A Beginner’s Guide to the Pixie Acquisition and Analysis Software System - Laughing Conqueror (PAASS-LC)

JJ van Zyl

April 2018

First Things First

1. Overview

The PAASS-LC DAQ consists of two main parts, the **poll2** digital acquisition backend, and the **utkscan** analyser (or unpacker, or sorter).

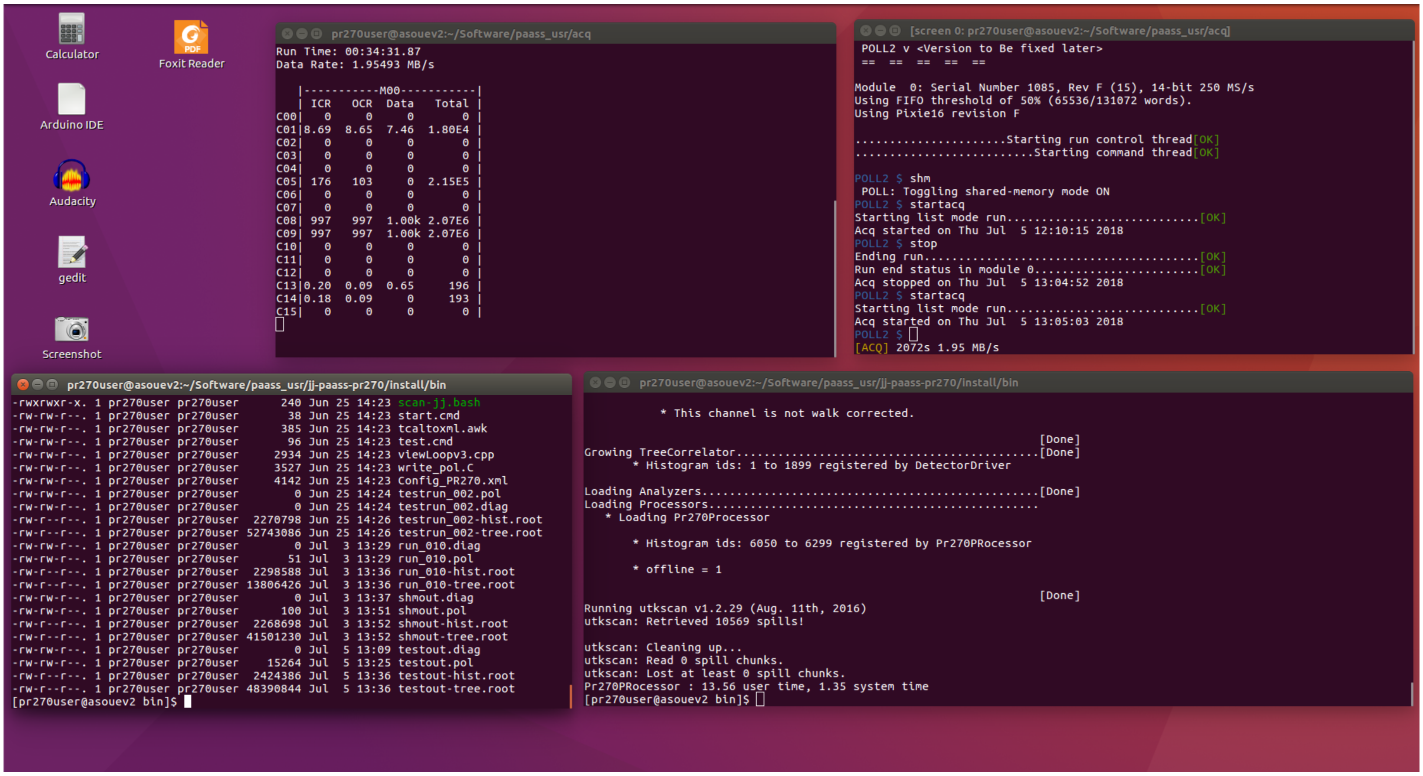
poll2

The poll2 software will capture the input channel signals, produce a digital trace of the signal, and run two main filters over this trace to get the signal timestamp and the signal energy. The **fast** or **trigger** filter finds the timestamp where the filter function crosses the preset trigger threshold (see “Arrival time” below). The **slow** or **energy** filter finds the energy amplitude of the signal, based on the channel and module parameters set by the user (see below). The poll2 interface will produce an **.ldf run file** when using the list mode for acquisition, but can also produce an **mca** output file if requested.

utkscan

For unpacking the .ldf file from poll2, we use **utkscan**. This unpacking (or sorting) of the ldf runfile is done with the help of a **Config.xml** file and a selection of experimental **processors** defined by the user. The Config.xml file contains the basic channel mapping and PIXIE16 module firmware and frequency setup (in order for the correct unpacking of the .ldf header file). The experimental processor(s) contains the basic event processing, logics and calculations needed by the experimenter to make sense of the acquired data - the building of the event tree.

I usually run the DAQ and analyser with the following arrangement:



- Top right terminal runs poll2 using a screen session. Screen sessions are used so that all logged in users have access to the single runcontrol terminal. This also allows for easy recovery if ssh connections are dropped.
- Top left terminal runs monitor to see rates if poll2 is run in shm mode.
- Bottom right terminal runs utkscan for analysis.
- Bottom left terminal runs plotting routines, e.g. RootHistogramViewer

2. Install the PXI crate and associated hardware

a. Note the jumper settings on each module for cross-module clock synchronisation (see the XIA manual for the PIXIE16 module)

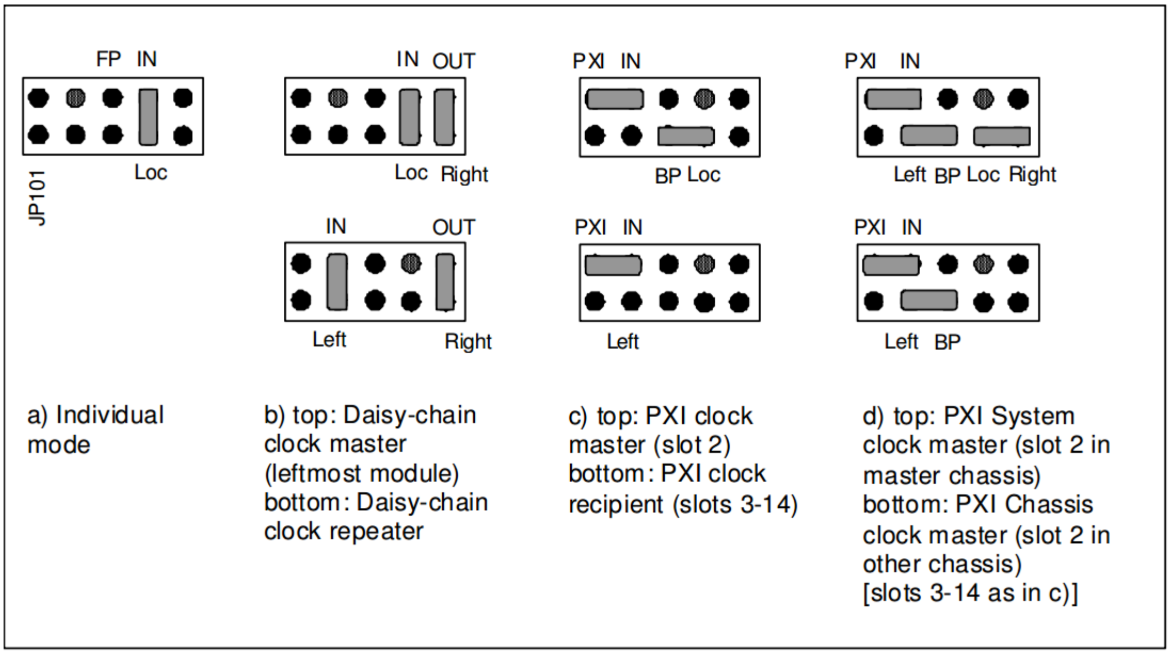


Figure 1: Jumper settings for different clock distribution modes. (a) An individual module uses its own local clock. (b) In a group of modules, there will be one daisy-chained clock master in the leftmost position and several repeaters. (c) Alternatively, the PXI clock distribution path can be used, with the module in slot 2 as the PXI clock master and the other modules as PXI clock recipients. (d) In multi-chassis systems, the module in slot 2 in the clock master chassis should be configured as shown in the top panel and the modules in slot 2 in all other chassis should be configured as shown in the bottom panel. Modules in any other slot should be configured as shown in the bottom panel of (c). The local clock can be substituted by an LVDS clock input on the front panel using the “Loc” pin instead of the “FP” pin and setting jumper JP5 to “Clk”. (Image from [XIA]).

3. Update the OS to the latest software releases

Make sure your OS is up to date. It is recommended to use CentOS 7.5 or Ubuntu 16/18.

- gcc-v5+ (since CentOS 7 comes with gcc-v4.8, I had to remove it (yum remove gcc -y) before installing the gcc-v8.2.0 from source according to this web: https://jdhao.github.io/2017/09/04/install-gcc-newer-version-on-centos/)
- cmake-v3+
- root-v6+
- See the pre-requirements in the paass-lc wiki

4. Install the XIA and PXI firmware

Note (Oct 2018) : When installing plx-api from Stan’s git repo, the PlxApi.a is not created and therefore not linked to the created libPlxApi.a file. The troubleshoot in the wiki also did not make sense.

So I took the latest release of the PLX software and unzipped into the folder /opt/plx, i.e.

Build the PLX API

Step 1 - Setup your environment

mkdir -p /opt/plx
mkdir -p /opt/plx/plx_v7_10
cd /opt/plx/plx_v7_10

Unzip the bundle into the plx version folder (e.g. plx_v7_10) created above:
unzip PLX_SDK_Linux_v7_10_Final_2013-08-09.zip
tar xvf PlxSdk.tar
ln -s plx_v7_10 /opt/plx/current

Step 2 - Compile the API Library

We then need to compile the API:
cd PlxSdk
export PLX_SDK_DIR=$PWD
make all

The directory structure will look like this:

/opt/plx:
current (-> plx_v7_10)
plx_v7_10
Documentation
PlxSdk
Bin
Driver
Include
Makefile
Makefiles
Samples
PlxApi
libPlxApi.a
PlxApi.a -> libPlxApi.a

Step 2.b - Create a symbolic link to the PlxApi.a file (as above):

(This step was not necessary for the PLX_SDK_Linux_v7_10_Final_2013-08-09.zip, as the 'make all' step above created the PlxApi.a file in the Library folder already (see also PAASS-LC installation notes here. If not, do this next step)

cd /opt/plx/plx_v7_10/PlxSdk/PlxApi/
ln -s libPlxApi.a PlxApi.a

Step 3 - Add the PLX_SDK_DIR to your environment

This step should **NOT** be done as super user, but as the user that you will be taking data with.
echo "export PLX_SDK_DIR=/opt/plx/current/PlxSdk" >> ${HOME}/.bashrc

Build the Driver

Step 1 - Build

Pixie-16 uses the PLX 9054 Chip Set.
cd Driver
./builddriver 9054

Build the XIA Pixie-16 API

Step 1 - Obtain the source code

cd ${HOME}
git clone https://github.com/spaulaus/xia-api.git
cd xia-api
mkdir build
cd build

(Here, I had to run the following commands to tell cmake where to look...)
export LD_LIBRARY_PATH=/usr/local/lib64/:$LD_LIBRARY_PATH

cmake ../ -DCMAKE_INSTALL_PREFIX=/opt/
make install -j4
After completing step 1, /opt/xia should have the following structure:

/opt/xia:
- api
- include
- share
  - slot_def.set
  - pxisys
    - pxisys_14e.ini pxisys_14w.ini pxisys_8.ini pxisys.ini
- firmwares
  - 2016-04-11-14b100m-general
    - dsp
      - Pixie16DSP_revfgeneral_14b100m_r34689.ldr
      - Pixie16DSP_revfgeneral_14b100m_r34689.var
      - Pixie16DSP_revfgeneral_14b100m_r34689.lst
    - firmware
      - fippixie16_revfgeneral_14b100m_r29406.bind
      - syspixie16_revfgeneral_adc100mhz_r33338.bin
  - 2016-04-11-14b250m-general
    - dsp
      - Pixie16DSP_revfgeneral_14b250m_r34455.ldr

Pixie16DSP_revfgeneral_14b250m_r34455.var
Pixie16DSP_revfgeneral_14b250m_r34455.lst
firmware
fippixie16_revfgeneral_14b250m_r33332.bin
Syspixie16_revfgeneral_adc250mhz_r33339.bin

The drivers and firmware files are obtainable directly from XIA. Before you start, make sure the prerequisite third-party packages are installed on your DAQ PC. You can find the list of required packages in the **PAASS-LC Wiki** instructions and sections PLX SDK and XIA Pixie-16 API.

The Pixie16 system makes use of PLX 9054 chips for communication. A driver and SDK need to be built and installed for PAASS-LC implementation. We recommend PLX version 7.1.0.
Carefully follow the outlined procedures. (See also my notes here...). I have found that I need to update my gls libraries as well.

The **firmware version** of the specific PIXIE16 module is the last 5 numbers in the DSP firmware file, i.e.
Pixie16DSP_revfgeneral_14b250m_r34455.ldr --> firmware version for 250 MHz mod is **34455**
Pixie16DSP_revfgeneral_14b100m_r34689.ldr --> firmware version **34689** for the 100 MHz mod

Note: Be aware that the crate needs to be connected to the computer and powered on prior to booting the computer to guarantee proper communication.

5. Install POLL2 (digital data acquisition software, PAASS-LC)

The POLL2 software is the acquisition part of the PAASS-LC package and used to acquire data with the XIA Pixie16 digital system. **Follow the PAASS-LC Wiki** instructions (https://github.com/spaulaus/paass-laughing-conqueror/wiki/Installation) and the section PAASS-LC.

We typically have two main installations of PAASS-LC:
1. Base installation for XIA setup and POLL2 acquisition (su user)
2. User installation for the UTKSCAN analysis software part

This will deal with the base installation of POLL2 for acquisition.

Installing PAASS-LC from GitHub repository

Below is my procedure of the instructions from the wiki, just to highlight the steps I found important.

We typically make a clone of this git-repository and store it in **/root/paass-lc**. This location ensures that standard users will not have access to the source and therefore cannot make any changes during an experiment.

$ su
$ git clone https://github.com/spaulaus/paass-lc.git **/root/paass-lc**
$ git checkout master

$ git pull   (This gets the information from the remote repository without applying any changes to the current branch) **--- Git asked me to first do: git branch --set-upstream-to=origin/master**

We first need to make a build directory to keep the build files separate from the source code.
$ mkdir /root/paass-lc/build
$ cd /root/paass-lc/build

PAASS-LC can be configured in a number of different ways using various CMake options. We recommend the optional ccmake package as it reduces the chance of making mistakes. Using CCMake we can simply invoke it with ccmake .. from the build directory (see the basic parameters in ccmake here...):

$ ccmake ../

...but we prefer the following command which sets the install prefix.

$ ccmake ../ -DCMAKE_INSTALL_PREFIX=/opt/paass-lc

- **Make sure the ROOTSYS variable points to the ROOT installation directory that contains the “bin” directory, e.g.**
  ROOTSYS = /home/Software/root/root-v6-12/build

- **Make sure the PLX_LIBRARY_DIR points to the correct folder, e.g.**
  /opt/plx/current/PlxSdk/PlxApi/Library

(more complete parameter values can be found in /root/paass-lc/build/CMakeCache.txt file...)

Currently we also have two base installations, one for poll2 with utkscan, and one for poll2 + pacman with utkscanor - this is installed by switching the HRIBF flag in ccmake ON. When using this HRIBF build, one must load the pixieSuite-hribf module (and unload the pixieSuite module), e.g.

$ module unload pixieSuite
$ module load pixieSuite-hribf

Note: When I first ran the make install command below, make gave me the error
/usr/bin/ld: cannot find -lPlxApi
collect2: error: ld returned 1 exit status
make[2]: *** [Acquisition/MCA/source/mca] Error 1

To fix this, I had to create a symbolic link to PlxApi.a, named libPlxApi.a, i.e.
ln -s /opt/plx/current/PlxSdk/PlxApi/Library/PlxApi.a /opt/plx/current/PlxSdk/PlxApi/Library/libPlxApi.a

In previous plx installations the file libPlxApi.a was also created, and we had to create a link from this libPlxApi.a file to PlxApi.a. In this installation (plx-v7.10), no such libPlxApi.a file was made, so I had the create it as a symbolic link above!

After CMake has completed generating the make file used for compilation we simply need to compile the package using the command make. To speed things up we can specify to use multiple threads for compilation with the optional argument -j [numberOfThreads].

$ make clean && make install -j8

The CMake files are capable of producing a guess at the configuration files need to run the poll2 and associated programs. These files are installed to ${CMAKE_INSTALL_PREFIX}/share/config and can be created by typing the following in the paass-lc/build directory.

$ make config
(note, make changes to the pixie.cfg file according to the example below:)
(what I do, is to copy current working pixie.cfg file to the /opt/paass-lc/share/config folder)

Adding PAASS-LC to the Environment
Finally, after configuration and installation is complete PAASS-LC needs to be added to the path. Traditionally this has been done by modifying your environment via a .bash_profile for example. We suggest instead you make use of environment-modules. Be sure to log out and log back in after installation of environment-modules.

After compilation is completed a module file has been created in /opt/paass-lc/share/modulesfiles/pixieSuite and we need to create a symbolic link to the module search path.

$ ln -s /opt/paass-lc/share/modulefiles/pixieSuite ${MODULESHOME}/modulefiles

We can then use the following command to source the PAASS environment

$ module load pixieSuite

PIXIE16 setup directory structure:

At this point the directory structure should look something like this:

/opt/paass-lc:
- bin (this is where the ccmake and install puts the executables, e.g. poll2 etc.)
- include
- lib
- share

/root/paass-lc: (this is where the Git repository source codes are saved)
- Acquisition
- Analysis
  - Resources ScanLibraries Scanor Utilities
  - Utkscan
    - analyzers core experiment processors share
- build
- Core

Doc
install
Resources
Share
ThirdParty

Set the module configuration - pixie.cfg and slot_def.set

The procedure is important. First make a few folders as user:

home/user/paass-lc/acq
home/user/paass-lc/analysis

Three files should be in the folder where the acquisition is run from, e.g. **.../paass_usr/acq**. These are: **pixie.cfg, slot_def.set** and **default.set**.

Note that poll2 (the command) is run from the /acq directory, but the actual executable resides under /opt/paass-lc/bin directory where it was installed (i.e. type: which poll2).

Much of the information about the Pixie setup is given through the use of a configuration file (typically **pixie.cfg**). This is simply a file which has a list of lines with a tag followed by blank space and a value which the interface interprets.

The pixie.cfg file contains a list of lines with a tag followed by blank space and a value which the interface interprets... see below. Lines starting with # are ignored, and all file paths are relative to the **PixieBaseDir** unless they begin with a '.' in which care they are taken relative to the current directory.

pixie.cfg

```
# Global Tags
PixieBaseDir      /opt/xia/current
CrateConfig       test/pxisys.ini
SlotFile          ./slot_def.set
DspSetFile        ./default.set
DspWorkingSetFile ./default.set

# Module Tags
ModuleType        14b100m-revf
ModuleBaseDir     /opt/xia/2016-04-11-14b100m-general
SpFpgaFile        firmware/fippixie16_revfgeneral_14b100m_r29406.bin
ComFpgaFile       firmware/syspixie16_revfgeneral_adc100mhz_r33338.bin
DspConfFile       dsp/Pixie16DSP_revfgeneral_14b100m_r34689.ldr
DspVarFile        dsp/Pixie16DSP_revfgeneral_14b100m_r34689.var

ModuleType        14b250m-revf
ModuleBaseDir     /opt/xia/2016-04-11-14b250m-general
SpFpgaFile        firmware/fippixie16_revfgeneral_14b250m_r33332.bin
```

```
ComFpgaFile       firmware/syspixie16_revfgeneral_adc250mhz_r33339.bin
DspConfFile       dsp/Pixie16DSP_revfgeneral_14b250m_r34455.ldr
DspVarFile        dsp/Pixie16DSP_revfgeneral_14b250m_r34455.var
```

A note on **CrateConfig**:
The **pxisys*.ini** file tells the PXI bus how to communicate with your crate. The version of the file you want depends on the type of crate you have. For example, if you have a 14 slot crate with a Weiner backplane then you want to use pxisys_14w.ini. These files should be supplied by XIA.

Notes on the **default.set** file:

- You do not need to create a default.set file. The one I gave you will be sufficient. You will be able to modify that file to your heart's content. In fact, that has been carried between several experiments. I suggest storing a backup copy of the file somewhere, so that it can be recovered in the event of another HD issue ;). I have about 10 copies of that file with various settings.
- You can use `set2ascii -h` or `set2root -h` for help on how to use those programs.
- There's a legacy "feature" in pixie.cfg. These two lines should be

  ```
  DspSetFile         ./default.set
  DspWorkingSetFile  ./default.set
  ```

  There's no longer any reason to have them as separate files.

The **SlotFile**, **slot_def.set** creates a slot map which is used in the Pixie initialisation prior to booting. The slots and modules are those listed in the pixie.cfg file above. The slot_def.set file has the format:

slot_def.set

```
2  Modules       ("2" is the number of modules in the crate)
2  Mod 0         ("2" and "3",… are the slot numbers in the crate where those
modules are in)
3  Mod 1
4  Mod 2
…
```

Only the first line is considered, starting with the number of modules and then the slot in which each listed module is plugged in. Extra numbers are ignored. This creates a slot map which is used in the Pixie initialization prior to booting. If a line begins with an asterisk, then the alternative values of the firmware configuration is used from the **pixie.cfg** file.

My current setup, having only one module in the first slot, is :

```
1  Modules
2  Mod 0
```

The **pxisys*.ini** file mentioned in the pixie.cfg file tells the PXI bus how to communicate with your crate. The version of the file you want depends on the type of crate you have. For example, if you have a 14 slot crate with a Weiner backplane then you want to use pxisys_14w.ini. These files should be supplied by XIA.

Startup of POLL2 DAQ

1. Switch ON PIXIE16 crate with modules in.
2. Boot-up DAQ PC, asouev2.tlabs.ac.za:
    user: pr270user
    passw: xxxxxxxx

Ideally everything should be connected and loaded. You can simply start the polling system with the poll2 command:

To run the poll2 polling software:
(This command is run from the folder where the **pixie.cfg, slot_def.set** and **default.set** files are.)

```
~/Software/paass_usr/acq/$: poll2
```

- you may need to run this command several times until the crate connects to the PCI card…
- you can start poll2 with -f flag to bypass full bootup, e.g. `$: poll2 -f`

See poll2 help for further instructions.

To access asouev2.tlabs.ac.za remotely from within local network:

$: ssh - XY pr270user@asouev2.tlabs.ac.za
(the -X/Y flag is needed for, amongst others, X11 forwarding which is used by the Xterm to display stuff, e.g. Gnuplot figures etc. The -Y is needed for permission for some login clients - I had to use -Y when connecting from the ion source PC.)

If you need to reload or unload the PLX driver (to be done after every reboot of PC):

(PIXIE crate needs to be ON before booting PC!)
(Ideally I have the startup script already enabled at boot, as per wiki ... but if not...)
login as su:
$: su
Password: xxxxxxxx

$: PLX_SDK_DIR=/opt/plx/current/PlxSdk /opt/plx/current/PlxSdk/Bin/Plx_load 9054

Wait for message below:
Install: Plx9054
  Load module......... Ok (Plx9054.ko)
  Verify load......... Ok
  Get major number.... Ok (MajorID = 244)
  Create node path.... Ok (/dev/plx)
  Create nodes........ Ok (/dev/plx/Plx9054)

Loading the PAASS-LC environment (This should happen automatically since it was added to the environment - see Wiki), or if you need to load a different poll2 installation e.g. poll2-hribf with pacman:

module load pixieSuite (or module unload pixieSuite)

Notes:

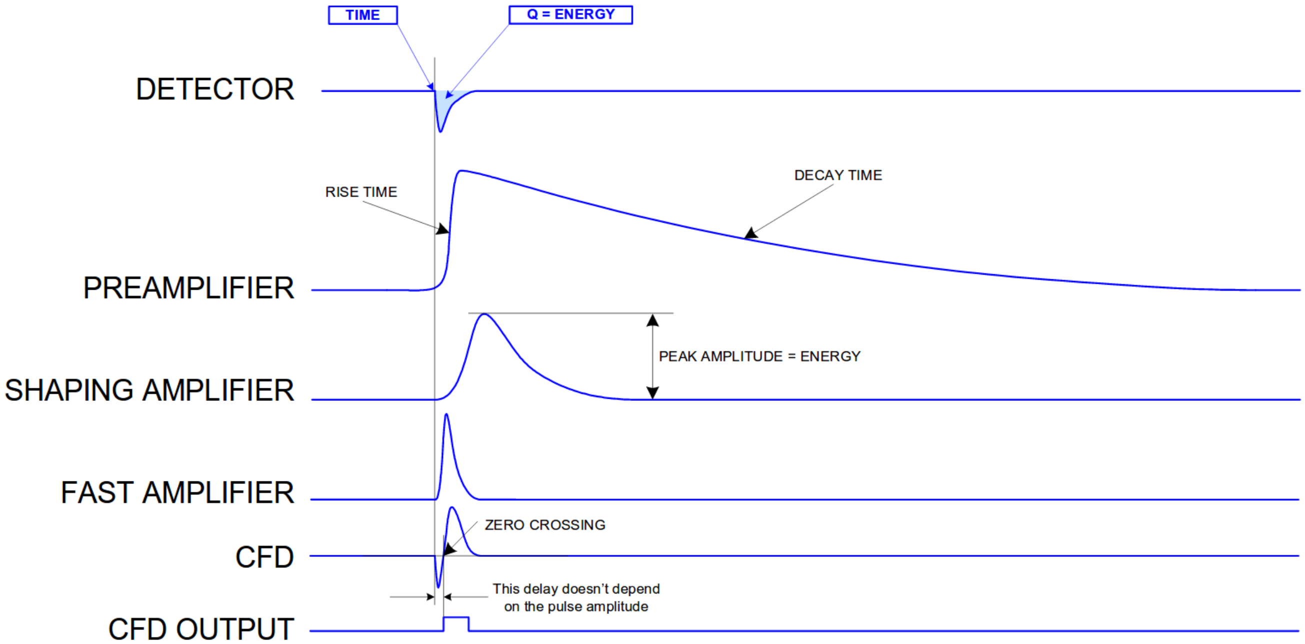
- The firmware version of the specific PIXIE16 card is the last 5 numbers in the DSP firmware file, i.e. Pixie16DSP_revfgeneral_14b250m_r34455.ldr --> firmware version for 250 MHz card is 34455 and for the 100 Mhz card: Pixie16DSP_revfgeneral_14b100m_r34689.ldr --> 34689.

- Multiple interfaces with poll2 are prevented from being opened on the same computer. This prevents hard crashes and safeguards the data acquisition from outside tampering. However, in the event of an error/crash the file which locks the interface (/tmp/PixieInterface) can be removed manually.

Basic signal input and channel parameter settings

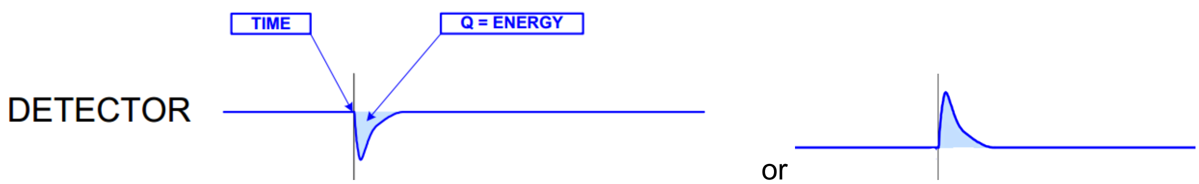
The following is a basic discussion on digital signal processing. Much is taken from the XIA PIXIE16 manual as well as the CAEN document on signal processing.

The traditional processing of analogue detector signals is illustrated in the figure below. Analog-to-digital-converters (ADC) measures the analog voltage value of the amplified signal and stores it as a number.



(Image from [CAEN]).

In the case of modern digital data acquisition systems, the detector output signal (top-most signal in the image above) is digitised by a Waveform Digitizer which operates similar to a digital oscilloscope.



When a trigger occurs, a certain number of samples is captured and saved into one memory buffer. This can be seen from the image below:

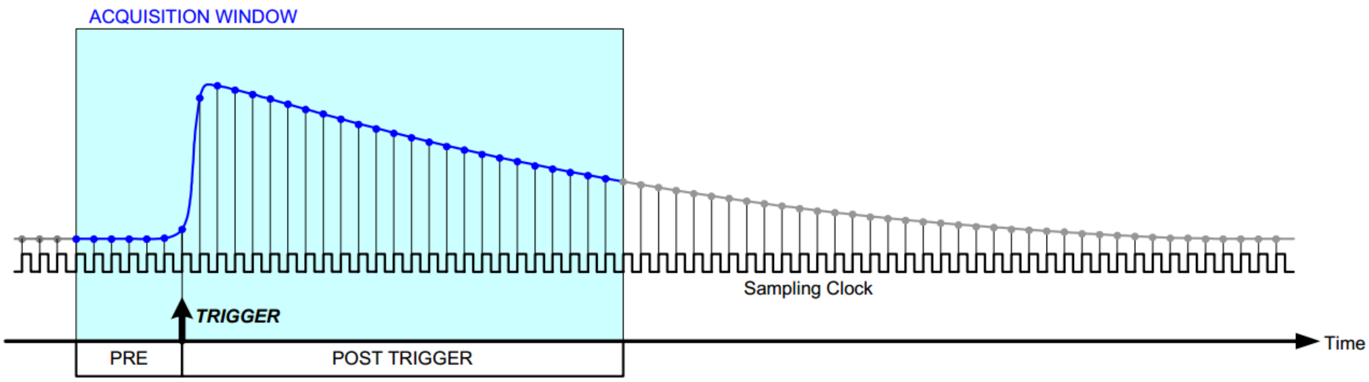
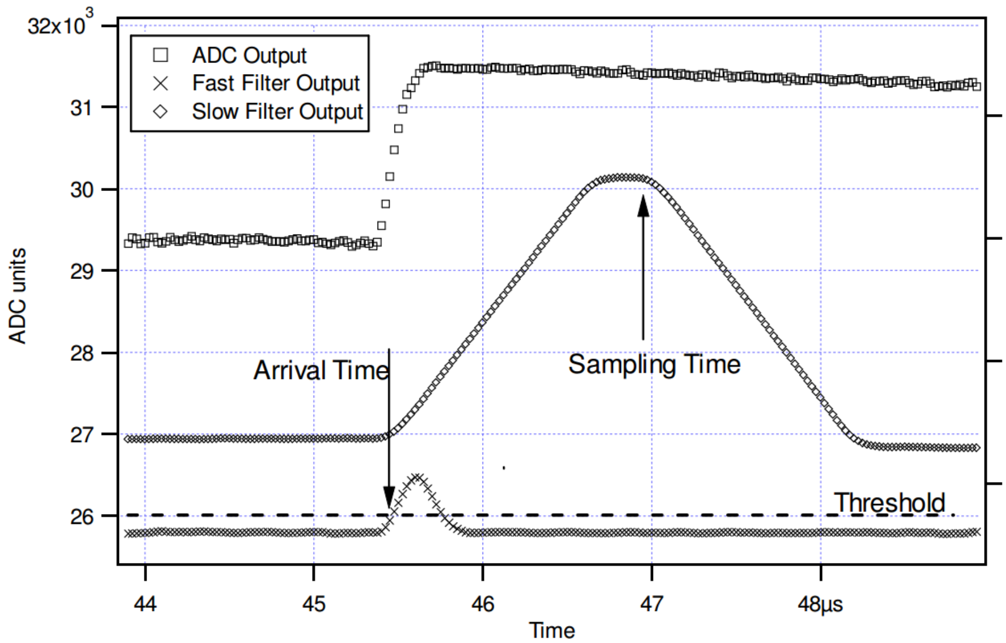


Fig. 1.6: Signal digitization and acquisition window defined by the trigger

(Image from [CAEN]).

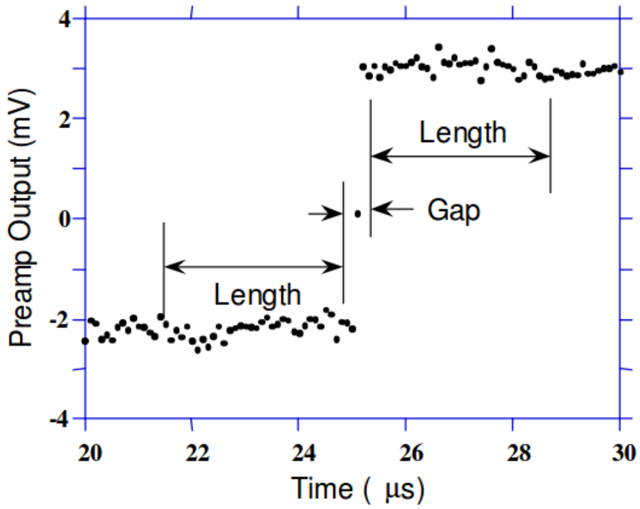
Once this **trace** of the signal is collected, further filtering and processing can be done at fast speeds. See the trace of the detector signal (ADC Output) in the figure below:

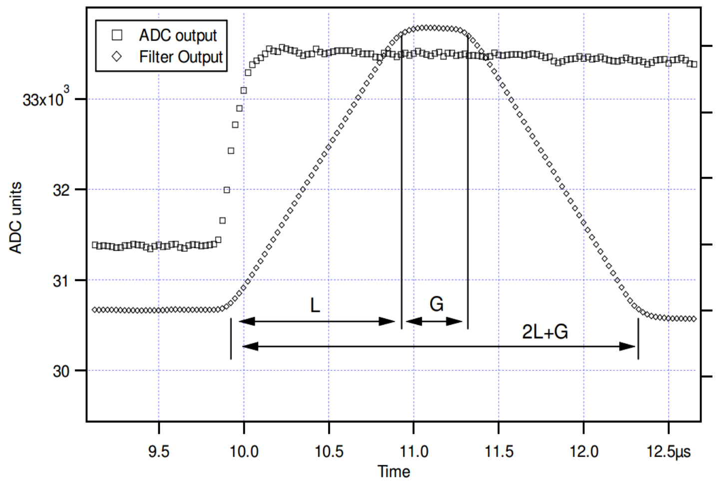


(Image from [XIA]).

In the above signal example, I consider a rise time (τ_r) of the input signal to be about 0.1 µs. The decay time (TAU, τ_d) in this example could be taken as about 25 µs.

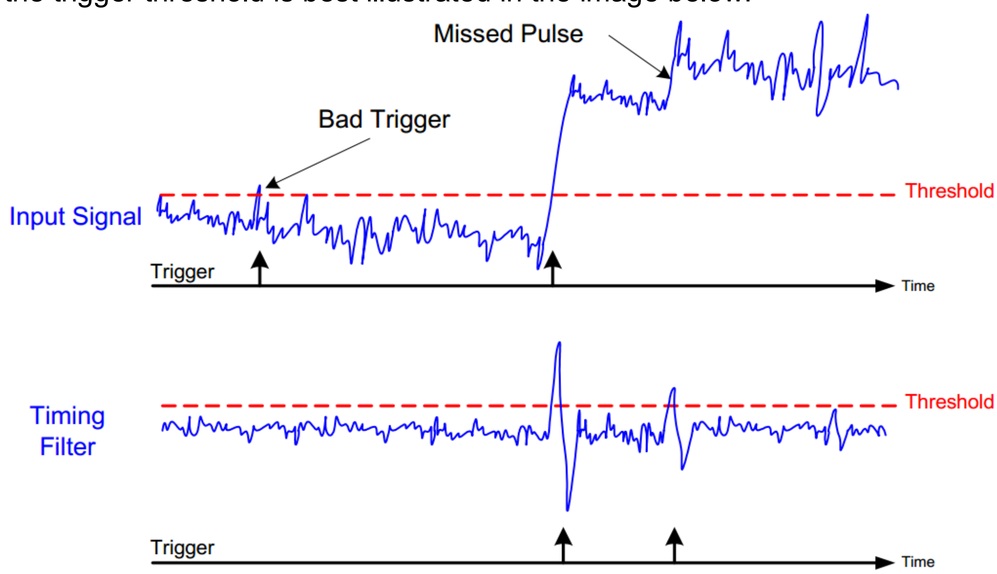
For the slow energy filter, we can use the following as a guide to setting the filter parameters. The typical length of the ENERGY_FLATTOP (G or Gap) of the slow energy filter is taken as at least the same as the signal rise time to about 3 times that, i.e. 0.35µs. The ENERGY_RISETIME (L or Length) of the slow filter is taken as about 4 times the FLATTOP, i.e. 1.2 µs.





(The figure left is from [CAEN], and the right is from [XIA].)

For the fast trigger filter, we can set the TRIGGER_RISETIME at about the same as the rise time, i.e. 0.1 µs. The event timestamp is taken at the time where the TRIGGER_THRESHOLD intercepts the fast trigger filter. The function of the trigger threshold is best illustrated in the image below:



(Image from [CAEN]).

For the trace, we can set the TRACE_LENGTH (the time over which the trace is captured) at about 2L + G. i.e. in this example, ~2.7 µs. A too long trace length will average out any subsequent signals that fall within that trace length. It is suggested that TRACE_DELAY be set to about 25% of the TRACE_LENGTH, i.e. in this case ~0.65 µs.

When setting up the PIXIE16 channel parameters, the user should consider the following:

- The energy resolution (i.e. how wide / narrow the peak of interest is) - this involves a proper choice of the slow [energy] filter settings, i.e. ENERGY_RISETIME (length L) and ENERGY_FLATTOP (gap G). For the energy (slow) filter, you need to balance throughput and energy resolution. Longer filters (i.e. long ENERGY_RISETIME) will mean that you're increasing the likelihood of pileup and effective deadtime of the system. A shorter rise time may lead to worse energy resolution. The likelihood of pileup is (if memory serves) proportional to the filter length and the rate. The suggestion for the energy filter is to keep it as short as possible while maintaining the desired energy resolution. A shorter energy filter means that you're measuring less of the signal and may calculate a "smaller" energy.

- If you make TAU bigger it means that the adjustment to the energy due to overlapping pulses is larger, and thus you will also have a smaller energy. According to [XIA] the signal decay time (TAU) is the most critical parameter for the energy computation. It is used to compensate for the falling edge of a previous pulse in the computation of the energy.

- The dynamic range (i.e. how much of your spectrum is filled up) - note, if the baselines of the signals are adjusted too high (e.g. 1600 instead of 600), you are losing a lot of the ADC range - see viewBaseline. Shifting the baselines lower will optimise the ADC range usage. The same applies for gains. Gain levels set too low means you are not using the full dynamic range of the ADCs.

- The throughput (i.e. how fast can I shove signals through the filter) - the longer the filter times, the slower the signal analysis, i.e. aim for the shortest realistic filter ranges.

- Pileup (i.e. how likely I am to have another signal arrive while my filter is still getting calculated) - same as above.

The user will have to set the following channel parameters from the standard set provided, or based on the input signals on a scope from within poll2 (see here...). These parameters and typical values are:

Pixie Parameter Name	Plastic Scint	Ge (Clover)	NaI (***)	TAC	Logic	CsI	LaBr3	Si
ENERGY_FLATTOP	0.048	0.768	0.512	1.024	0.096	0.048	0.045	1.982
ENERGY_RISETIME	0.128	4.864	1.15	1.024	1.008	0.48	0.576	6.114
SLOW_FILTER_RANGE	1	4	4	4	1	1	1	4
TAU	0.01	46	50	1000	1000	0.63	0.04	40
TRACE_DELAY	0.2	--	0.25	--	--	--	0.2	--
TRACE_LENGTH	0.496	--	1	--	--	--	0.496	--
TRIGGER_FLATTOP	0	0	0.25	0.504	0	0.072	0.016	0.104
TRIGGER_RISETIME	0.104	0.56	0.2	0.104	0.104	0.104	0.08	0.4
TRIGGER_THRESHOLD	4	60	20	5	60	25	100	200

**All times in μs.

*** These are my current NaI signal parameters based on the input detector signal with a tau ~ 80 μs.

The ENERGY_FLATTOP is usually more or less matched with the raw rise time of the signal.

See the ^{152}Eu gamma spectrum in Fig. 1 below. The blue line is with TRIGGER_RISETIME set to 0.6 μs (the actual signal has a rise time of about 0.1 μs). The black line is with TRIGGER_RISETIME set to a smaller 0.4 μs, and the red line with the TRIGGER_RISETIME set to 0.2 μs. Clearly, the smaller trigger risetimes cut into the lower energy signals. This is to be expected, since the trigger filter will not go far enough up the rising edge of the signal before being rejected by the trigger threshold, i.e. the trigger filter will be below the threshold and thus be rejected - see also the two TraceFilterJJ images below.

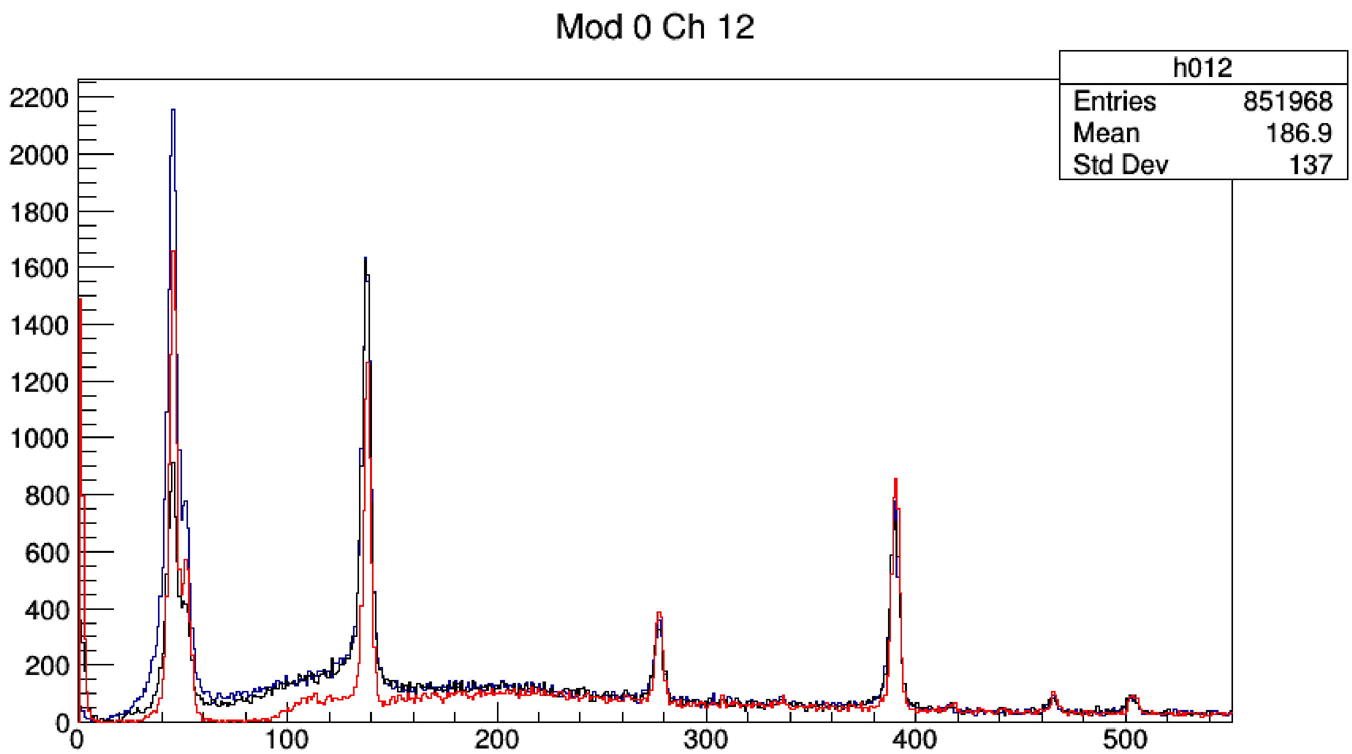


Fig.1: ^{152}Eu gamma spectrum with single Ge clover segment. The blue line is with TRIGGER_RISETIME set to 0.6 μs (the actual signal has a rise time of about 0.1 μs). The black line is with TRIGGER_RISETIME set to a smaller 0.4 μs, and the red line with the TRIGGER_RISETIME set to 0.2 μs.

TraceFilterJJ

I use the testing code TraceFilterJJ (adapted from the Stanley Paulauskas' TraceFilter code, https://github.com/spaulaus/TraceFilter). I added a loop of randomised signals around a set signal voltage to simulate the resolution of a detector, and see the effect of the filter parameters on the output trigger and energy.

Test example:

The following sample shows a simulated ^{228}Th spectrum with 5 alpha peaks around 6-8 MeV. The input signals are about 200 mV in amplitude on a baseline of 350, with rise times of 0.200 μs and decay times of 200 μs. The resulting trace, trigger filter and energy spectrum is shown below.

The trigger filter has a risetime of 0.080 μs (flattop of 0). Note, where the threshold crosses the trigger filter, the timestamp of the event is recorded. The current threshold is set to 200 ADC units, well below the trigger filter maximum, but above the noise.

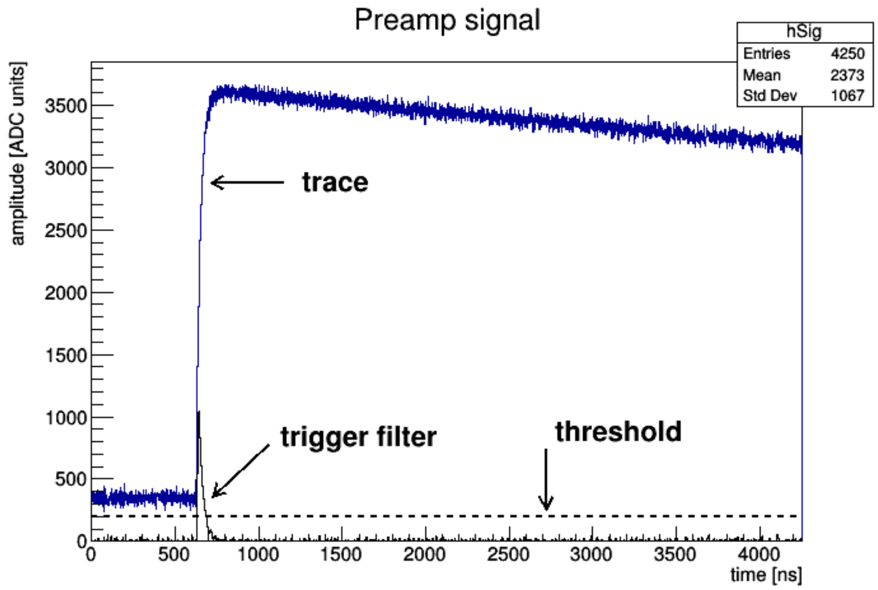


Fig. 2: This shows the resulting trace and trigger filter of the example discussed. The trigger filter has a risetime of 0.080 μs (flattop of 0). The current threshold is set to 200 ADC units.

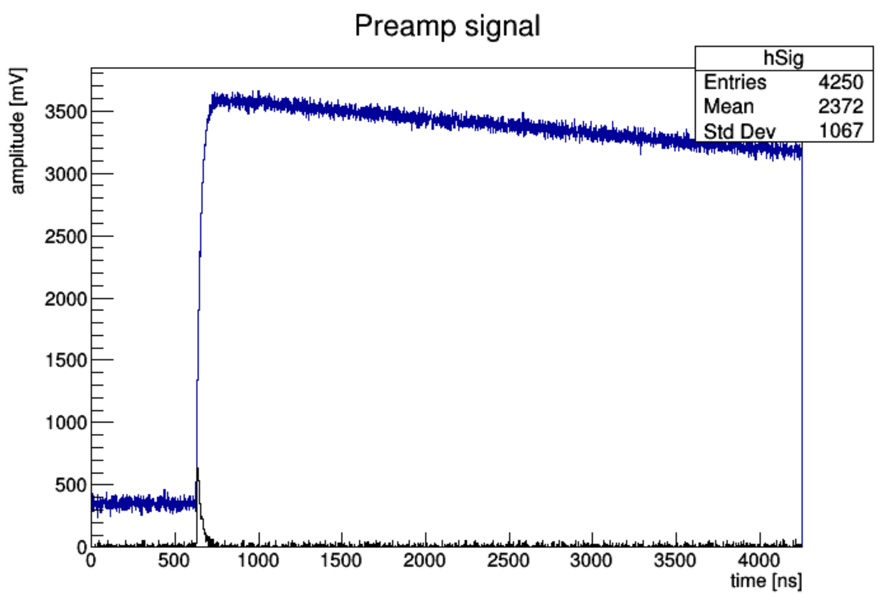


Fig. 3: Similar to the figure above, but we change the trigger risetime to a smaller 0.040 μs. The resulting filter has a smaller amplitude as it does not rise very high in the shorter 0.04 us. Chances are here greater that low energy signals may be rejected.

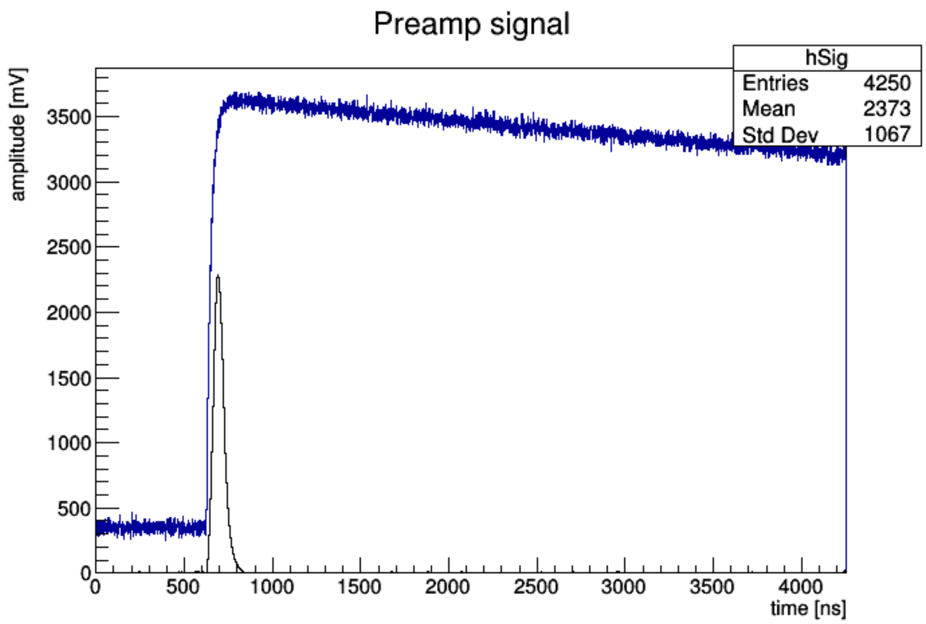


Fig. 4: Similar to the figure above, but we change the trigger risetime to a longer 0.40 μs, well above the threshold. The resulting filter amplitude is much larger because of the longer risetime.

The resulting energy spectrum is shown in Fig. 5 below.

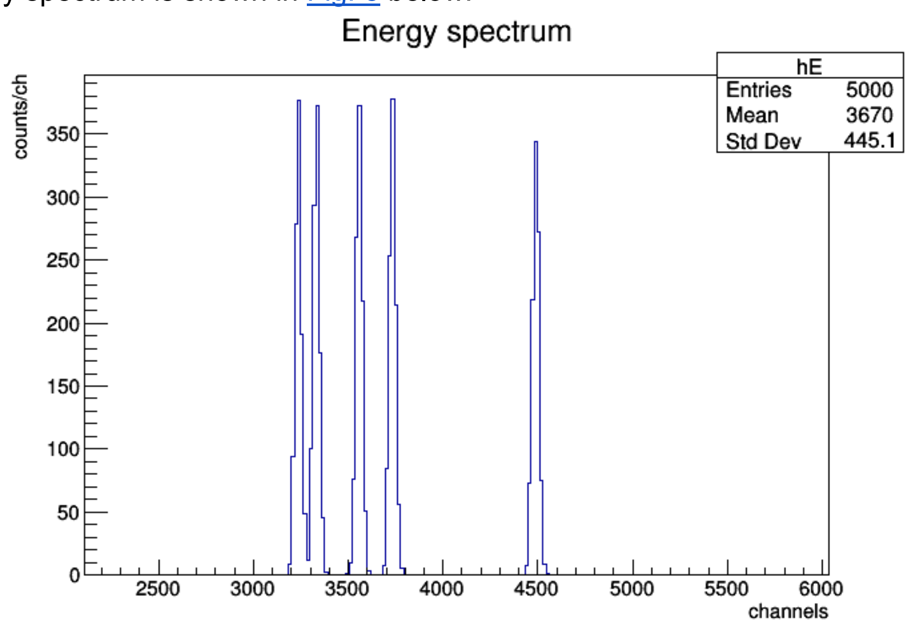
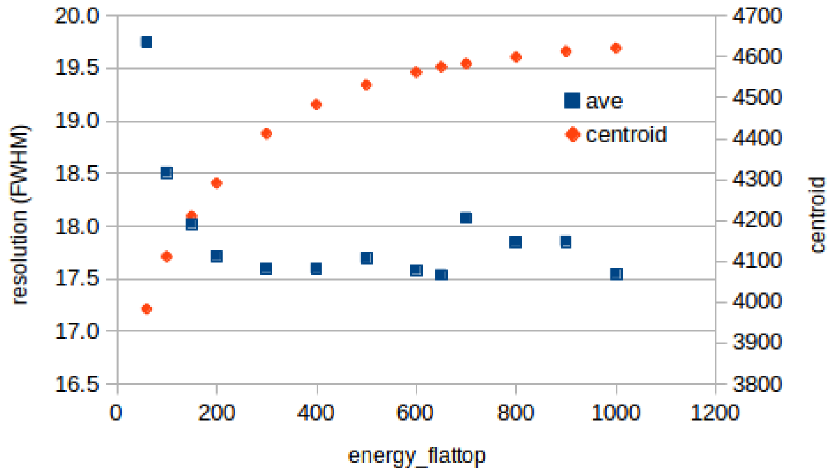
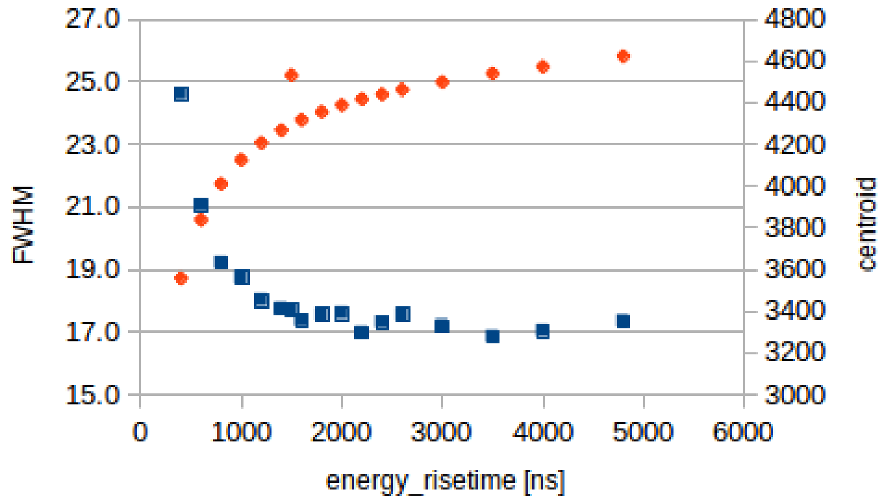


Fig. 5: Simulated ^{228}Th spectrum with 5 alpha peaks around 6-8 MeV. The resulting energy spectrum is shown, with energy filter parameters of risetime = 1.450 μs, flattop = 0.350 μs and TAU = 45 μs, giving a resolution of 0.9 %.

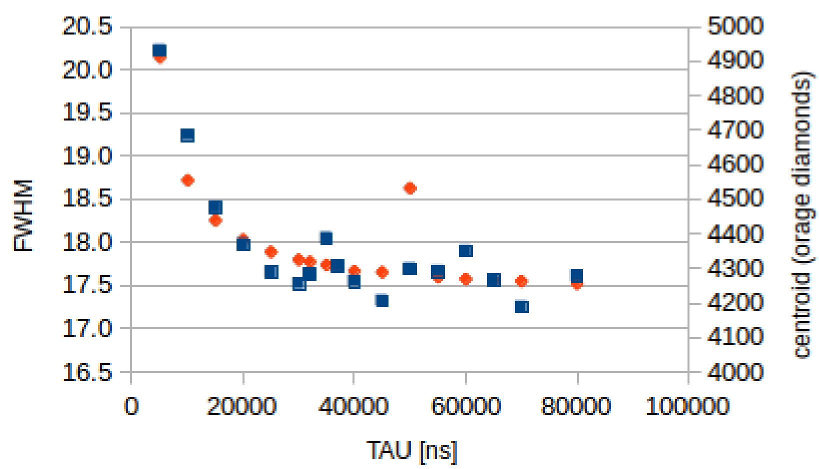
The above spectrum is from energy filter parameters of risetime = 1.450 μs, flattop = 0.350 μs and TAU = 45μs, giving a resolution of 0.9 %. If, e.g. we adjust the energy flattop to a smaller 0.100 μs, the resulting FWHM is worse, 1.1 %. The resolution as well as the centroid position is dependent on the energy flattop as seen in the graph below. Note, the signal risetime is around 200 μs, where the graph starts to flatten out.



Similarly, the resolution depends on the choice of energy risetime, as seen below,

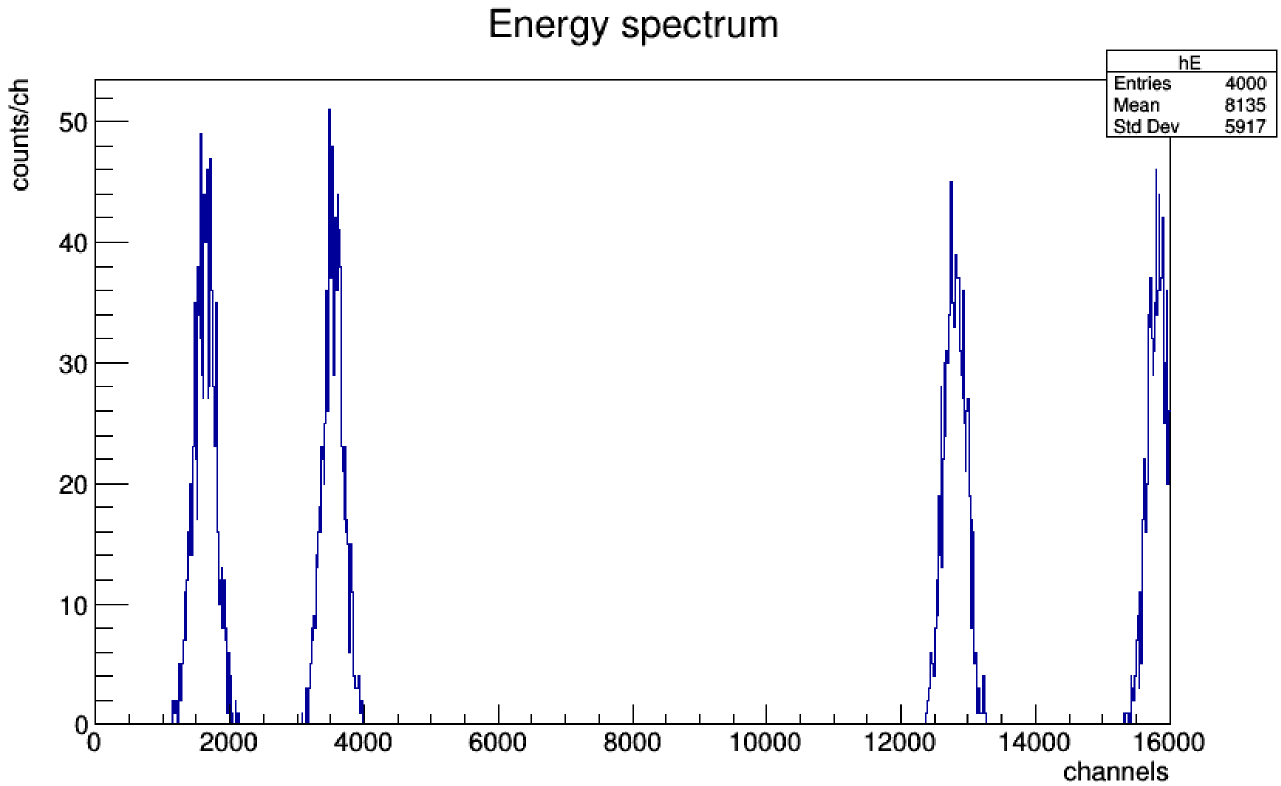


and also on the TAU value, as seen here.



We consider the example where a wide selection of different pre-amp signal amplitudes are given to see where on the raw “energy” spectrum x-axis the peak centroids are calculated.
In this example the four signal amplitudes are: 0.10 V, 0.22 V, 0.80 V and 0.99 V

The results of the energy filter is given in the MCA spectrum below. As I suspected, the pre-amp signal amplitudes are spread out over the 16000 channels of the MCA x-axis in a way proportional to the pre-amp amplitude out of the possible 1 V, i.e. a 1 V signal would give an energy output at channel no. ~16000.



Setting up external triggers and external time stamps

External timestamp

One option is to use the K600 spectrometer as external timestamp or as external fast trigger into the digital I/O ports of the four 250 MS/s Pixie16 Rev. F modules. If we use the front panel, then these are 5V TTL signals. When using the external timestamp, each channel signal's trigger will have two timestamps, one from local channel and one from external. You access these timestamps by calling the "GetExternalTimestamp()" method from "XiaData" in the processor.

Module Validation Trigger (MVT)

To set a single channel, e.g. CH0 in the master module (mod 0) as the source of the module validation trigger (MVT), do the following:

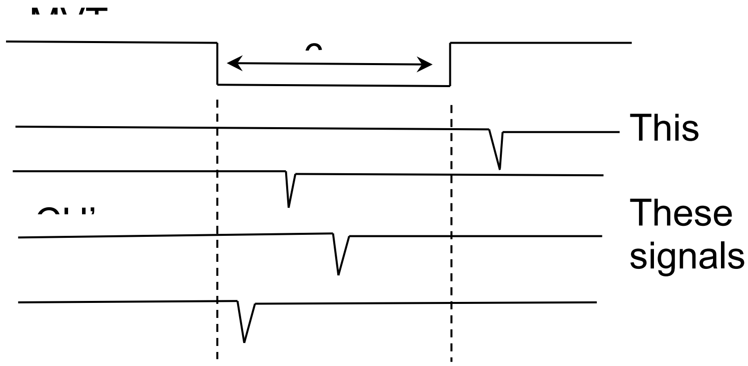
- Set **TrigConfig0[11:8]** = 0000 in mod 0 (this selects CH0 as the source of the internal validation trigger, Int_ValidTrig_Sgl)

- Set **TrigConfig0[27:26]** = 00 in mod 0 (this selects the external validation trigger, **Ext_ValidTrig_In**, above as the module validation trigger (MVT))

- Set **TrigConfig0[31:28]** = 0001 in mod 0 (this selects the source **Int_ValidTrig_Sgl** as the external validation trigger, **Ext_ValidTrig_In**)
- Set the Module Control Register B bit 6 (**MODULE_CSRB[6])** to 1 for mod 0 to enable the sharing of the **Ext_ValidTrig_In** from this module amongst the other modules in the crate.

The above choices gives the value of **TrigConfig0** = 2147516416.

The module validation trigger is then stretched long enough, e.g. 6 us to allow other channel signals from the clover detectors to fall within this period to be validated. Channel signals outside this window will be rejected.



To use the MVT as a validation trigger for all the channels, set the following bits in the **CHANNE_CSRA** parameter to 1 for each channel:

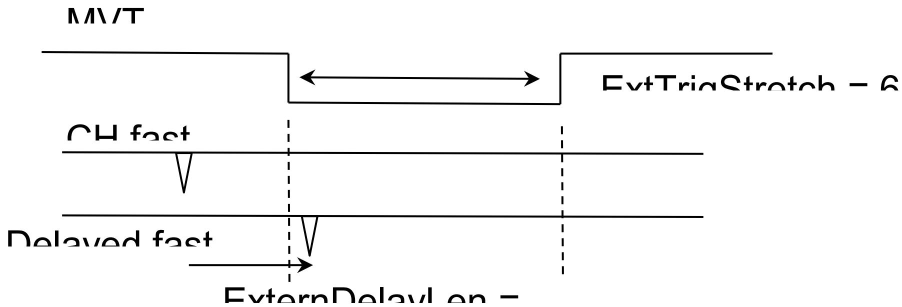
CHANNEL_CSRA[2] = 1 (Good channel)
CHANNEL_CSRA[11] = 1 (enable MVT for this channel)
CHANNEL_CSRA[21] = 1 (record external clock timestamp counter in event header - GetExternalTimestamp() in processor)

One will have to stretch the MVT to an appropriate length in order to allow local channel fast triggers to come in (as illustrated in the figure above), e.g. 6 us (for 250 MHz modules, 0.008 - 32.76 us). This is set in the **ExtTrigStretch** parameter for each channel, e.g.

pwrite -1 -1 ExtTrigStretch 6

To ensure the local fast triggers to come in after the MVT, one will have to delay the local triggers using the parameter **ExternDelayLen** (for 250 MHz modules, 0 - 4.088 us), e.g.

pwrite -1 -1 ExternDelayLen 0.5



Note, if a channel's fast trigger is used to generate the MVT, then that channel's ExternDelayLen should be 0!

Acquisition with POLL2

Start poll2 from the "acq" directory ~/Software/paass_usr/acq/… with:
$: poll2
 or
$: poll2 -f (for fast boot)

Wait for successful boot-up:
Module 0: Serial Number 1082, Rev F (15), 14-bit 100 MS/s
Module 1: Serial Number 1085, Rev F (15), 14-bit 250 MS/s
Using FIFO threshold of 50% (65536/131072 words).
Using Pixie16 revision F

Starting run control thread......................[OK]
Starting command thread..........................[OK]

[For poll2 commands, see here…]

Data can be acquired in both **mca mode** and **list mode**.

1. To save data in **mca mode**, i.e. save ROOT histograms of the channel energy, call the command:

POLL2: mca root <time> <outfile>

Where time is in sec. and <outfile> is the output root filename (without extension), e.g. mca_file.root. You can open this root file in ROOT the usual way, $ root -l mca_file.root, and draw the histos for each channel with h***->Draw(), where h*** refers to the 16 channels, i.e. h000 to h015.

2. In **list mode** the energy and time for each event are stored in a runfile (run_00*.ldf). We use UTKSCAN to unpack/ analyse this runfile into a ROOT event tree.

To write data to disk, call:
POLL2: run

This will save the run data in a file with default name “run_00*.ldf”. The file prefix can be changed by the user using the command **prefix [name]**, e.g:
POLL2: prefix runBaGeL
The next run will then be **runBaGeL_001.ldf**

You can change the runfile format to **pld** with the command **oform 1**:
POLL2: oform 1
The next run will then be **runBaGeL_001.pld**

To start the acquisition without saving anything to disk, call the command:
POLL2: startacq (and “stopacq” to stop)

To share the acquired buffer spill, start **shared memory mode** with
POLL2: shm
This will allow other processes such as *utkscan* to access the buffer.

Channel parameters

(in poll2, but not acquiring or writing to disk)

You can read the current module/channel parameters with:
pread <mod> <chan> <param>
e.g. pread 0 -1 CHANNEL_CSRA
(the “-1” for the channel number implies “all channels”, or just state the specific channel number, e.g. 5)

And write new parameters with:
pwrite <mod> <chan> <param> <val>
e.g. pwrite 0 -1 CHANNEL_CSRA 36

Here is a list of typical user parameters to be set before acquiring any signals.

Pixie Parameter Name			
	VANDLE	Plastic	
	Ge(Clo)	SiPm(Fast)	
	TAC	Logic	CsI
	LaBr3	BaF2	Si
ENERGY_FLATTOP	0.048	0.048	0.768
	0.048	1.024	0.096
	0.048	0.045	0.528
	1.982		
ENERGY_RISETIME	0.128	0.128	4.864
	0.064	1.024	1.008
	0.48	0.576	1.504
	6.114		
SLOW_FILTER_RANGE	1	1	4
	1	4	1
	1	1	1
	4		
TAU	0.01	0.01	46
	0.002	1000	1000
	0.63	0.04	0.9
	40		

TRACE_DELAY	0.344	0.2	--
	--	--	--
	0.2	--	--
TRACE_LENGTH	0.496	0.496	--
	--	--	--
	0.496	--	--
TRIGGER_FLATTOP	0	0	0
	0	0.504	0
	0.072	0.016	0.104
	0.104		
TRIGGER_RISETIME	0.104	0.104	0.56
	0.04	0.104	0.104
	0.104	0.08	0.4
	0.4		
TRIGGER_THRESHOLD	2	4	60
	40	5	60
	25	100	20
	200		

After gain changes or offset changes one can force PIXIE to adjust the channel offsets to a set fraction of the available ADC range by calling “adjust offsets <mod>”.

The **CHANNEL_CSRA** parameter’s bit control is as follows:

State	Bit	Bit Function
0	0	Fast trigger selection (local FiPPI trigger vs. external fast trigger from System FPGA)
0	1	Module validation signal selection (module validation trigger from System FPGA vs. module GATE from front panel)
1	**2**	**Good Channel**
0	3	Channel validation signal selection (channel validation trigger from System FPGA vs. channel GATE from front panel)
0	4	Block data acquisition if trace or header DPMs are full
1	**5**	**Input signal polarity control (default is NEG., switch ON if sig is POS. - THIS IS DIFFERENT FROM THE MANUAL, WHICH SAYS THAT THE DEFAULT IS POS...)**
0	6	Enable channel trigger veto
0	7	Histograms energy in the on-chip MCA
1	**8**	**Trace capture and associated header data**
0	9	QDC summing and associated header data
0	10	CFD for real time, trace capture and QDC capture
0	11	Require module validation trigger (triples)
0	12	Record raw energy sums and baseline in event header
0	13	Require channel validation trigger (doubles)
1	**14**	**Control input relay: =1: connect, gain factor x1; =0: disconnect, gain factor x1/4**
0	15	Control normal pileup rejection
0	16	Control Inverse pileup rejection
0	17	Enable “no traces for large pulses” feature
0	18	Group trigger selection (local FiPPI trigger vs. external group trigger from System FPGA)
0	19	Channel veto selection (front panel channel GATE vs. channel validation trigger)
0	20	Module veto selection (front panel module GATE vs. module validation trigger)
0	21	Recording of external clock timestamps in event header – 1: enable; 0: disable

For example, to set the CHANNEL_CSRA bits as above, one would call the command

pwrite 0 -1 CHANNEL_CSRA 16676

where the value at the end is the binary sum of the bits that are = 1,
e.g. $2^2 + 2^5 + 2^8 + 2^{14} = 16676$

Individual bits can also be se using the “toggle” command, e.g.:
toggle <module> <channel> <bit>
or
toggle_bit <mod> <chan> <param> <bit>
([For poll2 commands, see here…]())

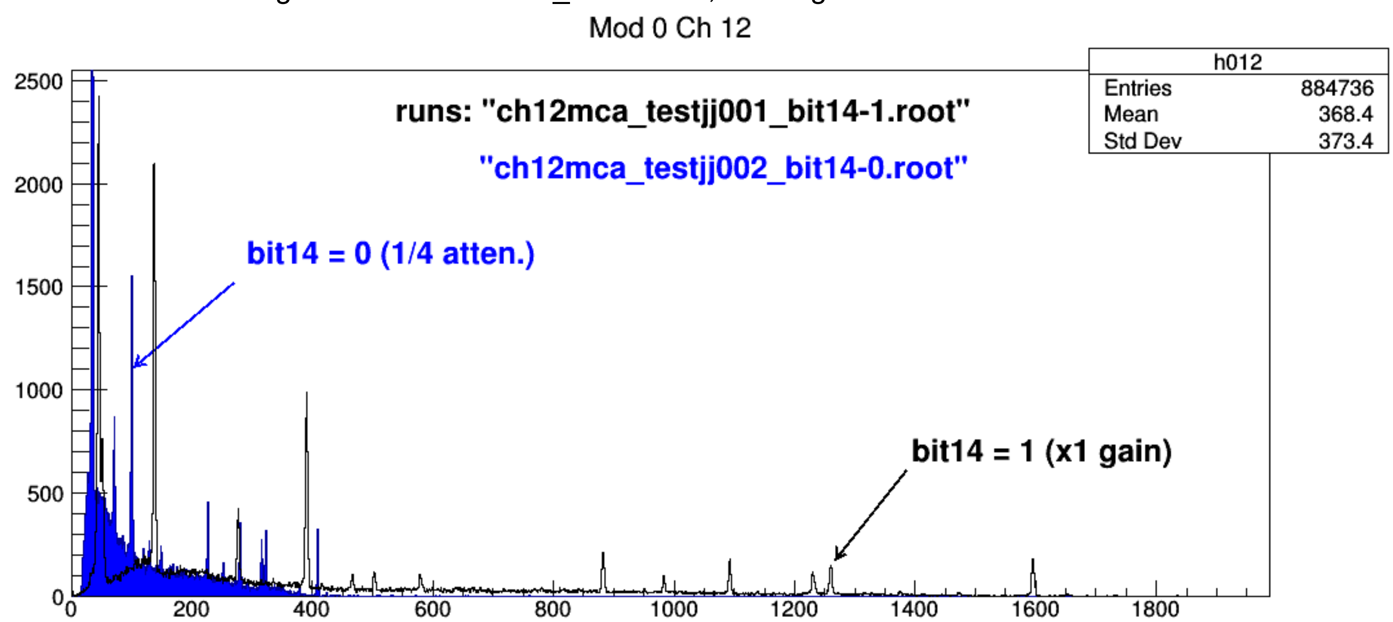
The fraction of the ADC where the baselines should be adjusted is set by changing the BASELINE_PERCENT parameter, e.g.

pwrite <mod> <chan> BASELINE_PERCENT 5 (to set to 5% of the ADC range)

More module/channel settings info can be found in the wiki [WIKI].

Gain/Attenuation factor

See the effect of setting bit 14 of CHANNEL_CSRA to 1, i.e. x1 gain/attenuation:



SCREEN

A good idea to access poll2 by different users and retain control, is to use “screen”. With screen a user can start poll2 and a different user can login to that screen session and stop or change parameters in poll2. Another advantage to screen is that when connection is lost, you can just restart that screen session and resume the poll2 control- this opposed to having to kill the lockfile created by the last user and possibly crash poll2.

Some basic screen commands are:

screen -ls (to list the available sessions)
screen -S <session name> (to start a new session with name <session name>, e.g. “screen -S jj-poll2” will create the session ****.jj-poll2)
screen -x <session name> will connect to the session with name <session name>

Once connected to a screen session, you can call screen commands by preceding them by “Ctrl+a”, e.g.:
Ctrl+a c (to create more “screens” (similar to terminal tabs))
Ctrl+a [spacebar] (to toggle between screens)
Ctrl+a d (to detach from the current screen session)

Note: The environment variables are not transferred to the screen login, so I have to add the library paths manually in a screen session

export LD_LIBRARY_PATH=/usr/local/lib/:${LD_LIBRARY_PATH}

viewBaseline

To set the input signals’ baselines properly one can run “viewBaseline” in a terminal where pixie.cfg is, i.e. from the poll2 directory - note, poll2 must NOT be running. (See also here…)

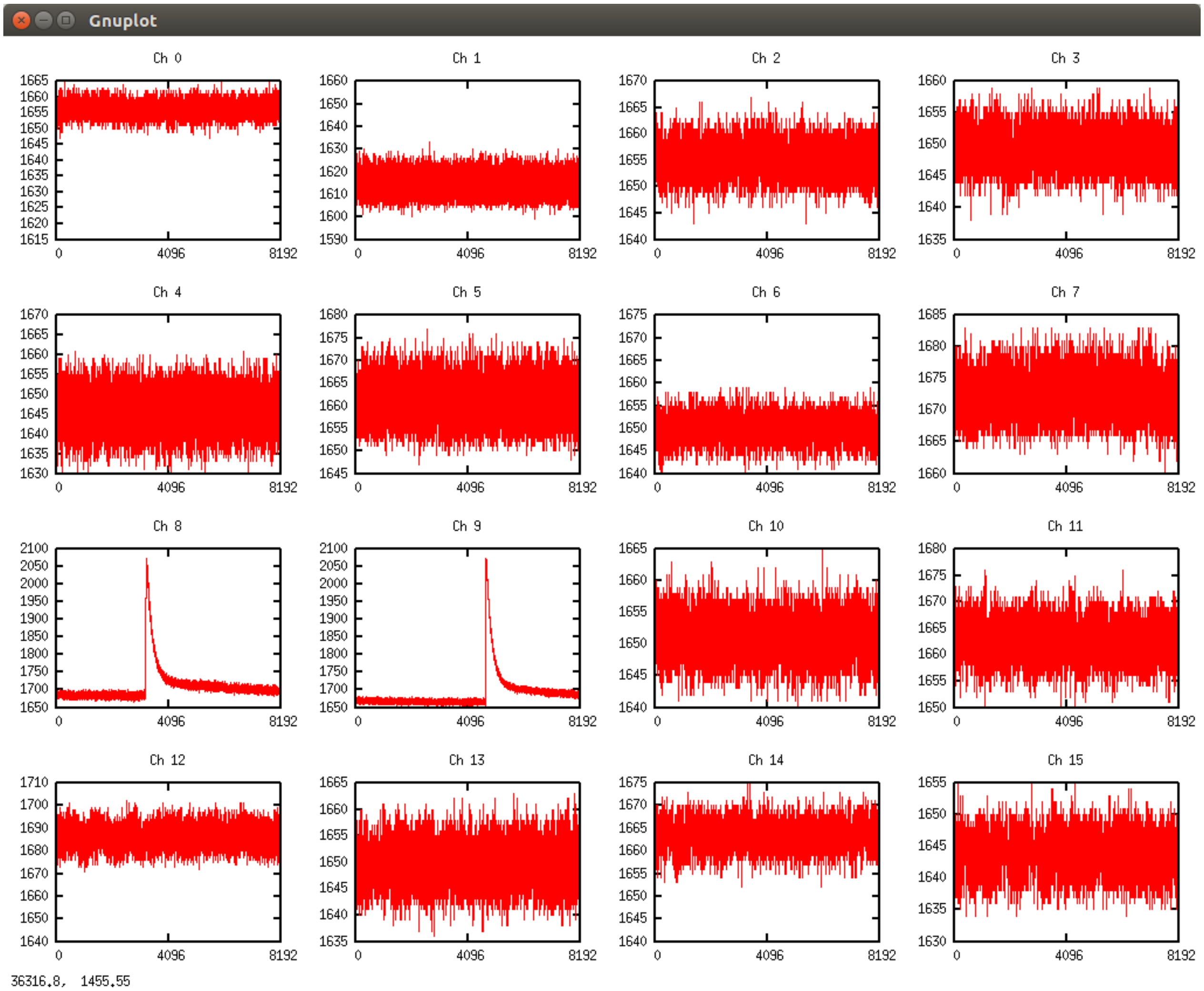
./viewBaseline 0 -1 (will plot the baseline of all the channels of module 0 in Gnuplot)

Note: some fixes are needed after installation - make sure the PAASS_BUILD_SETUP = ON in ccmake:
According to the repo issues one should:
Making local version of files:
cp /opt/paass/bin/viewBaseline /home/pr270user/Software/paass_usr/acq/
cp /opt/paass/share/traces/tra /home/pr270user/Software/paass_usr/acq/

Modify viewBaseline line 15 to read:
gnuplot -e "MOD=$mod;CH=$ch" tra

and modify tra line 16 to read
dir = "/opt/paass/share/traces/"

The initial baselines, after calling “adjust offsets 0” in poll2, were sitting around 1655 units as in the graphs below. This would have caused high amplitude signals to saturate. We readjust them to between channel 200 and 500 by changing the BASELINE_PERSENT value from 10% to 2%. This gives us more of the dynamic range to work with. We can also view any undershoots that may occur in the detectors. A review of the baselines show that there’s not too much noise. We can increase the trigger filters slightly to help cut down on any noise that might creep in.



monitor

One can monitor the data rates as the PIXIE system acquires data by running MONITOR:
// usage: once poll2 is acquiring data (startacq) in **shm** mode, run monitor in another terminal:
$: monitor <mod>
This will update every time the spill reached a set % of full buffer (50%) or you can force a spill by calling "spill" in poll2.
The columns in monitor refer to:

- **ICR** = Input Count Rate -> This is the average number of raw triggers that come into the channel.
- **OCR** = Output Count Rate -> This is the average number of validated triggers in the channel. Validated triggers pass through the coincidence logic before being "approved" for output.
- **Data** = The amount of data that this rate represents. @Karl may have more information here.
- **Total** = The total number of triggers that have passed through that channel.

For a system running in singles the ICR and OCR rates should match. If they are not within a few percent (5-10%), then you will need to reboot the module. It indicates there is some kind of issue.
Typical output, for only ch 8 and 9 running, is:

```
Run Time: 00:00:48.14
Data Rate: 1.91963 MB/s

   |-----------M00-----------|
   | ICR   OCR  Data   Total |
C00|   0     0     0       0 |
C01|   0     0     0       0 |
C02|   0     0     0       0 |
C03|   0     0     0       0 |
C04|   0     0     0       0 |
C05|   0     0     0       0 |
C06|   0     0     0       0 |
C07|   0     0     0       0 |
C08| 988   990   991  4.76E4 |
C09| 986   989   989  4.76E4 |
C10|   0     0     0       0 |
C11|   0     0     0       0 |
C12|   0     0     0       0 |
C13|   0     0     0       0 |
C14|   0     0     0       0 |
C15|   0     0     0       0 |
```

Analysis with UTKSCAN

UTKSCAN is the main run analyser (unpacker/sorter). We can analyse run data both *online* and *offline*.

Installation of paass-lc for UTKSCAN and analysis

Under the user installation, we can even hav e more than one version - just create a different installation directory and run the software from that directory, e.g.
<vestion1dir>/install/bin/ … ./utkscan
<vestion2dir>/install/bin/ … ./utkscan

We typically make a clone of this git-repository and store it in a new folder such as:
~/Software/paass_usr/jj-paass-pr270/.
$ git clone https://github.com/jjvz/paass-lc.git ~/Software/paass_usr/jj-paass-pr270/
$ git checkout feature-pr270
$ git pull --hard origin/feature-pr270

We first need to make a build directory to keep the build files separate from the source code.
$ mkdir build
$ cd build

PAASS-LC can be configured in a number of different ways using various CMake options. We recommend the optional ccmake package as it reduces the chance of making mistakes. Using CCMake we can simply invoke it with ccmake .. from the build directory (see the basic parameters in ccmake here...):

ccmake ../

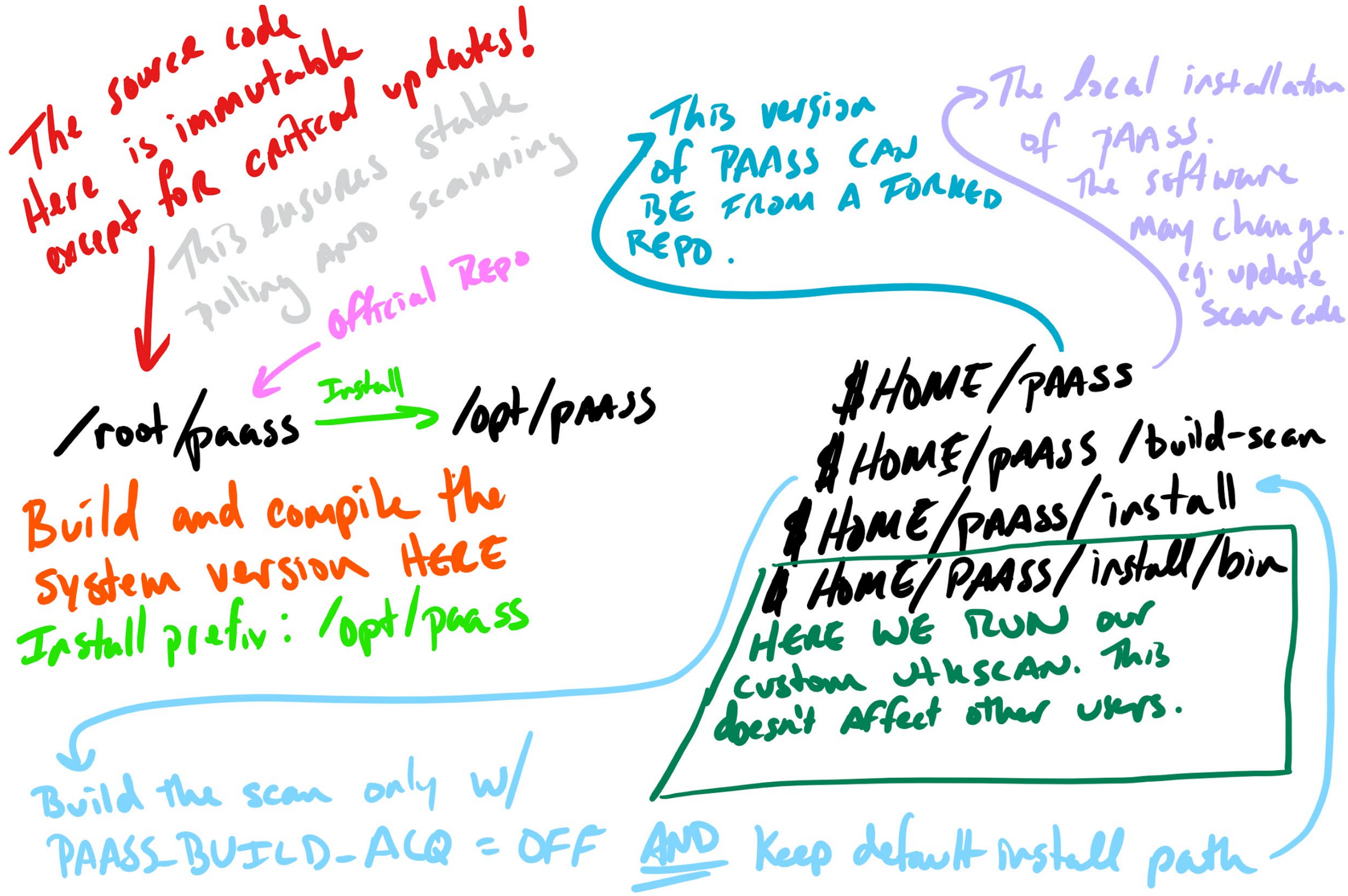
...but we prefer the following command which sets the install prefix.

ccmake ../ -DCMAKE_INSTALL_PREFIX=/install

Note, for the analyzer / utkscan unpacker installation only, i.e. no acquisition, one only needs to switch ON the following parameters in ccmake:
PAASS_BUILD_SHARED_LIBS
PAASS_BUILD_UTKSCAN
PAASS_USE_ROOT
PAASS_USE_DAMM -- and I'm not even sure this is necessary anymore after v3.1.0...

After CMake has completed generating the make file used for compilation we simply need to compile the package using the command make. To speed things up we can specify to use multiple threads for compilation with the optional argument -j [numberOfThreads].

$ make clean && make install -j8



My current directory environment - 26 Feb 2018

Base install:

/root/paass-lc/… ...cloned remote repo here
/root/paass-lc/build/... ...install poll from here into **/opt/paass/install/bin**
/opt/paass-lc/install/bin/… ...where the installation executables are (e.g. poll2 sits here)

User analysis install:

~/Software/paass_usr/acq/… ...run **poll2** from here, pixie.cfg and setfiles are here
~/Software/paass_usr/acq/… ...store temporary **runfiles** here
~/Software/paass_usr/analysis/… ...where **Config.xml** is
~/Software/paass_usr/jj-paass-pr270/install/bin/… ...run **utkscan** / **damm** from here
~/Software/paass_usr/jj-paass-pr270/build/… ...build utkscan software
~/Software/paass_usr/jj-paass-pr270/online/… ...backup runfiles here, once mounted the network drive

Run **poll2** from: (...this is where **pixie.cfg** is)**:**
/home/pr270user/Software/paass_usr/acq

Run **utkscan** from:
/home/pr270user/Software/paass_usr/jj-paass-pr270/install/bin, with the command:

$ ./utkscan -i /.../acq/<runfile>.ldf -c ~/Software/paass_usr/analysis/Config.xml -o <outfile>

Build base acquisition poll2 from:
/root/paass/build

Build utkscan from:
/home/pr270user/Software/paass_usr/jj-paass-pr270/build

What I did:
1) Created a folder for JJ: "/home/pr270user/Software/paass_usr/jj-paass-pr270"
2) cd ~/Software/paass_usr/jj-paass-pr270
3) Cloned his forked repo here: "git clone https://github.com/jjvz/paass-laughing-conqueror"
4) Checkout feature-pr270 branch : "git checkout feature-pr270"
5) Installed to: "/home/pr270user/Software/paass_usr/jj-paass-pr270/install"

Utkscan is run from where it's installed, i.e. here:
/home/pr270user/Software/paass_usr/jj-paass-pr270/install/bin

After e.g. copying an xml file from examples to Config.xml in the paass_usr folder, and editing the relevant parameters, e.g. firmware, modules etc., I run one of the following commands, depending on the module in the crate:

utkscan -f 34455 --frequency 250 -c ./Config_PIXIE16-testrun.xml -i run_011.ldf -o run_011_out -q
utkscan -f 34689 --frequency 100 -c ./Config_PIXIE16-testrun.xml -i run_011.ldf -o run_011_out -q

Note, these frequency and firmware parameters can be set in the config file on a "per module" basis. See here...

utkscan -i run_011.ldf -c ./Config_PIXIE16-testrun.xml -o run_011_out -q
or, for shm mode:
utkscan --shm -c ./Config_PIXIE16-testrun.xml -o run_011_out -q

1) For analysing **online**, polling data:
Make sure poll2 is running in another terminal. In poll2:
POLL2 $: shm   (toggle shared memory mode ON)
POLL2 $: startacq (and later 'stopacq') or "run" to write to disk. (poll2 will create a *.ldf output file)

Start UTKSCAN in another terminal, in **shm mode**:
utkscan **--shm** -c ./<config_file>.xml -o <output_filename>
Or if the frequency and firmware were not set in the Config.xml file:
utkscan **--shm** -f 34455 --frequency 250 -c ./<config_file>.xml -o <output_filename>

In UTKSCAN:
utkscan $: run   (Start the analyser)

This creates a set of **output files** from the initial *.ldf runfile:

<output_file>-hist.root (this creates the default histograms)

<output_file>-tree.root (this is created if you've set some ROOT event Tree with Brances and Leafs in the Processor code)

Note, UTKSCAN starts to unpack the incoming spills from poll2 only from when UTKSCAN was started. So when plotting histos from this utkscan *.poot output, it will only contain data from the start of UTKSCAN.

From 26 Feb 2018, I'm using the new version v3.1.0 of the laughing-conqueror repo, which has been modified to use ROOT for histogramming instead of DAMM

Online plotting with ROOT

Once UTKSCAN is running, you can view the .root output from utkscan. Utkscan produces two output root files, <output_file>-hist.root (this creates the default histograms), and <output_file>-tree.root (which contains the custom roottree and branches.

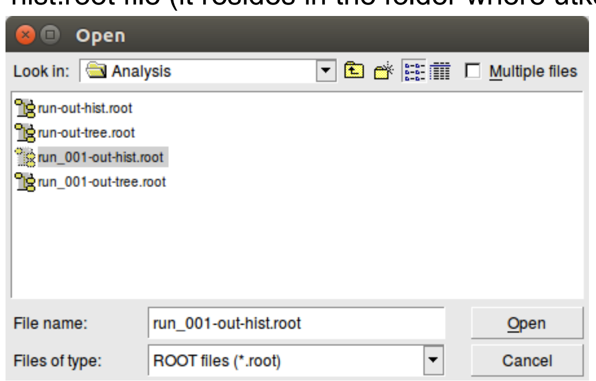
Method A

For online viewing of the default histograms that utkscan saves in the <output_file>-hist.root file, we currently are using a GUI script called RootHistogramGUI (found here: https://github.com/spaulaus/RootHistogramViewer).

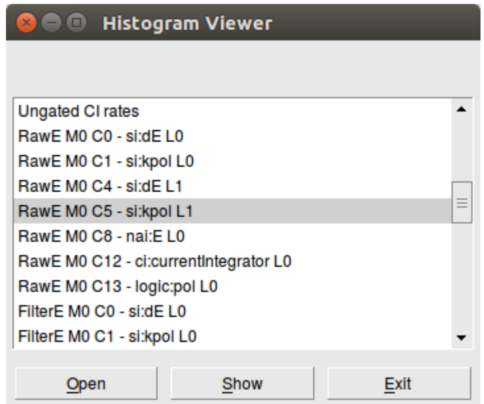
Install RootHistogramViewer in a folder (~/Software/paass_usr or wherever).

Open another terminal, and run the RootHistogramViewer script with
./issue126

Open the relevant <output_file>-hist.root file (it resides in the folder where utkscan is running, i.e. /install/bin).



Select the relevant spectrum to view, and press "Show"



The relevant histograms available are listed here:
Click on the “show” button again to refresh the histogram.

Histogram IDs

The set of predefined histograms created by default in UTKSCAN is given below.
HIS = histogram ID
DIM = the dimension of the histogram, 1D or 2D etc.
LEN(CH) = the channel resolution (binning) of the axis (see here…)

A set of default histograms is also defined and filled in the -hist.root runfile during list mode recording. These histograms can be commented out in the code **Analysis/Utkscan/core/source/DetectorDriver.cpp**.

Notes:
For channels 0 - 15, the IDs are 1 - 16, e.g. channel 7 has ID 8.
IDs 1 - 16 are the Raw energies
IDs 301 - 316 are the Filtered energies
IDs 601 - 616 are Scalar court rates
IDs 1201 - 1216 are Calibrated energies
IDs 18** are event statistics such as hits and buffer lengths and run times
IDs 6050 - 60** are for user defined histograms, defined in the processors

ID list:

1	2	3	4	301	302	303	304
601	602	603	604	1201	1202	1203	1204
1802	1803	1804	1805	1806	1807	1808	1811
1812	1813	6050	6051	6052			

HID	DIM	HWPC	LEN(CH)	COMPR	MIN	MAX	OFFSET	TITLE
1	**1**	**2**	**16384**	**1**	**0**	**16383**	**3277856**	**RawE M0 C0 - nai:stop L0**
2	**1**	**2**	**16384**	**1**	**0**	**16383**	**3408928**	**RawE M0 C1 - pulser:start L0**
3	**1**	**2**	**16384**	**1**	**0**	**16383**	**3540000**	**RawE M0 C2 - ge:E L0**
4	**1**	**2**	**16384**	**1**	**0**	**16383**	**3671072**	**RawE M0 C3 - plastic:E L0**
301	1	2	16384	1	0	16383	3310624	FilterE M0 C0 - nai:stop L0
302	1	2	16384	1	0	16383	3441696	FilterE M0 C1 - pulser:start L0
303	1	2	16384	1	0	16383	3572768	FilterE M0 C2 - ge:E L0
304	1	2	16384	1	0	16383	3703840	FilterE M0 C3 - plastic:E L0
601	1	2	16384	1	0	16383	3343392	**Scalar** M0 C0 - nai:stop L0
602	1	2	16384	1	0	16383	3474464	**Scalar** M0 C1 - pulser:start L0
603	1	2	16384	1	0	16383	3605536	**Scalar** M0 C2 - ge:E L0
604	1	2	16384	1	0	16383	3736608	**Scalar** M0 C3 - plastic:E L0

(These scalar values are the number of events per second for the total runtime to this point)

1201	**1**	**2**	**16384**	**1**	**0**	**16383**	**3376160**	**CalE M0 C0 - nai:stop L0**
1202	**1**	**2**	**16384**	**1**	**0**	**16383**	**3507232**	**CalE M0 C1 - pulser:start L0**
1203	**1**	**2**	**16384**	**1**	**0**	**16383**	**3638304**	**CalE M0 C2 - ge:E L0**
1204	**1**	**2**	**16384**	**1**	**0**	**16383**	**3769376**	**CalE M0 C3 - plastic:E L0**
1802	1	2	128	1	0	127	0	channel hit spectrum
1803	1	2	16384	1	0	16383	3146528	Time Between Channels in 10 ns / bin
1804	1	2	16384	1	0	16383	3179296	Event Length in ns
1805	1	2	16384	1	0	16383	3212064	Time Between Events in ns
1806	1	2	128	1	0	127	3244832	Number of Channels Event
1807	1	2	16384	1	0	16383	3245088	Buffer Length in ns
1808	2	1	16384	1	0	16383	256	run time - s
1811	2	1	16384	1	0	16383	1048832	run time - ms
1812	1	2	16	1	0	15	3145984	event counter
1813	1	2	256	1	0	255	3146016	channels with traces
6050	**2**	**1**	**4096**	**1**	**0**	**4095**	**3802144**	**Ungated dE vs. E - Telescope 0**
6051	**2**	**1**	**4096**	**1**	**0**	**4095**	**37356576**	**Ungated dE vs. E - Telescope 1**
6052	**1**	**2**	**8192**	**1**	**0**	**8191**	**70911008**	**NaI 0 Beam Up Gate**

(These last three plots have been defined in Pr270Processor.cpp)

One can access these also from the ROOT file with:

root [14] _file0->ls()

```
TFile**     out.root
 TFile*     out.root
  OBJ: TH1I   h610   Scalar M0 C9 - ci:currentIntegrator L0 : 0 at: 0x16ee000
  OBJ: TH1I   h601   Scalar M0 C0 - leps:E L0 : 0 at: 0x16bf810
  KEY: TH1I   h313;8   FilterE M0 C12 - leps:E L3
  KEY: TH1I   h601;8   Scalar M0 C0 - leps:E L0
  KEY: TH1I   h603;8   Scalar M0 C2 - plastic:dE L0
  KEY: TH1I   h605;8   Scalar M0 C4 - leps:E L1
  KEY: TH1I   h609;8   Scalar M0 C8 - leps:E L2
  KEY: TH1I   h613;8   Scalar M0 C12 - leps:E L3
  KEY: TH1I   h1201;8   CalE M0 C0 - leps:E L0
  KEY: TH1I   h1203;8   CalE M0 C2 - plastic:dE L0
  KEY: TH1I   h1205;8   CalE M0 C4 - leps:E L1
  KEY: TH1I   h1209;8   CalE M0 C8 - leps:E L2
  KEY: TH1I   h1213;8   CalE M0 C12 - leps:E L3
  KEY: TH1I   h1802;8   channel hit spectrum
  KEY: TH1I   h1803;8   Time Between Channels in 10 ns / bin
  KEY: TH1I   h1804;8   Event Length in ns
  KEY: TH1I   h1806;8   Number of Channels Event
  KEY: TH1I   h1807;8   Buffer Length in ns
  KEY: TH1I   h1812;8   event counter
  KEY: TH1I   h1813;8   channels with traces
  KEY: TH1I   h6051;8   Calculated rates of LaBr3 within threshold limits
  KEY: TH1I   h6052;8   Calibrated energy of LaBr3 within threshold limits
  KEY: TH1I   h6053;8   Calibrated energy of CI within threshold limits
  KEY: TH1I   h3;8   RawE M0 C2 - plastic:dE L0
  KEY: TH1I   h5;8   RawE M0 C4 - leps:E L1
  KEY: TH1I   h10;8   RawE M0 C9 - ci:currentIntegrator L0
  KEY: TH1I   h13;8   RawE M0 C12 - leps:E L3
  KEY: TH1I   h305;8   FilterE M0 C4 - leps:E L1
  KEY: TH1I   h1;8   RawE M0 C0 - leps:E L0
  KEY: TH1I   h9;8   RawE M0 C8 - leps:E L2
  KEY: TH1I   h301;8   FilterE M0 C0 - leps:E L0
  KEY: TH1I   h303;8   FilterE M0 C2 - plastic:dE L0
  KEY: TH1I   h306;8   FilterE M0 C5 - labr3:E L0
  KEY: TH1I   h309;8   FilterE M0 C8 - leps:E L2
  KEY: TH1I   h310;8   FilterE M0 C9 - ci:currentIntegrator L0
```

```
KEY: TH1I    h610;8    Scalar M0 C9 - ci:currentIntegrator L0
KEY: TH1I    h1206;8    CalE M0 C5 - labr3:E L0
KEY: TH1I    h1210;8    CalE M0 C9 - ci:currentIntegrator L0
KEY: TH1I    h1805;8    Time Between Events in ns
KEY: TH1I    h6;8    RawE M0 C5 - labr3:E L0
KEY: TH1I    h606;8    Scalar M0 C5 - labr3:E L0
KEY: TH2I    h1808;8    run time - s
KEY: TTree    DATA;3
KEY: TH2I    h1811;8    run time - ms
KEY: TH2I    h6050;8    Ungated dE vs. E - Telescope 0
```

Histogram x- and y-axis size constants

E.g.:
histo.DeclareHistogram1D(ungated::D_RATES, **SD**, "Ungated CI rates")
histo.DeclareHistogram2D(ungated::DD_TELESCOPE, **SC, SD**, "Ungated dE vs. E - Telescope L")

const int	S1 = 2 (2^1),	S2 = 4 (2^2),	S3 = 8 (2^3),
	S4 = 16 (2^4),	S5 = 32 (2^5),	S6 = 64 (2^6),
	S7 = 128 (2^7),	S8 = 256 (2^8),	S9 = 512 (2^9),
	SA = 1024 (2^10),	SB = 2048 (2^11),	SC = 4096 (2^12),
	SD = 8192 (2^13),	**SE = 16384 (2^14),**	**SF = 32768 (2^15)**

(From: *DammPlotIds.hpp*)
An x-axis size of SD implies a binning of 4 bins per channel.

TreeCorrelator… show me your secrets!

The TreeCorrelator provides user control and filtering on a level above the acquisition. The TreeCorrelator is managed from the Config.xml file which is run as part of the unpacking done by the analyser UTKSCAN. The TreeCorrelator creates and manages so-called “Places”.

Places

Places are entities that can act like familiar analog NIM modules such as SCAs and logic units. Places are characterized by a Boolean state (True / False) but may also store additional information (time, energy, etc.).

The typical Places are PlaceAND, PlaceOR, PlaceThreshold, PlaceCounter and PlaceDetector. The functioning of these Places are relatively clear from it name, e.g. a PlaceThreshold will have a True status if the constituent channel energy value is within the lower and upper limits set in the Place. Similarly, the status of a PlaceAND will be True is the daughter “channels” are True - the daughter components are not necessarily actual channels, but could be another Place. See e.g. this extract from a **Config.xml** file, mapping several channels:

```
<Map verbose_calibration="False" verbose_map="False" verbose_walk="False">
        <Module number="0" firmware="R34455" frequency="250">
                <Ch="0" type="scint" subtype= "beta" ></Channel>
                <Ch="1" type="scint" subtype= "beta" ></Channel>
                <Ch="2" type="ge" subtype= "clover_high" ></Channel>
                <Ch="3" type="ge" subtype= "clover_high" ></Channel>
                <Ch="4" type="ge" subtype= "clover_high" ></Channel>
```

```
                <Ch="5" type="ge" subtype= "clover_high" ></Channel>
        </Module>
</Map>

<TreeCorrelator name="root" verbose="False">
        <Place type="PlaceAND" name="GammaBeta">
                <Place type="PlaceOR" name="Beta">
                        <Place type="PlaceThreshold" name="scint_beta_0-1"
                        low_limit="10.0" high_limit="16382" replace="true"/>
                </Place>
                <Place type="PlaceOR" name="Gamma">
                        <Place type="PlaceOR" name="Clover0">
                                <Place type="" name="ge_clover_high_0-1"/>
                        </Place>
                        <Place type="PlaceOR" name="Clover1">
                                <Place type="" name="ge_clover_high_2-3"/>
                        </Place>
                </Place>
        </Place>

        <Place type= "PlaceDetector" name= "Beam" reset="false"/>
        <Place type= "PlaceDetector" name= "PrntDiagnostics" init="false" reset="false"/>
</TreeCorrelator>
```

Extract from an processor: **Processor.cpp**

```
------------------
Pr270Processor::Pr270Processor() : EventProcessor(OFFSET, RANGE, "Pr270PRocessor") {
associatedTypes.insert("scint");
associatedTypes.insert("ge");
….
bool PrntDiag = TreeCorrelator::get()->place("PrntDiagnostics")->status();
if(PrntDiag)  myfile<<"LaBr3 time = "<<(time-firsttm)*s2ns<<endl;
------------------
```

Note, Place names are referred to by: **type_subtype_location**, e.g. scint_beta_0, referring to type “scint”, subtype “beta” and location = 0 (for the first channels with this type, and location = 1 for the next. Other Place names are derived from others Places, e.g. “Clover1”or “GammaBeta”. Basic places are created automatically from entries in the Map Node of the Config.xml file using type_subtype_location pattern.

In the TreeCorrelator example one can see that the status of the PlaceThreshold named "**scint_beta_0-1**" will be True only if the energy of the 1st and 2nd locations of channels with “scint_beta” types are within the set limits. When either of the two daughter Places "**scint_beta_0**" or "**scint_beta_1**" is True, the PlaceOR named “Beta” will be True. In a similar argument, when the Place named “Gamma” is True AND the Place named “Beta” then the higher level PlaceAND named “GammaBeta” is True.

A virtual PlaceDetector can also be created such as “BeamOn” which will carry a logical status to the processor for further logical conditioning, e.g. I created a Placedetector called “PrintDiagnostics” with an initial value of True. In the processor I can read this boolean status to decide to print some diagnostics to a file or not.

The `**reset**="false"` flag in the Place definition tells the code to NOT reset that place at the end of an event. Set that flag to “true” and the place will reset at the end of an event (event, as defined by the EventWidth in the Config.xml file).

I think the `**replace**="true"` flag tells the TreeCorrelator to update this place with the most recent entry. I'm not 100% certain on that.

Note, the **associatedTypes** in the processor refers to the types in the Channel Map.

Available Places

PlaceThreshold(low_limit : double, high_limit : double, resettable : bool, max_size : unsigned)
+ activate(info : CorrEventData&)
+ getLowLimit() : double
+ getHighLimit() : double

PlaceAND(resettable : bool, max_size : unsigned)

PlaceDetector(resettable : bool, max_size : unsigned)

PlaceOR(resettable : bool, max_size : unsigned)

PlaceCounter(resettable : bool, max_size : unsigned)
+ activate(info : CorrEventData&)
+ deactivate(time : double)
+ reset()
+ getCounter() : int

PlaceThresholdOR(low_limit : double, high_limit : double, resettable : bool, max_size : unsigned)

PlaceLazy(resettable : bool, max_size : unsigned)
+ activate(info : CorrEventData&)
+ deactivate(time : double)

where

CorrEventData
+ status : bool
+ time : double
+ energy : double

Using places

- Accessing place’s status
  bool tapeMove = TreeCorrelator::get()->place("TapeMove")->status();

- Activating and deactivating place (if not done automatically!)

    - TreeCorrelator::get()->place("TapeMove")->activate(time);
    - CorrEventData info(time, energy);

    - TreeCorrelator::get()->place("TapeMove")->activate(info);
    - TreeCorrelator::get()->place("TapeMove")->deactivate(time);

- Accessing stored information
    - TreeCorrelator::get()->place("TapeMove")->last().time;
    - TreeCorrelator::get()->place("TapeMove")->secondlast().time;
        - rarely needed
    - TreeCorrelator::get()->place("TapeMove")->info_[i];

Editing and re-compiling code

Create and add new Processor

I have used the Pr270Processor.cpp/hpp as template and created a new experiment processor called **Pr248Processor**.cpp/hpp. I make sure to name the Processor class correctly!

To add the processor:

1. Place **cpp** in source and **hpp** in include folders :
   /Analysis/Utkscan/experiment/**source**/Pr270Processor.cpp
   /Analysis/Utkscan/experiment/**include**/Pr270Processor.hpp

2. Add the new processor to the **DetectorDriverXmlParser.cpp** code :
   /Analysis/Utkscan/core/source/DetectorDriverXmlParser.cpp :

   #ifdef useroot //Some processors REQUIRE ROOT to function
   #include "**Pr270Processor.hpp**"
   ...
   #ifdef useroot //Certain processors REQUIRE ROOT to actually work
   else if (name == "**Pr270Processor**") {
   vecProcess.push_back(new **Pr270Processor**());

3. Add the Processor source code to the **CMakeLists.txt** for building:
   /Analysis/Utkscan/experiment/source/CMakeLists.txt
   ...
   if (PAASS_USE_ROOT)
   list(APPEND EXPERIMENT_SOURCES
   Pr270Processor.cpp
)
4. Build the code again: cd build && make clean && make install -j8

Github

Here's my procedure for updating my adapted code to a **fork** of the paass-laughing-conqueror repository on Github (https://github.com/jjvz/paass-lc).

Note, follow the guide for updating the fork here:
https://github.com/spaulaus/paass-laughing-conqueror/wiki/Development-Workflow#fork-workflow

Current Setup

I have set **upstream** to be the original spaulaus/paass-lc repo:
$ git remote add **upstream** https://github.com/spaulaus/paass-lc.git
$ git fetch --all --prune

Local Repo on laptop (for editing code): ~/../PIXIE16/paass-lc
This was originally cloned from the upstream (spaulaus/paass-lc)

master -> upstream/master
feature-pr270 -> upstream/master + custom changes

Remote Fork: jj/paass-lc

master -> pushed from local master
feature-pr270 -> pushed from local feature-pr270

DAQ PC - client: asouev2 (~/Software/paass_usr/jj-paass-pr270/)

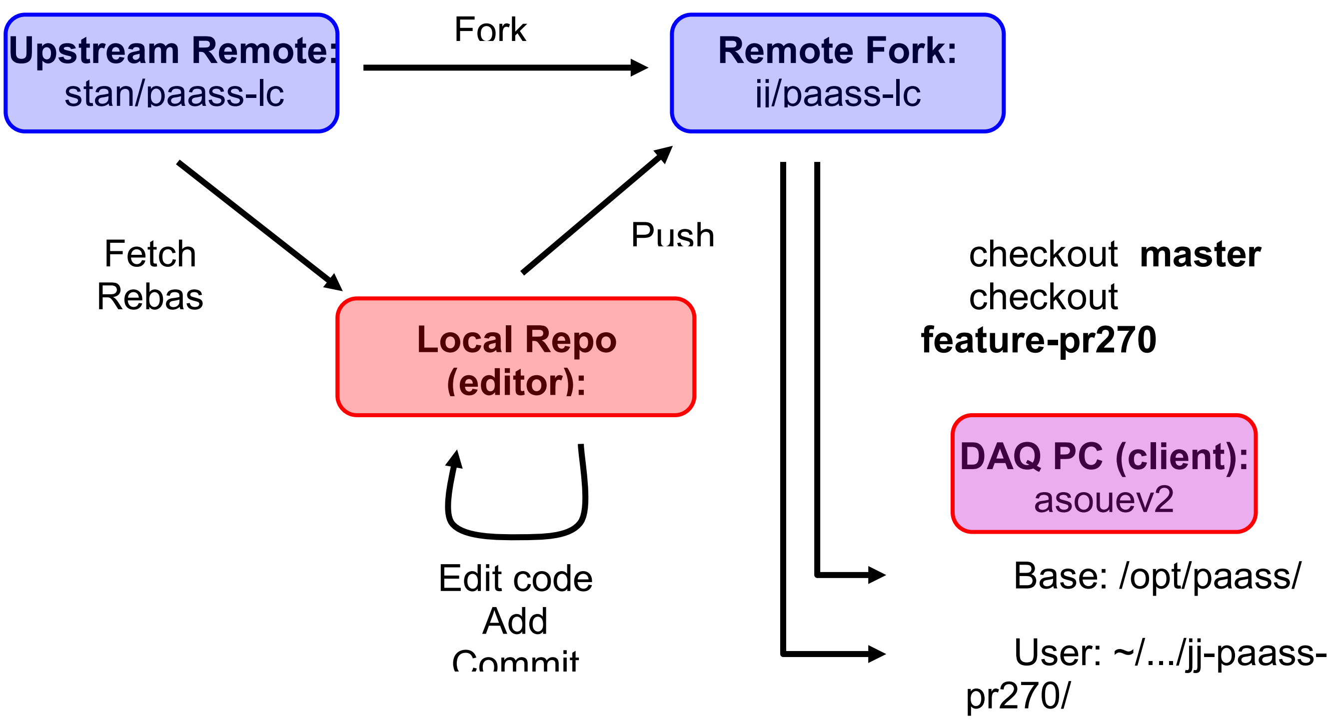
master -> cloned from remote fork master - for base installation (acquisition, poll2 etc.)
feature-pr270 -> pushed from local feature-pr270 - for user installation (analysis, utkscan, etc.)

You can change the remote origin[or upstream] URL with:

git remote set-url origin[upstream] git://new.url.here

To check which release version you have locally, call:

git describe --tags



Git Procedure

1. I regularly update the master and feature-pr270 branches of my local repo on my laptop with the upstream. From the local repo on laptop:
   $ git fetch --all --prune
   $ git checkout **master**
   $ git rebase -p upstream/master

$ git fetch --all --prune
$ git checkout **feature-pr270**
$ git rebase -p upstream/master

2. I update the local repo with my custom codes and changes:
   $ git checkout feature-pr270
   $ cp ~/codes/PIXIE16/**Backup_codes/**<file> ~/codes/PIXIE16/paass-lauging-conquerer/Analysis/Utkscan/experiment/source/<file>

   I put custom scripts in the following build directory: “paass-laughing-conqueror/Analysis/Utkscan/share/utkscan/**scripts**/”

   and config.xml file in the “cfgs” directory:
   “paass-laughing-conqueror/Analysis/Utkscan/share/utkscan/**cfgs**/”

   NOTE: My custom processors (cpp) and their headers (hpp) should be referred to in the **CMakeList.txt** file, as well as the **DetectorDriverXmlParser.xml**. Make sure these files are still correct according to the latest upstream build, because the “fetch” command does not override the existing files in the local repo.

3. I push local repo commits to remote fork:
   $ git checkout master
   $ git commit -a -m “...comment…”
   $ git push --force origin master
   $ git checkout feature-pr270 (to make sure you’re in the correct branch)
   $ git add -A **(to add new codes to git list)**
   $ **git commit** -a -m “...comment…” (commits the changes to Git, locally)
   $ **git push** --force origin feature-pr270 (saves the changes to the GitHub online repository)

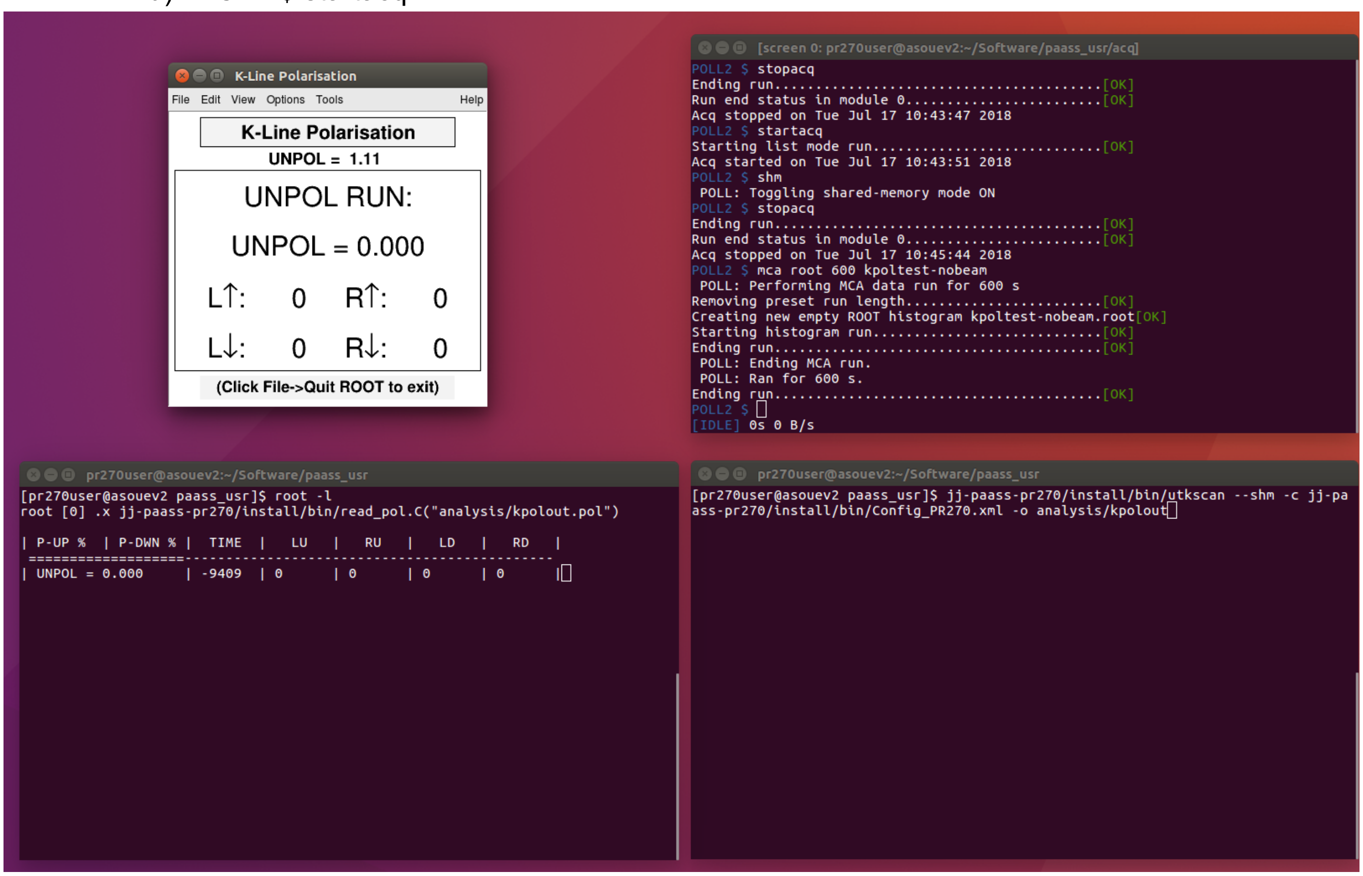
4. I pull latest remote fork to DAQ PC client:
   From the DAQ PC, and in the cloned folder where I run the analysis from (~**/Software/paass_usr/jj-paass-pr270/**):
   $ git checkout feature-pr270
   $ git pull (because I want this to be exact copy of remote repo)
   or force pull:
   $ git reset --hard origin/feature-pr270
   $ git pull origin feature-pr270

5. Rebuild and compile latest repo pull - code should be up to date, now recompile:
   $ cd build
   $ make clean && make install -j8
   or
   $ make clean
   $ make -j8
   $ make install

K-pol procedure to read polarization values

Initial startup (done once, usually by physicist):

1) ssh to the DAQ computer:
    a) ssh -Y pr270user@asouev2.tlabs.ac.za
    b) Open 3 terminals
2) In a terminal (1), start a “screen” session:
    a) screen -ls (to check if a session already exists)
    b) If a session already exists, connect to it (e.g. 1234.jjpoll2)
        i) screen -x 1234.jjpoll2
    c) If no session exists, create one:
        i) screen -S jjpoll2
3) start poll2 system in above screen session:
    a) $ cd ~/Software/paass_usr/acq/
    b) $ poll2 -f
    c) POLL $ shm
    d) POLL $ startacq



4) In another terminal (2): Start the UTKSCAN analyser:
    a) $ cd ~/Software/paass_usr/
    b) $ jj-paass-pr270/install/bin/utkscan --shm -c jj-paass-pr270/install/bin/Config_PR270.xml -o analysis/kpolout
    c) In UTKSCAN:

run

5) In another terminal (3): Call the “read_pol” script in ROOT:
   a) $ cd ~/Software/paass_usr
   b) $ root -l
   c) ROOT [0] .x jj-paass-pr270/install/bin/read_pol.C("analysis/kpolout.pol")

where "kpolout.pol" is the name of the polarisation value file set in the Config.xml file (read by UTKSCAN).

Recurring startup (done by operator)

1. **Restart UTKSCAN in terminal 2:**

jj-paass-pr270/install/bin/utkscan --shm -c jj-paass-pr270/install/bin/Config_PR270.xml -o analysis/kpolout

2. **Restart ROOT in terminal 3:**

root -l

3. **Restart script “read_pol” in ROOT:**

.x jj-paass-pr270/install/bin/read_pol.C("analysis/kpolout.pol")

4. **Quit ROOT:**
   a. Click on **File->Quit ROOT** in the pol window
   b. Type **stop** and **quit** in UTKSCAN terminal

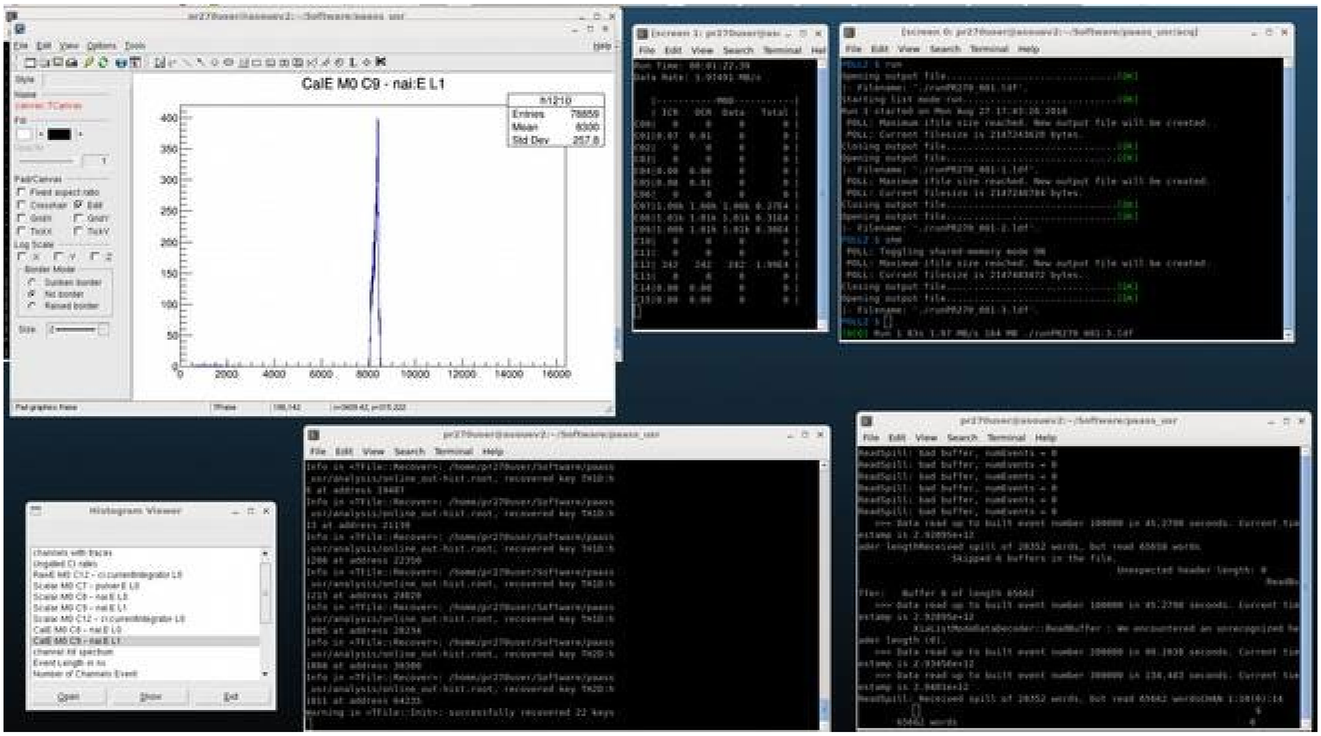
I have a testing procedure where I write some simulated polarisation events to a file, and then use the “read_pol.C” script to read these values as if it was from UTKSCAN:

In one ROOT terminal:
.x write_pol.C("kpolout.pol")

In another ROOT terminal:
.x read_pol.C("kpolout.pol")

When a new setting is implemented:
1. stop and quit UTKSCAN (type ‘stop’, then ‘quit’ in UTKSCAN terminal)
2. start a new analyser run: go to step 3 above.

PR270 - Guidelines for the Shifter



The terminals / windows

POLL2: The online DAQ polling system – records the actual run data (run ONLY with “screen” session). Run from ~/Software/paass_usr/acq/
MONITOR: Shows the data rates as the DAQ records them (run ONLY with “screen” session). Run from ~/Software/paass_usr/
ANALYZER: This is the event unpacker/sorter/analyzer (originally called “utkscan”). Run from ~/Software/paass_usr/
Plot Control: Terminal window from which various scripts to e.g. start ROOT displays or editors can be launched. Run from ~/Software/paass_usr/
Histrogram Selection: GUI code to open histograms filled by the analyzer, which will be displayed in the Histogram Display ROOT window. Run from ~/Software/paass_usr/

Table of Contents

To start and stop a run in POLL2

In the POLL2 (top right) terminal, at the prompt POLL2 $
(terminal title: screen 0: pr270user@asouev2:~/Software/paass_user/acq)
POLL2$ run

POLL2 will then display the run number, e.g., “Run 1 started on Wed Aug 29 11:00:00 2018”
Record this in the logbook

To stop the run, enter the command
POLL2$ stop

Record the stop time in the logbook

Things to monitor during a standard run
In the MONITOR (top left) terminal
(terminal title: screen 1: pr270user@asouev2:/Software/paass_usr/acq)

Observe the event rates displayed in the 'Data' column for the active channels (namely, C00, C01, C04, C05, C07, C08, C09, C12, C14, C15) – see logbook p.10 for channel assignments.

The input rates (ICR) and output rates (OCR) should be the same (ish – so within roughly 1% of each other). If they differ markedly contact JJ van Zyl, LM Donaldson or R Neveling, because the power to the crate must be recycled.

Halo conditions

Ideally <100 Hz on the NaI detectors (C08 & C09) for a beam current of ~1 nA. Note, there are 1000 Hz pulser signals on C08 & C09 as well.

Keep an eye on the Data rates of C14 & C15. These two channels get signals from the ion source indicating UP and DOWN polarisation states. They should alternate (~10 sec. intervals) and show ~equal Total rates. If they do not alternate, or show 0 all the time, then the WF/SF timer is still OFF (unpolarised), or the baselines are incorrect. To correct the baselines, switch OFF the WF/SF timer and in POLL2 (not acquiering) type

POLL2$ adjust offsets 0

C12 shows the Current Integrator rate on the beamstop. This rate has a full scale of 1000 Hz which represents the Range of the Current Integrator unit. E.g. a rate of 200 Hz out of a full 1000 Hz is 200/1000 = 0.2. If the Range is set to e.g. 6 nA, then this 200 Hz is 0.2 of 6 nA, i.e. 1.2 nA.

To run the analyser on a saved run or shm spill...

Analyse a saved run, e.g. runPR270_001.pld
In the analyser terminal, i.e. at [pr270user@asouev2 paass_usr]$ prompt type ./analyser.sh
[pr270user@asouev2 paass_usr]$ ./analyser.sh

Type the 3 digit run number, e.g. 001

When the analyser is done, type (...and this may take a few minutes...):
utkscan $ stop
utkscan $ quit

One can view the resulting histograms while the analyser is still busy: in another terminal, run the alias histos and select the hist file, e.g. analysis/runPR270_001-hist.root.

Analyse the current online shm spills

Make sure the shared memory mode (shm) is active in the POLL2 terminal : shm

In the analyser terminal, i.e. at [pr270user@asouev2 paass_usr]$ prompt type online_scan
[pr270user@asouev2 paass_usr]$ online_scan

In the analyser prompt, type run:
utkscan $ run

One can view the resulting histograms while the analyser is still busy: in another terminal, run the alias histos and select the hist file analysis/online_out-hist.root.

Note: the analyser sometimes start to miss shm buffer spills, so I would not run this online_scan too long – few minutes maybe.

To quit the analyser (before every new run), at the analyser's command promp, enter the commands stop and then quit: utkscan $ stop   and   utkscan $ quit

...and this may take a few minutes...

To view histograms (online or offline)

I have installed the latest RootHistogramViewer, but had to export the following first, before compiling:
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH

In the bottom left control terminal (terminal title: pr270user@asouev2:/Software/paass_usr) start the ROOT histogram viewer with the command histos

[pr270user@asouev2 paass_usr]$ histos

Two new windows will appear. In the smaller window that lists files and directories, select the analysis folder and select and open the *-hist.root (histogram) file, e.g. run_001-hist.root, or online_out-hist.root.

A Histogram Viewer will pop up and a list of predefined histograms will be available. (See logbook for channel assignments.)

Click on Show to refresh the selected histogram.

Note that clicking “Show” on the Histogram Viewer will refresh the spectra. They do not update automatically.

To exit the Histogram Viewer, click Exit. (Don't try to quit this by Ctrl+c/z or the quit X on the window, it will cause a crash – then you are stuck!)

Measuring polarisation in the A-line S.C.

Move the telescopes to the correct angles:

Upper Arm (UA) to 23.5o + offset.

Lower Arm (LA) to 336.5o + offset.

Change target to 12C / CH

Start with an unpolarised measurement to get the UNPOL factor:
Switch OFF (down) the WF/SF timer at the ion source.
Start a new run in the POLL2 terminal: POLL2$ run
Wait for a few minutes then stop the run: POLL2$ stop
Run the analyser with command ./uti.sh
Check the elastic peak limits in the CallE spectra (C08 & C09) by viewing the histograms. Type in the plot control terminal the alias:
[pr270user@asouev2 paass_usr]$ histos

Set these limits in the Config_PR270.xml file:
vim jj-paass-pr270/install/bin/Config_PR270.xml

Look for the following in this file:
<Place type="PlaceAND" name="TelLgate" reset="true">
and
<Place type="PlaceAND" name="TelRgate" reset="true">

Run the analyser again (with the new gates) with command:
[pr270user@asouev2 paass_usr]$ ./uti.sh

In the plot terminal, run the script:
[pr270user@asouev2 paass_usr]$ ./startroot.sh
to see the UNPOL value as the data stats come in.

Write the UNPOL value in the script (line 17) read_pol.C with:
vim jj-paass-pr270/install/bin/read_pol.C

Now do a beam polarization measurement:
Switch ON (up) the WF/SF timer at the ion source.
Start a new run in POLL2 for few minutes.

Run the analyser again on the new run:
[pr270user@asouev2 paass_usr]$ ./uti.sh

In plot terminal run the bash script startroot.sh again
[pr270user@asouev2 paass_usr]$ ./startroot

The up and down polarisation values should come in with increasing certainty. Write these pol up and down values in the logbook.

Measuring beam offset in S.C. arm angles

Move the telescopes to the correct angles:
Upper Arm (UA) to 100o
Lower Arm (LA) to 20o

Change target to 12C / CH
Start with an unpolarised measurement - Switch OFF (down) the WF/SF timer at the ion source.
Start a new run in the POLL2 terminal: POLL2$ run
Wait for a few minutes (e.g. 3 min.) then stop the run: POLL2$ stop
Check the elastic peak limits in the CalIE spectrum of the LA (C09) by viewing the histograms. Type in the plot control terminal the alias:
[pr270user@asouev2 paass_usr]$ histos

Set these limits in the offsets.C script: vim offsets.C
int binmin = 8370;
int binmax = 8700;

Run the analyser on the last run with command ./offset_analyse.sh

Once the analyser is done, run the script offsets.C in ROOT. i.e.
root -l offsets.C

Now follow beam offset procedure to find optimum beam zero.

Notes on data analysis

How to look at online/shared memory data of the experiment
Make sure the shared memory mode is active in the POLL2 terminal : shm
In the ANALYZER terminal analyze the data in the shared memory (the shm) with the command:
[pr270user@asouev2 paass_usr]$ online_scan
Then move to the Plot Control terminal and start the ROOT histogram viewer with the command
[pr270user@asouev2 paass_usr]$ histos
Two new windows will appear. In the smaller window that lists files and directories, select the analysis folder and select and open the run file online_out-hist.root.
The Histogram Selection viewer will pop up and a list of predefined histograms will be available.

Note that clicking "Show" on the Histogram Viewer will refresh the spectra.
Histograms do not update automatically.

To exit the Histogram Viewer, click Exit.

To quit the analyser (before every new run):
Go to the ANALYZER termina and enter the commands stop and quit
utkscan $ stop
utkscan $ quit
which means you are now out of utkscan and back to the terminal commandline.

How to look at finished datarun nr XYZ (effectively offline data analysis)

In the ANALYZER terminal analyze the finished data run by entering the command [pr270user@asouev2 paass_usr]$ ./analyser.sh
You will be asked for the runnumber
You will see the data analysis progress in the ANALYZER terminal window. Once it is stopped you can move to the Plot Control terminal and start the ROOT histogram viewer for the analyzed run with the command
[pr270user@asouev2 paass_usr]$ histos
Be sure to select the relevant root file: in the window that lists files and directories select the analysis folder and then select and open the run file runPR270_XYZ-hist.root and click on OPEN.
The Histogram Selection viewer will pop up and a list of predefined histograms will be available.

Summary of commands

In POLL2

Start data run: run

Stop data run: stop

In ANALYZER

Analyze saved data run: ./analyser.sh
Analyze saved kpol data run: ./uti.sh
Analyze shm spill: online_scan
Analyze shm spill for kpol: kpol
Analyzse saved runfile for offsets: offset_analyse.sh
Analyze shm spill for offsets: offset_scan

In control terminal

Start the ROOT Histrogram Viewer: histos
Calculate Polarisation: ./startroot.sh
Calculate beam offset: offsets

Summary of bash and alias scripts

analyser.sh
jj-paass-pr270/install/bin/utkscan -i acq/runPR270_***.ldf -c jj-paass-pr270/install/bin/Config_PR270.xml -o analysis/runPR270_***

histos
./RootHistogramGUI/build/issue126

offset_analyse.sh
jj-paass-pr270/install/bin/utkscan -b -i acq/runPR270_***.ldf -c jj-paass-pr270/install/bin/Config_PR270.xml -o analysis/offsets

uti.sh
jj-paass-pr270/install/bin/utkscan -b -i acq/runPR270_***.ldf -c jj-paass-pr270/install/bin/Config_PR270.xml -o analysis/kpolout

kpol
jj-paass-pr270/install/bin/utkscan --shm -c jj-paass-pr270/install/bin/Config_PR270.xml -o analysis/kpolout

online_scan
jj-paass-pr270/install/bin/utkscan --shm -c jj-paass-pr270/install/bin/Config_PR270.xml -o analysis/online_out

offset_scan
jj-paass-pr270/install/bin/utkscan --shm -c jj-paass-pr270/install/bin/Config_PR270.xml -o analysis/offsets

offsets
root -l offsets.C

startroot.sh
root -l jj-paass-pr270/install/bin/read_pol.C("analysis/kpolout.pol")

PR270 DAQ - Serious Troubleshooting:

If no events are coming into the ANALYZER, maybe the “shm” (shared memory mode) is OFF To switch it ON:

In the top right terminal, at the prompt POLL2 $
(terminal title: screen 0: pr270user@asouev2:~/Software/paass_user/acq)

Stop the run (refer to appropriate section) and enter the command shm

POLL2$ shm

You should see:

POLL: Toggling shared-memory mode ON

IF it is off, type 'shm' again to turn it ON.

If the Pol_UP and Pol_DOWN signals (C14 & C15) are not switching / alternating or not counting at the same Total rates:
Go to ion source and switch OFF the WF/SF timer
In POLL2 (stop any runs) type
adjust offsets 0
The offsets for C14 & C15 should both be ~0.7
Switch ON the WF/SF timer again
Start new run

How to start the monitor windown in the Control Room if you got kicked off the session

Log onto pr270user@asouev2

Connect to the existing screen session that is running POLL2. To find out which this is you can type:
screen -ls

Then connect to that screen (e.g 2802.jjpoll2) with:
screen -x 2802.jjpoll2

The resulting terminal title will be either screen 0 or screen 1...
[screen 0: pr270userasouev2:~/Software...] or [screen 1: pr270userasouev2:~/Software...]

monitor usually runs in screen 1, but if the screen session starts in screen 0, use the “CTRL+A and then spacebar” command to toggle to the monitor screen.

To start the monitor if it is not there at all, go to Software/paass_usr/acq and type:
monitor 0

Note: monitor will only initialise if POLL2 is in shm mode and acquiering a run.

ROOT procedures

(This is old stuff and some variables are not used or have been renamed… I keep it here as a quick reference)
The Pr270Processor creates a tree called "DATA"
The tree has two brances, "telescope0" and "telescope1", each containing variables (leafs) "si', "nai" and "polarization".

Drawing the leafs in root:

TFile *file0 = TFile::Open("PIXIE16-testrun_001_out.root")

<tree>->Draw("<branch>.<leaf> >> h01(1000,0,25000)","","")
E.g.: DATA->Draw("telescope1.nai>>h01(1000,0,25000)","","")

Rootscan (not used anymore…)

(Does not necessarily works anymore… I have not yet had a chance to investigate after v3.1.0). ROOTSCAN generates an "online" datafile "histScanner.root" with the following tree structure
(13 possible modules, 16 channels per module):

```
TTree: data
   Branch: eventData
          Leafs: mult[13][16] : Int_t
                 filterEn[13][16] : Float_t
                 peakAdc[13][16] : Float_t
                 traceQdc[13][16] : Float_t
                 baseline[13][16] : Float_t
                 timeStampNs[13][16] : Double_t
                 cfdBin[13][16] : Float_t
                 timeCfdNs[13][16] : Double_t
```

E.g. you can open the histScanner.root file offline in root and plot the leafs usually with:

```
data->Draw("filterEn[0][2]>>h01(1000,1,25000)","","")
```

...will give the mca content of PIXIE module 0, channel 2.

If we are only using one module, mod=0, then all the above variables will be e.g. mult[0][16]:

1) For playing back existing runfile:

```
rootscan -f 34455 --frequency 250
```

Once rootscan is running:

```
file <filename.ldf>
clear
plot <mod> <chan> <expr> [pad]   or plot <expr> [pad]     e.g. plot 0 1 filterEn, or plot filterEn[0][1]
run
```

2) For analysing **online** polling data:
Make sure poll2 is running in another terminal. In poll2:

```
POLL2 $: shm   (toggle shared memory mode ON)
POLL2 $: startacq  (and later 'stopacq') or "run" to write to disk.
```

In another terminal, start rootscan (in shm mode):
rootscan **--shm** -f 34455 --frequency 250

Once rootscan is running:

```
rootscan $: run
```

rootscan $: divide 1 2   (1 row, 2 columns)
rootscan $: plot <mod> <chan> <expr> [pad]   or plot <expr> [pad]   e.g. plot 0 1 filterEn 1, or plot filterEn[0][1] 1

And then wait. The interface will indicate: "Waiting for Unpacker..."

Once done:
rootscan $: stop
rootscan $: quit

Typical commands are: plot, refresh, zero, clear, and divide
Possible plot <expr> options are:
You can plot the following: mult, filterEn, peakAdc, traceQdc, baseline, timeStampNs, cfdBin, timeCfdNs
(Some of those don't work as well. It's a work in progress. The first 5 should work though…)

DAMM

We do not use DAMM for plotting anymore. We keep this section here for reference only… as it took me so long to get this!

DAMM basics

Creating and altering histograms

Damm can be started offline by itself in a terminal once a suitable <outfile>.his file has been created by utkscan.

Start damm with:
$: damm

Some starter options:

DAMM->in <utkscan_outfile>.his

DAMM->fig 1 or
DAMM->fig 11 plots histogram axes in the layout specified below.

DAMM->d 1
DAMM->d 3
DAMM->d 301
DAMM->d 601
DAMM->dd 1811 (2D hist)
etc. fills the fig above with content of the data specified by the 'dir' command, e.g.:

Or you could just call them all at the same time for a "same" plot on the same axis, e.g.:
DAMM->d 1 3,601

Calling "d" by itself again reload last draw request, e.g:
DAMM->d

For e.g. four windows with different values plotted:
DAMM-> fig 4
DAMM-> win i (i=1..4)
DAMM-> d 3

To scale x-axis:
DAMM->dl lo, hi (e.g. dl 0, 1000)
DAMM->d 3 (again to redraw)

To get log y-axis:

DAMM->log

To expand a region (zoom in to x-axis region):
Plot the histo with the cursor option, i.e.
DAMM-> c 2 (instead of d 2) for ID 2
Move cursor over plot, and at the lower point where you want the region to start, press “<” on keyboard.
Move to higher position with cursor, and press “>” on keyboard.
Now press “e” to expand this region between < and >

Listing available histograms

Below is a list of histogram ID's (HID). You can access this list in DAMM by calling “ddir”, e.g.:
DAMM->ddir

You can list the available plots via the <utkscan-outfile>.list file created by utkscan, e.g.:

Useful information

Some more tutorial guides:
(Also handy help file here: http://webpages.ursinus.edu/lriley/doc/damm/damm.html#DS)

Summary of plotting procedure

Here is my quick plotting procedure:

DAMM-> in <outfile>.his
DAMM-> fig 1
DAMM-> win 1
DAMM-> d 1
DAMM-> dl 0,2000
DAMM-> d 1
DAMM-> c
(on the plot, move curse to lo position, press “<”. Move cursor to hi position, press “>”. Then press “e” for expand.)
Now to find a peak and fit it:
DAMM-> find
DAMM-> ds 1,600,1400 … this should now show several found peaks, marked with green lines.
Use the cursor to set lo and hi for fit with “[“ and “]” keys. Quit c mode with “q”., then
DAMM-> gfit 1 x … should now print the fit parameters and display the yellow fitteds gauss peak on background.

Programs installed in PAASS-LC

List of programs

[pr270user@asouev2.tlabs.ac.za/]$ ls-lrt /opt/paass/bin/

viewBaseline
set2ascii
set2root
hexReader
monitor // usage: once poll2 is acquiring data (startacq), run monitor in another terminal
listener
utkscan
skeleton
scope // usgae: scope --firmware 34689 --frequency 100
headReader // usage: headReader pulser_003.ldf
eventReader // usgae: eventReader --firmware 34689 --frequency 100
dataGenerator
set_pileups_reject
rootscan
rate
pwrite
pread
get_traces
copy_params
boot
adjust_offsets
trace
toggle
set_standard
set_pileups_only
set_hybrid
pmwrite
pmread
MCA
find_tau
csr_test
paramScan
poll2

scope

Scope shows the signal as it was sampled by the ADC. E.g., if it only has the rising edge, it means you had the TRACE_DELAY too long (i.e. where the trigger falls in the trace), or the TRACE_LENGTH (i.e. how many samples to view) was too short. Maybe a combo of both!
TRACE_LENGTH denotes how many samples should be acquired for the trace. TRACE_LENGTH = 800 ns will have 100 samples (i.e. 800 ns / 8 ns / sample).
TRACE_DELAY parameter tells the system where the trigger should fall in the waveform (typically about 20% of the length).

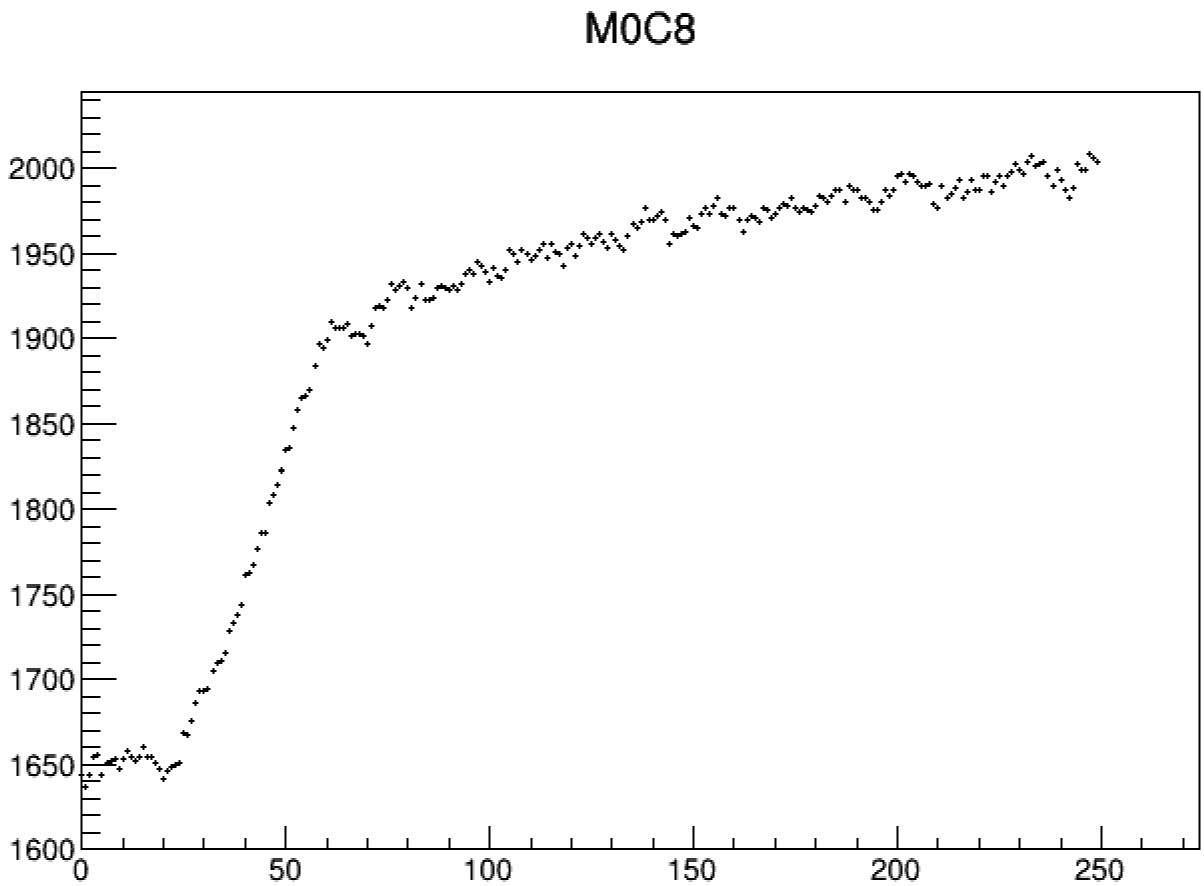
// usage:

(Note, for this the trace needs to be active for the channel, see parameter, CHANNEL_CSRA, bit #8)

- Run poll2 in **shared memory mode (shm)** and start the acquisition (i.e. in a terminal poll2->**shm-**>startacq)
- In another terminal, I run scope from the .../acq folder where poll2 is run from:
  $: scope --shm -f 34455 --frequency 250 -c ../jj-paass-pr270/install/bin/Config_PR270.xml -o scopeout (similar to utkscan)
  Scope $
  Scope $ set 0 8   (to set the <mod> and <chan> number)

```
Scope $ h
  Help:
   debug   - Toggle debug mode flag (default=false)
   file    - Usage : file <fileName> | Load an input file.
   help    - Usage (h)elp : <cmd> | Display this dialogue or just the dialog for <cmd>
   quiet   - Toggle quiet mode flag (default=false)
   quit    - Close the program
   rewind  - Usage : rewind [offset] | Rewind to the beginning of the file or to the requested number of words
   run     - Start acquisition
   stop    - Stop acquisition
   sync    - Wait for the current run to finish
   version - Usage : (v)ersion | Display version information.
   avg     - Usage : avg <number> | Set the number of waveforms to average.
   cfd     - Usage : cfd <F> <D> [L] Turn on cfd analysis of waveform. Set [F] to "off" to disable.
   clear   - Clear all stored traces and start over.
   delay   - Usage: delay <val> | Set the delay between drawing traces in seconds. Default = 1 s).
   fit     - Usage : fit <low> <high> | Turn on fitting of waveform. Set <low> to "off" to disable.
   log     - Toggle log/linear mode on the y-axis.
   save    - Usage : save <fileName> | Save the next trace to the specified file name. Do not provide the extension!
   set     - Usage : set <module> <channel> | Set the module and channel of signal of interest (default = 0, 0).
   single  - Perform a single capture.
   thresh  - Usage: <low> [high] | Set the plotting window for trace maximum.
```

With TRACE_LENGTH = 1 us (i.e. for 250 MHz/s card, 1000 ns / (8 ns/sample) = 125 samples):



headReader

```
// usage:
$: headReader pulser_003.ldf
```

eventReader

```
// usage e.g.:
$ ./eventReader -f 34455 --frequency 250 -i ~/DATA/PR283/JJDAQ/runBaGeL_337.pld -o ~/DATA/PR283/JJDAQ/runBaGeL_337-evt
```

Or

```
$: eventReader --firmware 34689 --frequency 100
        file <output_file>.ldf
        run

$ ./eventReader -h
 usage: ./eventReader [options]
  Available options:
```

```
  --batch (-b)                 - Run in batch mode (i.e. with no command line)
  --config (-c) <path>         - Specify path to setup to use for scan
  --counts                     - Write all recorded channel counts to a file
  --debug                      - Enable readout debug mode
  --dry-run                    - Extract spills from file, but do no processing
  --fast-fwd <word>            - Skip ahead to a specified word in the file (start of file at zero)
  --firmware (-f) <firmware>           - Sets the firmware revision for decoding the data. See the wiki or
                                                  HelperEnumerations.hpp for more information.
  --frequency <frequency in MHz or MS/s>- Specifies the sampling frequency used to collect the data.
  --help (-h)                  - Display this dialogue
  --input (-i) <filename>              - Specifies the input file to analyze
  --output (-o) <filename>             - Specifies the name of the output file. Default is "out"
  --quiet (-q)                 - Toggle off verbosity flag
  --shm (-s)                   - Enable shared memory readout
  --version (-v)               - Display version information
  --skip (-S) <N>              - Skip the first N events in the input file.

Help:
debug - Toggle debug mode flag (default = false)
file - Usage : file <name> | load input file
help
quiet
quit
rewind
run
stop
sync
version
```

References

[XIA] XIA PIXIE16 manual: http://www.xia.com/Manuals/Pixie16UserManual.pdf
[CAEN] Caen: http://www.caen.it/documents/News/20/WP2081_DigitalPulseProcessing_Rev_2.1.pdf)
[WIKI] PAASS-LC wiki: https://github.com/spaulaus/paass-laughing-conqueror/wiki

Appendix

Troubleshooting

gsl (GNU Scientific Library)

To install **gsl (GNU Scientific Library)** in Ubuntu, I followed these instructions (from https://astrointro.wordpress.com/2017/05/17/installing-gnu-scientific-library-gsl-in-ubuntu-16-04-and-compiling-codes/):
This is a step by step guide to installing GSL and compiling your first C code with GSL. The official page of this amazing project is https://www.gnu.org/software/gsl/

Step 1
Fetch GSL tarball from GSL website or the FTP given by ftp://ftp.gnu.org/gnu/gsl/
I use the gsl-2.3, given at the end of the list, which works in Ubuntu 16.04 very well.

Step 2
Unzip the tarball with archive manager and go to the folder that is created. It will be something like gsl-2.3.
Open a terminal and cd to this folder.

Step 3
Run the following commands in the terminal.
./configure
make
sudo make install
The process can take a few minutes to complete. Especially the make process.

Step 4
Add the following lines to your .bashrc file in home folder (this is hidden usually, so do Ctrl+H to see it)
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib/

POLL2 $ help

(see also here: https://github.com/spaulaus/paass-laughing-conqueror/wiki/Poll2-Commands)

Help:
run - Start data acquisition and start recording data to disk
stop - Stop data acqusition and stop recording data to disk
startacq (startvme) - Start data acquisition
stopacq (stopvme) - Stop data acquisition
timedrun <seconds> - Run for the specified number of seconds
acq (shm) - Run in "shared-memory" mode
spill (hup) - Force dump of current spill
prefix [name] - Set the output filename prefix (default='run_#.ldf')

fdir [path] - Set the output file directory (default='./')
title [runTitle] - Set the title of the current run (default='PIXIE Data File)
runnum [number] - Set the number of the current run (default=0)
oform [0|1|2] - Set the format of the output file (default=0)... not sure which is which (ldf / pld / ?)
reboot - Reboot PIXIE crate
stats [time] - Set the time delay between statistics dumps (default=-1)
mca [root|damm] [time] [filename] - Use MCA to record data for debugging purposes
dump [filename] - Dump pixie settings to file (default='Fallback.set')
pread <mod> <chan> <param> - Read parameters from individual PIXIE channels
pmread <mod> <param> - Read parameters from PIXIE modules
pwrite <mod> <chan> <param> <val> - Write parameters to individual PIXIE channels
pmwrite <mod> <param> <val> - Write parameters to PIXIE modules
adjust_offsets <module> - Adjusts the baselines of a pixie module
find_tau <module> <channel> - Finds the decay constant for an active pixie channel
toggle <module> <channel> <bit> - Toggle any of the 19 CHANNEL_CSRA bits for a pixie channel
toggle_bit <mod> <chan> <param> <bit> - Toggle any bit of any parameter of 32 bits or less
csr_test <number> - Output the CSRA parameters for a given integer
bit_test <num_bits> <number> - Display active bits in a given integer up to 32 bits long
save [setFilename] - Writes the DSP Parameters to [setFileName] (default='active .set from pixie_cfg')
get_traces <mod> <chan> [threshold] - Get traces for all channels in a specified module
status - Display system status information
thresh [threshold] - Modify or display the current polling threshold.
debug - Toggle debug mode flag (default=false)
quiet - Toggle quiet mode flag (default=false)
quit - Close the program
help (h) - Display this dialogue
version (v) - Display Poll2 version information

Rootscan help

----
clear - Usage : clear [padNum] | Removes all histograms on the specified pad. If none are specified the canvas is cleared.
debug - Toggle debug mode flag (default=false)

divide - Usage: divide <numPads>
Usage: divide <numXPads> <numYpads> | Divides the canvas into the selected number of pads.

file - Usage: file <fileName> | Load an input file.
help - Usage: (h)elp : <cmd> | Display this dialogue or just the dialogue for <cmd>

plot - Usage 1 : plot <mod> <chan> <val> [pad]
Usage 2 : plot <val> [pad] | Plots a new histogram for module, chan, and value specified.
If no pad is specified the plot is added to the currently selected pad.

quiet - Toggle quiet mode flag (default=false)
quit - Close the program
refresh - Usage : refresh [delayInSec] | Refreshes histograms after the delay specified has elapsed. If no delay is specified a force refresh occurs.
rewind - Usage : rewind [offset] | Rewind to the beginning of the file or to the requested number of words
run - Start acquisition
stop - Stop acquisition
sync - wait for the current run to finish
version - Usage : (v)ersion | Display version information.
zero - Usage : zero | Zeros all histograms and stored data.

Config.xml

Typical Config.xml file for utkscan:

```
<Configuration>
        <Author>
                <Name>JJ van Zyl</Name>
                <Email>jjvz AT sun DOT ac DOT za</Email>
                <Date>Sept. 28, 2017</Date>
        </Author>
        <Description>
                A simple multidetector setup for testing purposes. A single 250 MHz PIXIE16 module.
        </Description>

        <Global>
                <Revision version="F"/>
                <EventWidth unit="s" value="1e-6"/>
                <HasRaw value="true"/>
        </Global>

        <DetectorDriver>
                <Analyzer name="WaveformAnalyzer"/>
                <Processor name="Pr270Processor"/>
        </DetectorDriver>

        <Map verbose="true">
                <Module number="0" firmware="34455" frequency="250">
                        <Channel number="0" type="nai" subtype="stop">
                                <Calibration model="linear" min="0" max=”32000”>
                                        -2.384 13.61
                                </Calibration>
                        </Channel>
                        <Channel number="1" type="pulser" subtype="start"/>
                        <Channel number="2" type="ge" subtype="E"/>
                        <Channel number="3" type="plastic" subtype="E"/>
                </Module>
        </Map>

        <TreeCorrelator name="root" verbose="false"><Place type="PlaceDetector" name="Beam"
        reset="false"/>
        </TreeCorrelator>
</Configuration>
```

Pr270Processor.cpp

Typical Pr270Processor code:

```
/// @file Pr270Processor.cpp
/// @brief Experiment processor for the PR270 Experiment at iThemba labs.
/// @author S. V. Paulauskas, adapted by JJ van Zyl
/// @date January 15, 2018
/// @copyright Copyright (c) 2018 S. V. Paulauskas.
/// @copyright All rights reserved. Released under the Creative Commons Attribution-ShareAlike 4.0
International License
#include "Pr270Processor.hpp"

#include "BarBuilder.hpp"
#include "DammPlotIds.hpp"
#include "DetectorDriver.hpp"
#include "RawEvent.hpp"

///For now we're just going to do this brute force.

struct Telescope {
	double Det_energy;
	int Det_time;
	bool polarization;
};

static Telescope nai0_;
static Telescope pulser0_;
static Telescope leps0_;
static Telescope plastic0_;

namespace dammIds {
	namespace experiment {
	namespace ungated {
		const int DD_TELESCOPE0 = 0; //!< Ungated dE vs. E for Telescope 0
		const int DD_TELESCOPE1 = 1; //!< Ungated dE vs. E for Telescope 1
	}

	namespace polarizaitonUp {
		const int D_ENERGY_NAI0 = 2; //!< NaI 0 Energy Gated on Up polarized beam
	}
	}
}

using namespace std;
using namespace dammIds::experiment;

void Pr270Processor::DeclarePlots(void) {
	DeclareHistogram2D(ungated::DD_TELESCOPE0, SC, SD, "Ungated dE vs. E - Telescope 0");
	DeclareHistogram2D(ungated::DD_TELESCOPE1, SC, SD, "Ungated dE vs. E - Telescope 1");

	DeclareHistogram1D(polarizaitonUp::D_ENERGY_NAI0, SD, "NaI 0 Beam Up Gate");
}

Pr270Processor::Pr270Processor() : EventProcessor(OFFSET, RANGE, "Pr270PRocessor") {
	associatedTypes.insert("nai");
	associatedTypes.insert("pulser");
	associatedTypes.insert("leps");
```

```
	associatedTypes.insert("plastic");

	stringstream rootname;
	rootname << Globals::get()->GetOutputPath() << Globals::get()->GetOutputFileName() <<
".root";
	cout << rootname.str() << endl;
	rootfile_ = new TFile(rootname.str().c_str(), "RECREATE");
	roottree_ = new TTree("DATA", "");

	roottree_->Branch("Nai0", &nai0_, "Det_energy/D:Det_time/I:polarization/b");
	roottree_->Branch("Pulser0", &pulser0_, "Det_energy/D:Det_time/I:polarization/b");
	roottree_->Branch("Leps0", &leps0_, "Det_energy/D:Det_time/I:polarization/b");
	roottree_->Branch("Plastic0", &plastic0_, "Det_energy/D:Det_time/I:polarization/b");

}

Pr270Processor::~Pr270Processor() {
	rootfile_->Write();
	rootfile_->Close();
	delete rootfile_;
}

bool Pr270Processor::Process(RawEvent &event) {
	if (!EventProcessor::Process(event))
	return false;

	static const auto &nai_evt = event.GetSummary("nai")->GetList();
	static const auto &pulser_evt = event.GetSummary("generic")->GetList();
	static const auto &leps_evt = event.GetSummary("ge")->GetList();
	static const auto &plastic_evt = event.GetSummary("plastic")->GetList();

	bool isUpPolarized = TreeCorrelator::get()->place("Beam")->status();

	for (const auto &it : nai_evt) {
	auto location = it->GetChanID().GetLocation();
	auto energy   = it->GetCalibratedEnergy();
	auto time     = it->GetWalkCorrectedTime();


		nai0_.Det_energy = energy;
		nai0_.Det_time = time;
//		nai0_.Det_time *= (Globals::get()->GetClockInSeconds() * 1.e9);//in ns now
		if (isUpPolarized)
		plot(polarizaitonUp::D_ENERGY_NAI0, energy);
	}

	for (const auto &it : pulser_evt) {
	auto location = it->GetChanID().GetLocation();
	auto energy   = it->GetCalibratedEnergy();
	auto time     = it->GetWalkCorrectedTime();

	pulser0_.Det_energy = energy;
	pulser0_.Det_time = time;
	}
	for (const auto &it : leps_evt) {
	auto location = it->GetChanID().GetLocation();
	auto energy   = it->GetCalibratedEnergy();
	auto time     = it->GetWalkCorrectedTime();
```

```
        leps0_.Det_energy = energy;
        leps0_.Det_time = time;
        }
        for (const auto &it : plastic_evt) {
        auto location = it->GetChanID().GetLocation();
        auto energy   = it->GetCalibratedEnergy();
        auto time     = it->GetWalkCorrectedTime();

        plastic0_.Det_energy = energy;
        plastic0_.Det_time = time;
        }


        nai0_.polarization = pulser0_.polarization = leps0_.polarization = plastic0_.polarization =
isUpPolarized;

        roottree_->Fill();

        nai0_.Det_energy = pulser0_.Det_energy = leps0_.Det_energy = plastic0_.Det_energy = -
9999.0;
        nai0_.Det_time = pulser0_.Det_time = leps0_.Det_time = plastic0_.Det_time = -9999;

        EndProcess();
cout << "...Current time = " <<Globals::get()->GetClockInSeconds()* 1.e9 << endl;
        return true;
}
```

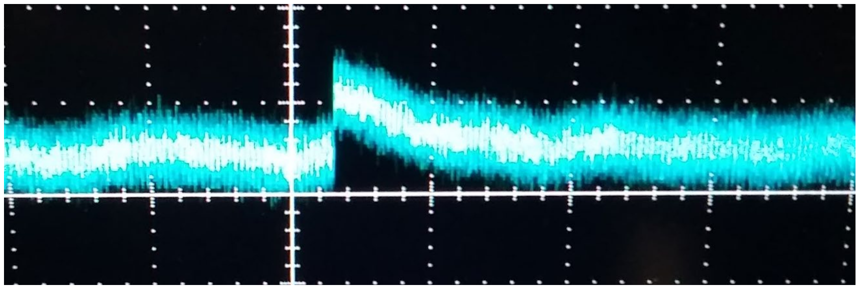
ccmake ../

CMAKE_AR	/usr/bin/ar
CMAKE_BUILD_TYPE	Release
CMAKE_COLOR_MAKEFILE	ON
CMAKE_CXX_COMPILER	/usr/bin/c++
CMAKE_CXX_COMPILER_AR	/usr/bin/gcc-ar
CMAKE_CXX_COMPILER_RANLIB	/usr/bin/gcc-ranlib
CMAKE_CXX_FLAGS	
CMAKE_CXX_FLAGS_DEBUG	-g
CMAKE_CXX_FLAGS_MINSIZEREL	-Os -DNDEBUG
CMAKE_CXX_FLAGS_RELEASE	-O3 -DNDEBUG
CMAKE_CXX_FLAGS_RELWITHDEBINFO	-O2 -g -DNDEBUG
CMAKE_C_COMPILER	/usr/bin/cc
CMAKE_C_COMPILER_AR	/usr/bin/gcc-ar
CMAKE_C_COMPILER_RANLIB	/usr/bin/gcc-ranlib
CMAKE_C_FLAGS	
CMAKE_C_FLAGS_DEBUG	-g
CMAKE_C_FLAGS_MINSIZEREL	-Os -DNDEBUG
CMAKE_C_FLAGS_RELEASE	-O3 -DNDEBUG
CMAKE_C_FLAGS_RELWITHDEBINFO	-O2 -g -DNDEBUG
CMAKE_EXE_LINKER_FLAGS	
CMAKE_EXE_LINKER_FLAGS_DEBUG	
CMAKE_EXE_LINKER_FLAGS_MINSIZE	
CMAKE_EXE_LINKER_FLAGS_RELEASE	
CMAKE_EXE_LINKER_FLAGS_RELWITH	
CMAKE_EXPORT_COMPILE_COMMANDS	OFF
CMAKE_Fortran_COMPILER	/usr/bin/gfortran
CMAKE_Fortran_COMPILER_AR	/usr/bin/gcc-ar
CMAKE_Fortran_COMPILER_RANLIB	/usr/bin/gcc-ranlib
CMAKE_Fortran_FLAGS	
CMAKE_Fortran_FLAGS_DEBUG	-g
CMAKE_Fortran_FLAGS_MINSIZEREL	-Os -DNDEBUG -Os
CMAKE_Fortran_FLAGS_RELEASE	-O3 -DNDEBUG -O3
CMAKE_Fortran_FLAGS_RELWITHDEB	-O2 -g -DNDEBUG
CMAKE_INSTALL_PREFIX	/opt/paass
CMAKE_LINKER	/usr/bin/ld
CMAKE_MAKE_PROGRAM	/usr/bin/gmake
CMAKE_MODULE_LINKER_FLAGS	
CMAKE_MODULE_LINKER_FLAGS_DEBU	
CMAKE_MODULE_LINKER_FLAGS_MINS	
CMAKE_MODULE_LINKER_FLAGS_RELE	
CMAKE_MODULE_LINKER_FLAGS_RELW	
CMAKE_NM	/usr/bin/nm
CMAKE_OBJCOPY	/usr/bin/objcopy
CMAKE_OBJDUMP	/usr/bin/objdump
CMAKE_RANLIB	/usr/bin/ranlib
CMAKE_SHARED_LINKER_FLAGS	
CMAKE_SHARED_LINKER_FLAGS_DEBU	
CMAKE_SHARED_LINKER_FLAGS_MINS	
CMAKE_SHARED_LINKER_FLAGS_RELE	
CMAKE_SHARED_LINKER_FLAGS_RELW	
CMAKE_SKIP_INSTALL_RPATH	OFF
CMAKE_SKIP_RPATH	OFF
CMAKE_STATIC_LINKER_FLAGS	
CMAKE_STATIC_LINKER_FLAGS_DEBU	
CMAKE_STATIC_LINKER_FLAGS_MINS	

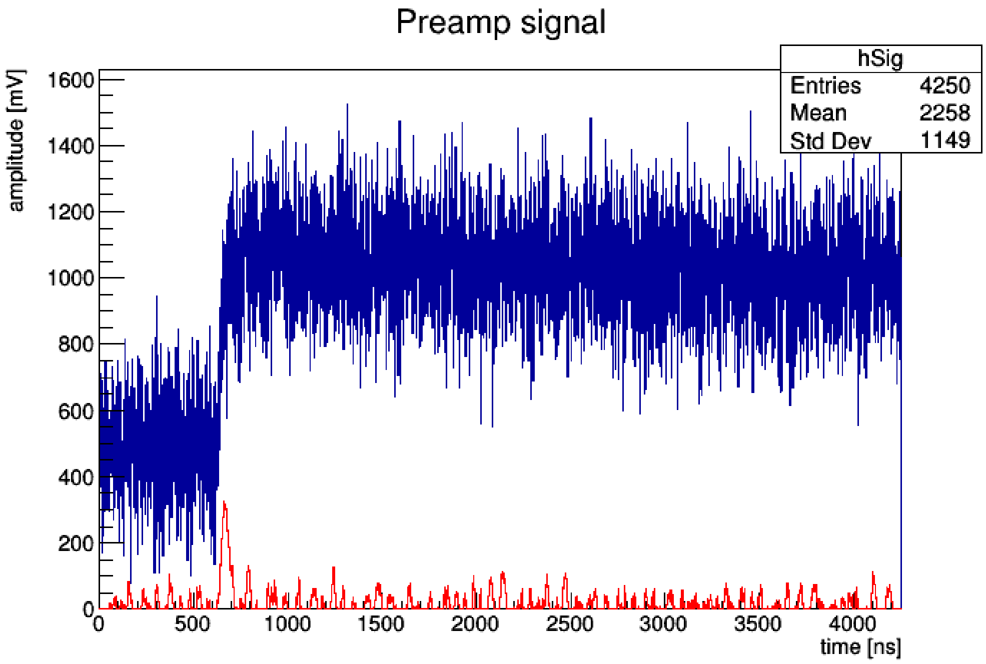
CMAKE_STATIC_LINKER_FLAGS_RELE	
CMAKE_STATIC_LINKER_FLAGS_RELW	
CMAKE_STRIP	/usr/bin/strip
CMAKE_VERBOSE_MAKEFILE	OFF
CURSES_CURSES_LIBRARY	/usr/lib64/libcurses.so
CURSES_FORM_LIBRARY	/usr/lib64/libform.so
CURSES_INCLUDE_PATH	/usr/include
CURSES_NCURSES_LIBRARY	/usr/lib64/libncurses.so
GENREFLEX_EXECUTABLE	/home/pr270user/Software/root/root-v5-34/bin/genreflex
GSL_CONFIG_EXECUTABLE	/usr/local/bin/gsl-config
PAASS_BUILD_ACQ	ON
PAASS_BUILD_SCAN_UTILITIES	ON
PAASS_BUILD_SETUP	ON
PAASS_BUILD_SHARED_LIBS	ON
PAASS_BUILD_TESTS	OFF
PAASS_BUILD_UTKSCAN	ON
PAASS_USE_DAMM	ON
PAASS_USE_GSL	ON
PAASS_USE_HRIBF	OFF
PAASS_USE_ROOT	ON
PAASS_UTKSCAN_GAMMA_GATES	OFF
PAASS_UTKSCAN_ONLINE	OFF
PAASS_UTKSCAN_TREE_DEBUG	OFF
PAASS_UTKSCAN_VERBOSE	OFF
PLX_LIBRARY_DIR	**/opt/plx/current/PlxSdk/PlxApi/Library**
ROOTCINT_EXECUTABLE	/home/pr270user/Software/root/root-v5-34/bin/rootcint
ROOTSYS	/home/pr270user/Software/root/root-v5-34
ROOT_CONFIG_EXECUTABLE	/home/pr270user/Software/root/root-v5-34/bin/root-config
XIA_FIRMWARE_DIR	/opt/xia

Some experimentation with Tracefilter and simulated signals

This is an example of a bad Si detector signal from k-line polarimeter, with vertical axis set at 10 mV/div, and horizontal axis at 400 μs/div:



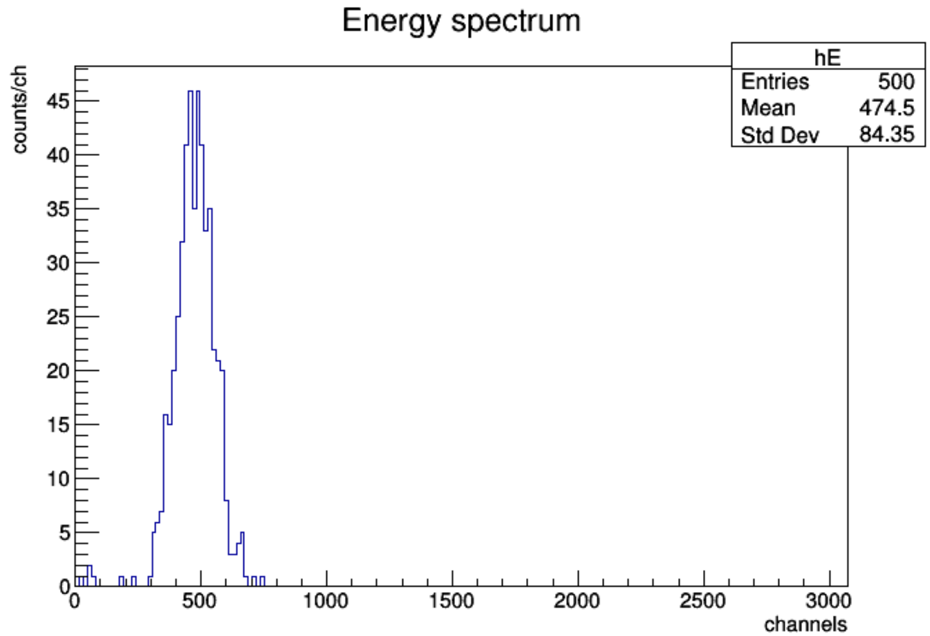
I simulate this signal and do a trace of it with a fast and slow filter to see the effects of the channel (filter) parameters on the resulting trigger and energy response. The simulated signal (blue) and resulting trace (red) is seen below.



The blue line is the simulated detector signal. The red line is the resulting trace, showing the time of the trigger above the set threshold (set to 175 adc units). If the threshold is set any higher, we miss the trace, or if the threshold is set lower, we trigger too soon (in the noise). The signal and trace parameters are, respectively:

<u>Signal</u>		<u>Trace filter</u>	
Voltage:	0.035 V	TAU:	100000 ns
Noise (fraction of signal ampl.):	0.25	Trigger rise time:	200 ns
Decay time:	200000 ns	Trigger flattop:	20 ns
Rise time:	200 ns	Trigger threshold:	175 adc units
Baseline:	500 adc units	Energy Rise time:	400 ns
		Energy Flattop:	200 ns

The corresponding calculated energy is shown below.



Utkscan options

(Taken from: Analysis/ScanLibraries/source/ScanInterface.cpp)

```
//Setup all the arguments that are known to the program.
	optionExt("batch", no_argument, NULL, 'b', "", "Run in batch mode (i.e. with no command line)"),
	optionExt("config", required_argument, NULL, 'c', "<path>", "Specify path to setup to use for scan"),
	optionExt("counts", no_argument, NULL, 0, "", "Write all recorded channel counts to a file"),
	optionExt("debug", no_argument, NULL, 0, "", "Enable readout debug mode"),
	optionExt("dry-run", no_argument, NULL, 0, "", "Extract spills from file, but do no processing"),
	optionExt("fast-fwd", required_argument, NULL, 0, "<word>",
		"Skip ahead to a specified word in the file (start of file at zero)"),
	optionExt("firmware", required_argument, NULL, 'f', "<firmware>", "Sets the firmware revision for decoding the data. "
		"See the wiki or HelperEnumerations.hpp for more information."),
	optionExt("frequency", required_argument, NULL, 0, "<frequency in MHz or MS/s>",
		"Specifies the sampling frequency used to collect the data."),
	optionExt("help", no_argument, NULL, 'h', "", "Display this dialogue"),
	optionExt("input", required_argument, NULL, 'i', "<filename>", "Specifies the input file to analyze"),
	optionExt("output", required_argument, NULL, 'o', "<filename>",
		"Specifies the name of the output file. Default is \"out\""),
	optionExt("quiet", no_argument, NULL, 'q', "", "Toggle off verbosity flag"),
	optionExt("shm", no_argument, NULL, 's', "", "Enable shared memory readout"),
	optionExt("version", no_argument, NULL, 'v', "", "Display version information")
```

Class definitions

If documented in XIA's manual -> **XiaData**
if it is something that happens after data leaves the module -> **ProcessedXiaData**

ProcessedXiaData inherits from **XiaData**
If you have a **ProcessedXiaData** object, you have access to everything in **XiaData**

XiaData header file

(**Analysis/ScanLibraries/include/XiaData.hpp**)

```
/*! \brief A pixie16 channel event
 *
 * All data is grouped together into channels.  For each pixie16 channel that
 * fires the energy, time (both trigger time and event time), and trace (if
 * applicable) are obtained.  Additional information includes the channels
 * identifier, calibrated energies, trace analysis information.
 * Note that this currently stores raw values internally through pixie word types
 *   but returns data values through native C types. This is potentially non-portable.
 */
class XiaData {
public:
	/// Default constructor.
	XiaData() { Initialize(); }

	///@brief A method that will compare the times of two XiaData classes
	/// this method can be used in conjunction with sorting methods
	///@param[in] lhs : A pointer to the left hand side of the comparison
	///@param[in] rhs : A pointer to the right hand side of the comparison
	///@return True if the time of arrival for right hand side is later than
	/// that of the left hand side.
	static bool CompareTime(const XiaData *lhs, const XiaData *rhs) {
	return lhs->GetFilterTime() < rhs->GetFilterTime();
	}

	///@brief A method that will compare the unique ID of two XiaData classes
	///@param[in] lhs : A pointer to the left hand side of the comparison
	///@param[in] rhs : A pointer to the right hand side of the comparison
	///@return Return true if left hand side has a lower ID than the right
	/// hand side.
	static bool CompareId(const XiaData *lhs, const XiaData *rhs) { return (lhs->GetId() < rhs->GetId()); }

	///@return The status of the CFD Forced Trigger Bit
	bool GetCfdForcedTriggerBit() const { return cfdForceTrig_; }

	///@return The status of the CFD Trigger bit.
	bool GetCfdTriggerSourceBit() const { return cfdTrigSource_; }

	///@return True if we had a pileup detected on the module
	bool IsPileup() const { return isPileup_; }

	///@return True if the trace was flagged as a pileup
	bool IsSaturated() const { return isSaturated_; }
```

In the **Trace.hpp** header, the description is rather: “Sets to true if the trace was flagged as **saturated by the electronics**.”
In XiaListModeDataDecoder.cpp, a code piece does:

```
///@TODO This needs to be revised to take into account the bit
/// resolution of the modules. I've currently set it to the maximum
/// bit resolution of any module (16-bit).
if (data->IsSaturated())
       data->SetEnergy(65536);
```

```
///@return True if this channel was generated on the module
bool IsVirtualChannel() const { return isVirtualChannel_; }

///@return The baseline as calculated on-board the Pixie-16 modules using
/// the energy filter. This parameter is only set if the data set
/// contains the Energy Sums in the list mode data. This baseline cannot
/// be used in conjunction with trace information.
double GetFilterBaseline() const { return filterBaseline_; }

///@return The energy that was calculated on the module
double GetEnergy() const { return energy_; }

///@return The external timestamp that was recorded on the module
double GetExternalTimestamp() const { return externalTimestamp_; }

///@return The time for the channel including all of the CFD information
/// when available.
double GetTime() const { return time_; }

///@return The arrival time of the signal without any CFD information in
/// the calculation
double GetFilterTime() const { return filterTime_; }

///@return The CFD fractional time in clockticks
unsigned int GetCfdFractionalTime() const { return cfdTime_; }

///@return The Channel number that recorded these data
unsigned int GetChannelNumber() const { return chanNum_; }

///@return The crate number that had the module
unsigned int GetCrateNumber() const { return crateNum_; }

///@return The upper 16 bits of the event time
unsigned int GetEventTimeHigh() const { return eventTimeHigh_; }

///@return The lower 32 bits of the event time
unsigned int GetEventTimeLow() const { return eventTimeLow_; }

///@return The upper 16 bits of the external time stamp provided to the
/// module via the front panel
unsigned int GetExternalTimeHigh() const { return externalTimeHigh_; }

///@return The lower 32 bits of the external time stamp provided to the
/// module via the front panel
unsigned int GetExternalTimeLow() const { return externalTimeLow_; }

///@return The unique ID of the channel.
///We can have a maximum of 208 channels in a crate, the first module (#0) is always in the second slot of the crate, and
/// we always have 16 channels
unsigned int GetId() const { return crateNum_ * 208 + GetModuleNumber() * 16 + chanNum_; }

///@return the module number
```

```
unsigned int GetModuleNumber() const { return slotNum_ - 2; }

///@return The slot that the module was in
unsigned int GetSlotNumber() const { return slotNum_; }

///@return The energy sums recorded on the module
std::vector<unsigned int> GetEnergySums() const { return eSums_; }

///@return the QDC recorded on the module
std::vector<unsigned int> GetQdc() const { return qdc_; }

///@return The trace that was sampled on the module
std::vector<unsigned int> GetTrace() const { return trace_; }

///@brief This value is set to true if the CFD was forced to trigger
///@param[in] a : The value to set
void SetCfdForcedTriggerBit(const bool &a) { cfdForceTrig_ = a; }

/// @brief Sets the value of the concatenated external timestamp
/// @param [in] a : The value that we're going to set
void SetExternalTimestamp(const double &a) { externalTimestamp_ = a; }

///@brief Sets the baseline recorded on the module if the energy sums
/// were recorded in the data stream
///@param[in] a : The value to set
void SetFilterBaseline(const double &a) { filterBaseline_ = a; }

///@brief Sets the CFD fractional time calculated on-board
///@param[in] a : The value to set
void SetCfdFractionalTime(const unsigned int &a) { cfdTime_ = a; }

///@brief Sets the CFD trigger source
///@param[in] a : The value to set
void SetCfdTriggerSourceBit(const bool &a) { cfdTrigSource_ = a; }

///@brief Sets the channel number
///@param[in] a : The value to set
void SetChannelNumber(const unsigned int &a) { chanNum_ = a; }

///@brief Sets the crate number
///@param[in] a : The value to set
void SetCrateNumber(const unsigned int &a) { crateNum_ = a; }

///@brief Sets the energy calculated on-board
///@param[in] a : The value to set
void SetEnergy(const double &a) { energy_ = a; }

///@brief Sets the energy sums calculated on-board
///@param[in] a : The value to set
void SetEnergySums(const std::vector<unsigned int> &a) { eSums_ = a; }

///@brief Sets the upper 16 bits of the event time
///@param[in] a : The value to set
void SetEventTimeHigh(const unsigned int &a) { eventTimeHigh_ = a; }

///@brief Sets the lower 32 bits of the event time
///@param[in] a : The value to set
```

```
void SetEventTimeLow(const unsigned int &a) { eventTimeLow_ = a; }

///@brief Sets the upper 16 bits of the external event time
///@param[in] a : The value to set
void SetExternalTimeHigh(const unsigned int &a) { externalTimeHigh_ = a; }

///@brief Sets the lower 32 bits of the external event time
///@param[in] a : The value to set
void SetExternalTimeLow(const unsigned int &a) { externalTimeLow_ = a; }

///@brief Sets if we had a pileup found on-board
///@param[in] a : The value to set
void SetPileup(const bool &a) { isPileup_ = a; }

///@brief Sets the QDCs that were calculated on-board
///@param[in] a : The value to set
void SetQdc(const std::vector<unsigned int> &a) { qdc_ = a; }

///@brief Sets the saturation flag
///@param[in] a : True if we found a saturation on board
void SetSaturation(const bool &a) { isSaturated_ = a; }

///@brief Sets the slot number
///@param[in] a : The value to set
void SetSlotNumber(const unsigned int &a) { slotNum_ = a; }

///@brief Sets the calculated arrival time of the signal
///@param[in] a : The value to set
void SetTime(const double &a) { time_ = a; }

///@brief Sets the calculated arrival time of the signal sans the CFD
/// fractional time components.
///@param[in] a : The value to set
void SetFilterTime(const double &a) { filterTime_ = a; }

///@brief Sets the trace recorded on board
///@param[in] a : The value to set
void SetTrace(const std::vector<unsigned int> &a) { trace_ = a; }

///@brief Sets the flag for channels generated on-board
///@param[in] a : True if we this channel was generated on-board
 void SetVirtualChannel(const bool &a) { isVirtualChannel_ = a; }
```

ProcessedXiaData

This class contains additional information about the XiaData after additional processing has been done. The processing includes, but is not limited to energy/time calibrations, high resolution timing analysis, trace analysis, etc.

(/Analysis/ScanLibraries/include/**ProcessedXiaData.hpp**)

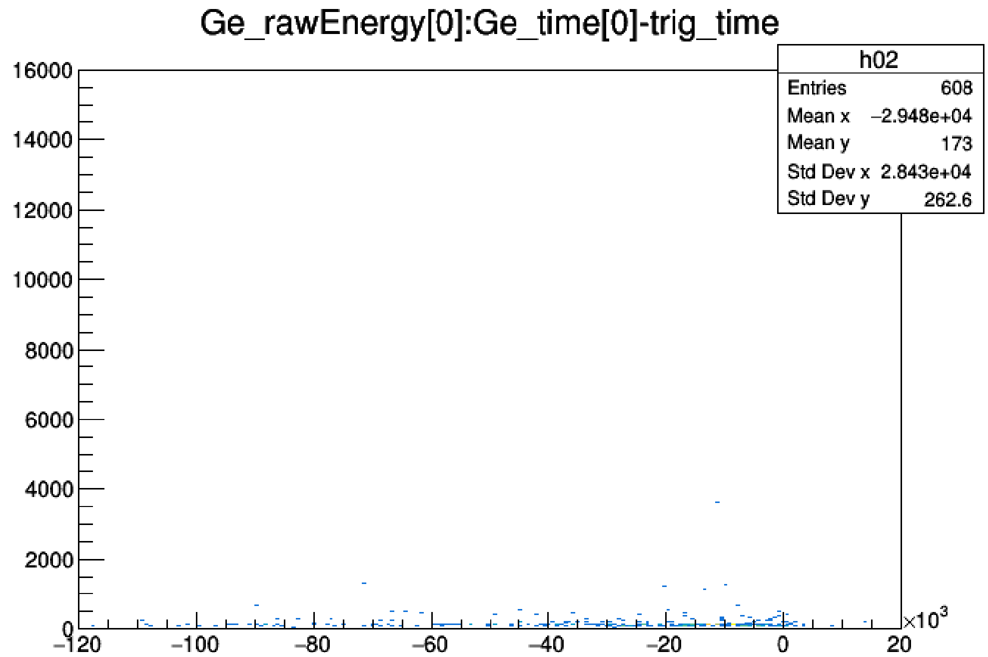
```
///This class contains additional information about the XiaData after
/// additional processing has been done. The processing includes, but is not
/// limited to energy/time calibrations, high resolution timing analysis,
/// trace analysis, etc.
class ProcessedXiaData : public XiaData {
```

```
public:
	/// Default constructor.
	ProcessedXiaData() {}

	///Constructor taking the base class as an argument so that we can set
	/// the trace information properly
	///@param[in] evt : The event that we are going to assign here.
	ProcessedXiaData(XiaData &evt) : XiaData(evt) {
	trace_ = evt.GetTrace();
	trace_.SetIsSaturated(evt.IsSaturated());
	walkCorrectedTime_ = 0;
	};

	/// Default Destructor.
	~ProcessedXiaData() {}

	///@return The calibrated energy for the channel
	double GetCalibratedEnergy() const { return calibratedEnergy_; }

	///@return The sub-sampling arrival time of the signal in nanoseconds.
	double GetHighResTimeInNs() const { return highResTimeInNs_; }

	///@return A constant reference to the trace.
	const Trace &GetTrace() const { return trace_; }

	///@return An editable trace.
	Trace &GetTrace() { return trace_; }

	///@return The Walk corrected time of the channel
	double GetWalkCorrectedTime() const { return walkCorrectedTime_; }

	///Set the calibrated energy
	///@param [in] a : the calibrated energy
	void SetCalibratedEnergy(const double &a) { calibratedEnergy_ = a; }

	///Set to True if we would like to ignore this channel
	///@param[in] a : The value that we want to set
	void SetIsIgnored(const bool &a) { isIgnored_ = a; }

	///Set to true if the energy and time are not bogus values.
	///@param[in] a : The value that we would like to set
	void SetIsValidData(const bool &a) { isValidData_ = a; }

	///Set the high resolution time (Filter time (sans CFD) + phase ).
	///@param [in] a : the high resolution time
	void SetHighResTime(const double &a) { highResTimeInNs_ = a; }

	///Sets the trace appropriately
	///@param[in] a : The trace that we want to set
	void SetTrace(const std::vector<unsigned int> &a) { trace_ = a; }

	///Set the Walk corrected time
	///@param [in] a : the walk corrected time */
	void SetWalkCorrectedTime(const double &a) { walkCorrectedTime_ = a; }
```

PR283 notes

DATA->Draw("Ge_rawEnergy[0]:Ge_time[0]-trig_time>>h02(200,-120000,20000,1600,0,16000)","","col")

Gives me this:



DATA->Draw("tac_energy[0]:trig_time-tac_time>>h02(2000,-8,0,1000,0,12000)","","col")

Gives me: