DAQ C++ Software Manual

Contents

Chapter 1

Introduction

This manual is meant to help someone install the necessary software to run the DAQ C++ system and describe how the DAQ C++ software working.

1.1 Installing Required Packages

The DAQ C++ system depends on several external applications, which must first be installed. They are:

- Linux
- gnuplot 4.4.0
- C++11

In addition to these required applications, a MySQL server is recommended to be installed for convenience.

1.2 Creating Required Folders

We need to create several folders under readout directory to store data. Make sure we have all the folders listed in the path below (/home/ is the local path to readout folder):

/home/readout/data1/CRTTime/DATA
/home/readout/data1/OVDAQ/DATA

If we want to get a normalized plot for PMT data finally, the “Scripts_mb” folder should be created under readout directory and the “normalized.txt” file should be copied into that folder.

1.3 Readout Program

There are two ways to start the readout program on Linux.

- **Run the readout file manually**. Firstly, locate the "readout" file (usually in /home/readout/bin folder, /home/ is the local path to readout folder). Secondly, check if DCONLINE_PATH is defined in the current environment (use the "env" command"). If not, we can use the following command to update it:

  ```
  export DCONLINE_PATH="/home/readout"
  $DCONLINE_PATH              //this command is used to check if we set
                              //the right path
  ```

  Then, run the following command in the bin folder:

  ```
  ./readout –d 1 –g 1
  ```

  Usually the readout file should start running smoothly. We can list the readout using command:

  ```
  ps aux | grep readout
  ```

  If we want to kill that readout:

  ```
  kill -9 xxxxxx (the readout id)
  ```

  If nothing listed after we started readout. One thing could happen is that we didn't give the write access to the USB, check the USB writing permission:

  ```
  ls –l /dev/bus/usb/00*
  ```

  We can give the writing permission to the USB:

  ```
  sudo chmod o+w (or a+w) /dev/bus/usb/00*
  ```

  Type in the user password if asked and then start the readout again.

- **Using the script to start readout**. The other way is much easier if we have the start_readout.sh script. We could search for that file in the system or it usually locate at /home/readout/script. If it exists, we can start it with command:

  /home\/readout\/script\/start_readout.sh “readout” “1” “1” “none”

1.4 Debugging

Once the readout program is running, a few things should be checked (We could also check it when readout program can’t be started). All the log files for output and errors should be located at /home/readout/DCOV/log folder.

- Check that the correct USB number is shown in the usb_main_out.log file.

  tail –f usb_main_out.log

  If not, usually the readout is not working properly, check the usb_main_err.log for errors.

- Check that there aren’t any errors in the usb_(usb number)_err.log file.

- Check that the readout is working well using usb_(usb_number)_out.log file. Keep the file open so we can monitor the USB when we are taking data.

The above completes the preparation for the readout of the OVDAQ.

Chapter 2

OVDAQ

This OVDAQ system is based on C++. For the integrity of the system, please make sure all the files needed are in the cpp_readout folder.

```
backup      CRT.h       Makefile       sevenboard.cpp  usbreadout.cpp
baselines   decode      Makefile1      unpackbaseline  usbreadout.h
CRT.cpp     histogram   oneboard.cpp   unpacksignal

./backup:
CRT.cpp  usbreadout.cpp

./baselines:
baselines  baselines.cpp

./decode:
decode  decode.cpp

./histogram:
histogram  histogram.cpp

./unpackbaseline:
unpackbaseline  unpackbaseline.cpp

./unpacksignal:
unpacksignal  unpacksignal.cpp_
```

The backup folder is just used to store the copies of original file in case the files are changed accidently.

2.1 Compiling C++ Program

The program has Makefile to help us generate executable oneboard or sevenboard program. However, this Makefile can only compile the CRT.cpp and usbreadout.cpp files. If we made any changes in other files (baselines, histogram,

unpackbaseline, unpacksignal), please make sure they are compiled before using. A separate compilation example: If baselines code is changed, using the command followed to compile it again:

```
g++ -o baselines baselines.cpp –std=c++0x
```

The Makefile could execute two files depends on which one we need. To execute oneboard program only:

```
make oneboard
```

To execute sevenboard only:

```
make sevenboard
```

To execute both programs:

```
make
```

To clean all these:

```
make clean
```

2.2 Functionality of the C++ Program

The oneboard and sevenboard programs we have work exactly the same as the programs in Perl language. Here shows how the functions in oneboard program work step by step.

- **Initialize values and settings**. We have three functions to initialize settings. And two modes are available for choosing: ‘debug’ and ‘mysql’. ‘debug’ mode could load the settings in the function and could be changed directly in these functions. ‘mysql’ mode could load the settings from MySQL table, and this mode is much convenient as we only need to update the SQL table if we want to make any change. The functions we can choose are listed:

loadconfig (string mode_local, int usb_board, int pmt_board, int triggerbox);
loadpmtdata_auto (int usb, int pmt, int pmtserialnumber, string mysql_table);
loadpmtdata (int usb, int pmt, int SWITCH, string gate_override);

In the loadconfig function, the 'mode_local' input decides which mode will be used.
In the loadpmtdata_auto function, if we do not define a 'pmtserialnumber' value, then the system will use 'debug' mode, otherwise, it will load the settings from MySQL table.
In the loadpmtdata function, the 'SWITCH' input controls which mode we will use. If SWITCH = 0, 'mysql' mode will be used. Otherwise, 'debug' mode will be used.

- **Test system time**. Before taking baseline data, we can use the test_system_time function to do a test run to see if everything is working properly.

  test_system_time (int usb, int pmt);
  check_system_time ();

  When we run the test_system_time function, the PMT will keep taking data until we run the initializeboard function. And the check_system_time function will check if the USB number is right and if we are taking data.

- **Take baseline and initialize PMT board**. The function we use here is called initializeboard:

  initializeboard (string define_runnumber, int trigger_num, int pmtini, int pmtfin);

  In the initializeboard function, we will create data folder for current run, set the run number, take the baseline data, initialize the pmt board and write the summary.txt file.
  The 'trigger_num' input defines how many trigger times we want. If we only have one PMT and PMT board, we can leave pmtini and pmtfin undefined.

- **Start and stop taking data**. We can use the starttakedata and stoptakedata functions to take data as long as PMT is connected.

  ```
  starttakedata (int pmtini, int pmtfin, int boxini, int boxfin);
  stoptakedata (int pmtini, int pmtfin, int boxini, int boxfin);
  ```

  If we only have one pmtboard and box, it is ok to leave all the values undefined (give 0 to all the input values).

- **Data Processing**. After taking data, we use generatecsv, signal (called by generatecsv) and plotdatamb to process the data.

  ```
  generatecsv (int usb, int pmt);
  signal (string dir, string file, string baselines, int pmt_board, string homedir);
  plotdatamb (int usb, int pmt);
  ```

  The generatecsv function generates the baseline.dat file and call the signal function. Then the signal function will deal with the data taken and generate all the plots we need. The plotdatamb function is not necessary, it just processes the data recorded in the summary.csv file.

- **Difference between oneboard and sevenboard**. The only difference between oneboard and sevenboard functions is data processing. For the sevenboard, we are using the generavecsv2 to process data:

  ```
  generatecsv2(int usb, int pmt);
  ```

  This function could process data for each PMT separately and save all the plots and data files into the separate PMT folders.

2.3 Running C++ Program

Here shows a run example for oneboard function.

```
Found previous instances of readout. Killing...
Starting
Enter pmt serial number without PA (example 4673) for board 3:
5268                                                                            ]
Re-enter pmt serial number without PA (example 4673) for board 3:
5268                                                                            ]
Connected to MySQL database
Found info for: PMT_serial = 5268, board_number = 3

Please check that your setup has the following setting:
USB address = 33
PMT serial number=5268
Board number=7
HV setting=702
DAC threshold = 938
Use of maroc2 gain constants = yes
Gate = on
hdelay = 5
trigger_mode = 0
Please enter if you want to take data with that settings. If instead you would like to
 change any settings type change.
yes                                                                             ]


Please enter comments for this run (max 50 characters), press enter when done
test                                                                            ]


off
Warning! Gate override initiated by user! Now using gate = off
Initializing time check for the CRT ..
start data taking ....
Finish test system time
DataPath=/OVDAQ/DATA/, Disk=1
pmtini = 1, pmtfin = 1
Bseline data taking .....................: 5 sec
Initializing .
Loading PMT: 3
finished initializing
usb_local=33 and pmt_local=3.
File open for writing
..... Taking data .....
10 sec...shutting down ...........
Done check rate!
PMT 3:   hits: 500
Getting data for typical module...
Getting baselines...
Getting signal file.../e/h.0/localdev/readout/data1/OVDAQ/DATA/Run_0001025/binary/sign
al.dec
Calculating..
Histogramming...
Plotting...
Plotting data for USB=33 and PMT=3
Done!
```