MRD DAQ
Operations Manual

Tommaso Boschi

July 21, 2016

Contents

1 DAQ overview

The MRD DAQ is based upon the ToolDAQ Framework, developed by Dr Benjamin Richards [1]. ToolDAQ is designed to incorporate the best features of other frameworks along with:

- being very easy and fast to develop DAQ implementations in a very modular way;
- including dynamic service discovery and scalable network infrastructure to allow its use on large scale experiments.

The main executable creates a *ToolChain* which is an object that holds user-defined *Tools*. *Tools* are modular classes that can be daisy-chained to a *ToolChain* and then handled sequentially by the main executable. A *ToolChain* also handles the more complicated aspects of a DAQ system, like DAQ remote control, service discovery, status while allowing for easy sequential Initialisation, Execution and Finalisation of the contained *Tools*. The *ToolChain* also holds a user defined DataModel which each tool has access too and can read, update and modify. This is the method by which data is passed between *Tools*.

Users need to write their own modular *Tools* for hardware control and data processing while the software provides a powerful set of features out-of-the-box:

- three execution modes: Interactive, Inline and Remote;
- built-in distriuted Network DAQ control through command line or Web Interface
- built-in Dynamic Service Discovery and Publishing
- Remote or Local Logging modes
- simple threaded scalable Fault tolerant NtoN;
- networking technology provided by ZMQ
- configuration file tools using a universal data storage class.

ToolDAQ is an open source data acquisition Framework, available on GitHub [2]. One of the branches of the repository holds the MRD DAQ, in which the ToolDAQ Framework implements purpose-written classes which interface with the front-end electronics.

2 CAMAC classes

The MRD and the FACC veto of the ANNIE experiment rely on CAMAC electronics for front-end read out of the PMTs. In particular, Lecroy modules are employed, such as the Lecroy3377 32-channels 10-bit TDC [4] and the Lecroy4300B 16-channels 11-bit ADC [5]. All the cards are addressed via the Weiner CCUSB controller module [3]. The vendor provides a `C++` class to interface the controller with a CPU, called `libxxusb`. In turn, the controller usb class makes use of a kernel-level `C++` usb class.

In order to handle more easily the CAMAC modules, some user classes were developed:

- a base class, `CamacCrate`;
- a derived class for the TDC, `Lecroy3377`;
- a derived class for the ADC, `lecroy4300b`.

2.1 `CamacCrate`

The base class takes care of opening the usb connection for the CCUSB controller. It also takes care of assigning IDs to the derived objects, which correspond to CAMAC modules. In this way, it is feasible to store informations on the cards, such as the Slot number. It allows the configuration of the USB stack, a sequence of CAMAC commands to be executed sequentially. Related to the stack, functions to use the CCUSB's FIFO are available. CAMAC functions are delivered by the `READ` and `WRITE` procedures, using NAF addressing[1]. These functions are nothing but wrapper functions of CAMAC own functions.

Furthermore, as a base class, it exploits polymorphism so as to make the main code lighter (see 3.2), when using the derived classes.

2.2 `Lecroy3377`

This derived class allows an easy handling of the TDC modules. Single word mode is supported only, both Common Stop or Common Start version. Every CAMAC function is implemented, as long as a private system for registers setting.

To read data from the TDC, the function `GetData` should be called. This procedure smartly reads the card's FIFO storing data in a STL map container. It automatically uses the proper readout operation, depending on the register setting. The retrieved map has as "key" the channel number, while as "value" the datum itself.

2.3 `Lecroy4300b`

This derived class allows an easy handling of the ADC modules. Every CAMAC function is implemented, as long as a private system for the register setting.

To read data from the ADC, the function `GetData` should be called. This procedure smartly reads the card's channels toring data in a STL map container. It automatically changes reading method for either compressed data or sequential readout. The retrieved map has as *key* the channel number, while as *value* the datum itself.

3 MRD DAQ tools and configuration files

The MRD DAQ's *Tools* employs the CAMAC classes to interface with the controller and the cards. In order to communicate with each other, some variables were added to the DataModel. A single structure, called `Module`, handles both the class objects and the data structures. The Camac classes (`CC`) are saved as a map of vectors of pointers to the `CamacCrate` class. The *key* is a string and defines the type of the *value*, which is a pointer to one of the CAMAC classes:

`TDC` the vector holds `Lecroy3377` pointers;

`ADC` the vector holds `Lecroy4300b` pointers.

[1] Module addressing is achieved knowing the slot **N**umber, the sub**A**ddress, and the **F**unction code.

The data retrived from the CCUSB controller are also stored in the same structure as a map. Even in this case, the *key* distinguish the TDC's data from the ADC's. The *value* is another structure, called `Card`. Since it's not guaranteed that every card will give some outpu, this structure contains two synchronised objects:

- a vector holding of the Slot number, to identify the read card;
- a vector of the definitive map, which is handed to the `GetData` functions.

The corresponding code is the following:

```
struct Module
{
        std::map<std::string, Card> Data;
        std::map<std::string, std::vector<CamacCrate*> > CC;
}

struct Card
{
        std::vector<Channel> Num;
        std::vector<int> Slot;
};

struct Channel
{
        std::map<int, int> ch;
};
```

Three *Tools* have been written for the MRD DAQ:

- a class for triggering and initialising the DataModel objects: `CCTrigger`;
- a class for setting up the modules and reading them: `Lecroy`;
- a class for saving retrived data to a ROOT file: `TreeRecorder`.

Each *Tool* has its own configuration file.

3.1 CCTrigger

This *Tool* is set by the `configfiles/TRGConfig` configuration file.

`configcc` is the path to the crate configuration file;

`verbose` is the *Tool* verbosity;

`percent` is the probabilty of card firing, valid for soft triggering only;

`trg_mode` is the trigger mode.

The existing cate configuration file is `configfiles/TRGConfig`. It's used by the *Tool* in the Initialise procedure to set the `CC` entry of DataModel's `Module`.

Three are the Suported types:

`ADC` stands for Lecroy4300B;

`TDC` stands for Lecroy3377;

`TRG` stands for Lecroy3377 designated as trigger board.

If no card is appointed as `TRG`, the the first TDC found is used. In order to create the objects, the slot number and the register configuration file are also needed. The CAMAC classes take care of properly construction of the objects.

Example of configuration files are `configfiles/TDCreg` and `configfiles/ADCreg`. Please refer to the Lecroy manual in roder to set the cards properly.

Trigger is made reading the FIFO of the specified card: if there's data in it, then all cards are read. When a trigger is found, a DataModel bool variable is set by this *Tool* which allows the Execution of the other *Tools*.

There are three modes for the trigger:

0: external trigger, provided by module tpye TRG;

1: software trigger, cards accessed randomly, with programmable probability;

2: software trigger and test functions are called.

3.2 Lecroy

This *Tool* is meant to work for both the TDCs and ADCs cards. If only either TDCs or ADCs are employed, then only one tool should be added in the *ToolChain*. Otherwise, if both are used, then two *Lecroy Tools* are required. In the corresponding configruation file the *Tool* is choosen to be for TDC or ADC. Example of these files are `configfiles/TDCConfig` or `configfiles/ADCConfig`.

If a trigger is found, then the Exectuion functions is allowed to be run. This procedure test the cards if data are available, and if so, collected and stored in the `Data` entry of DataModel's `Module`.

3.3 TreeRecorder

The last *Tool* fills a ROOT tree and save it to file. The output path and output name are set by `configfiles/TreeConfig`. In the same configuration file is possible to set the entry cap, to which a new file is opened.

The tree has the following branches:

Trigger: incremental number of the trigger;

OutNumber: number of channels read;

TDC: array of OutN storing the TDC values;

TSlot: array of OutN storing the slot number of TDCs;

TChannel: array of OutN storing the channel number of TDCs;

ADC: array of OutN storing the ADC values;

ASlot: array of OutN storing the slot number of ADCs;

AChannel: array of OutN storing the channel number of ADCs;

TimeStamp: time stamp of the entry, in unix time.

The TimeStamp is computed from the time specified in the *Tool* configration file. It's advisable to use Epoch time, 1970/01/01.

4 Building and Usage

Since the MRD DAQ is part of the ToolDAQ Framework, the executable is built with

```
make clean
make
```

and in Interactive mode it is run with

```
./main
```

5 Post processing

The ROOT file generated by the DAQ as conceived to be as light as possible, in order to limit writing-to-disk dead times. Hence, the raw ROOT files must be processed, for many reasons.

The Timestamps saved in the ROOT file are given by from the computer (annielx01 for now) time on which the code is run. On the other hand, local time is obtained from NTP servers, within an accuracy less than 40 ms. The delay from NTP server is not constant, therefore an algorithm is required to fix the timestamps with the help of IFBeam database. Doing so, it will be possible to relates MRD data with PMT data.

Moreover, the TDCs and ADCs values are converted, from raw bit number to physical quantity (time and charge) and each channel is mapped to the corresponding PMT.

The post processor requires as sole argument the raw ROOT file. Timestamps are retrieved from the database by `wget` and saved to a text file. Tue URL employed is

```
http://ifb-data.fnal.gov:8100/ifbeam/data/data?e=e%2C1d&b=BNBBPMTOR&
        f=csv&tz=&action=Show+device&t0={0}&t1={1}
```

where t0 is the start time and t1 the end time, in unix time with millisecond precision.

References

[1] Richards, B. (2016). *ToolDAQ Framework (7th Open Meeting for the Hyper-Kamiokande Project)* [pdf slides]. Retrived from `http://indico.ipmu.jp/indico/materialDisplay.py?contribId=26&materialId=slides&confId=94`

[2] ANNIEDAQ repository on GitHub `https://github.com/ANNIEDAQ/ANNIEDAQ`

[3] Manual of the CC-USB CAMAC Controller with USB interface by Weiner `file.wiener-d.com/documentation/CC-USB/WIENER_CC-USB_Manual_6.00.pdf`

[4] Technical datasheet of the Lecroy3377 module `http://teledynelecroy.com/lrs/dsheets/3377.htm`

[5] Technical datasheet of the Lecroy4300B module `http://teledynelecroy.com/lrs/dsheets/4300b.htm`