

Wiener-DAQ program for data acquisition with Wiener CC-USB CAMAC Controller

Strahinja Lukić ¹⁾*

** Vinča Institute, University of Belgrade, Serbia*

Abstract

This document provides information and the user manual for the Wiener-DAQ software for data acquisition with the Wiener CC-USB CAMAC controller.

This version refers to Code Revision ≥ 209

¹slukic@vinca.rs

1 Introduction

The program `Wiener-DAQ` has been made primarily for the purpose of studies at Fermilab of scintillator modules with WLS fibers for future collider detectors. Its main goal is to provide automated acquisition of amplitude and timing data via the Plein&Baus Wiener CC-USB CAMAC crate controller [1]. In the present code revision (see title page) the program provides for the acquisition and monitoring of 4 ADC and 4 TDC channels.

The program relies on `libxxusb` library provided by the the manufacturer (Plein&Baus GmbH) for the communication with the crate controller [1], and on CERN `ROOT` [2] libraries for the GUI, monitoring and storage of the data.

When started, the program will look for Wiener controllers at USB ports of the computer, and connect to the first CC-USB crate found, or report an error if no crates are found. Three GUI windows will be opened:

Acquisition Control window (Fig. 1) contains control buttons to `Start` and `Stop` the acquisition, to `Save` the collected data in a `ROOT` file, to `Export Text` format data and to `Exit` the program. In addition, the number of collected events is displayed in this frame.

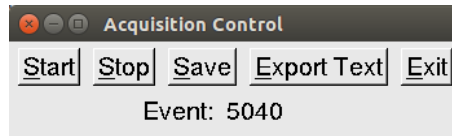


Figure 1: Screenshot of the Acquisition Control window.

SiPM Monitor window (Fig. 2) displays histograms of collected data from ADC channels 1 and 2 and TDC channels 1 and 2 in 1D, as well as 2D histograms of ADC1 vs. ADC2 and TDC1 vs. TDC2.

Scintillator Monitor window has identical layout as the SiPM window. It displays histograms of collected data from ADC channels 3 and 4 and TDC channels 3 and 4 in 1D, as well as 2D histograms of ADC3 vs. ADC4 and TDC3 vs. TDC4.

2 How to get and install

The software and this manual are available from the subversion repository <http://vinhep.vin.bg.ac.rs/strahinja-software/SiPM>. Makefiles for Windows (Makefile.win32) and Linux (Makefile) are provided. In the following, prerequisites and instructions for installation are given for linux and Windows.

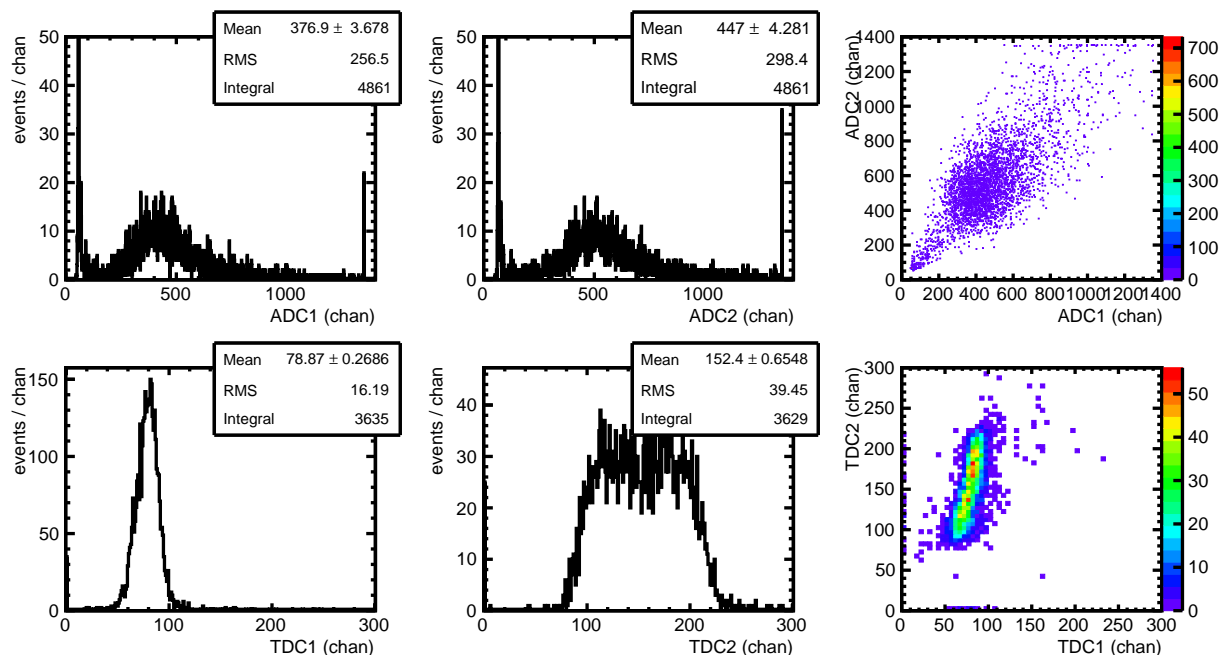


Figure 2: Layout of the "SiPM Monitor" window.

2.1 Linux

Installation requirements:

1. A working installation of GCC. Any version supporting C++11 should be fine, although the program has so far been tested only with GCC v4.9.2.
2. CERN ROOT data analysis framework. Tested with ROOT version 5.34.
3. libusb and libusb-dev packages
4. libxxusb library by the manufacturer available at <http://file.wiener-d.com/cd/XX-USB/>.

Installation instructions:

In Makefile, adapt the lines defining XXUSB_HOME and ROOTSYS to point to the folders where libxxusb and ROOT are installed on your system.

Then do:

```
make wiener-daq
```

It should install the executable into your personal \$HOME/bin folder.

Depending on your system, it may happen that the usb device is automatically mounted as owned by root. In that case libusb cannot access it if run with normal user privileges. The solution is to

add a file with the following content to your `/etc/udev/rules.d/` folder (For linux variants other than Ubuntu/Debian, the exact location of the rules folder may vary):

```
# Rules to grant normal users rw privileges to Wiener USB devices
# for CAMAC and VME crate controllers
SUBSYSTEM=="usb", ATTRSIdVendor=="16dc", MODE:="0666"
```

This file is also found in the subversion repository under the name `wiener.rules`.

2.2 Windows

Installation requirements:

1. A working installation of MS Visual C++.
2. CERN ROOT data analysis framework binary compiled with the same version of MS Visual C++ as the one used for the compilation of Wiener-DAQ. Tested with ROOT version 5.34. Tested with root 6 in simulation mode.
3. `libusb-win32` package
4. `libxxusb` library by the manufacturer available at <http://file.wiener-d.com/cd/XX-USB/>.

Installation instructions:

Prior to the installation, the batch file for the setup of the MS Visual C++ environment variables should be executed, then `%ROOTSYS%\bin\thisroot.bat`. After this, compile the code with `nmake`. Typically, a sequence similar to the following will do the job,

```
call "C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat"
call "C:\Root_v5_34\bin\thisroot.bat"
nmake /f Makefile.win32 wiener-daq.exe
```

3 How to use the program

Before using the program, make sure that the relevant environment variables are properly set.

For linux, this is normally accomplished by sourcing `thisroot.sh`, and making the location where the compiled `libxxusb` library resides (by default `XXUSB_HOME/lib`) known to your system either via `ldconfig` or by adding it to the `$LD_LIBRARY_PATH` variable. The program is invoked from the command line as,

```
wiener-daq <configuration file>
```

On windows, the batch file setting the environment variables of MS Visual C++ must be executed, and then `thisroot.bat` to set the environment for the ROOT libraries. Then the program is invoked as,

```
wiener-daq.exe <configuration file>
```

3.1 Configuration file

The configuration file contains all the relevant parameters for the communication with the crate controller, and the display parameters for monitoring the acquisition. Wiener-DAQ parses the configuration file line by line, looks for parameter keywords/names, and if a parameter keyword is found in a line, reads the parameter values from the same line or, in some cases, from the subsequent lines. If no recognized parameter keyword is found in a line, the line is ignored. If the first character in a line is a hashtag (#) or an exclamation mark (!), the line is considered a comment and is ignored by the program regardless of content.

The structure of the configuration file is fairly free. Keywords are case-insensitive. Occurrence of characters or patterns other than the keyword and the numerical values of the parameters is ignored, so that special characters such as ":" or "-" can be used to improve the readability of the config file by the user.

In the following the rules for setting of individual parameters in the config file are described.

3.1.1 Monitoring display parameters

Window keyword in the same line with two integer numbers defines the initial size in pixels of the monitor windows displaying histograms of collected data.

Example:

```
Window: 900 x 500
```

sets the window size to 900 pixels wide and 500 pixels high. The characters ":" and "x" are ignored by the parser, and are present just as a reading aid to the user.

ADC n , where $n = (1,2,3,4)$, in the same line with two integer numbers defines the histogram limits of the ADC channel $\#n$ for the monitor display.

Example:

```
ADC3: 0 - 1400
```

sets the histogram limits for the display of data from the ADC channel #3.

TDC n , where $n = (1,2,3,4)$, in the same line with two integer numbers defines the histogram limits of the TDC channel $\#n$ for the monitor display.

Example:

```
TDC2: 0 - 400
```

sets the histogram limits for the display of data from the TDC channel #2.

Reduction keyword, in the same line with three integer values defines the reduction factors for the display of the 2D histograms in the monitor windows. As 2D histograms contain a large number of bins, their plotting can be fairly slow. To avoid slowing down the acquisition and losing events, this keyword offers the possibility to reduce the frequency of updating the 2D histograms (first value), as well as to set the bin width in x (second value) and y (third value) to more than 1, thus reducing the number of bins to plot.

Example:

```
Reduction:  f 10, x 4, y 4
```

reduces the update frequency of the 2D histograms to each 10th data transfer from the crate controller, and sets the bin width in x and y directions to 4, thus reducing the number of bins by a factor 16. The characters ":", "f", ",", "x" and "y" are ignored by the parser, and are present just as a reading and orientation aid to the user.

3.1.2 Crate-controller parameters

Stack keyword announces that the lines to follow contain the stack of CAMAC NAF commands in hexadecimal format to be transferred to the crate controller. This sequence of commands will be executed by the crate controller upon reception of a NIM trigger, a LAM signal, or a USB trigger, depending how the acquisition is configured. For a detailed description refer to the CC-USB manual [1].

The first line following the keyword "Stack" represents the length of stack, l , in words. The following l lines beginning with "0x" will be interpreted as stack commands in hexadecimal format. Comment lines may be inserted between stack lines.

Example:

```
stack:
# Length of stack
9
# NAF commands given in hex format here
# Read ADC1 (slot 8, channel 0)
0x1000
# Read ADC2 (slot 8, channel 1)
0x1020
# Read ADC3 (slot 8, channel 2)
0x1040
# Read ADC4 (slot 8, channel 3)
0x1060
# Read TDC1 (slot 12, channel 0)
0x1800
```

```
# Read TDC2 (slot 12, channel 1)
0x1820
# Read TDC3 (slot 12, channel 2)
0x1840
# Read TDC4 (slot 12, channel 3)
0x1860
# Clear all channels
0x393d
```

Len and **buffer** keywords found in the same line with an integer value set the CC-USB data buffer length in 16-bit words. Wiener CC-USB crate controller allows buffer lengths of a single event, 64 words, 128, 256, 512, 1024, 2048 or 4096 words. The value found in this line will be rounded up to the next allowed value, unless it is smaller than 32, in which case, the single-event length will be set. If a keyword "single" is found in the line, the single-event length will be set.

Example:

```
Length of buffer: 512
```

Trig and **delay** keywords found in the same line with an integer value set the delay time in microseconds between the arrival of the trigger event to the crate controller and the execution of the stack. This delay should be long enough to allow all modules which will be read out to finish digitization of signals.

Example:

```
Trigger delay: 100
```

LAM and **timeout** keywords found in the same line with an integer value set the LAM waiting timeout in the LAM mode.

LAM and **mask** keywords found in the same line with an integer value set the LAM mask. The lowest 24 bits of the integer value will define whether LAM signals will be accepted (bit set to 1) or ignored (bit set to 0) from the corresponding CAMAC slot.

Bulk and **buff** keywords found in the same line with an integer value set the number of data buffers to be collected for bulk data transfer. This option allows optimal utilization of the high speed of the USB data transfer in measurements with high event rates.

Bulk and **timeout** keywords found in the same line with an integer value define the waiting timeout in seconds in excess of 1 s, after which the collected data will be transferred even if the number of buffers for bulk transfer has not been filled.

Simulation keyword declares a simulation run. The program will not attempt to connect to the crate controller, but will work with data from random-number generator as if they were coming from the CC. This keyword is for debugging purposes only.

An example configuration file with comments to get you started is available at

<http://vinhep.vin.bg.ac.rs/strahinja-software/SiPM/example.conf>

4 Final remarks

For further questions, comments and requests email slukic@vinca.rs

References

- [1] *CC-USB User Manual*, tech. rep. Revision 6.00, Plein&Baus GmbH, 2012.
- [2] .