GDDAQ

发布 *beta*

Hongyi Wu(吴鸿毅)

2019 年 10 月 23 日

content:

Welcome to PKUXIADAQ’s guides.

CHAPTER 1

Introduction

1.1 Version

We recommend users to download a stable version

1.1.1 Stable Version

Stable Version Version:2019.10.23

Download the latest version, please click: PKUXIADAQ stable

For web page manual, please click: English/简体中文

- manual version in reStructuredText: README/
- manual version in offline web: docs/
- manual version in pdf: README_en.pdf README_ch.pdf

1.1.2 Pre-alpha

Pre-alpha Version:2019.10.23

Download the latest version: PKUXIADAQ

manual version in web page: 简体中文/English

- If you have any comments or suggestions for this project(function addition or improvement), please send an email to Hongyi Wu(wuhongyi@qq.com).
- We will improve the Chinese/English manuals of the software as soon as possible. Currently, we mainly explain the use of the software through our demonstration.

1.2 About

This manual applies only to XIA LLC Pixie-16

- This program is developed by the Group of Experimental Nuclear Physics, Peking University .
- The earliest graphical interface development of this program is based on NSCL DDAS Nscope .
- Thanks to *Hui Tan' s(XIA LLC)* support for our development.

Technical adviser:

- Zhihuan Li 李智焕
- Hui Tan 谭辉 (XIA LLC)
- Wolfgang Hennig(XIA LLC)

Software Developer:

- **2015 - 2016**
  - Jing Li 李晶 (lijinger02@126.com)
- **2016 - now**
  - Hongyi Wu 吴鸿毅 (wuhongyi@qq.com)

Principal author of the instruction:

- Diwen Luo 罗迪雯
- Hongyi Wu 吴鸿毅
- Xiang Wang 王翔

Art director:

- Yi Song 宋🄵

The development of this program is supported by the following:

- XIA LLC
- Institute of Modern Physics, Chinese Academy of Sciences(IMP)
- China Institute of Atomic Energy(CIAE)
- The University of Hong Kong(HKU)
- Shandong University, Weihai(SDU)
- …

This program is applicable to XIA Pixie16 module, which supports the 100/250/500 MHz sampling rates(specifically, information about the module can be found in File->About in the graphics software) and supports up to 8-chassis synchronous operation, that maeans, at least 1600-channel signals are simultaneously collected. **This package requires the CERN ROOT6 version and the resolution display above 1920x1080**.

The program is designed to be compatible with the 100/250/500 MHz modules. Simply add the firmware location of the corresponding sample rate modules to cfgPixie16.txt. The program can automatically identify the module' s type and load the corresponding firmware.

Currently we have tested most types of modules, so you can run the type of modules of our tested by default. To support other types, please contact XIA LLC to obtain the corresponding firmware or contact Hongyi Wu(wuhongyi@qq.com).

1.3 File contents

The following files/folders are included in the user’s use package:

- Decode(Converting raw binary data to ROOT file)
- docs(Web page manual)
- **firmware**
  - firmware/firmware.md(History of firmware)
- GUI
- MakeEvent(event builder，optional)
- NOGUI(Non-graphics software)
- OnlineStattics(Online monitoring program)
- parset(parameter setting file)
- PlxSdk.tar.gz(Plx9054 driver)
- README(manual version in markdown)
- README.md(introduction for home page)
- README.pdf(manual version in pdf)
- software(non-standard pixie16 driver API revised by Hongyi Wu)
- TestTool(testing tool for developer, not necessary for users)

1.4 Update plan

- The main control interface development based on the ROOT GUI is highly complex, which is difficult for users to modify now. It is not easy for users to develop their own version based on this program.
- **We are also developing acquisition software for online/offline analysis based on web control:**
  - Django
  - ZeroMQ
  - JSROOT
  - …

1.5 License

CHAPTER 2

Installation of Software

Installation for this software is requrired by

- **CERN ROOT 6**
  - GCC >= 4.8
- FFTW3

The operating system tested by this program includes CentOS7 / Scientific Linux 7.2/7.3/7.4

> **危险：** Graphical interface programs and non-graphical interface programs cannot run at the same time!
>
> Graphical interface programs and non-graphical interface programs cannot run at the same time!
>
> Graphical interface programs and non-graphical interface programs cannot run at the same time!

2.1 The steps for Installation

- Delete the old version of the PKUXIADAQ folder in your personal directory
- Extract this package into your personal directory ($HOME)
- Set up environment variables
- Compile Plx9054 driver
- Compile pixie16 driver API (this API has been modified by Wu Hongyi, driven by unofficial standards)
- Compile graphical acquisition software
- Compile non-graphical acquisition software
- Compile online monitor program
- Compile data converter program
- Compile event reconstruction program (optional)

```
## Set up environment variables


# Add the follwing content into .bashrc file
export PLX_SDK_DIR=$HOME/PKUXIADAQ/PlxSdk

# put PKUXIADAQ.tar.gz(or PKUXIADAQ-master.tar.gz) in the personal directory /home,
↪ the position ~/

tar -zxvf PKUXIADAQ.tar.gz
or
tar -zxvf PKUXIADAQ-master.tar.gz
mv PKUXIADAQ-master  PKUXIADAQ

# Acquire PKUXIADAQ directory
```

```
## Compile Plx9054 driver

# Open a new terminal
cd ~
cd PKUXIADAQ/
#Delete the undeleted driver that may exist. If there is no such directory, you do␣
↪not need to execute the command.
rm -rf PlxSdk
tar -zxvf PlxSdk.tar.gz
cd PlxSdk/PlxApi/
make clean
make
# If it succeeds,you will see Library "Library/PlxApi.a" built successfully


cd ../Samples/ApiTest/
make clean
make
#if it succeeds,you will see Application "App/ApiTest" built successfully


cd ../../Driver/
./builddriver 9054

# If it succeeds,you will see Driver "Plx9054/Plx9054.ko" built sucessfully
```

```
## Compile pixie16

cd ~
cd PKUXIADAQ/software/
make clean
make

# As long as no error is reported, the libPixie16App.a libPixie16Sys.a will be␣
↪generated in the folder
```

```
# Modify settings parameters
cd ~
cd PKUXIADAQ/parset/


# Modify cfgPixie16.txt file。
# The value after CrateID indicates the chassis number, and the value is allowed␣
↪to be 0-15. If there is only a chassis, the parameter is set freely (usually the␣
↪default 0 is used). If multiple chassis are running synchronously, make sure␣
↪that the number of each chassis is set to a different value.
```

(下页继续)

(续上页)

```
# SettingPars Following is the parameter setting file and write the parameter␣
↪configuration file to be used.
# ModuleSlot The first value number indicates the number of plugins, and if there␣
↪are 3 plugins, it is 3. The following numbers are for each plug-in in the slot␣
↪position of the chassis (the slot position is counted from 2), and there are␣
↪three plugins followed by 2 3 4 respectively.
#AutoRunModeTimes The following values are the time for automatic switching in␣
↪automatic operation mode.
# Parameter ModuleSampingRate and ModuleBits only take effect in offline mode.␣
↪When the main interface is initialized in Offline mode, this parameter is read.


# Modify the Run.config file, the first line in the file is the original data␣
↪storage path, and the second is the file name.
# Modify the RunNumber file, the value in this file is the run number of the␣
↪actual run.
```

```
## Compile graphical acquisition software

cd ~
cd PKUXIADAQ/GUI/
make clean
make
```

```
## Compile non-graphical acquisition software

cd ~
cd PKUXIADAQ/NOGUI/
make clean
make
```

```
## Compile online monitor program

cd ~
cd PKUXIADAQ/OnlineStattics/

make clean
make
```

```
## Compile data converter program

cd ~
cd PKUXIADAQ/Decode/

# Modify UserDefine.hh according to the instructions in the program.

make clean
make
```

```
## Compile event reconstruction program

cd ~
cd PKUXIADAQ/MakeEvent/

# Modify UserDefine.hh according to the instructions in the program.

make clean
make
```

2.2 Instruction for use

- Restart the computer after booting the chassis (the computer must be open later than the chassis)
- Load Plx9054 driver under ROOT permission after opening the chassis
- Normal acquisition

```
## Load Plx9054 driver under ROOT permission


cd ~
cd PKUXIADAQ/PlxSdk/Bin/
su #input ROOT password
./Plx_load 9054

# You Will see a prompt to load successfully

exit  #Exit ROOT permission  退出 ROOT 权限
```

```
## Start the graphical interface program

cd ~
cd ~/PKUXIADAQ/GUI
./pku

# The graphical interface will pop up.
# You can choose Online/Offline Mode then press Boot to initialize.
# After waiting for initialization, you can modify the output data file path, file␣
↪name, and run number. Press the Complete button to confirm.
# The LSRunStart button becomes operational at this time. You can start pressing␣
↪Start and then press Stop for the second time.
# Online Statistics option selections mean sending online statistics
# Update Energy Monitor: Each time you select it, the energy spectrum information␣
↪is read from the plug-in and sent to the online program (frequent select
```

```
## Start the non-graphical interface program

cd ~
cd ~/PKUXIADAQ/NOGUI
./pku
```

```
## Start online monitor program


cd ~
cd PKUXIADAQ/OnlineStattics/
./online

# The graphical interface will pop up.
# Press RunStart to start monitoring and update the input rate and output rate of␣
↪each channel every 3 seconds. (The first time you enable the program after␣
↪opening the chassis, you need to enable it after the acquisition is turned on)
# In the lower right corner of the monitoring interface, there is monitoring of␣
↪the amount of hard disk usage for writing data.



# EnergyMonitor page is used to view the spectrum. Due to the internal register␣
↪size limitations of the plug-in, this energy spectrum differs from the actual␣
↪spectrum in channel range.
```

```
## execute data converter program


cd ~
cd PKUXIADAQ/Decode/

# After the last run of acquisition, we can convert the previous run of data to␣
↪ROOT file.

./decode xxx
#xxx indicates Run Number
```

CHAPTER 3

Guide

User’s Manual *Digital Gamma Finder (DGF) PIXIE-16* **Version 1.40, October 2009**

Pixie-16 User Manual Version 3.00 August 21, 2018

重要: The Pixie-16 is designed for single exponentially decaying signals.

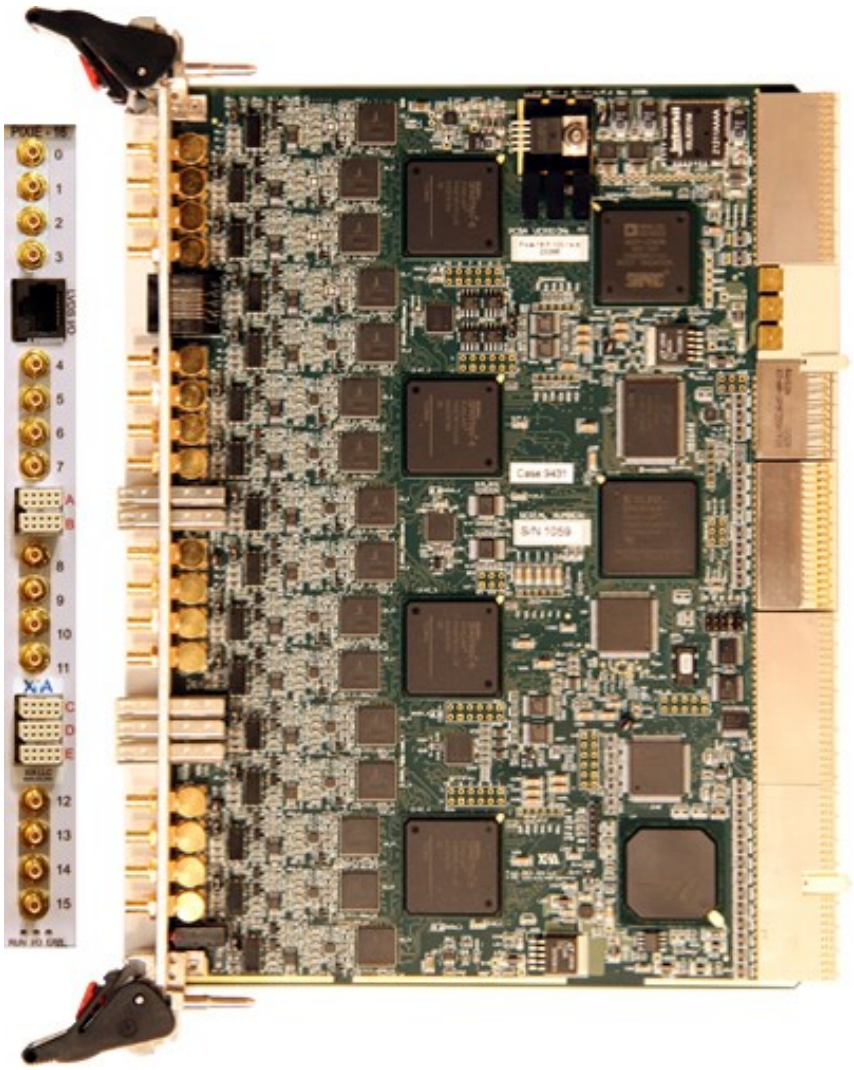
Step pulses or short non-exponential pulses can be accommodated with specific parameter settings.

Staircase type signals from reset preamplifiers generally need to be AC coupled.

危险: The amplitude of the detector output signals is not recommended to exceed +/-3.5V if 50Ohm input termination jumper is installed and the 1:4 attenuation is not used.

Do Not Hot-Swap!

To avoid personal injury, and/or damage to the DGF-Pixie-16, always turn off crate power before removing the DGF-Pixie-16 from the crate!



The DGF Pixie-16 is a 16-channel all-digital waveform acquisition and spectrometer card based on the CompactPCI/PXI standard for fast data readout to the host. It combines spectroscopy with waveform digitizing and the option of on-line pulse shape analysis. The Pixie-16 accepts signals from virtually any radiation detector. Incoming signals are digitized by 12/14/16-bit 100/250/500 MSPS ADCs. Waveforms of up to 163.8 μs in length for each channel can be stored in a FIFO.

The waveforms are available for onboard pulse shape analysis, which can be customized by adding user functions to the core processing software. Waveforms, timestamps, and the results of the pulse shape analysis can be read out by the host system for further off-line processing. Pulse heights are calculated to 16-bit precision and can be binned into spectra with up to 32K channels. The Pixie-16 supports coincidence spectroscopy and can recognize complex hit patterns.

Data readout rates through the CompactPCI/PXI backplane to the host computer can be up to 109 Mbyte/s. The

standard PXI backplane, as well as additional custom backplane connections are used to distribute clocks and trigger signals between several Pixie-16 modules for group operation. A complete data acquisition and processing systems can be built by combining Pixie-16 modules with commercially available CompactPCI/PXI processor, controller or I/O modules in the same chassis.

The Pixie-16 is an instrument for waveform acquisition and MCA histogramming for arrays of gamma ray or other radiation detectors such as

- **100 MSPS**
  - Segmented HPGe detectors.
  - Scintillator/PMT combinations: NaI, CsI, BGO and many others.
  - Gas detector.
  - Silicon strip detectors.
- **250 MSPS**
  - Scintillator
  - LaBr3
- **500 MSPS**
  - Scintillator
  - LaBr3

The Pixie-16 modules must be operated in a custom 6U CompactPCI/PXI chassis providing high currents at specific voltages not included in the CompactPCI/PXI standard 1 . Currently XIA provides a 14-slot chassis. Put the host computer(or remote PXI controller) in the system slot (slot 1) of your chassis. Put the Pixie-16 modules into any free peripheral slot (slot 2-14) with the chassis still powered down. After modules are installed, power up the chassis (Pixie-16 modules are not hot swappable). If using a remote controller, be sure to boot the host computer after powering up the chassis.

3.1 Data Structures

Event header as the first 4 words
RevD(12-bit,100MHz),RevF(14-bit,100MHz)

Index	Data					
0	[31]	[30:17]	[16:12]	[11:8]	[7:4]	[3:0]
	Finish Code	Event Length	Header Length	CrateID	SlotID	Chan#
1	[31:0]					
	EVTTIME_LO[31:0]					
2	[31]		[30:16]		[15:0]	
	CFD forced trigger bit		CFD Fractional Time[14:0] x 32768		EVTTIME_HI[15:0]	
3	[31]		[30:16]		[15:0]	
	Trace Out-of-Range Flag		Trace Length		Event Energy	

(EVTTIME_LO[31:0]+EVTTIME_HI[15:0]x2^{32}+(CFD_Fractional_Time[14:0]/32768))x10ns

Finish Code: 0-good event,1- pileup event
CFD forced trigger bit: 0- valid ,1-invalid (Threshold was set too high)
Trace Out-of-Range Flag: 0- trace in range, 1- trace is out of range

Event header as the first 4 words RevF(12/14/16-bit,250MHz)

Index	Data					
0	[31]	[30:17]	[16:12]	[11:8]	[7:4]	[3:0]
	Finish Code	Event Length	Header Length	CrateID	SlotID	Chan#
1	[31:0]					
	EVTTIME_LO[31:0]					
2	[31]	[30]	[29:16]			[15:0]
	CFD forced trigger bit	CFD trigger source bit	CFDFractionalTime[13:0]x16384			EVTTIME_HI[15:0]
3	[31]		[30:16]			[15:0]
	Trace Out-of-Range Flag		Trace Length			Event Energy

((EVTTIME_LO[31:0]+EVTTIME_HI[15:0]x2^{32})x2 - CFD trigger source bit + (CFD_Fractional_Time[13:0]/16384))x4ns

Finish Code: 0-good event,1- pileup event
CFD forced trigger bit: 0- valid ,1-invalid (Threshold was set too high)
CFD trigger source bit:
Trace Out-of-Range Flag: 0- trace in range, 1- trace is out of range

Event header as the first 4 words RevF(12/14-bit,500MHz)

Index	Data					
0	[31]	[30:17]	[16:12]	[11:8]	[7:4]	[3:0]
	Finish Code	Event Length	Header Length	CrateID	SlotID	Chan#
1	[31:0]					
	EVTTIME_LO[31:0]					
2	[31:29]		[28:16]			[15:0]
	CFD trigger source bits[2:0]		CFD Fractional Time[12:0] x 8192			EVTTIME_HI[15:0]
3	[31]		[30:16]			[15:0]
	Trace Out-of-Range Flag		Trace Length			Event Energy

(EVTTIME_LO[31:0]+EVTTIME_HI[15:0]x2^{32})x10 + ((CFD_Fractional_Time[12:0]/8192)+ CFD trigger source bits[2:0]-1)x2ns

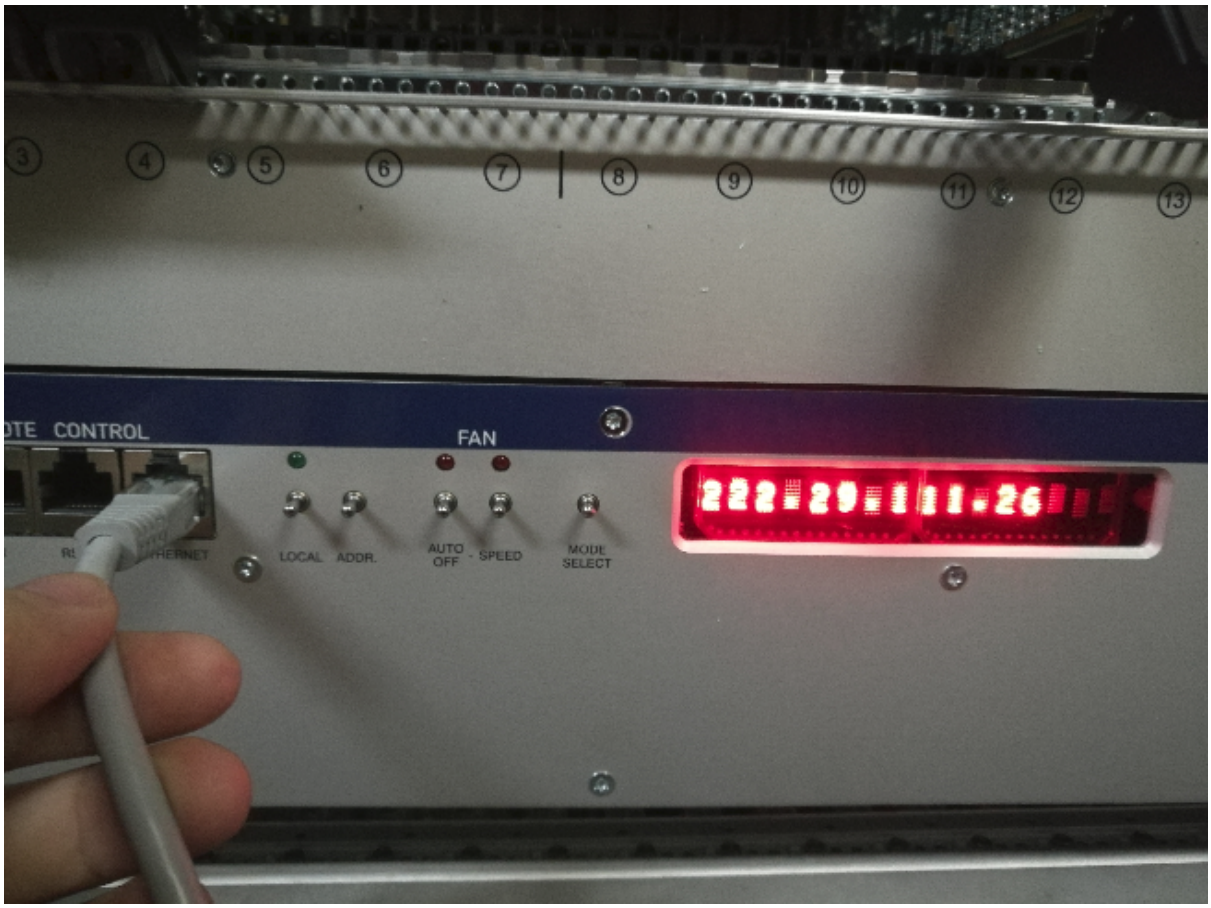
Finish Code: 0-good event,1- pileup event
CFD trigger source bits:
Trace Out-of-Range Flag: 0- trace in range, 1- trace is out of range

If trace recording is enabled, trace data will immediately follow the last word of the event header. Since raw ADC data points are 12/14/16-bit number, two 12/14/16-bit numbers are packed into one 32-bit word. Since the event header could have variable length(4,6,8,10,12,14 ,16 or 18 words) depending on the selection of various output data options, the header length, event length and trace length that are recorded in the first 4 words of the event header should be used to navigate through the output data stream.

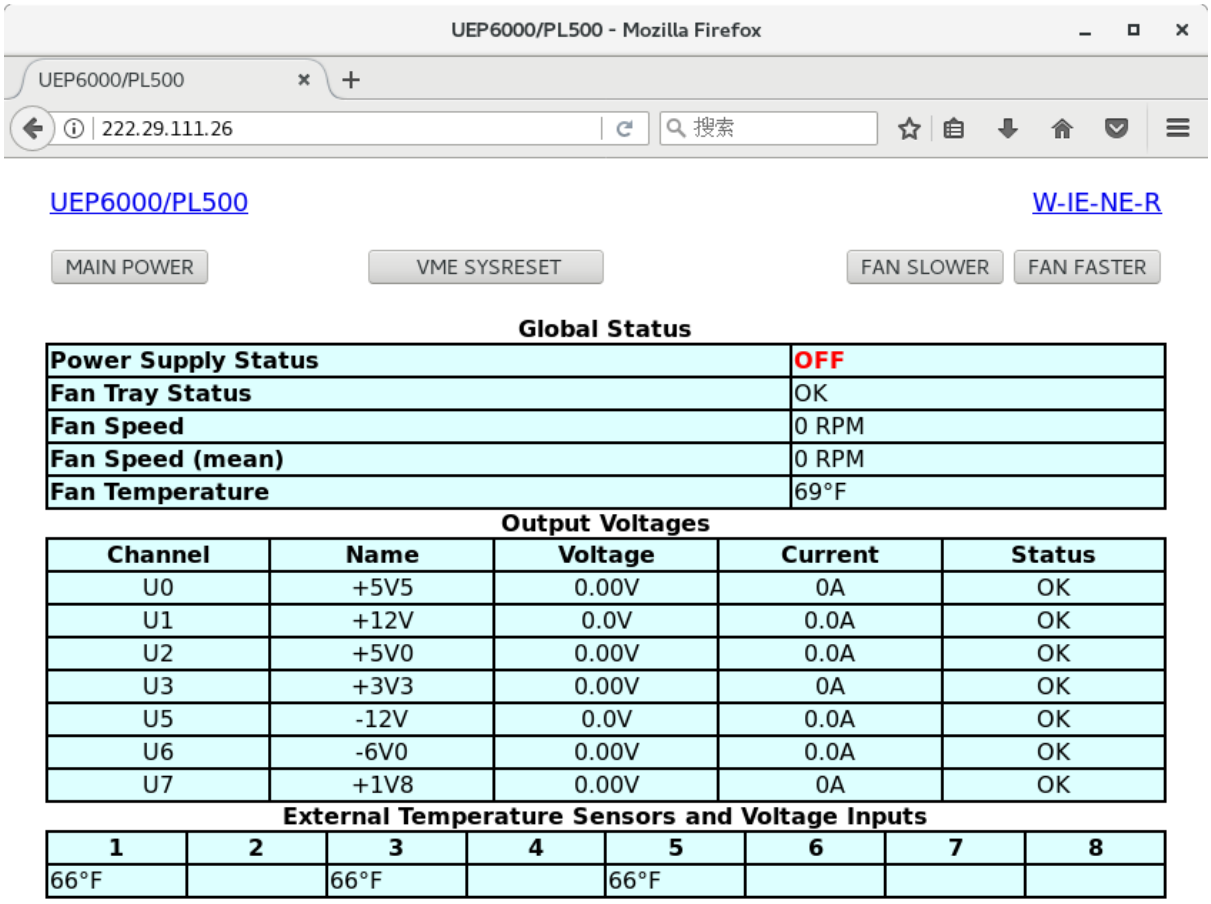
CHAPTER 4

Crate

4.1 remote control



Connect the chassis network port with the network cable as shown in the figure above, and then the screen on the right side will quickly flash over the assigned IP. For example,the IP here is 222.29.111.26. If users don't see the IP clearly, the network cable can be pulled out and reconnected.



Global Status

Power Supply Status	**OFF**
Fan Tray Status	OK
Fan Speed	0 RPM
Fan Speed (mean)	0 RPM
Fan Temperature	69°F

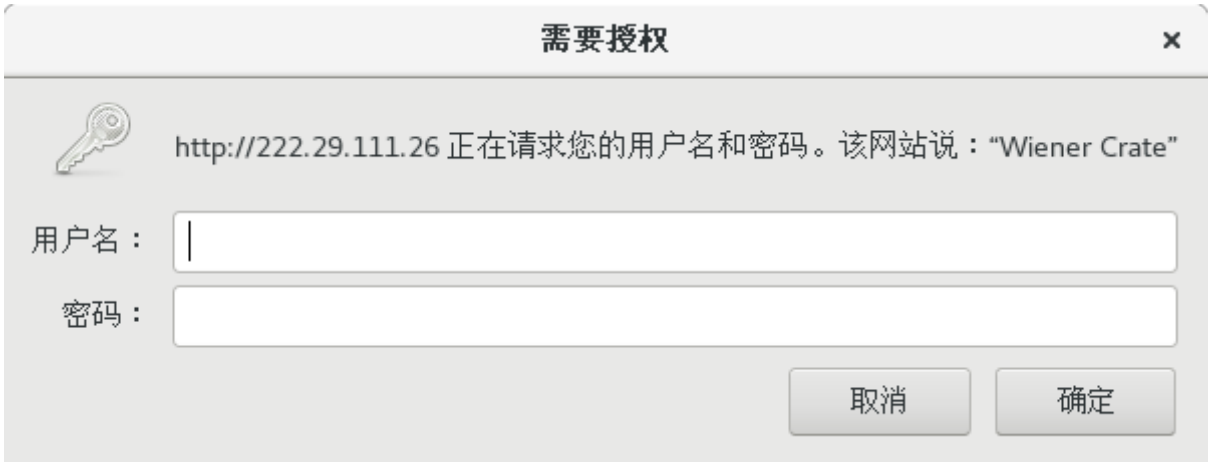
Output Voltages

Channel	Name	Voltage	Current	Status
U0	+5V5	0.00V	0A	OK
U1	+12V	0.0V	0.0A	OK
U2	+5V0	0.00V	0.0A	OK
U3	+3V3	0.00V	0A	OK
U5	-12V	0.0V	0.0A	OK
U6	-6V0	0.00V	0.0A	OK
U7	+1V8	0.00V	0A	OK

External Temperature Sensors and Voltage Inputs

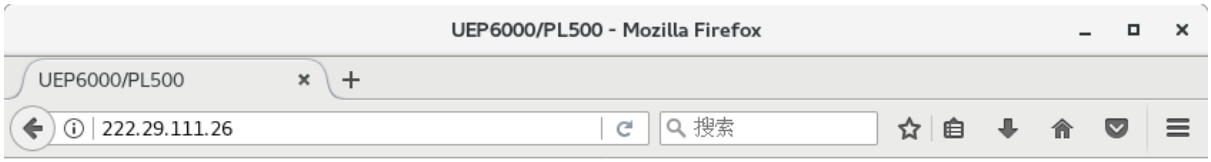
1	2	3	4	5	6	7	8
66°F		66°F		66°F			

As shown in the figure above, users can enter the control page by typing IP in the browser and it is shown that the chassis is off in the figure.



The button **MAIN POWER** is used to control the opening and closing of the chassis. The login box above will pop up when you click it for the first time.

Type the username "private", and the default password is "private".



UEP6000/PL500 W-IE-NE-R

MAIN POWER VME SYSRESET FAN SLOWER FAN FASTER

Global Status

Power Supply Status	ON
Fan Tray Status	OK
Fan Speed	3300 RPM
Fan Speed (mean)	3290 RPM
Fan Temperature	64°F

Output Voltages

Channel	Name	Voltage	Current	Status
U0	+5V5	5.49V	6A	OK
U1	+12V	12.0V	0.2A	OK
U2	+5V0	5.00V	0.2A	OK
U3	+3V3	3.31V	5A	OK
U5	-12V	12.0V	0.0A	OK
U6	-6V0	6.00V	4.5A	OK
U7	+1V8	1.81V	1A	OK

External Temperature Sensors and Voltage Inputs

1	2	3	4	5	6	7	8
64°F		66°F		64°F			

After the chassis is open， the monitoring parameters is shown in the figure above.

Wherein, the buttons in the upper right corner **FAN SLOWER** and **FAN FASTER** are used to regulate the speed of the fan.

4.2 slot

There are 14 slots in the chassis, and the Numbers 1-14 are marked on the bottom, in which slot 1 is for the controller slot and slots 2-14 are for the acquisition modules.

CHAPTER 5

FIRMWARE

Peking University Custom Firmware

Added the following features based on standard firmware：

- **100MHz 12 bit(pixie16_revd_12b100m_firmware_release)**
  - standard firmware
- **100MHz 14 bit(pixie16_revf_general_14b100m_firmware_release)**
  - standard firmware
- **100MHz 14 bit(pixie16_revfpku_14b100m_firmware_release_10142019)**
  - 4-ch debug signals of the front panel A to the chassis backplane’ s TriggerAll bits 28 to 31. TrigConfig3[0] control
  - multiplicity results can be output regardless of MultiplicityMaskHigh[31]=0 or 1. Output from front panel A and RJ45.
  - The value is set to 0 when the calculated energy is negative.
  - The pileup event energy is not set to 0, output calculated values directly.
  - In the record waveform mode, when the waveform buffer is full, the module is not busy, and the header continues to record. In this case, the output event data has no waveform.
  - The record waveform mode with down frequency output. The strategy adopted is to select the output of 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128 frequency, ie how many points retain one point. The points retained are the averaged values.
  - tried to speed up the event processing, by removing some of the unnecessary wait when reading trace from each channel.
  - **Also removed a few unnecessary processing routines in the DSP code:**
    - (1) no longer process pile-up rejection or inverse pile-up rejection, all events will be accepted, but still with pileup flag in the event header;
    - (2) removed “no traces for large pulses” feature.
- **100MHz 14 bit(pixie16_revfpku_14b100m_firmware_release_09272018)**
  - multiplicity results can be output regardless of MultiplicityMaskHigh[31]=0 or 1. Output from front panel A and RJ45.

  - The value is set to 0 when the calculated energy is negative.
  - The pileup event energy is not set to 0, output calculated values directly.
  - In the record waveform mode, when the waveform buffer is full, the module is not busy, and the header continues to record. In this case, the output event data has no waveform.
  - The record waveform mode with down frequency output. The strategy adopted is to select the output of 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128 frequency, ie how many points retain one point. The points retained are the averaged values.
  - tried to speed up the event processing, by removing some of the unnecessary wait when reading trace from each channel.
  - **Also removed a few unnecessary processing routines in the DSP code:**
    - (1) no longer process pile-up rejection or inverse pile-up rejection, all events will be accepted, but still with pileup flag in the event header;
    - (2) removed "no traces for large pulses" feature.

- **100MHz 14 bit(pixie16_revfpku_14b100m_dsp_update_05082018)**
  - multiplicity results(front panel A) can be output regardless of MultiplicityMaskHigh[31]=0 or 1
  - The value is set to 0 when the calculated energy is negative.
  - The pileup event energy is not set to 0, output calculated values directly.
  - In the record waveform mode, when the waveform buffer is full, the module is not busy, and the header continues to record. In this case, the output event data has no waveform.
  - The record waveform mode with down frequency output. The strategy adopted is to select the output of 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128 frequency, ie how many points retain one point. The points retained are the averaged values.
- **250MHz 12bit(pixie16_revf_12b250m_firmware_release)**
  - standard firmware
- **250MHz 14bit(pixie16_revfpku_14b250m_release_06092019) some bug**
  - multiplicity results can be output regardless of MultiplicityMaskHigh[31]=0 or 1. Output from front panel A and RJ45.
  - The value is set to 0 when the calculated energy is negative.
  - The pileup event energy is not set to 0, output calculated values directly.
- **250MHz 16bit(pixie16_revfpku_16b250m_release_06092019) some bug**
  - multiplicity results can be output regardless of MultiplicityMaskHigh[31]=0 or 1. Output from front panel A and RJ45.
  - The value is set to 0 when the calculated energy is negative.
  - The pileup event energy is not set to 0, output calculated values directly.
- **500MHz 14bit(pixie16_revf_14b500m_firmware_release)**
  - standard firmware

CHAPTER 6

Decode

The Decode program is used to convert data collected by different modules in the same run into a ROOT file. The user's physical analysis is based on the ROOT file generated by the program.

The user first needs to modify the definition in the **UesrDefine.hh** file.

```
#define RAWFILEPATH "/home/wuhongyi/data/"   //The path of the original binary
#define RAWFILENAME "data"                   //The filename of the original binary
#define ROOTFILEPATH "/home/wuhongyi/data/"  //The path to generate the ROOT file

#define TimesHist 3600  // second   Histogram parameters should be set longer than␣
↪the running time of the run.

#define Crate0
#define Crate0num 5   //Number of modules used in this crate
const int Crate0SamplingRate[Crate0num] = {100,100,100,250,250};//Specify the␣
↪sampling rate for each module separately, 100/250/500 three sampling rates, 0␣
↪means skipping the module.
```

Users need to modify the following concents:

- The directory where the original binary files are stored.
- The directory where the generated ROOT files are stored.
- File name.
- Number of modules used in the crate.
- The sampling frequency corresponding to each module. If the sampling frequency is set to 0, the data of the module is ignored.

After modifying, execute the following command to compile the program:

```
make clean
make
```

After the compilation is successful, an executable file **decode** will be generated, and the program will run as follows:

```
./decode [RuNnumber]
```

[RuNnumber] is the file run number you want to convert.

For example:

```
./decade  3
```

ROOT File Branch：

- sr(short): sample rate，100/250/500，This value is specified in UesrDefine.hh.
- pileup(bool): pile-up flag
- outofr(bool): the overrange flag
- cid(short): crate number
- sid(short): slot number
- ch(short): channel number
- evte(unsigned short): energy
- ts(long int 64 bit): timestamps
- ets(long int 64 bit): external timestamps
- cfd(short): cfd value
- cfdft(bool): Is the cfd value valid? yes or no
- cfds(short):cfd source，only for 250/500 MHz modules
- trae(unsigned int): energy trapezoidal rising segment integral
- leae(unsigned int): energy trapezoidal falling segment integral
- gape(unsigned int): energy trapezoidal gaps segment integral
- base(double):the baseline of the energy trapezoidal algorithm
- qs(unsigned int): the integral of eight QDC areas
- ltra(unsigned short): number of waveform acquisition points
- data(unsigned short): waveform data
- dt(unsigned short): In order to view each waveform directly, an array of values from 0 - N-1 is added
- nevt(unsigned int): the number of this event in this ROOT file

The following figure shows the Branch definition in a file:

```
[wuhongyi@ScientificLinux data]$ root data_R0595.root
root [0]
Attaching file data_R0595.root as _file0...
(TFile *) 0x1f6edd0
root [1] .ls
TFile**         data_R0595.root
 TFile*         data_R0595.root
  KEY: TTree    tree;175        PKU XIA Pixie-16 Data
  KEY: TTree    tree;174        PKU XIA Pixie-16 Data
root [2] tree->Print()
******************************************************************************
*Tree    :tree      : PKU XIA Pixie-16 Data                                  *
*Entries :  1123666 : Total =     13993888785 bytes  File  Size = 3708728891 *
*        :          : Tree compression factor =   3.77                       *
******************************************************************************
*Br    0 :sr        : sr/S                                                   *
*Entries :  1123666 : Total  Size=    2263185 bytes  File Size  =     167039 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=  13.53     *
*.............................................................................*
*Br    1 :pileup    : pileup/O                                               *
*Entries :  1123666 : Total  Size=    1140233 bytes  File Size  =      30956 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=  36.71     *
*.............................................................................*
*Br    2 :outofr    : outofr/O                                               *
*Entries :  1123666 : Total  Size=    1140233 bytes  File Size  =      23284 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=  48.81     *
*.............................................................................*
*Br    3 :cid       : cid/S                                                  *
*Entries :  1123666 : Total  Size=    2263364 bytes  File Size  =      27637 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=  81.76     *
*.............................................................................*
*Br    4 :sid       : sid/S                                                  *
*Entries :  1123666 : Total  Size=    2263364 bytes  File Size  =     175529 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=  12.87     *
*.............................................................................*
*Br    5 :ch        : ch/S                                                   *
*Entries :  1123666 : Total  Size=    2263185 bytes  File Size  =     284004 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=   7.96     *
*.............................................................................*
*Br    6 :evte      : evte/s                                                 *
*Entries :  1123666 : Total  Size=    2263543 bytes  File Size  =    1631103 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=   1.39     *
*.............................................................................*
*Br    7 :ts        : ts/L                                                   *
*Entries :  1123666 : Total  Size=    9020679 bytes  File Size  =    3066162 *
*Baskets :      349 : Basket Size=      51200 bytes  Compression=   2.94     *
*.............................................................................*
*Br    8 :ets       : ets/L                                                  *
*Entries :  1123666 : Total  Size=    9021032 bytes  File Size  =      73195 *
*Baskets :      349 : Basket Size=      51200 bytes  Compression= 123.15     *
*.............................................................................*
```

```
*Br    9 :cfd       : cfd/S                                                  *
*Entries :  1123666 : Total  Size=    2263364 bytes  File Size  =    2191516 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=   1.03     *
*.......................................................................*
*Br   10 :cfdft     : cfdft/O                                                *
*Entries :  1123666 : Total  Size=    1140054 bytes  File Size  =      30291 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=  37.51     *
*.......................................................................*
*Br   11 :cfds      : cfds/S                                                 *
*Entries :  1123666 : Total  Size=    2263543 bytes  File Size  =     269187 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=   8.39     *
*.......................................................................*
*Br   12 :trae      : trae/i                                                 *
*Entries :  1123666 : Total  Size=    4510879 bytes  File Size  =    2642761 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=   1.71     *
*.......................................................................*
*Br   13 :leae      : leae/i                                                 *
*Entries :  1123666 : Total  Size=    4510879 bytes  File Size  =    2886877 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=   1.56     *
*.......................................................................*
*Br   14 :gape      : gape/i                                                 *
*Entries :  1123666 : Total  Size=    4510879 bytes  File Size  =    2982289 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=   1.51     *
*.......................................................................*
*Br   15 :base      : base/D                                                 *
*Entries :  1123666 : Total  Size=    9021385 bytes  File Size  =    3984210 *
*Baskets :      349 : Basket Size=      51200 bytes  Compression=   2.26     *
*.......................................................................*
*Br   16 :qs        : qs[8]/i                                                *
*Entries :  1123666 : Total  Size=   35989106 bytes  File Size  =   23047394 *
*Baskets :      354 : Basket Size=     174592 bytes  Compression=   1.56     *
*.......................................................................*
*Br   17 :ltra      : ltra/s                                                 *
*Entries :  1123666 : Total  Size=    2263543 bytes  File Size  =     230891 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=   9.79     *
*.......................................................................*
*Br   18 :data      : data[ltra]/s                                           *
*Entries :1123666 : Total  Size= 6945634237 bytes  File Size  = 3206301603 *
*Baskets :     1689 : Basket Size=   25600000 bytes  Compression=   2.17     *
*.......................................................................*
*Br   19 :dt        : dt[ltra]/s                                             *
*Entries :1123666 : Total  Size= 6945630851 bytes  File Size  =  457034676 *
*Baskets :     1689 : Basket Size=   25600000 bytes  Compression=  15.20     *
*.......................................................................*
*Br   20 :nevt      : nevt/I                                                 *
*Entries :  1123666 : Total  Size=    4510879 bytes  File Size  =    1581484 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=   2.85     *
*.......................................................................*
```

At the end of each run of data conversion, a **txt** file will be generated in this folder, which counts the following information for each channel of the modules:

- Mod: Module number, starting from 0.
- Channel: Channel marker, 0 - 15.
- OutOfRange: Number of events whose signal amplitude exceeds the range of the analog-to-digital conversion module.
- Pileup: Number of events marked as pile-up.
- CfdForcedTrigger: Number of events forced by cfd (cfd does not exceeded threshold).

- Energy->0: Calculate the number of events with trapezoidal energy less than 0 (the result is less than 0 and is directly marked as 0).
- WaveformCount: Number of events recording the waveform.
- TotalEvent: Total number of output events.

At the end of each run of data conversion, a **ROOT** file is generated in this folder, which counts the counting rate of each channel of all the modules.

CHAPTER 7

GUI

After setting the parameter file in the **parset**, enter the GUI directory and execute the following command to pop up the main control interface.

```
./pku
```

7.1 The main control interface

At the top of the main interface are five drop-down columns: File, Base, Expert, Debug, and Offiline. The submenu inside is as follows:

- **File**
  - Exit
  - About
- **Base**
  - Base Setup
  - Trigger Filter
  - Energy
  - CFD
  - QDC
  - Decimation
  - Copy Pars
  - Save2File
- **Expert**
  - Module Variables
  - CSRA
  - Logic Set
- **Debug**
  - Hist & XDT
  - Trace & Baseline
- **Offiline**
  - Adjust Par
  - Simulation(Not yet implemented)

After the main interface is open, select the **Online Mode** option. You need to connect to the chassis. All functions (including offline analysis) can be used in this mode. If the **Online Mode** option is not selected, it means that the offline mode is enabled, and the acquired parameters can be set and modified or analyze the acquired waveform.

After selecting or not selecting the **Online Mode** option, press the **Boot** button to start the initialization process and see the status changes in the *Information* section at the bottom.

After the system is successfully initialized, confirm the file storage path, file name, and file number in the *Setup* column. If there is any problem, modify it directly. After confirming, press **Complete**.

After confirming the information in the *Setup* column, the main button **RunStart** in the *Control* column is open. Click this button to get data acquisition open, the button status changes to **RunStop**, click the button again, the data acquisition completed, and the *Run Num* number is automatically added. Click **RunStart** again to open the next run.

Currently, you can adjust and modify the parameters through the submenu in the top drop-down bar before you acquire data. Do not manipulate all options except the *Control* bar when acquiring data.

The **Online Statistics** option in the *Control* column is turned on to get the input rate and output rate information for each signal sent to the *OnineStatistics* program every **3 s**.

Clicking the Update Energy Monitor option once will send the one-dimensional spectrum of each channel in all the internal registers of the modules to the Online Statistics program. Sending this information will cause a certain dead time. Please do not click this option frequently.

7.2 File

Here it has no practical use.

7.2.1 About

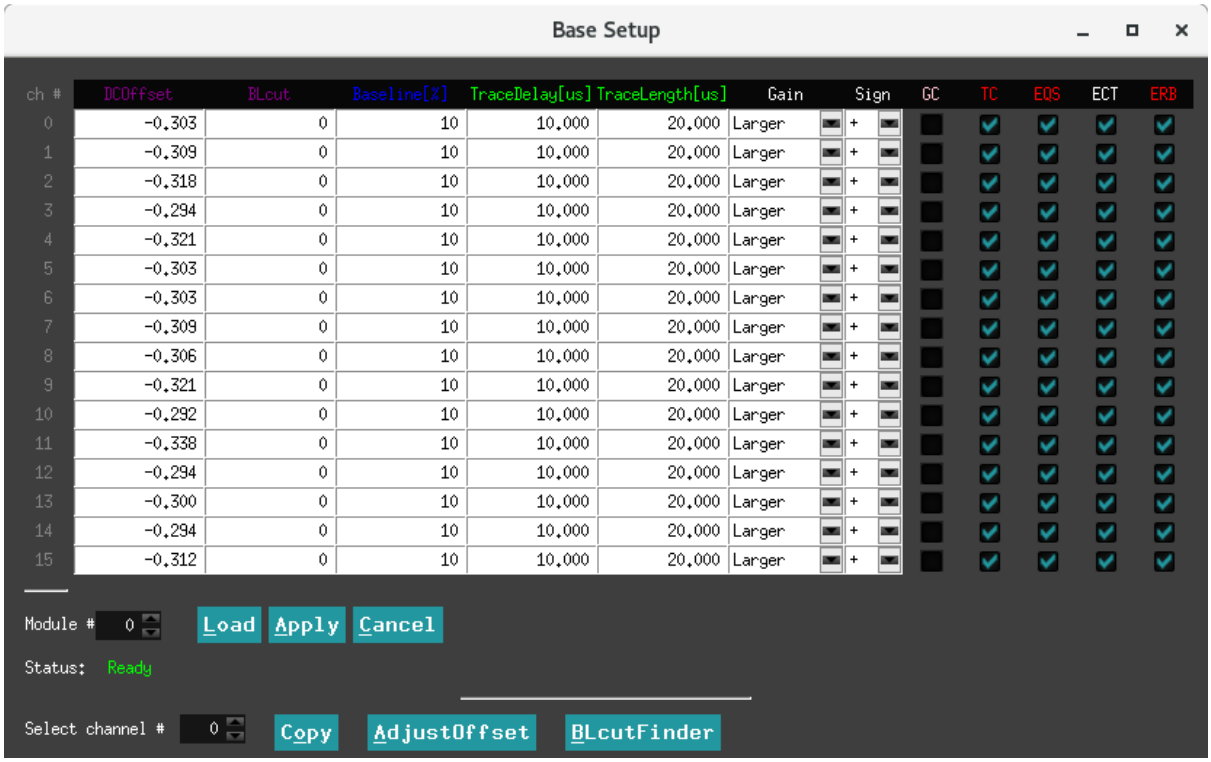
The introduction of software developer. The basic operating instructions of the main program will be added later.

7.3 Base

The adjustments in this drop-down column are the basics, and anyone using the Pixie16 acquisition system should be familiar with and master the operating techniques.

7.3.1 Base Setup

Interface



The status below the interface shows a **green Ready** to indicate that the interface can be operated, otherwise you need to wait.

In the interface, the parameters behind the **Module** are used to select the adjusted module, **Load** is used to read the parameter value of the modules, and **Apply** is to write the value in the interface to the module.

The parameter following the **Select Channel** at the bottom of the interface indicates that you can choose to copy the parameters of the channel on the interface to other channels, click **Copy** to complete the copy, and then **Apply** to write the parameters to the module.

The **Base Setup** page controls the analog gain, offset and polarity for each channel. It is useful to click on **Trace & Baseline** in the top control **Monitor** bar to view the signal read from the ADCs while adjusting these parameters. The display shows one or all 16 channels of a module; you can set the sampling interval for each block to capture a longer time frame in **Hist & XDT** page. Click **Draw** to update the graph.

Pulses from the detector should fall in the range from 0 to 16383(14 bit), with the baseline at ~1638 to allow for drifts and/or undershoots and no clipping at the upper limit. If there is clipping, adjust the Gain and Offset or click on the *AdjustOffset* button to let the software set the DC offsets to proper levels.

Since the trigger/filter circuits in the FPGA only act on rising pulses, negative pulses are inverted at the input of the FPGA, and the waveforms shown in the ADC trace display include this optional inversion. Thus set the channel’ s Polarity such that pulses from the detector appear with positive amplitude (rising edge).

In the **Base Setup** tab, you can set the total trace length and the pre-trigger trace delay for the waveforms to be acquired in list mode runs.

The trace delay cannot be longer than the trace length, and for each Pixie-16 variant, there is also a limit for the maximum value of trace delay and trace length.

Parameters introduction

- The option *Gain* indicates the gain adjustment. The user can select the *Larger* or *Small*. The gain parameters corresponding to the two files of each module can be tested by the user or be chosen by consulting the manufacturer.

- The option *Sign* indicates the polarity of the selected input signal. The positive signal selects "+" and the negative signal selects "- ".
- The option *GC* indicates whether the channel is data recorded. If it is checked, it means that the channel is recorded. If it is not checked, it means that it is not recorded.
- The option *ECT* means to enable CFD trigger. Otherwise, regular trapezoidal fast trigger will be used.

The red *TC*, *EQS*, *ERB* are used to select which raw data to output:

- The option *TC* indicates the recording waveform. At this time, *TraceDelay* and *TraceLength* are valid. If not selected, the waveform is not recorded.
- The option *EQS* means that the scores of the eight QDCs are recorded, and if they are not selected, they are not recorded.
- The option *ERB* represents the three-part area integral of the energy trapezoid(raw energy sums) and the baseline value of the trapezoidal calculation.

The green *TraceDelay* and *TraceLength* are the points of the output data. The parameter is divided by the nominal sampling rate of the acquisition card to calculate the actual output data points of the waveform:

- *TraceDelay* indicates the length of the acquired waveform before the trigger.
- *TraceLength* represents the entire waveform acquisition length.

It should be specially noted that when using the down frequency mode, the actual waveform length is TraceDelay x 2^N / TraceLength x 2^N (N is the down-frequency parameter).

The blue *Baseline* is used to adjust the baseline position and adjust the baseline to the user's expected position by voltage compensation:

- The *Baseline* adjustable range is 0 - 100, which indicates the percentage of the waveform's baseline that falls within the full scale. For example, for a vertical precision 14-bit capture card, setting this parameter to 10 means that the baseline reduction compensation is adjusted to around 10% of the 16384 full-scale, near 1638.
- The purple *DCOffset* and *BLcut* users do not need to be modified, and the parameters can be adjusted automatically. After modifying *Baseline*, *Gain*, and *Sign* in this submenu, you need to press the bottom of the *AdjustOffset*, and then press *BLcutFinder* to automatically adjust these two parameters.

Important note

注解：trace length in 500 MHz

For the 500 MHz Pixie-16 modules, the ADCs are running at 500 MHz, but the traces are recorded with 100 MHz clocks in the FPGA with 5 ADC samples captured in each 10 ns interval. In addition, the data packing from the FPGA to the onboard External FIFO is two sets of 5 ADC samples in one transfer. So the trace length should be multiples of 20 ns, i.e., 20 ns, 40 ns, ⋯for instance, a trace length of 500 ns and a trace delay of 200 ns.

小技巧：Good channel

Only channels marked as good will have their events recorded.

This setting has no bearing on the channel's capability to issue a trigger.

There can be a triggering channel whose data are discarded.

Channels not marked as good will be excluded from the automatic offset adjustment.

Baseline measurements

The Pixie-16 constantly takes baseline measurements when no pulse is detected and keeps a baseline average to be subtracted from the energy filter output during pulse height reconstruction. Baseline measurements that differ from the average by more than the BaselineCut value will be rejected as they are likely contaminated with small pulses below the trigger threshold.

A series of baseline measurements for each channel can be viewed in **Trace & Baseline** page, and in the BASELINE panel a histogram of baselines can be built to verify that the Baseline Cut does not reject measurements falling into the main (ideally Gaussian) peak in the baseline distribution.

Usually, it is sufficient to keep Baseline Cut at its default value.

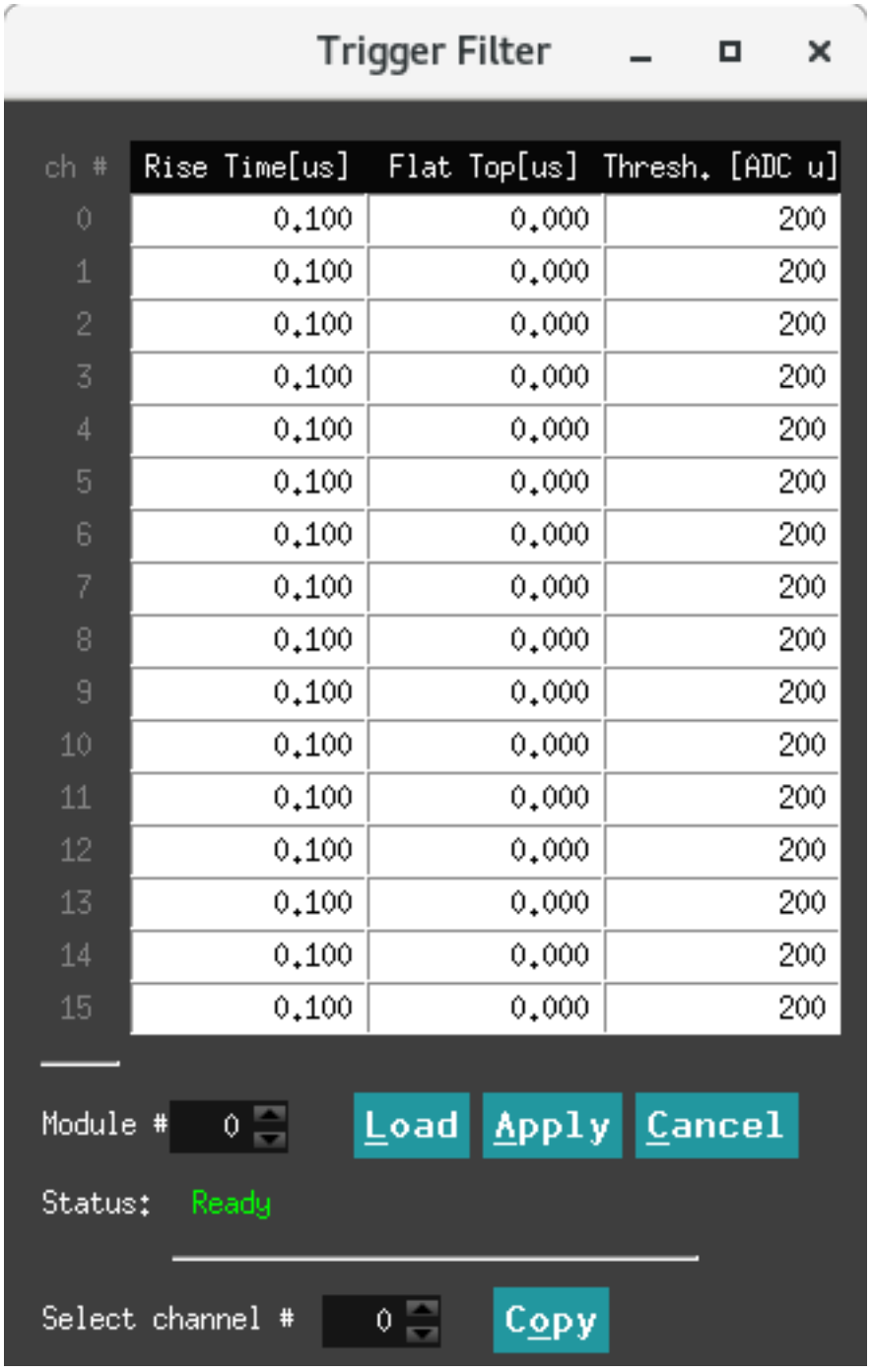
Note: Since the baseline computation takes into account the exponential decay, no pulses should be noticeable in the baseline display if

- a) the decay time is set correctly and
- b) the detector pulses are truly exponential.

Baseline Percent is a parameter used for automatic offset adjustment; by clicking on the *AdjustOffses* button, offsets will be set such that the baseline seen in the ADC trace display falls at the Baseline Percent fraction of the full ADC range (e.g. for a 12-bit ADC and Baseline Percent = 10% the baseline falls at ADC step 409 out of 4096 total).

7.3.2 Trigger Filter

Interface



When the status below the interface is displaying **green Ready**, it means that the interface can be operated, otherwise you need to wait. The operation of the bottom button is the same as above.

- The parameter *Rise Time* mean trigger filter rise time.
- The parameter *Flat Top* mean trigger filter flat top time.
- The parameter *Thresh.* represents the threshold, which is set relative to the fast filter waveform.

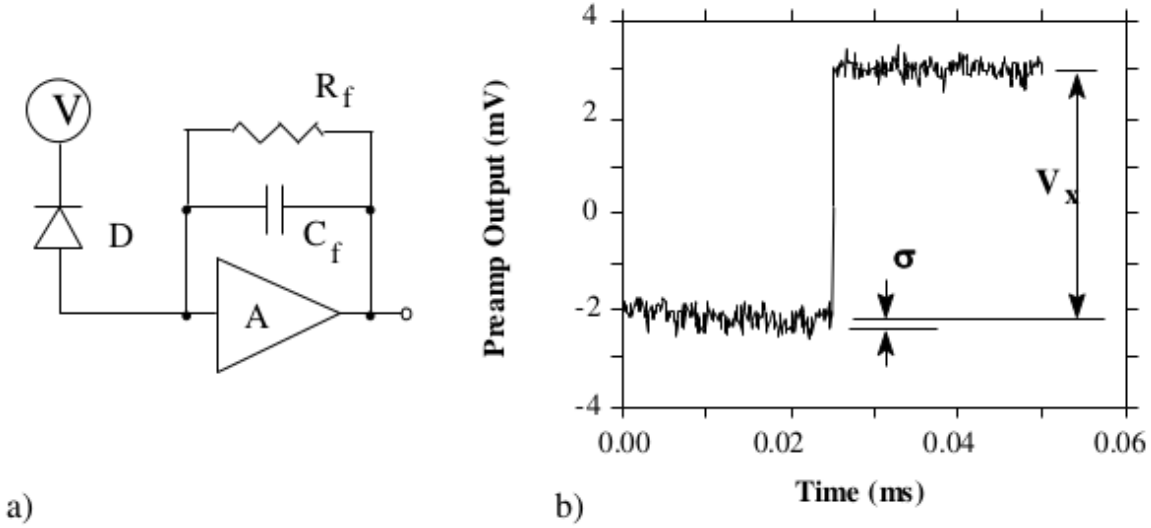
General rules of thumb for the following important parameters are:

- A longer trigger filter rise time averages more samples and thus allows setting lower thresholds without triggering on noise.
- Typically the threshold should be set as low as possible, just above the noise level.

- A longer trigger filter flat top time makes it easier to detect slow rising pulses.

Digital Filters

Energy dispersive detectors, which include such solid state detectors as Si(Li), HPGe, HgI2 , CdTe and CZT detectors, are generally operated with charge sensitive preamplifiers as shown in Figure. Here the detector **D** is biased by voltage source **V** and connected to the input of **preamplifier A** which has feedback capacitor **Cf** and feedback resistor **Rf** .



Reducing noise in an electrical measurement is accomplished by filtering. Traditional analog filters use combinations of a differentiation stage and multiple integration stages to convert the preamp output steps, such as shown in Figure (b), into either triangular or semi-Gaussian pulses whose amplitudes (with respect to their baselines) are then proportional to **Vx** and thus to the gamma-ray’ s energy.

Digital filtering proceeds from a slightly different perspective. Here the signal has been digitized and is no longer continuous. Instead it is a string of discrete values as shown in Figure. Figure is actually just a subset of Figure (b), in which the signal was digitized by a Tektronix 544 TDS digital oscilloscope at 10 MSPS (mega samples per second). Given this data set, and some kind of arithmetic processor, the obvious approach to determining Vx is to take some sort of average over the points before the step and subtract it from the value of the average over the points after the step. That is, as shown in follow Figure, averages are computed over the two regions marked “Length” (the “Gap” region is omitted because the signal is changing rapidly here), and their difference taken as a measure of **Vx** . Thus the value Vx may be found from the following equation:

$$V_{x,k} = -\sum_{i(before)} W_i V_i + \sum_{i(after)} W_i V_i$$

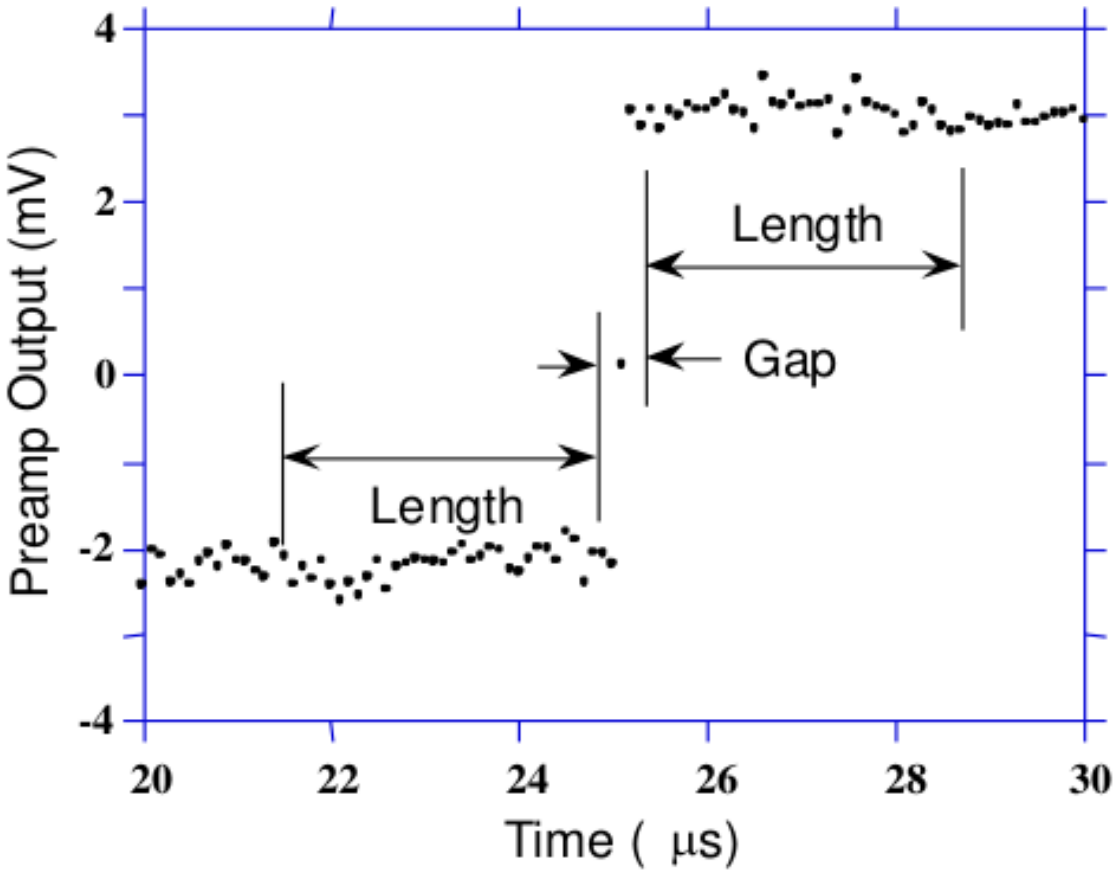
Where the values of the weighting constants **Wi** determine the type of average being computed. The sums of the values of the two sets of weights must be individually normalized.

The primary differences between different digital signal processors lie in two areas: what set of weights Wi is used and how the regions are selected for the computation of Equation.

Thus, for example, when larger weighting values are used for the region close to the step while smaller values are used for the data away from the step, Equation produces “cusp-like” filters. When the weighting values are constant, one obtains triangular (if the gap is zero) or trapezoidal filters. The concept behind cusp-like filters is that, since the points nearest the step carry the most information about its height, they should be most strongly weighted in the averaging process. How one chooses the filter lengths results in time variant (the lengths vary from pulse to pulse) or time invariant (the lengths are the same for all pulses) filters. Traditional analog filters are time invariant. The concept behind time variant filters is that, since the gamma-rays arrive randomly and the lengths between them vary accordingly, one can make maximum use of the available information by setting the length to the interpulse spacing.

In principle, the very best filtering is accomplished by using cusp-like weights and time variant filter length selection. There are serious costs associated with this approach however, both in terms of computational power required to

evaluate the sums in real time and in the complexity of the electronics required to generate (usually from stored coefficients) normalized **Wi** sets on a pulse by pulse basis.



The Pixie-16 takes a different approach because it was optimized for high speed operation.

It implements a fixed length filter with all **Wi** values equal to unity and in fact computes this sum afresh for each new signal value k. Thus the equation implemented is:

$$LV_{x,k} = -\sum_{i=k-2L-G+1}^{k-L-G} V_i + \sum_{i=k-L+1}^{k} V_i$$

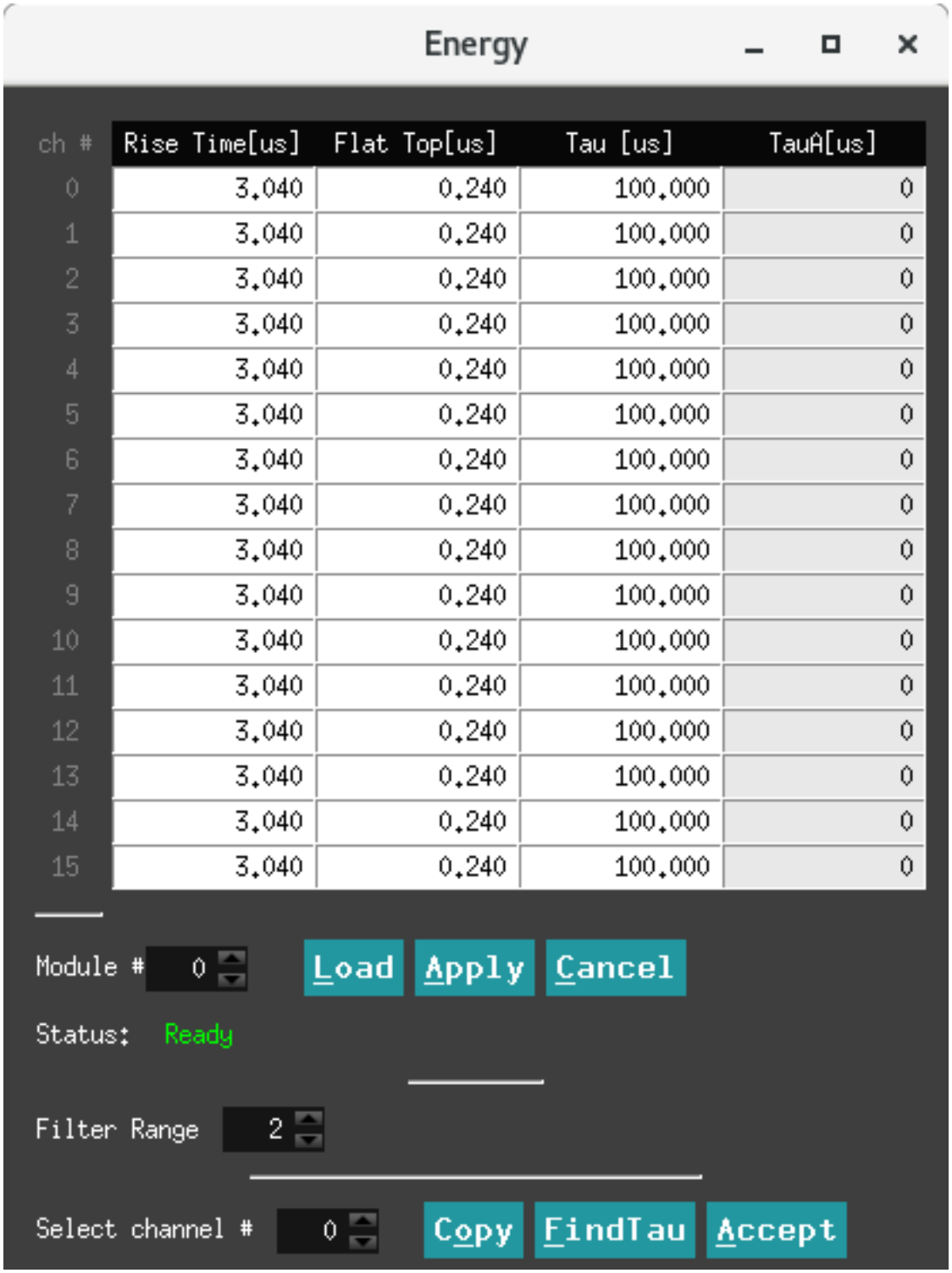
Where the filter length is **L** and the gap is **G**. The factor **L** multiplying $V_{x,k}$ arises because the sum of the weights here is not normalized. Accommodating this factor is trivial.

While this relationship is very simple, it is still very effective. In the first place, this is the digital equivalent of triangular (or trapezoidal if G ≠ 0) filtering which is the analog industry's standard for high rate processing. In the second place, one can show theoretically that if the noise in the signal is white (i.e., Gaussian distributed) above and below the step, which is typically the case for the short shaping times used for high signal rate processing, then the average in Equation actually gives the best estimate of **Vx** in the least squares sense. This, of course, is why triangular filtering has been preferred at high rates.

Triangular filtering with time variant filter lengths can, in principle, achieve both somewhat superior resolution and higher throughputs but comes at the cost of a significantly more complex circuit and a rate dependent resolution, which is unacceptable for many types of precise analysis. In practice, XIA's design has been found to duplicate the energy resolution of the best analog shapers while approximately doubling their throughput, providing experimental confirmation of the validity of the approach.

7.3.3 Energy

Interface



When the status below the interface is displaying **green Ready**, it means that the interface can be operated, otherwise you need to wait. The operation of the bottom button is the same as above.

- The parameter Rise Time, please refer to *Trapezoidal Filtering*
- The parameter Flat Top, please refer to *Trapezoidal Filtering*
- The parameter Tau, please refer to *Baselines and Preamp. Decay Times*
- The parameter filter range, please refer to *Filter Range*

The most critical parameter for the energy computation is the signal decay time Tau. It is used to compensate for the falling edge of a previous pulse in the computation of the energy. You can either enter Tau directly for each channel, or enter an approximate value in the right control, select a channel, and click Find it to let the software determine the decay time automatically.

Click Accept it to apply the found value to the channel. (If the approximate value is unchanged,the software could not find a better value.)

At high count rates, pulses overlap with each other at higher frequency. In order to compute the energy or pulse height of those pulses accurately without the need to wait until they decay back to baseline level completely, the pulse height computation algorithm implemented in the Pixie-16 uses the decay time to compute and remove the contribution from the exponentially decaying tail of the overlapping prior pulse when computing the pulse height of the current pulse.

危险：single exponential decay constant

It is assumed the pulses have only a single exponential decay constant. If pulses have multiple decay constants, it might be possible to use the decay constant that dominates the decay of the pulse, but the accuracy of pulse height computation will be degraded.

General rules of thumb for the following important parameters are:

- The energy filter flat top time should be larger than the longest pulse rise time.
- The energy filter rise time can be varied to balance the resolution and throughput.
- In general, energy resolution improves with the increase of energy filter rise time, up to an optimum when longer filters only add more noise into the measurement.
- The energy filter dead time TD is about 2×($T_{rise} + T_{flat}$), and the maximum throughput for Poisson statistics is 1/(TD x e). For HPGe detectors, a rise time of 4-6us and a flat top of 1us are usually appropriate.
- Choose the smallest energy filter range that allows setting the optimum energy filter rise time. Larger filter ranges allow longer filter sums, but increase the granularity of possible values for the energy filter rise time and flat top time and increase the jitter of latching the energy filter output relative to the rising edge of the pulse. This is usually only important for very fast pulses.

Filter Range

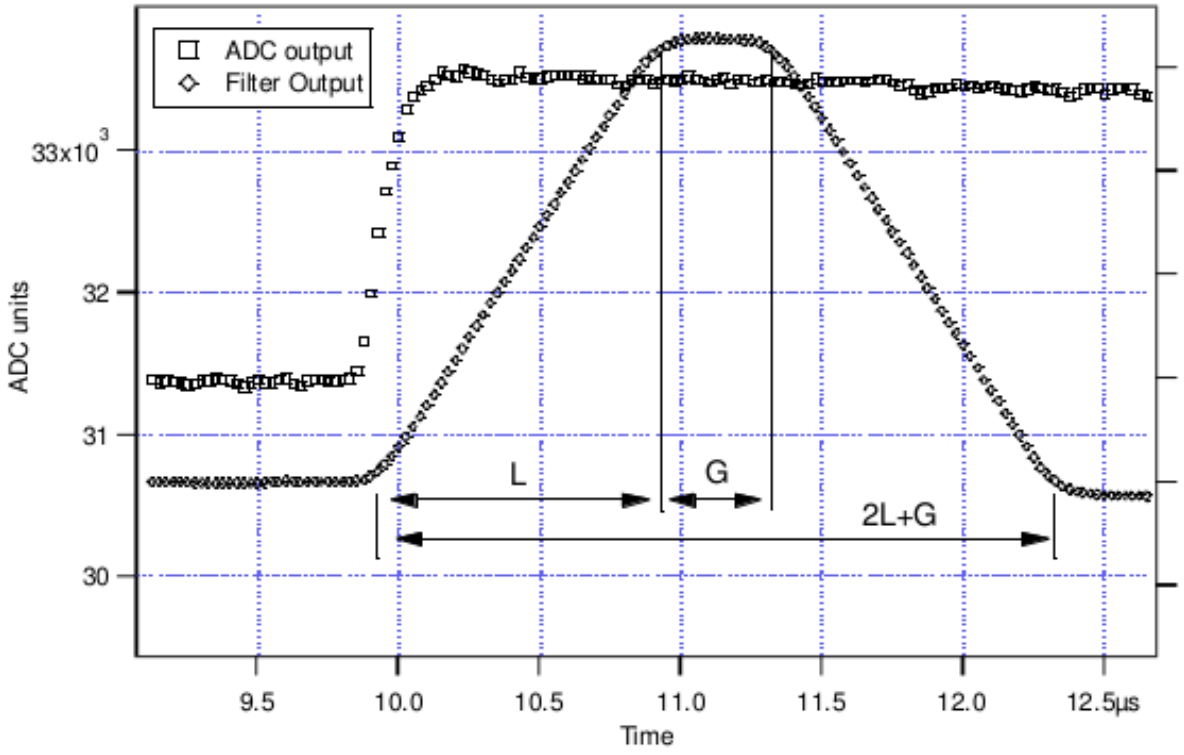
To accommodate a wide range of energy filter rise times from tens of nanoseconds to tens of microseconds, the filters are implemented in the FPGA with different clock decimations(filter ranges). The ADC sampling rate is either 2ns, 4ns, or 10ns depending on the ADC variant that is used, but in higher clock decimations, several ADC samples are averaged before entering the energy filtering logic. In filter range 1, 2^1 samples are averaged, 2^2 samples in filter range 2, and so on. Since the sum of rise time and flat top is limited to 127 decimated clock cycles, filter time granularity and filter time are limited to the values listed in Table .

Filter range	Filter granularity	max. T_{rise}+T_{flat}	min. T_{rise}	min. T_{flat}
1	0.02μs	2.54μs	0.04μs	0.06μs
2	0.04μs	5.08μs	0.08μs	0.12μs
3	0.08μs	10.16μs	0.16μs	0.24μs
4	0.16μs	20.32μs	0.32μs	0.48μs
5	0.32μs	40.64μs	0.64μs	0.96μs
6	0.64μs	81.28μs	1.28μs	1.92μs

Filter range	Filter granularity	max. T_{rise}+T_{flat}	min. T_{rise}	min. T_{flat}
1	0.016µs	2.032µs	0.032µs	0.048µs
2	0.032µs	4.064µs	0.064µs	0.096µs
3	0.064µs	8.128µs	0.128µs	0.192µs
4	0.128µs	16.256µs	0.256µs	0.384µs
5	0.256µs	32.512µs	0.512µs	0.768µs
6	0.512µs	65.024µs	1.024µs	1.536µs

Trapezoidal Filtering

From this point onward, only trapezoidal filtering will be considered as it is implemented in a Pixie-16 module according to Equation $LV_{x,k} = -\sum_{i=k-2L-G+1}^{k-L-G} V_i + \sum_{i=k-L+1}^{k} V_i$. The result of applying such a filter with Length L=1 us and Gap G=0.4 us to a gamma-ray event is shown in Figure. The filter output is clearly trapezoidal in shape and has a rise time equal to L, a flattop equal to G, and a symmetrical fall time equal to L. The basewidth, which is a first-order measure of the filter’ s noise reduction properties, is thus 2L+G.



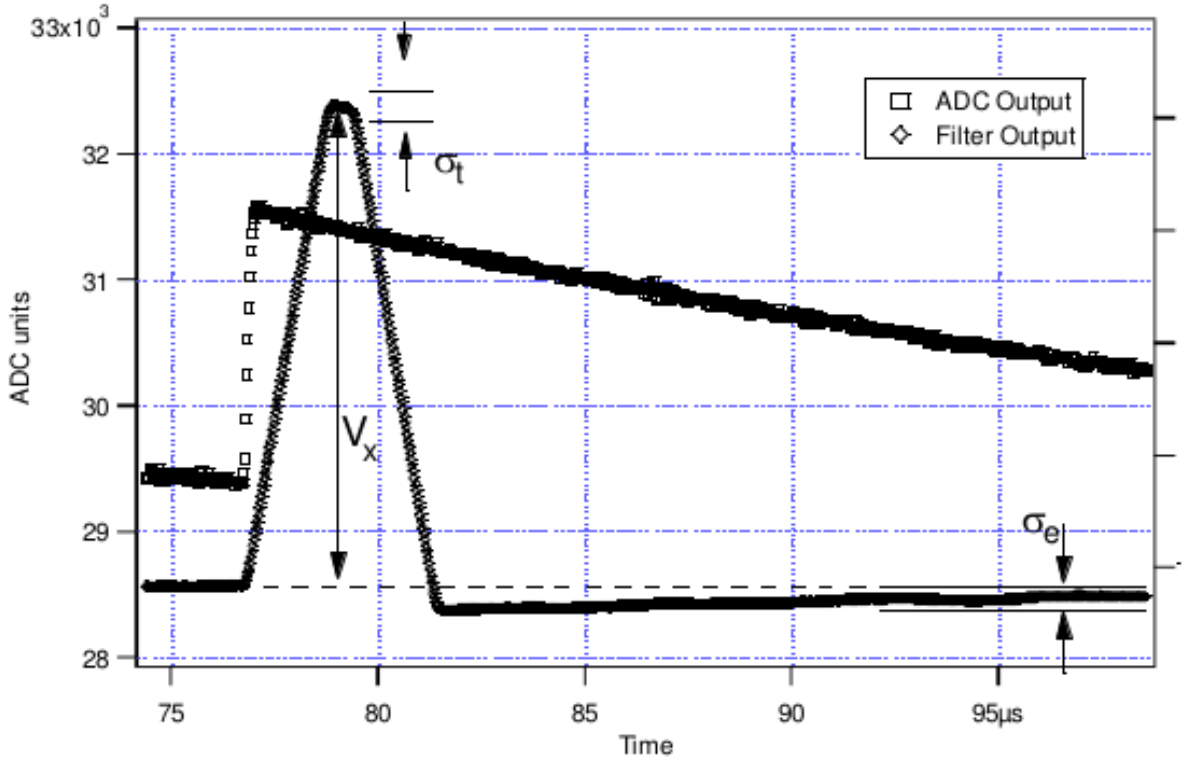
This raises several important points in comparing the noise performance of the Pixie-16 module to analog filtering amplifiers.

- **First, semi-Gaussian filters are usually specified by a shaping time.**
  - Their rise time is typically twice this and their pulses are not symmetric so that the basewidth is about 5.6 times the shaping time or 2.8 times their rise time.
- **Thus a semi-Gaussian filter typically has a slightly better energy resolution than a triangular filter of the same rise tim**
  - This is typically accommodated in amplifiers offering both triangular and semi-Gaussian filtering by stretching the triangular rise time a bit, so that the true triangular rise time is typically 1.2 times the selected semi-Gaussian rise time.
  - This also leads to an apparent advantage for the analog system when its energy resolution is compared to a digital system with the same nominal rise time.

One important characteristic of a digitally shaped trapezoidal pulse is its extremely sharp termination on completion of the basewidth 2L+G. This may be compared to analog filtered pulses whose tails may persist up to 40% of the rise time, a phenomenon due to the finite bandwidth of the analog filter. As can be seen below, this sharp termination gives the digital filter a definite rate advantage in pileup free throughput.

Baselines and Preamp. Decay Times

Figure shows an event over a longer time interval and how the filter treats the preamplifier noise in regions when no gamma-ray pulses are present.



As may be seen the effect of the filter is both to reduce the amplitude of the fluctuations and reduce their high frequency content. This region is called the baseline because it establishes the reference level from which the gamma-ray peak amplitude **Vx** is to be measured. The fluctuations in the baseline have a standard deviation σ_e which is referred to as the electronic noise of the system, a number which depends on the rise time of the filter used. Riding on top of this noise, the gamma-ray peaks contribute an additional noise term, the Fano noise, which arises from statistical fluctuations in the amount of charge **Qx** produced when the gamma-ray is absorbed in the detector. This Fano noise σ_f adds in quadrature with the electronic noise, so that the total noise σ_t in measuring **Vx** is found from:

$$\sigma_t = \sqrt{\sigma_f^2 + \sigma_e^2}$$

The Fano noise is only a property of the detector material. The electronic noise, on the other hand, may have contributions from both the preamplifier and the amplifier. When the preamplifier and amplifier are both well designed and well matched, however, the amplifier’ s noise contribution should be essentially negligible. Achieving this in the mixed analog-digital environment of a digital pulse processor is a non-trivial task, however.

With a RC-type preamplifier, the slope of the preamplifier is rarely zero. Every step decays exponentially back to the DC level of the preamplifier. During such a decay, the baselines are obviously not zero. This can be seen in Figure, where the filter output during the exponential decay after the pulse is below the initial level. Note also that the flat top region is sloped downwards.

Using the decay constant τ, the baselines can be mapped back to the DC level. This allows precise determination of gamma-ray energies, even if the pulse sits on the falling slope of a previous pulse. The value of τ, being a characteristic of the preamplifier, has to be determined by the user and host software and downloaded to the module.

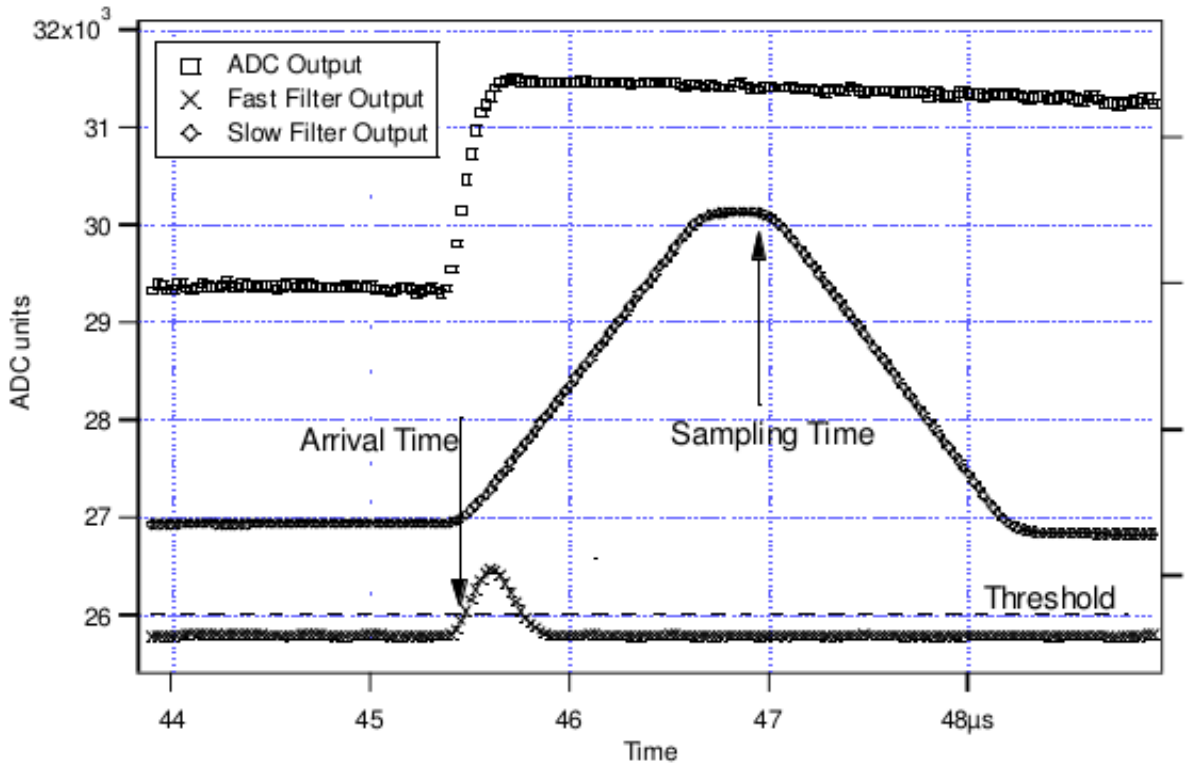
Pileup Inspection

As noted above, the goal is to capture a value of **Vx** for each gamma-ray detected and use these values to construct a spectrum.

注解: This process is also significantly different between digital and analog systems. In the analog system the peak value must be "captured" into an analog storage device, usually a capacitor, and "held" until it is digitized. Then the digital value is used to update a memory location to build the desired spectrum. During this analog to digital conversion process the system is dead to other events, which can severely reduce system throughput. Even single channel analyzer systems introduce significant deadtime at this stage since they must wait some period (typically a few microseconds) to determine whether or not the window condition is satisfied.

Digital systems are much more efficient in this regard, since the values output by the filter are already digital values. All that is required is to take the filter sums, reconstruct the energy Vx , and add it to the spectrum. In the Pixie-16, the filter sums are continuously updated in the FPGA, and are captured into event buffers. Reconstructing the energy and incrementing the spectrum is done by the DSP, so that the FPGA is ready to take new data immediately (unless the buffers are full). This is a significant source of the enhanced throughput found in digital systems.

The peak detection and sampling in a Pixie-16 module is handled as indicated in follow Figure. Two trapezoidal filters are implemented, a fast filter and a slow filter. The fast filter is used to detect the arrival of gamma-rays, the slow filter is used for the measurement of **Vx** , with reduced noise at longer filter rise times. The fast filter has a filter length Lf = 0.1us and a gap Gf = 0.1us. The slow filter has Ls = 1.2us and Gs = 0.35us.

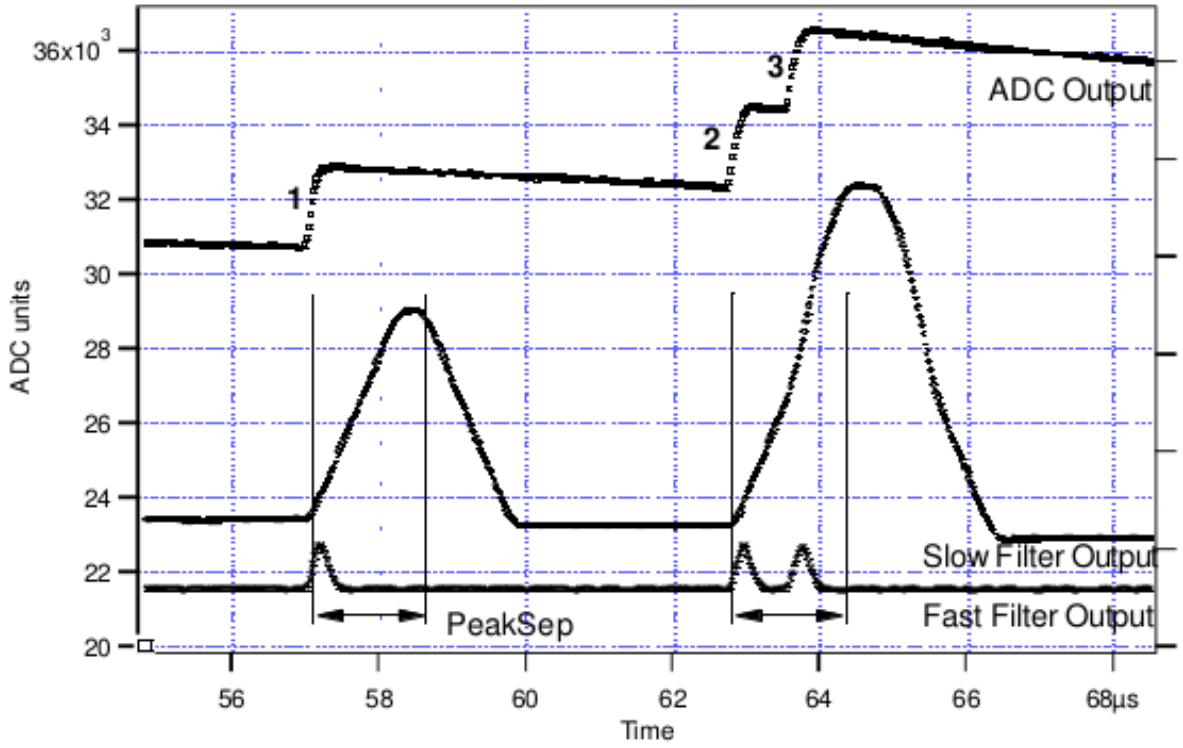


The arrival of the gamma-ray step(in the preamplifier output) is detected by digitally comparing the fast filter output to **THRESHOLD**, a digital constant set by the user. Crossing the threshold starts a delay line to wait **PEAKSAMP** clock cycles to arrive at the appropriate time to sample the value of the slow filter. Because the digital filtering processes are deterministic, **PEAKSAMP** depends only on the values of the fast and slow filter constants.

The slow filter value captured following **PEAKSAMP** is then the slow digital filter's estimate of **Vx** . Using a delay line allows to stage sampling of multiple pulses even within a **PEAKSAMP** interval (though the filter values themselves are then not correct representations of a single pulse's height).

The value **Vx** captured will only be a valid measure of the associated gamma-ray's energy provided that the filtered pulse is sufficiently well separated in time from its preceding and succeeding neighbor pulses so that their peak

amplitudes are not distorted by the action of the trapezoidal filter. That is, if the pulse is not piled up. The relevant issues may be understood by reference to Figure, which shows 3 gamma-rays arriving separated by various intervals. The fast filter has a filter length Lf = 0.1us and a gap Gf =0.1us. The slow filter has Ls = 1.2us and Gs = 0.35us.



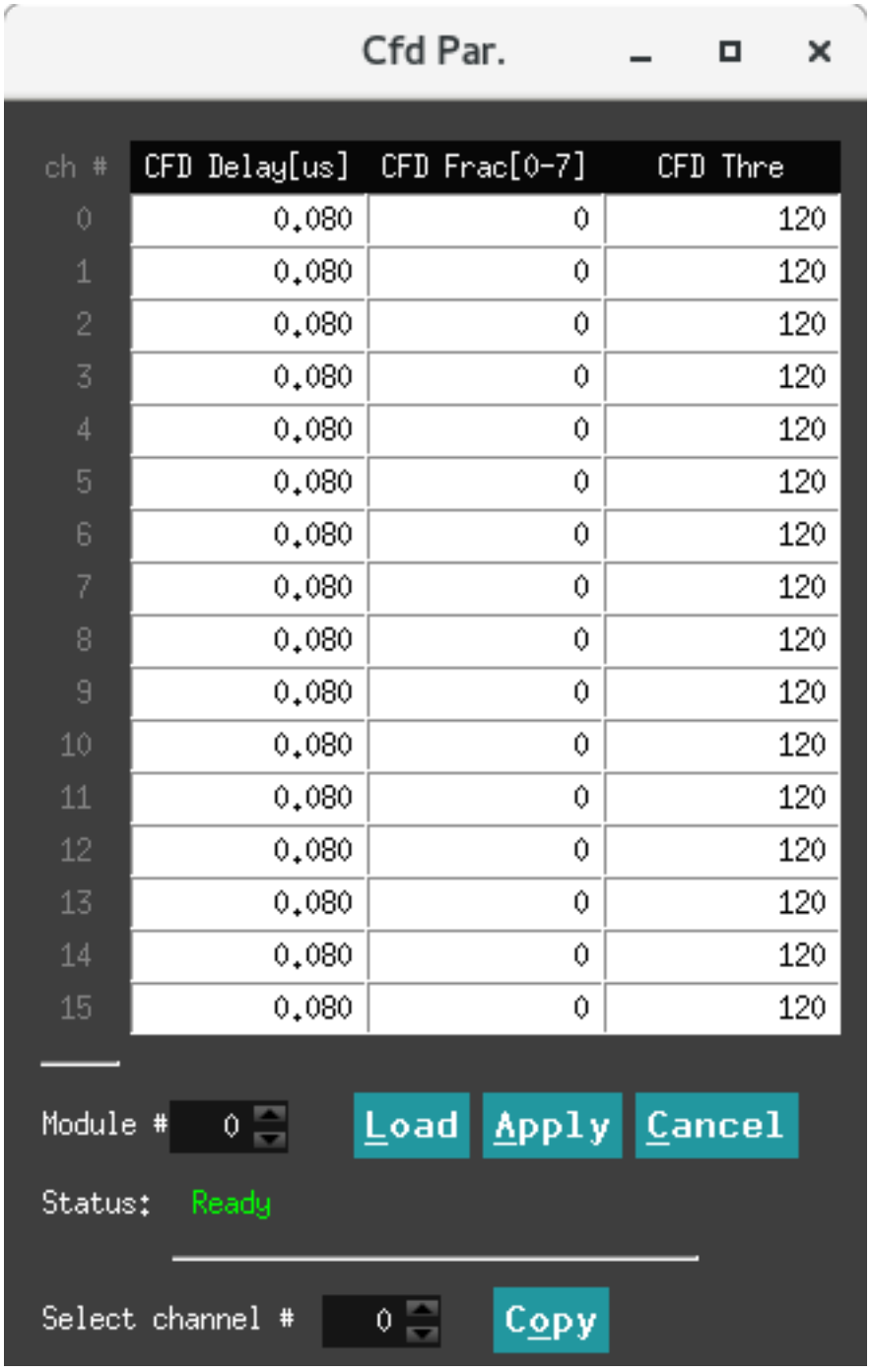
Because the trapezoidal filter is a linear filter, its output for a series of pulses is the linear sum of its outputs for the individual members in the series. Pileup occurs when the rising edge of one pulse lies under the peak (specifically the sampling point) of its neighbor. Thus, in Figure , peaks 1 an 2 are sufficiently well separated so that the leading edge of peak 2 falls after the peak of pulse 1. Because the trapezoidal filter function is symmetrical, this also means that pulse 1’ s trailing edge also does not fall under the peak of pulse 2. For this to be true, the two pulses must be separated by at least an interval of L+G. Peaks 2 and 3, which are separated by less than 1.0 us, are thus seen to pileup in the present example with a 1.2 us rise time.

This leads to an important point: whether pulses suffer slow pileup depends critically on the rise time of the filter being used. The amount of pileup which occurs at a given average signal rate will increase with longer rise times.

Because the fast filter rise time is only 0.1 us, these gamma-ray pulses do not pileup in the fast filter channel. The Pixie-16 module can therefore test for slow channel pileup by measuring the fast filter for the interval PEAKSEP after a pulse arrival time. If no second pulse occurs in this interval, then there is no trailing edge pileup and the pulse is validated for acquisition. **PEAKSEP** is usually set to a value close to L+G+1. Pulse 1 passes this test, as shown in Figure. Pulse 2, however, fails the **PEAKSEP** test because pulse 3 follows less than 1.0 us. Notice, by the symmetry of the trapezoidal filter, if pulse 2 is rejected because of pulse 3, then pulse 3 is similarly rejected because of pulse 2.

7.3.4 CFD

Interface



TODO

100 MHz and 250 MHz modules

The following CFD algorithm is implemented in the signal processing FPGA of the 100 MHz(Rev. B, C, D and F) and 250 MHz(Rev. F) Pixie-16 modules.

Assume the digitized waveform can be represented by data series Trace[i], i = 0, 1, 2, ⋯First the fast filter re-

sponse(FF) of the digitized waveform is computed as follows:

$$FF[i] = \sum_{j=i-(FL-1)}^{i} Trace[j] - \sum_{j=i-(2\times FL+FG-1)}^{i-(FL+FG)} Trace[j]$$

Where FL is called the fast length and FG is called the fast gap of the digital trapezoidal filter. Then the CFD is computed as follows:

$$CFD[i+D] = FF[i+D] \times (1 - w/8) - FF[i]$$

Where D is called the CFD delay length and w is called the CFD scaling factor(w=0, 1,⋯, 7).

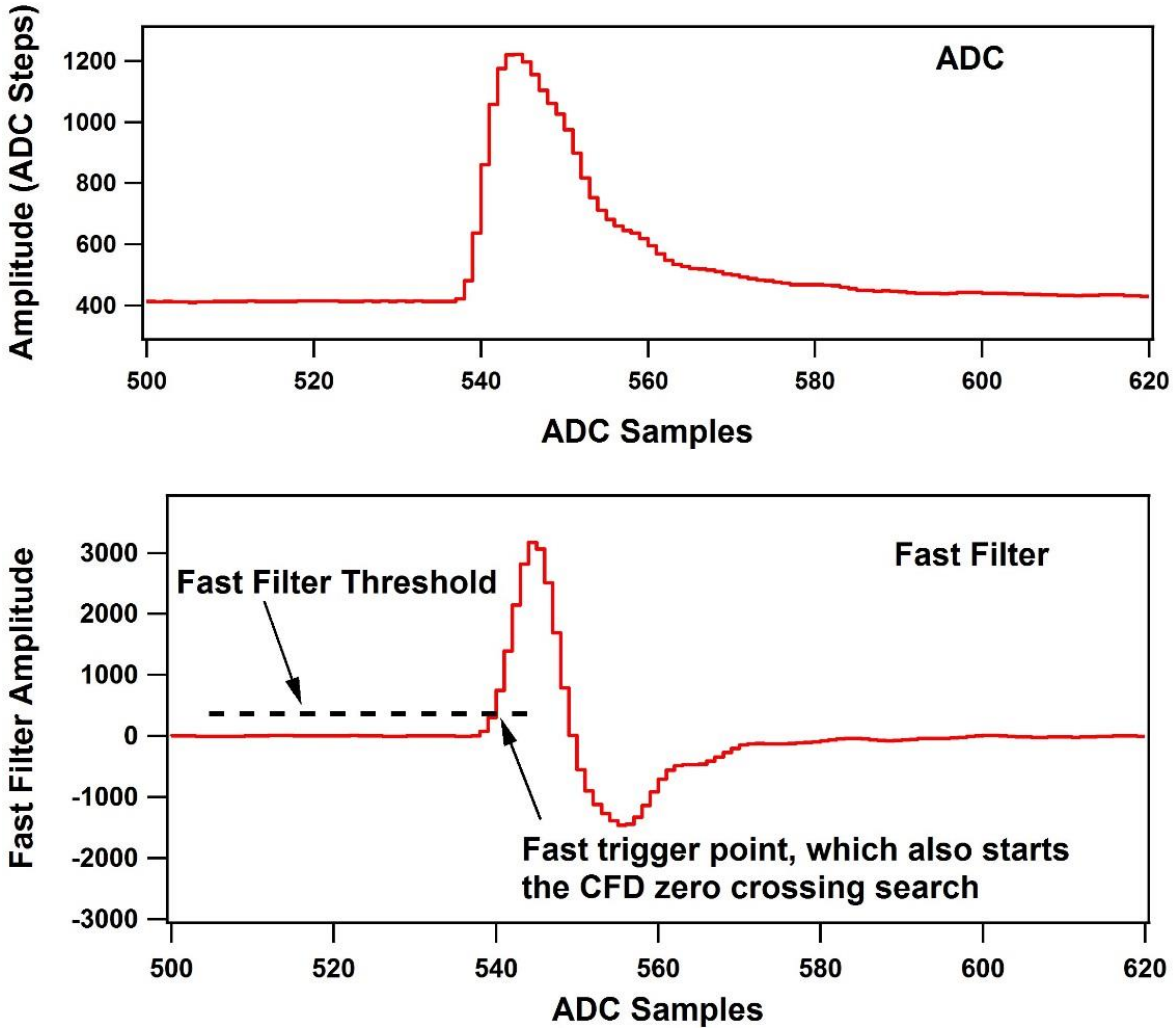
The CFD zero crossing point(ZCP) is then determined when $CFD[i] \geq 0$ and $CFD[i+1] < 0$. The timestamp is latched at Trace point i, and the fraction time f is given by the ratio of the two CFD response amplitudes right before and after the ZCP.

$$f = \frac{CFDout1}{CFDout1 - CFDout2}$$

Where CFDout1 is the CFD response amplitude right before the ZCP, and CFDout2 is the CFD response amplitude right after the ZCP(subtraction is used in the denominator since CFDout2 is negative). The Pixie-16 DSP computes the CFD final value as follows and stores it in the output data stream for online or offline analysis.

$$CFD = \frac{CFDout1}{CFDout1 - CFDout2} \times N$$

Where N is scaling factor, which equals to 32768 for 100 MHz modules and 16384 for 250 MHz modules, respectively.



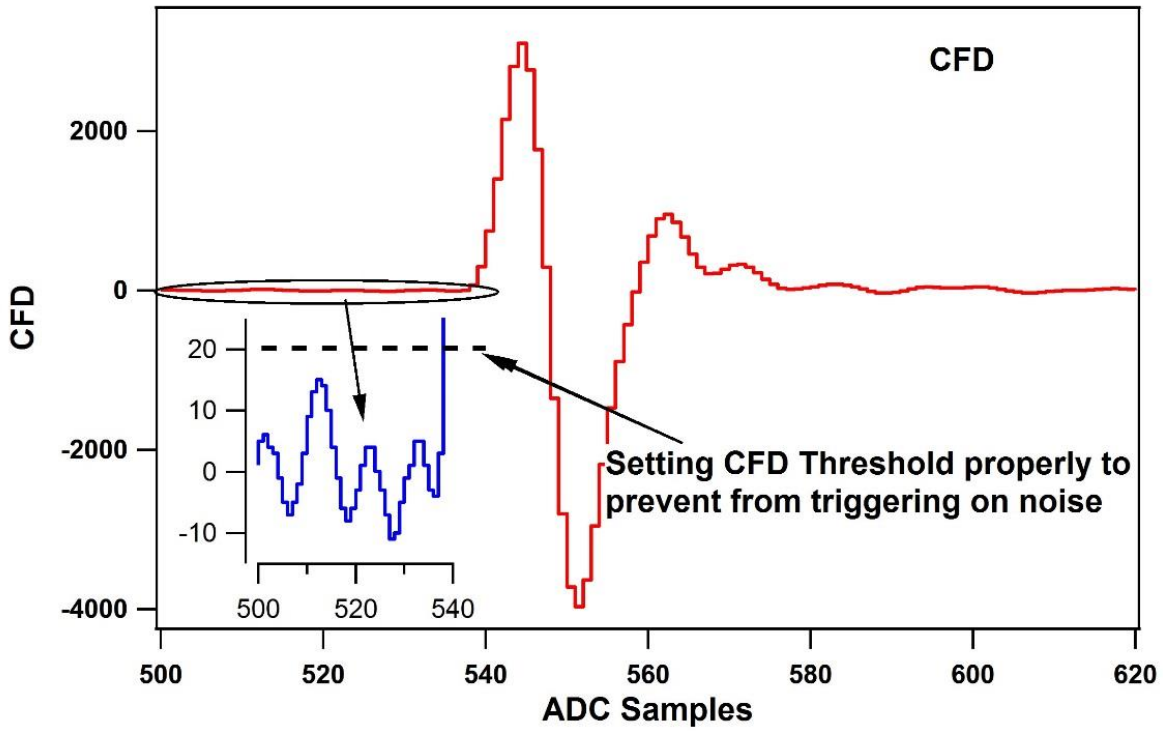


Figure shows a sample ADC trace, its fast filter response and its CFD response, respectively.

The top figure shows a raw ADC trace. After computing the fast filter response on the raw ADC trace using Equation $FF[i]$, the fast filter response is compared against the fast filter threshold as shown in the middle figure. The ADC sample where the fast filter response crosses the fast filter threshold is called the fast trigger point, which also starts the search for the CFD zero crossing point.

The CFD response is computed using Equation $CFD[i + D]$ and is shown in the bottom figure(for actual implementation in the firmware, the fast filter response FF is delayed slightly before being used for computing the CFD response so that there are sufficient number of CFD response points to look for the zero crossing point after the fast trigger). To prevent premature CFD trigger as a result of the noise in the CFD response before the actual trigger, a DSP parameter called CFDThresh is used to suppress those noise-caused zero crossing. However, if a zero crossing point cannot be found within a certain period after the fast trigger (typically 32 clock cycles), e.g., due to unnecessarily high CFDThresh, a forced CFD Trigger will be issued and a flag will be set in an event header word to indicate that the recorded CFD time for this event is invalid.

However, the event will still have a valid timestamp which is latched by the fast filter trigger when fast filter crosses over the trigger threshold. The aforementioned CFD parameters correspond to the following DSP parameters.

CFD Parameters	DSP Parameters
FL	FastLength
FG	FastGap
Fast Filter Threshold	FastThresh
D	CFDDelay
W	CFDScale (valid values: 0, 1, 2, … and 7)
CFD Threshold	CFDThresh

注解：250 MHz

In the 250 MHz Pixie-16 modules, the event timestamp is counted with 125 MHz clock ticks, i.e., 8 ns intervals, and two consecutive 250 MHz ADC samples are captured in one 8 ns interval as well.

The CFD trigger also runs at 125 MHz, but the CFD zero crossing point is still reported as a fractional time between two neighboring 250 MHz ADC samples, which are processed by the FPGA in one 125 MHz clock cycle.

However, the CFD zero crossing point could be in either the odd or even clock cycle of the captured 250 MHz ADC waveforms.

Therefore, the firmware outputs a "CFD trigger source" bit in the output data stream to indicate whether the CFD zero crossing point is in the odd or even clock cycle of the captured 250 MHz ADC waveforms.

注解: 100 MHz

In the 100 MHz Pixie-16 modules, event timestamp, CFD trigger, and ADC waveform capture are all carried out with the same 100 MHz clock. So there is no need to report "CFD trigger source" for the 100 MHz Pixie-16 modules.

500 MHz modules

The CFD algorithm discussed in the previous section for the 100 MHz and 250 MHz Pixie-16 modules can also be written in the following format:

$$CFD(k) = w \cdot \left(\sum_{i=k}^{k+L} a(i) - \sum_{i=k-B}^{k-B+L} a(i) \right) - \left(\sum_{i=k-D}^{k-D+L} a(i) - \sum_{i=k-D-B}^{k-D-B+L} a(i) \right)$$

Where a(i) is the ADC trace data, k is the index, and w, B, D, and L are CFD parameters.

The CFD algorithm implemented in the 500 MHz Pixie-16 modules is special when compared to the one implemented in the 100 MHz and 250 MHz Pixie-16 modules in terms of the ability to adjust parameters w, B, D, and L.

The reason for this is that in the 500 MHz Pixie-16 modules, ADC data that come into the FPGA at the speed of 500 MHz is first slowed down with a ratio of 1:5, in other words, the FPGA captures 5 ADC samples at the rate of 100 MHz, i.e., every 10 ns. The FPGA then tries to find the CFD trigger point between any two adjacent 2-ns ADC samples within that 10 ns by first building sums of ADC samples and then calculating differences between delayed and non-delayed sums until the zero crossing point is found. However, in the 500 MHz Pixie-16 modules, the FPGA does not have enough resources to build sums for 5 ADC samples in parallel with variable delays. Therefore, the CFD algorithm for the 500 MHz modules was implemented using a set of fixed CFD parameters as shown in Table *Fixed CFD Parameter Values for 500 MHz Pixie-16 Modules*. Tests show these fixed parameters give best performance for LaBr3(Ce) detectors.

CFD Parameters	Fixed Values for 500 MHz Modules
w	1
B	5
D	5
L	1

The CFD time given by the 500 MHz Pixie-16 modules consists of two parts: a shift within the 5 ADC samples and a fractional time between two ADC samples where the CFD zero crossing occurred. The shift within the 5 ADC samples is reported as the 3-bit CFD trigger source[2:0] is defined as follows.

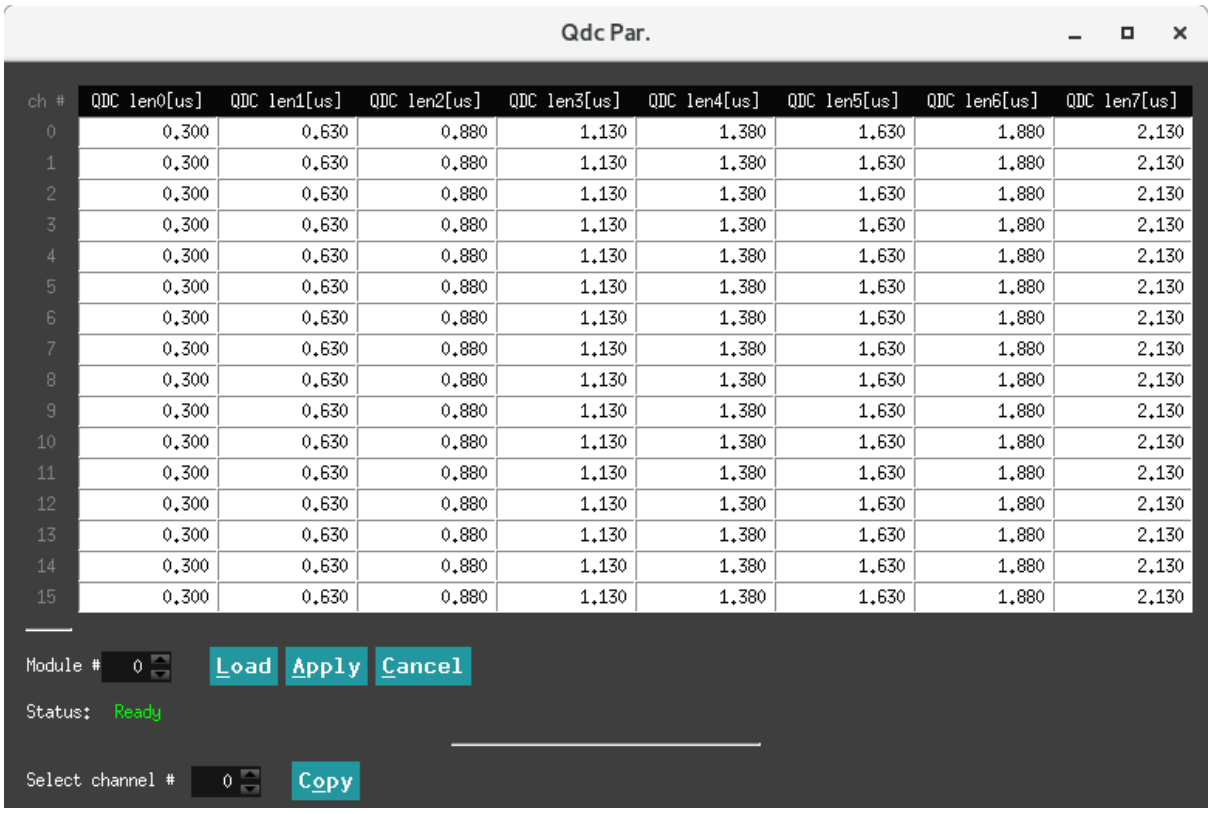
CFD Trigger Source [2:0]	Zero Crossing Point (ZCP) Location
000	ZCP occurred between the 5th ADC sample of the previous 5-sample group and the 1st ADC sample of the current 5-sample group
001	ZCP occurred between the 1th ADC sample of the current 5-sample group and the 2nd ADC sample of the current 5-sample group
010	ZCP occurred between the 2nd ADC sample of the current 5-sample group and the 3rd ADC sample of the current 5-sample group
011	ZCP occurred between the 3rd ADC sample of the current 5-sample group and the 4th ADC sample of the current 5-sample group
100	ZCP occurred between the 4th ADC sample of the current 5-sample group and the 5th ADC sample of the current 5-sample group
101	Not used
110	Not used
111	CFD trigger is forced, so CFD time is invalid

CFD 分数时间如下：

$$CFD = \frac{CFDout1}{CFDout1 - CFDout2} \times 8192$$

7.3.5 QDC

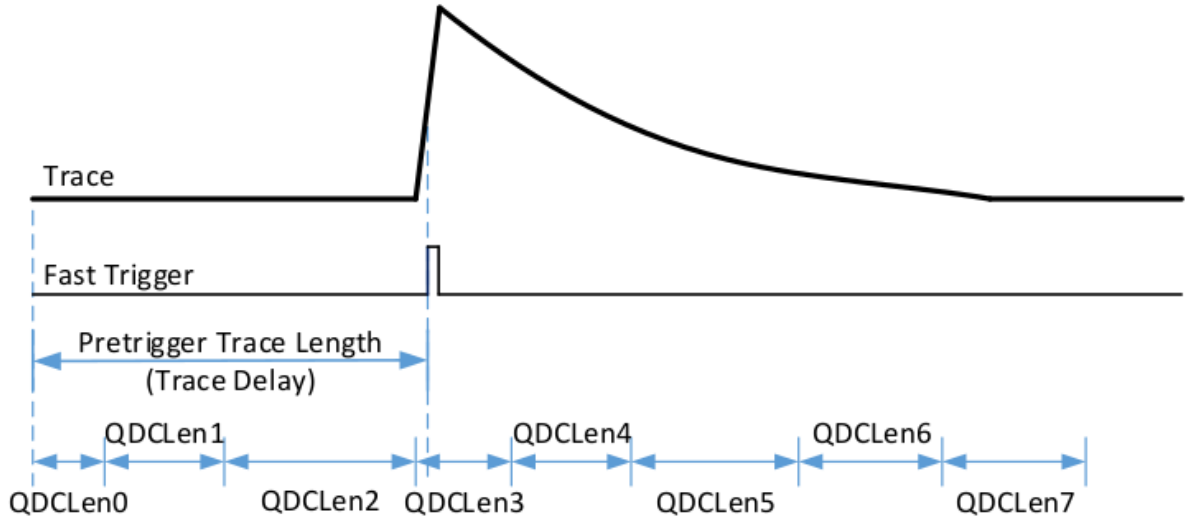
Interface



Eight QDC sums, each of which can have different lengths, are computed in the Signal Processing FPGA of a Pixie-16 module for each channel and the sums are written to the list mode output data stream if the user requests so.

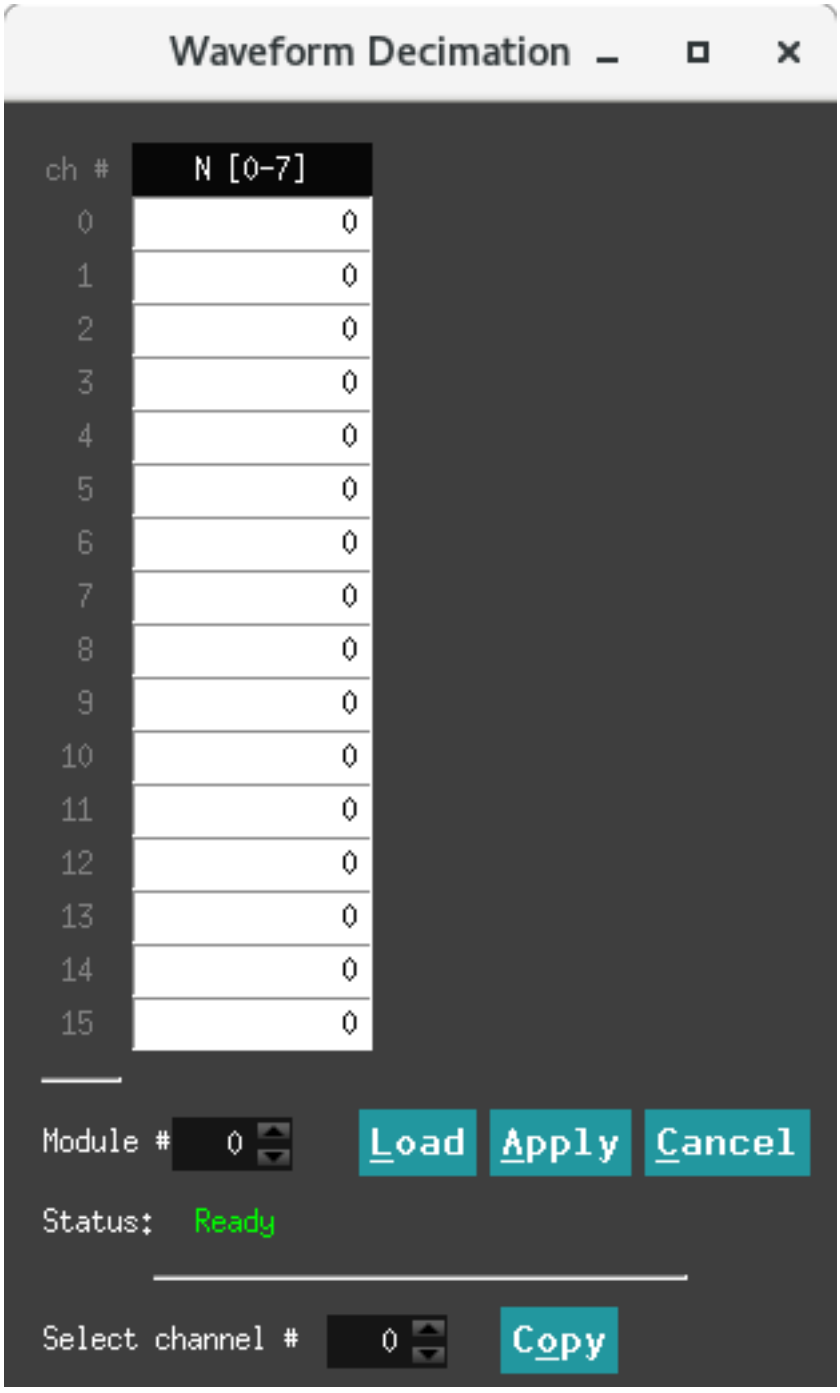
The recording of QDC sums starts at the waveform point which is *Pre-trigger Trace Length* or *Trace Delay* earlier than the trigger point, which is either the CFD trigger or channel fast trigger depending on whether or not CFD trigger mode is enabled.

The eight QDC sums are computed one by one continuously, but they are not overlapping. The recording of QDC sums ends when the eight intervals have all passed.



7.3.6 Decimation

Interface

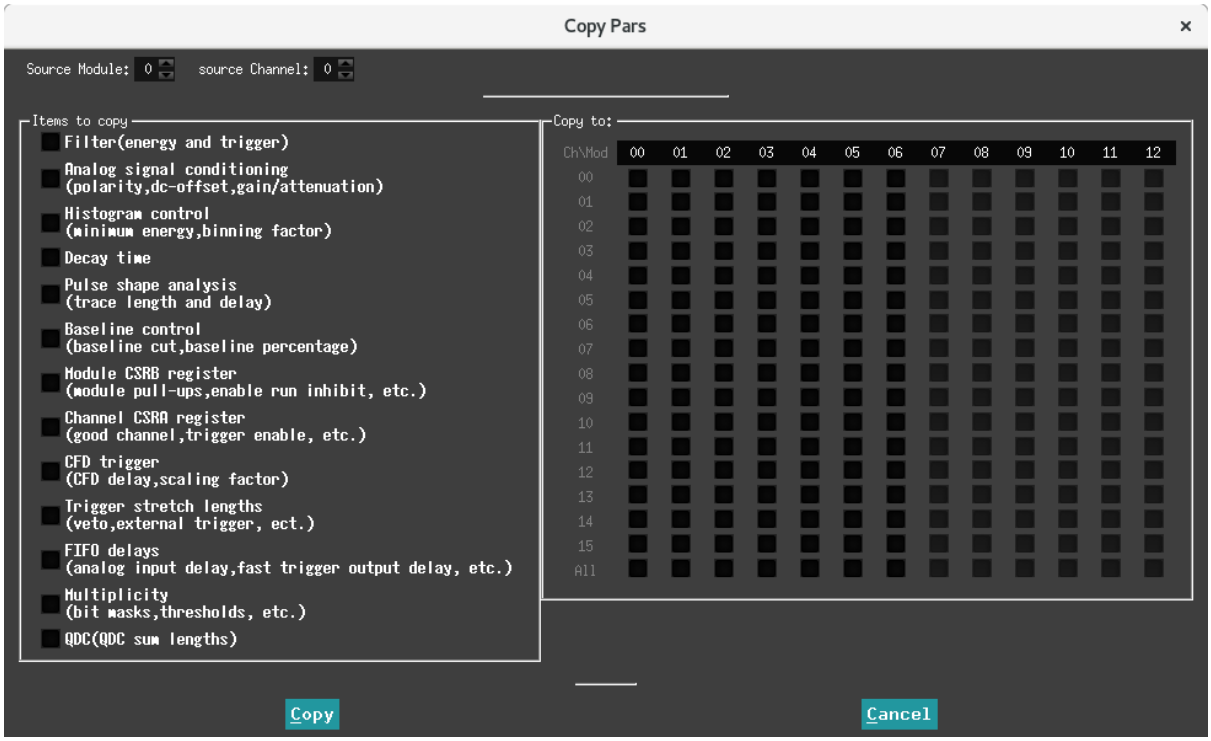


Waveform down frequency outputs are customized for the 100 MHz module in the PKU firmware.

The record waveform mode with down frequency output. The strategy adopted is to select the output of 1, 1/2, 1/4, 1/8, 1/16, 1/32, 1/64, 1/128 frequency, ie how many points retain one point. The points retained are the averaged values.

7.3.7 Copy Pars

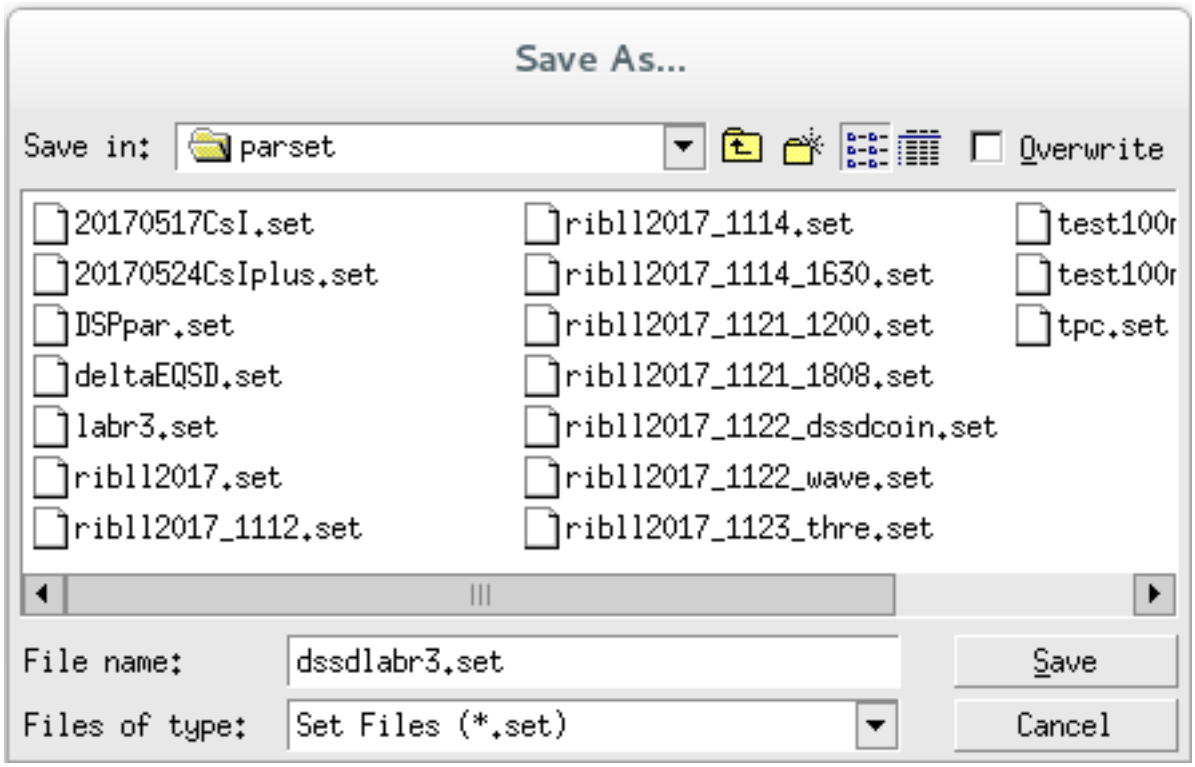
Interface



This window is used to quickly copy parameters between channels.

7.3.8 Save2File

Interface



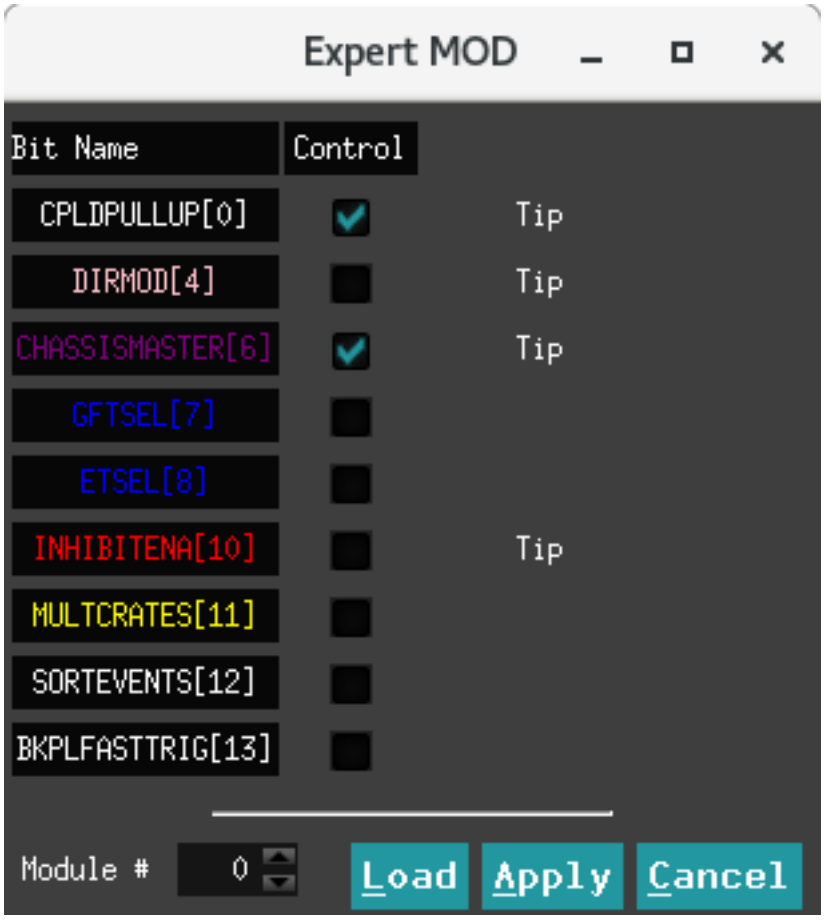
As you know, the DSP parameters are stored in a file with the suffix "set". Once you have configured the parameters, you should save them for later use.

7.4 Expert

The contents in this drop-down column are high-level, and you need to learn to master the acquisition logic.

7.4.1 Module Variables

Interface



In addition to distributing the global clock signal, the Pixie-16 rear I/O trigger module can also share global triggers and run synchronization signals. The global trigger signals include the global validation trigger and global fast trigger, plus the Pixie-16 FPGA data storage buffers' full flag signal. The run synchronization signals include synchronous run start and stop signals that can be shared among multiple crates.

In order to enable the distribution of such global triggers and run synchronization signals, certain Pixie-16 parameters have to be set properly. The parameter that controls the trigger distribution and run synchronization is the Module Control Register B (ModCSRB).

ModCSRB is a 32-bit parameter with each of 32 bits controlling different operation modes of the Pixie-16 module.

注解: Trigger Distribution and Run Synchronization

For the System Director module that is installed in the Master crate, bits 0, 4, 6 and 11 of ModCSRB should be set to 1 (checked & enabled).

For the Crate Master module that is installed in the Slave crate, bits 0, 6 and 11 of ModCSRB should be set to 1 (checked & enabled).

For the General modules that are installed in both the Slave crate and Master crate, bit 11 of ModCSRB should be set to 1 (checked & enabled).

Register definition

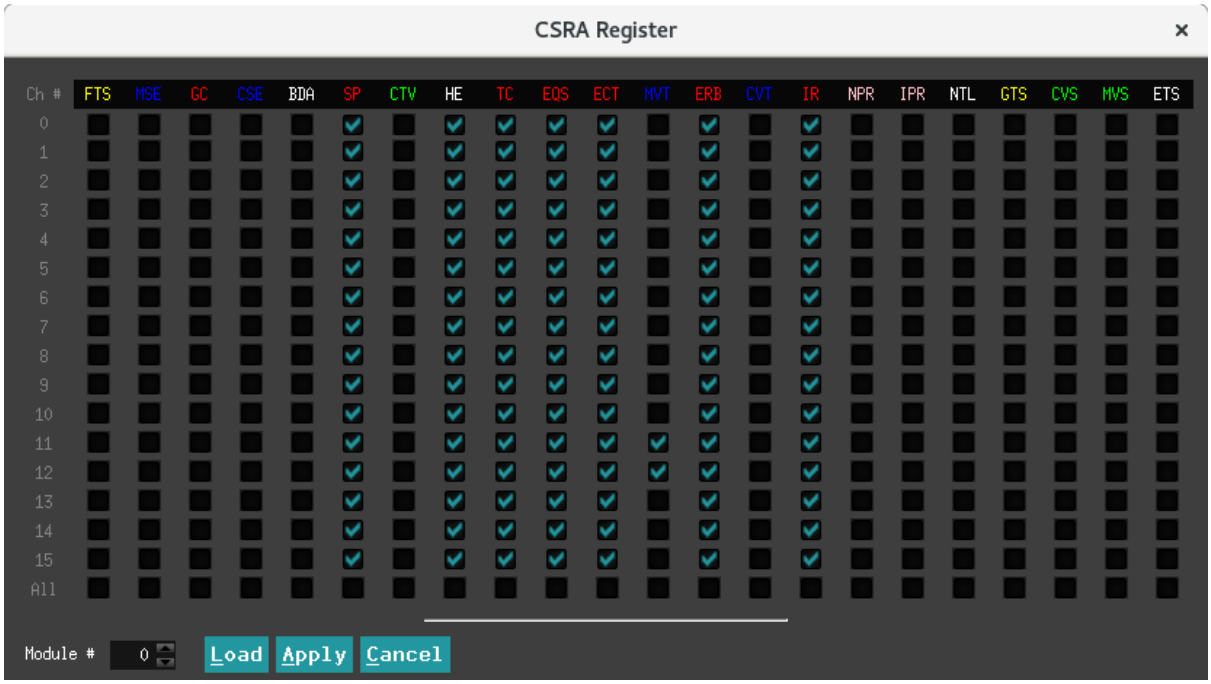
Module Control Register B affecting the module as a whole.

- **bit 0 - MODCSRB_CPLDPULLUP**
  - Enable pullups for PXI trigger lines on the backplane through an onboard CPLD.
  - With the pullups, those PXI trigger lines default to logic high state.
  - Only when one module actively pulls a line to logic low state will such a line be in the low state.
  - Therefore signals transmitted over those PXI trigger lines are actively low signals.
  - **Note: enable this bit only for one module per crate (e.g. the crate master module)**
- **bit 4 - MODCSRB_DIRMOD**
  - Set this module as the Director module so that it can send triggers, trace and header DPM full signal and run synchronization signal to all crates through the rear I/O trigger modules.
  - Here triggers include fast trigger and validation trigger
  - **Note: enable this bit only for one module among all crates (e.g. the system director module in multi-crate configuration)**
- **bit 6 - MODCSRB_CHASSISMASTER**
  - Set this module as the chassis master module so that it can send triggers, trace and header DPM full signal and run synchronization signal to the backplane of the local crate.
  - Here triggers include fast trigger and validation trigger
  - **Note: enable this bit only for one module per crate(e.g. the crate master module)**
- **bit 7 - MODCSRB_GFTSEL**
  - Select external fast trigger source(=1: external validation trigger, =0: external fast trigger, in case these two signals are swapped at the Pixie-16 front panel input connectors)
- **bit 8 - MODCSRB_ETSEL**
  - Select external validation trigger source(=1: external fast trigger, =0: external validation trigger, in case these two signals are swapped at the Pixie-16 front panel input connectors)
- **bit 10 - MODCSRB_INHIBITENA**
  - Enable(=1) or disable(=0) the use of external INHIBIT signal.
  - When enabled, the external INHIBIT signal in the logic high state will prevent the run from starting until this external INHIBIT signal goes to logic low state.
- **bit 11 - MODCSRB_MULTCRATES**
  - Set this module to run in the multi-crate mode(=1) or in the local-crate mode(=0).
  - If the module is running in multi-crate mode, it will use the trace and header DPM full signal and run synchronization signal that are generated and distributed among multiple crates.
  - If the module is running in local-crate mode, it will use the trace and header DPM full signal and run synchronization signal generated in the local crate.

- **bit 12 - MODCSRB_SORTEVENTS**
  - Sort(=1) or do not sort(=0) events from all 16 channels of a Pixie-16 module based on the timestamps of the events, before storing the events in the external FIFO.
  - Note: all 16 channels must have the same DAQ parameters setting to use this feature
- **bit 13 - MODCSRB_BKPLFASTTRIG**
  - Enable(=1) or disable(=0) the sending of 16 local fast triggers to the 16 lines on the backplane of the crate.
  - **Note: only one module can enable this option in each PCI bus segment of a crate(not limited to the crate master module, e.g. any module in each PCI bus segment)**

7.4.2 CSRA

Interface



- **The yellow FTS and GTS combinations are used to select the channel fast trigger:**
  - When both are not selected, it is local fast trigger.
  - When selecting FTS, it is a latched module fast trigger.
  - FTS is not selected, GTS is selected as a latched channel validation trigger.
- **Blue MSE, CSE, MVT, CVT are used to select module/channel validation trigger:**
  - MVT is the module validation trigger.
  - CVT is the channel validation trigger.
  - MSE select module validation trigger depends on System FPGA or front panel module GATE.
  - CSE selection channel validation trigger depends on System FPGA or front panel channel GATE.
- **The pink NPR, IPR combination is used to select the treatment of the pileup event:**
  - All events are logged when both are not selected, and the accumulated event energy value is invalid.

  - NPR selection does not record stacking events when IPR is not selected.
  - When the NPR is not selected, the event record waveform is accumulated, and the waveform is not recorded when it is not stacked.
  - When both NPR and IPR are selected, only the stacked events are recorded.

- **Green CTV, CVS, MVS are used to select module/channel veto:**
  - MVS select module veto depending on the front panel module GATE or module validation trigger.
  - CVS select channel veto depending on the front panel channel GATE or channel validation trigger.
  - CTV is whether to enable channel trigger veto.
- **The Red contents are the basic settings.**
  - Black NTL is to keep the waveform out of range.
  - Black ETS is to record the data of the external clock.
- The remaining BDA does not need to be selected, HE does not matter.

Register definition

Channel Control Register A affecting each channel individually

- **bit 0 - CCSRA_FTRIGSEL**
  - Channel fast trigger selection(=1: module fast trigger from the System FPGA; =0: the selection depends on the value of another bit CCSRA_GROUPTRIGSEL – if CCSRA_GROUPTRIGSEL = 1, select the channel validation trigger from the System FPGA, and if CCSRA_GROUPTRIGSEL = 0, select this channel’s local fast trigger)
- **bit 1 - CCSRA_EXTTRIGSEL**
  - Module validation trigger selection(=1: module gate input from the Pixie-16 front panel Module Gate LVDS connector; =0: module validation trigger from the System FPGA)
- **bit 2 - CCSRA_GOOD**
  - Set this channel as a Good channel(=1) or a not Good channel(=0).
  - **When a channel is set to be a not Good channel, it still generates local fast triggers, which could be used in multiplicity computation, etc., but this channel will not record list mode data or MCA data, and will not update its baseline value**
- **bit 3 - CCSRA_CHANTRIGSEL**
  - Channel validation signal selection(=1: channel gate input from the Pixie-16 front panel Channel Gate LVDS connector; =0:channel validation trigger from the System FPGA)
- **bit 4 - CCSRA_SYNCDATAACQ**
  - Choose the level of synchronous data acquisition for this channel(=1: stops taking data when the trace or header DPM for any channel of any Pixie-16 module in the system is full; =0: stops taking data only when the trace or header DPM for this channel of this Pixie-16 module is full)
- **bit 5 - CCSRA_POLARITY**
  - Choose this channel’s input signal polarity(=1: invert input signal’s polarity; =0: do not invert input signal’s polarity).
  - **Please note in Pixie-16, signal processing requires positive rising input signal. So if input signal has a negative falling edge, it should be inverted by setting this CCSRA_POLARITY bit to 1**
- **bit 6 - CCSRA_VETOENA**
  - Enable(=1) or disable(=0) this channel’s veto.

  - If veto is enabled, this channel’s fast trigger will be vetoed by either the module veto signal(see bit 20 CCSRA_MODVETOSEL below) or channel veto signal(see bit 19 CCSRA_CHANVETOSEL below).
  - But if veto is disabled, this channel’s fast trigger will not be vetoed by either veto signal, even if either veto signal is present

- **bit 7 - CCSRA_HISTOE**
  - Enable(=1) or disable(=0) the histogramming of pulse energy values in the onboard MCA memory.
  - However, the current Pixie-16 firmware always histograms pulse energy values in the onboard MCA memory.
  - So this CCSRA_HISTOE is essentially not in use at the moment
- **bit 8 - CCSRA_TRACEENA**
  - Enable(=1) or disable(=0) trace capture in the list mode run for this channel
- **bit 9 - CCSRA_QDCENA**
  - Enable(=1) or disable(=0) QDC sums recording in the list mode run for this channel.
  - There are a total of 8 QDC sums for each event
- **bit 10 - CCSRA_CFDMODE**
  - Enable(=1) or disable(=0) CFD trigger in the list mode run for this channel.
  - CFD trigger is used to latch sub-sample timing for the event time of arrival or timestamp
- **bit 11 - CCSRA_GLOBTRIG**
  - Enable(=1) or disable(=0) the requirement of module validation trigger for this channel.
  - If enabled, only when module validation trigger overlaps the channel fast trigger will the events be recorded for this channel
- **bit 12 - CCSRA_ESUMSENA**
  - Enable(=1) or disable(=0) the recording of raw energy sums and baseline values in the list mode run for this channel.
  - There are a total of three raw energy sums and one baseline value for each event.
  - **Please note the baseline value is stored in the format of 32-bit IEEE float point(IEEE 754)**
- **bit 13 - CCSRA_CHANTRIG**
  - Enable(=1) or disable(=0) the requirement of channel validation trigger for this channel.
  - If enabled, only when channel validation trigger overlaps the channel fast trigger will the events be recorded for this channel
- **bit 14 - CCSRA_ENARELAY**
  - Switch between two attenuations or gains for the input signal in this channel through an input relay(=1: close the input relay resulting in no input signal attenuation; =0: open the input relay resulting in a 1⁄4 input signal attenuation)
- **bit 15/16 - CCSRA_PILEUPCTRL/CCSRA_INVERSEPILEUP**
  - Control normal pileup rejection(bit 15) and inverse pileup rejection(bit 16) for list mode runs:
  - 1) Bits [16:15] = 00, record all events
  - 2) Bits [16:15] = 01, only record single events, i.e., reject piled up events
  - 3) Bits [16:15] = 10, record everything for piled up events, but will not record trace for single events even if trace recording is enabled, i.e., only record event header
  - 4) Bits [16:15] = 11, only record piled up events, i.e., reject single events

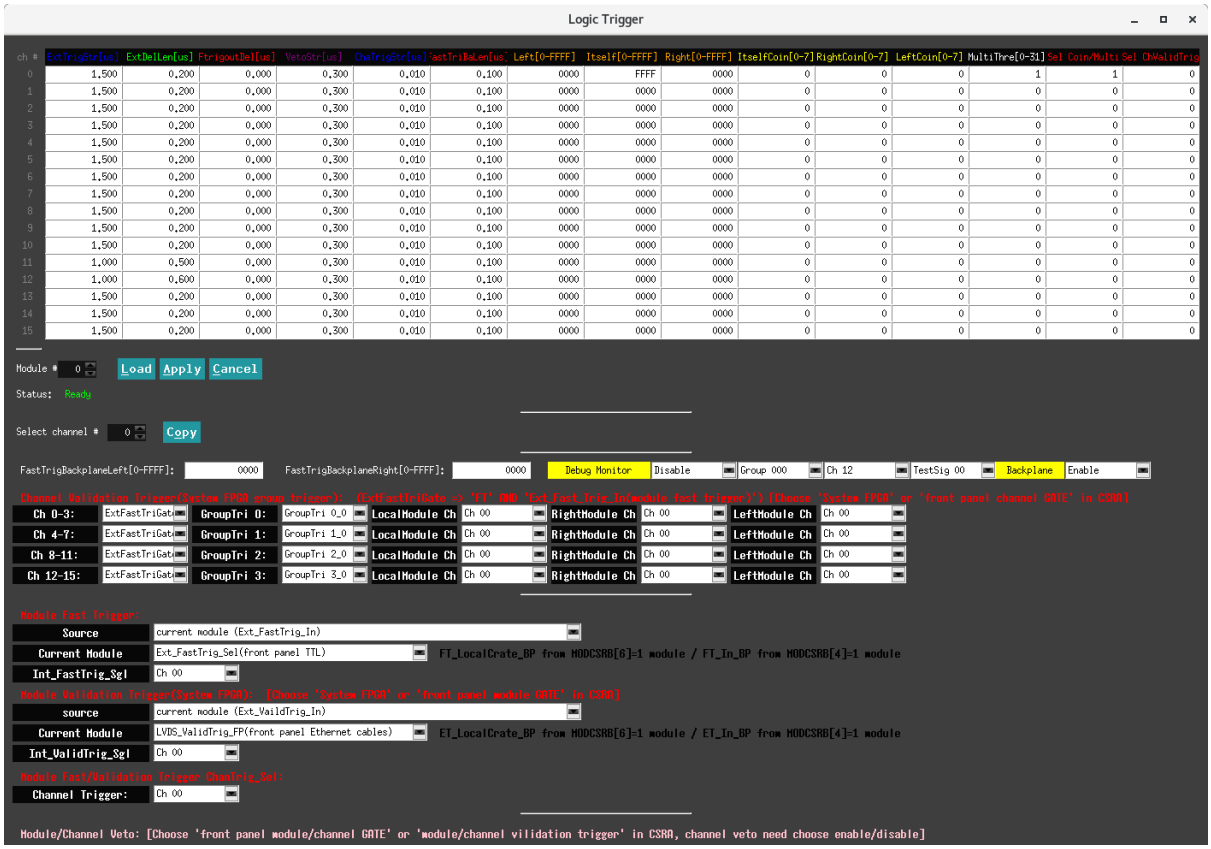
  - In all cases, if the event is piled up, no energy will be computed for such event

- **bit 17 - CCSRA_ENAENERGYCUT**
  - Enable(=1) or disable(=0) the “no traces for large pulses” feature.
  - If enabled, trace will be not be recorded if the event energy is larger than the value set in DSP parameter EnergyLow
- **bit 18 - CCSRA_GROUPTRIGSEL**
  - Select channel fast trigger – this bit works together with the CCSRA_FTRIGSEL bit(bit 0): if CCSRA_FTRIGSEL=1, this CCSRA_GROUPTRIGSEL bit has no effect; if CCSRA_FTRIGSEL=0, then if CCSRA_GROUPTRIGSEL=1, select the channel validation trigger from the System FPGA, and if CCSRA_GROUPTRIGSEL=0, select this channel’s local fast trigger
- **bit 19 - CCSRA_CHANVETOSEL**
  - Channel veto signal selection(=1: channel validation trigger from the System FPGA; =0: channel gate input from the Pixie-16 front panel Channel Gate LVDS connector)
- **bit 20 - CCSRA_MODVETOSEL**
  - Module veto signal selection(=1: module validation trigger from the System FPGA; =0: module gate input from the Pixie-16 front panel Module Gate LVDS connector)
- **bit 21 - CCSRA_EXTTSENA**
  - Enable(=1) or disable(=0) the recording of the 48-bit external clock timestamp in the event header during list mode run for this channel

---

7.4.3 Logic Set

Interface



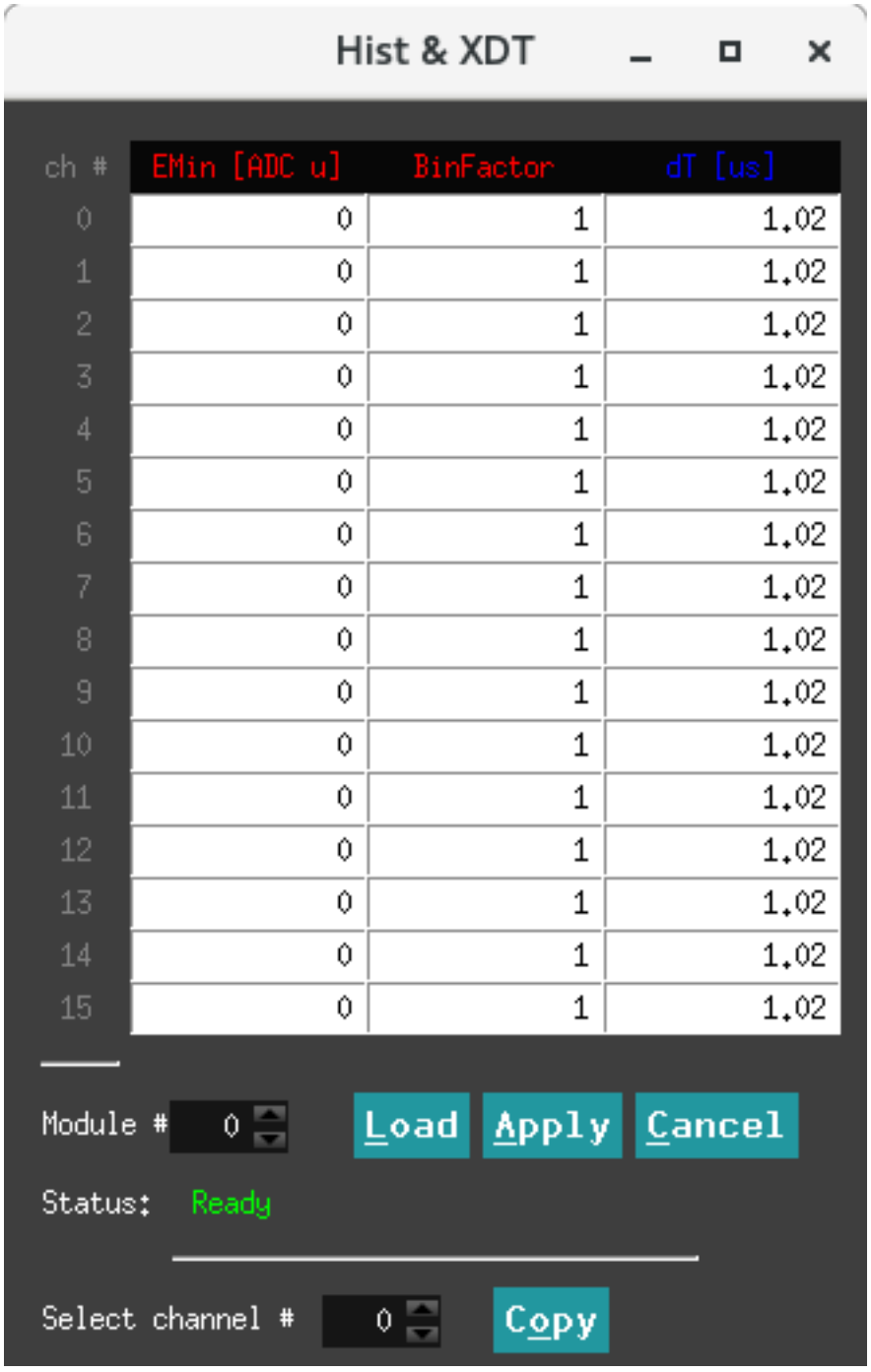
			Range[us]
1	Fast trigger stretch length	FT门宽	0.01-40.95
2	Fast trigger delay length	FT延迟	0-5.11
3	Extern delay	采集信号延迟	0-5.11
4	External trigger stretch length		0.01-40.95
5	Channel trigger stretch length		0.01-40.95
6	Veto stretch length		0.01-40.95

7.5 Debug

The adjustment in this drop-down column is to monitor the waveform noise level, baseline distribution, and so on.

7.5.1 Hist & XDT

Interface



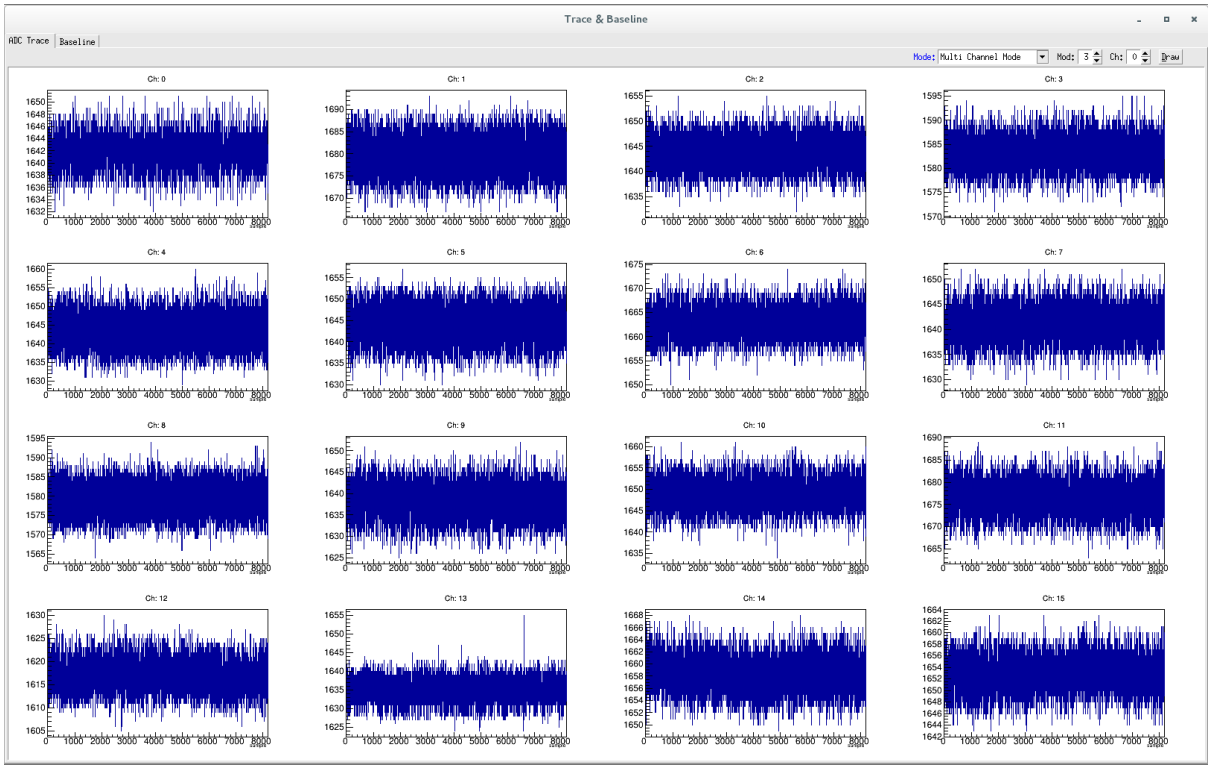
The binning factor controls the number of MCA bins in the spectrum. Energies are computed as 16 bit numbers, allowing in principle 64K MCA bins.

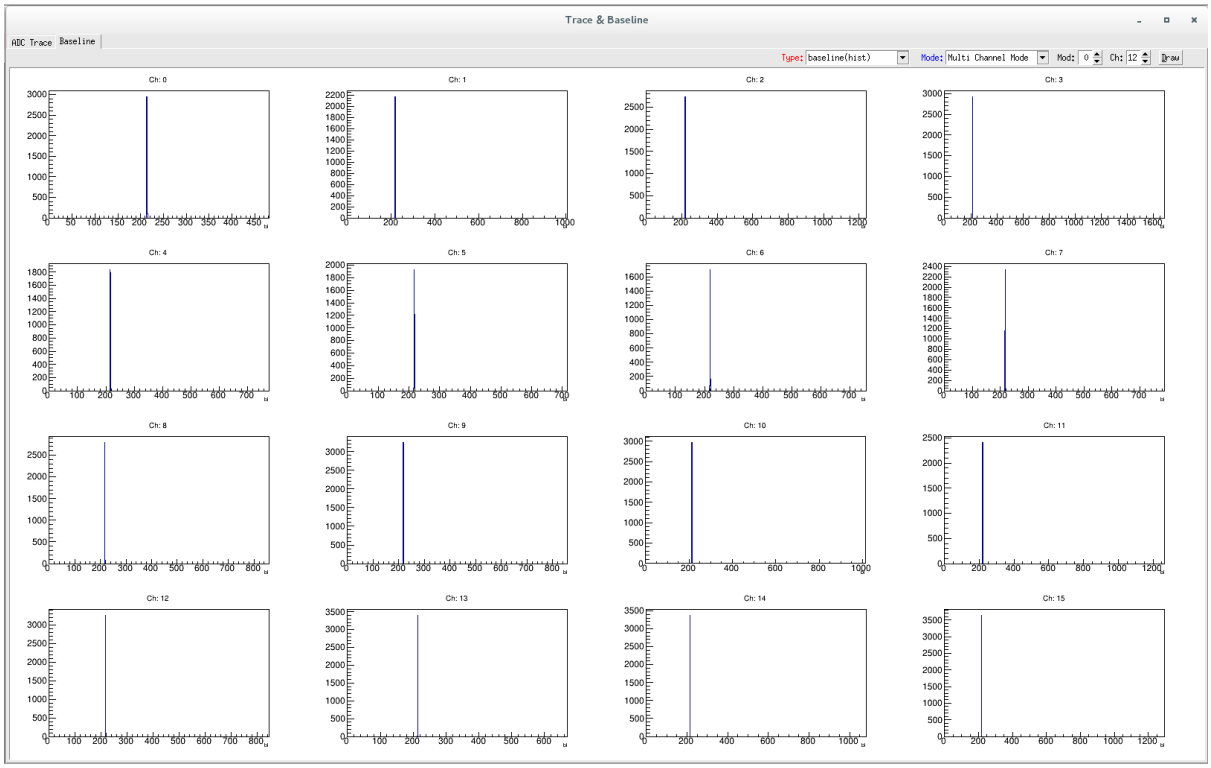
However, spectrum memory for each channel is limited to 32K bins, so computed energy values are divided by $2^{binningfactor}$ before building the histogram. Binning Factor is usually set to 1, but for low count rates and wide peaks, it might be useful to set it to a larger value to obtain a spectrum with fewer bins, but more counts per bin.

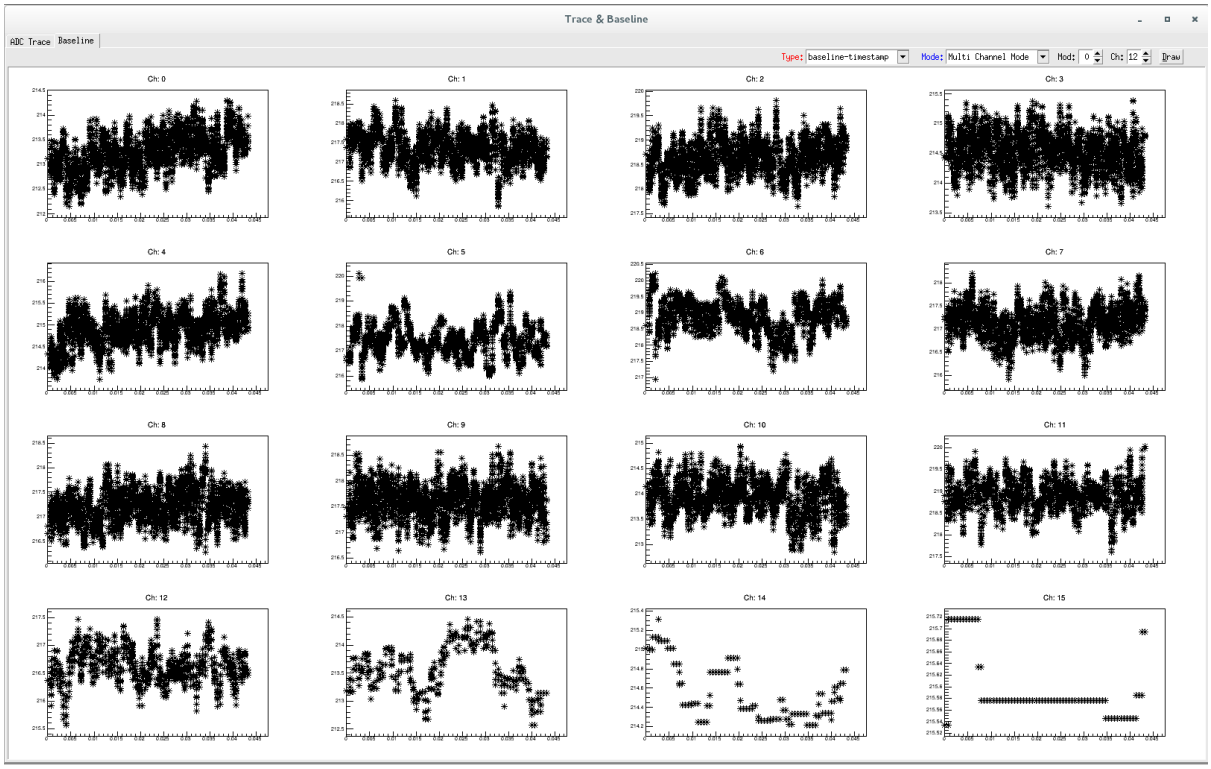
E_{min} is reserved for a future function to subtract a constant "minimum energy" from the computed energy value before binning to essentially cut off the lower end of the spectrum.

7.5.2 Trace & Baseline

Interface





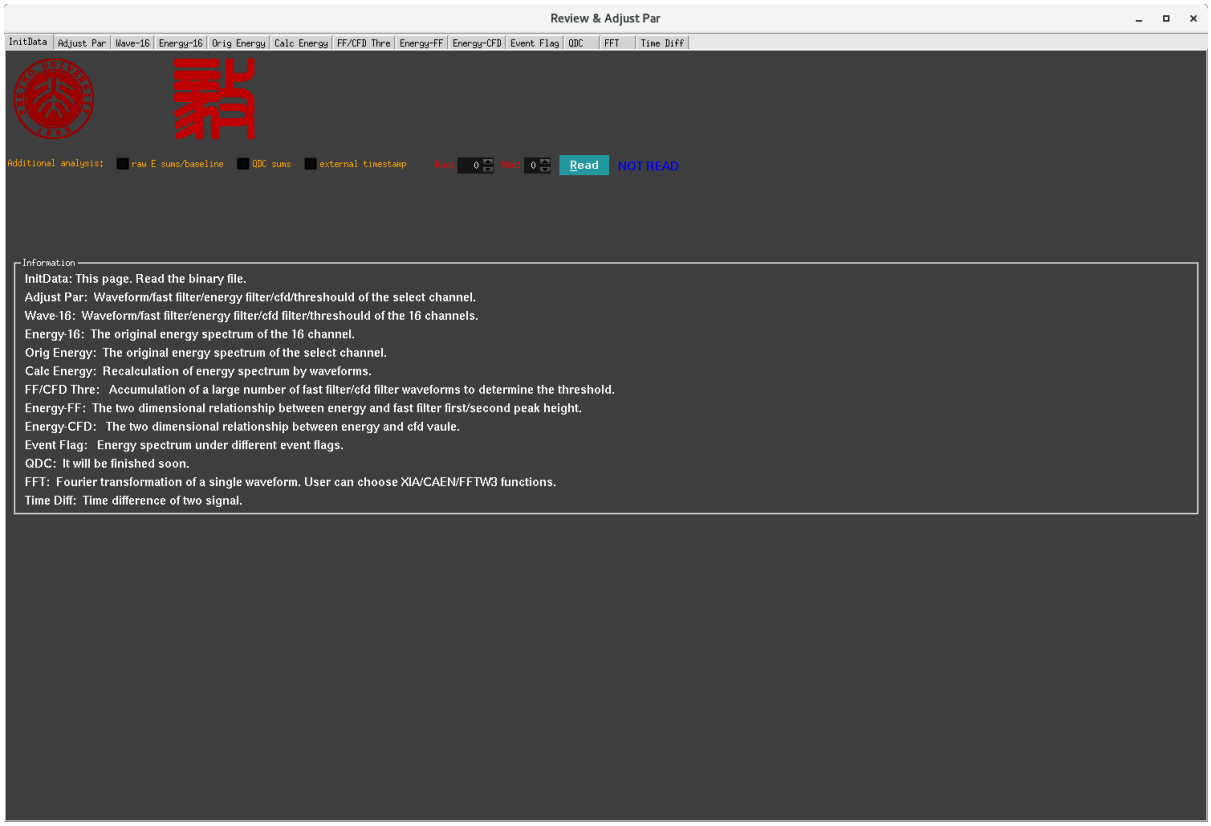


。。**TODO**。。

7.6 Offiline

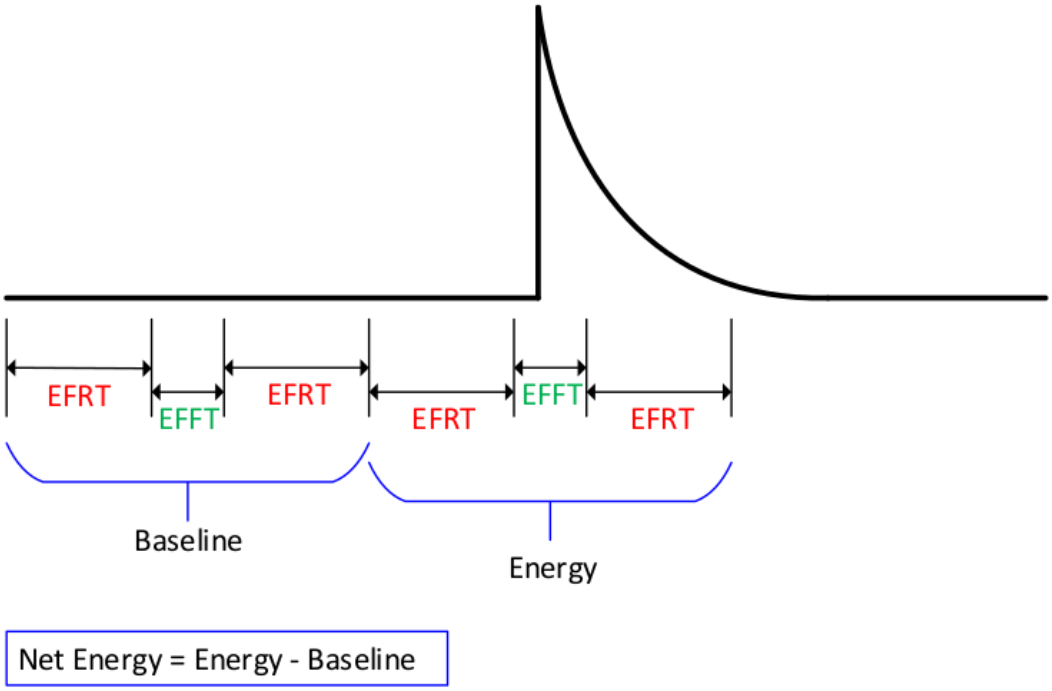
The contents in this drop-down column are the offline parameter optimization adjustment.

7.6.1 InitData



- *Run*: Select the file run number to be read, *Mod*: selects to read which module, and the button *Read* loads the file main information (channel address, energy, waveform position, etc.) into the memory.
- Additional analysis: Among the three options, the selection indicates that the information is included when reading the file data into memory. Some analysis methods can only be enabled if this data is read. But the premise is that this information needs to be recorded during data collection.
- InitData: This page. Read the binary file.
- Adjust Par: Waveform/fast filter/energy filter/cfd/threshould of the select channel.
- Wave-16: Waveform/fast filter/energy filter/cfd filter/threshould of the 16 channels.
- Energy-16: The original energy spectrum of the 16 channel..
- Orig Energy: The original energy spectrum of the select channel.
- Calc Energy: Recalculation of energy spectrum by waveforms.
- FF/CFD Thre: Accumulation of a large number of fast filter/cfd filter waveforms to determine the threshold.
- Energy-FF: The two dimensional relationship between energy and fast filter first/second peak height.
- QDC: It will be finished soon.
- FFT: Fourier transformation of a single waveform. User can choose XIA/CAEN/FFTW3 functions.
- Time Diff: Time difference of two signal.

7.6.2 Adjust Par



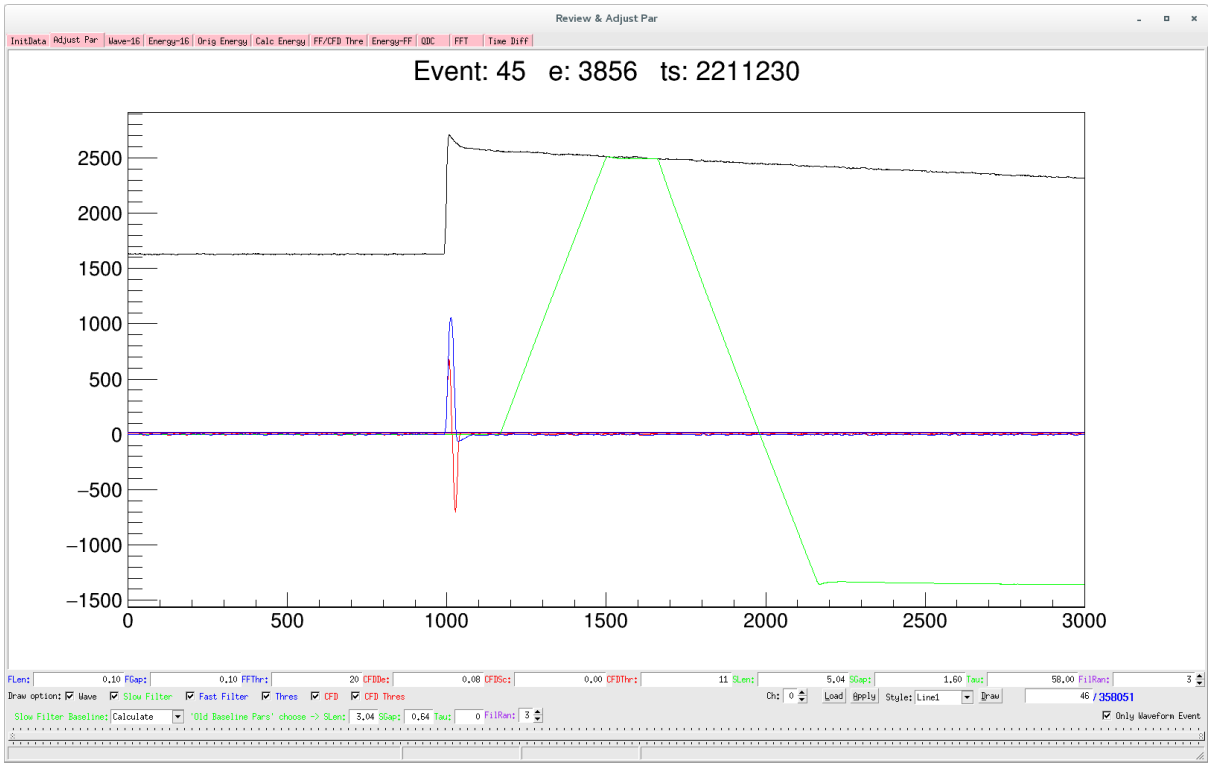
To calculate the fast filter and slow filter cfd curves offline from the acquired waveform, the following requirements are imposed on the acquired waveform. In the above figure, the calculated energy is the difference between the energy of the algorithm and the baseline of the algorithm. To get the correct trapezoid, the premise is that there is a long enough point in front to calculate the baseline.

In the figure, EFRT stands for Energy Filter Rise Time and EFFT stands for Energy Filter Flat Top.

To compute energy filter response offline, the ideal settings are:

- Total trance length > 2 x (2 x EFRT + EFFT)
- Pre-trigger trace (Trace-delay) length > (3 x EFRT + EFFT)

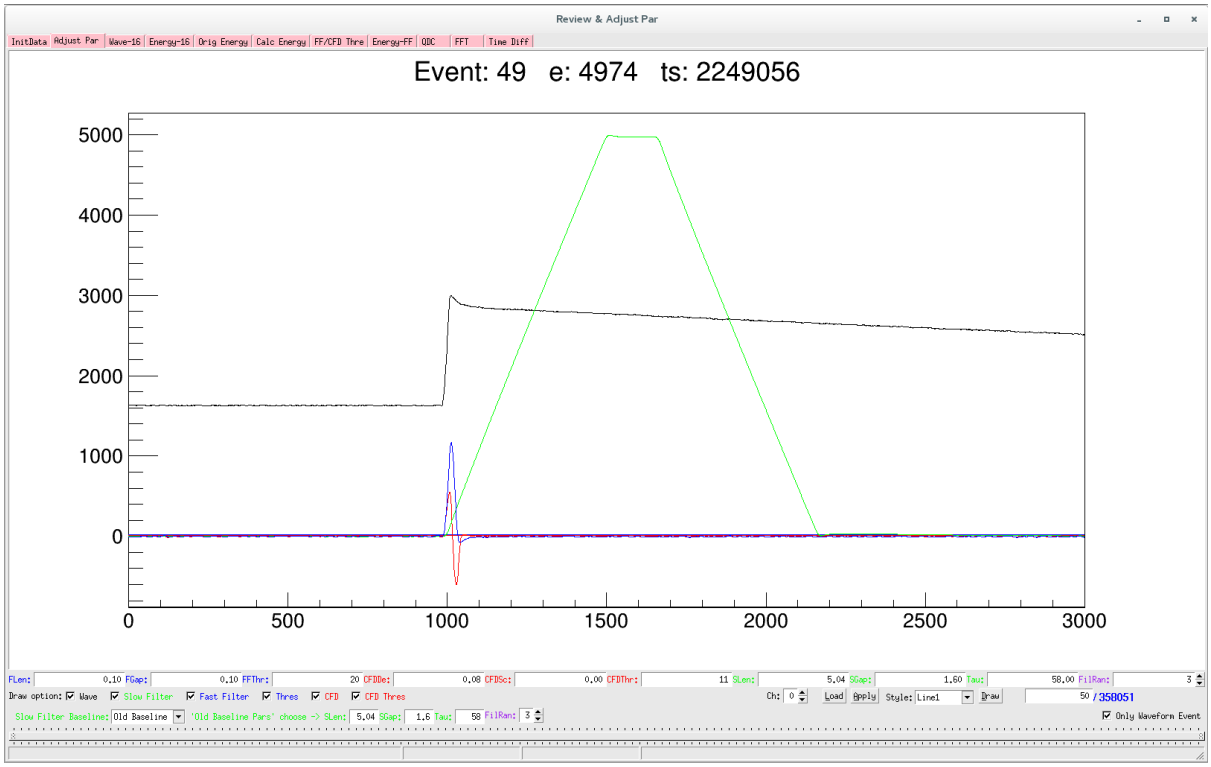
Of course, this is just one way to calculate the trapezoid. If we record the baseline of the energy trapezoid for each event and use the average of the pre-trigger partial points as the infinite extension to the left of the waveform, then this is not limited by this condition Pre-trigger trace length > (3 x EFRT + EFFT) . In the following pages, when using the Old Baseline method to calculate the energy trapezoid, there is a premise that the pre-trigger trace length requires at least 200 points because the left side of the waveform is averaged by the first 200 points.



When the acquired waveform pre-trigger trace length > 3 x EFRT + EFFT, the pre-trigger trace provides enough points to calculate the baseline, and the SF BL algorithm can choose Calculate, otherwise the Old Baseline algorithm needs to be selected. When recording data with the premise of the Old Baseline algorithm, select to turn on the baseline for the trapezoid and the raw E sums/baseline option for the InitData page. When the Old Baseline algorithm is selected, the next four option parameters take effect, which are the parameters of the energy trapezoid used in the data acquisition.

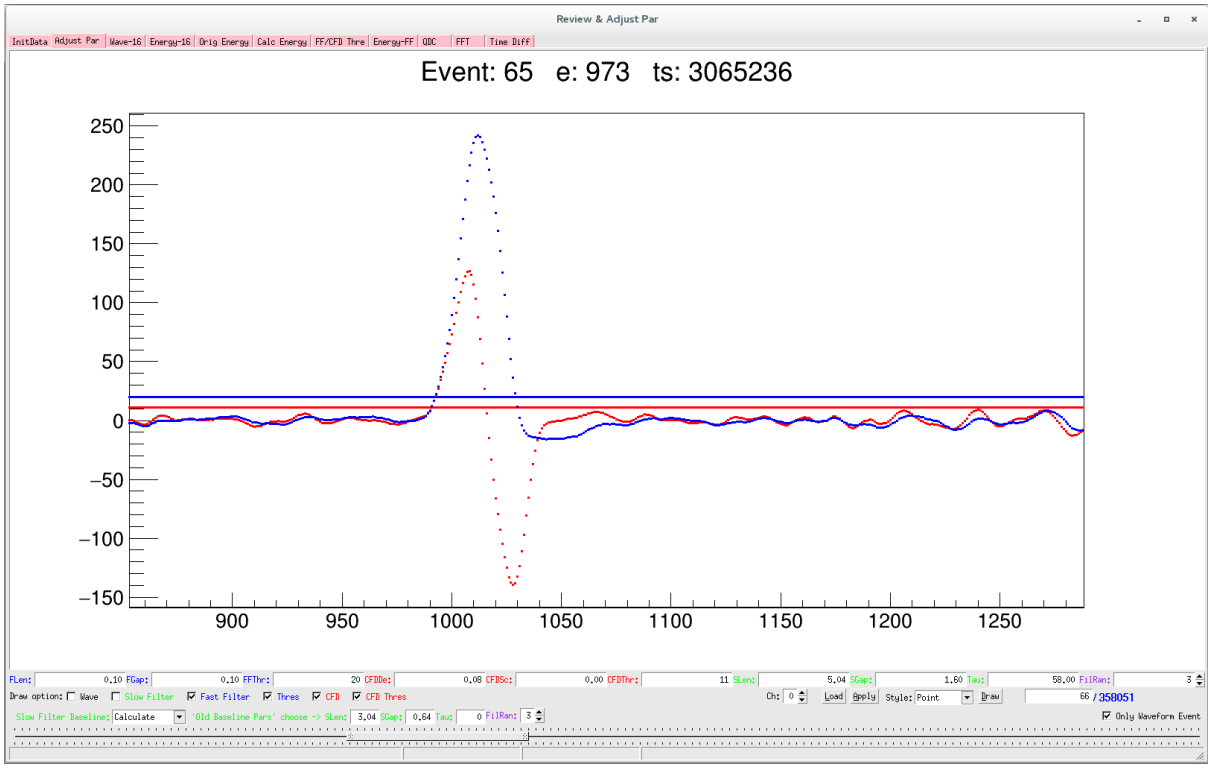
The green curve in the above figure is a typical condition that does not satisfy the pre-trigger trace length > 3 x EFRT + EFFT when the Calculate algorithm is used. The figure shows a pre-trigger trace length of 10 us, an EFRT of 5.04 us, and an EFFT of 1.60 us.

In this case, you should use the Old Baseline algorithm shown below.



The user can choose to view the channel of the waveform. The button *Load* can read and display the current parameter setting. When modifying the above parameters, you need to press the *Apply* button to make it effective. The button *Draw* is used to display the event waveform of the next channel.

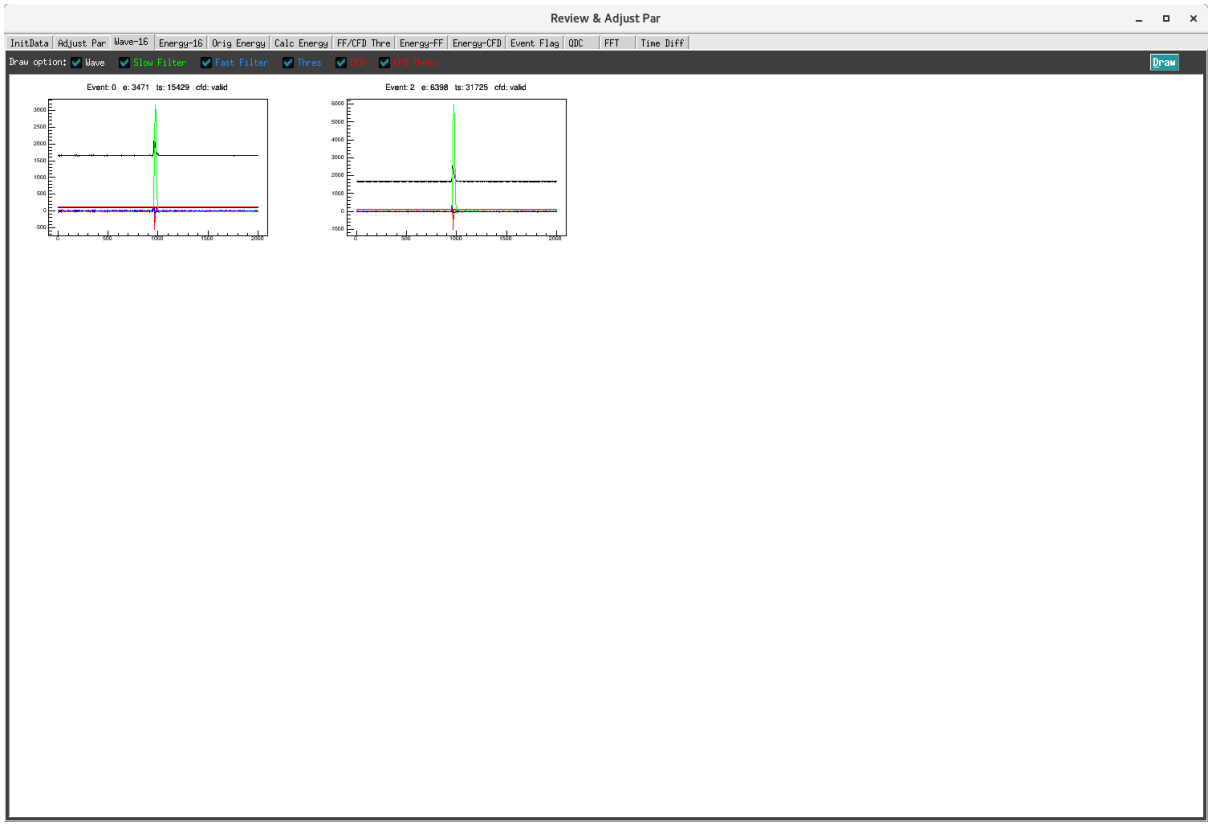
Users can choose to display multiple waveforms in **Wave / Slow Filter / Fast filter / Thres / CFD / CFD Thres** or select the drawing style of the curve.



The figure above shows four waveforms， fast filter, Thres, CFD, and CFD Thres. The pattern displays in dots. The lowest level horizontal bar can be dragged at both ends, and the user can pull to control the display range of the waveform abscissa, as shown in the figure of 800 - 1300 points. In this case, clicking the Draw button will hold the

specified coordinate range.

7.6.3 Wave-16

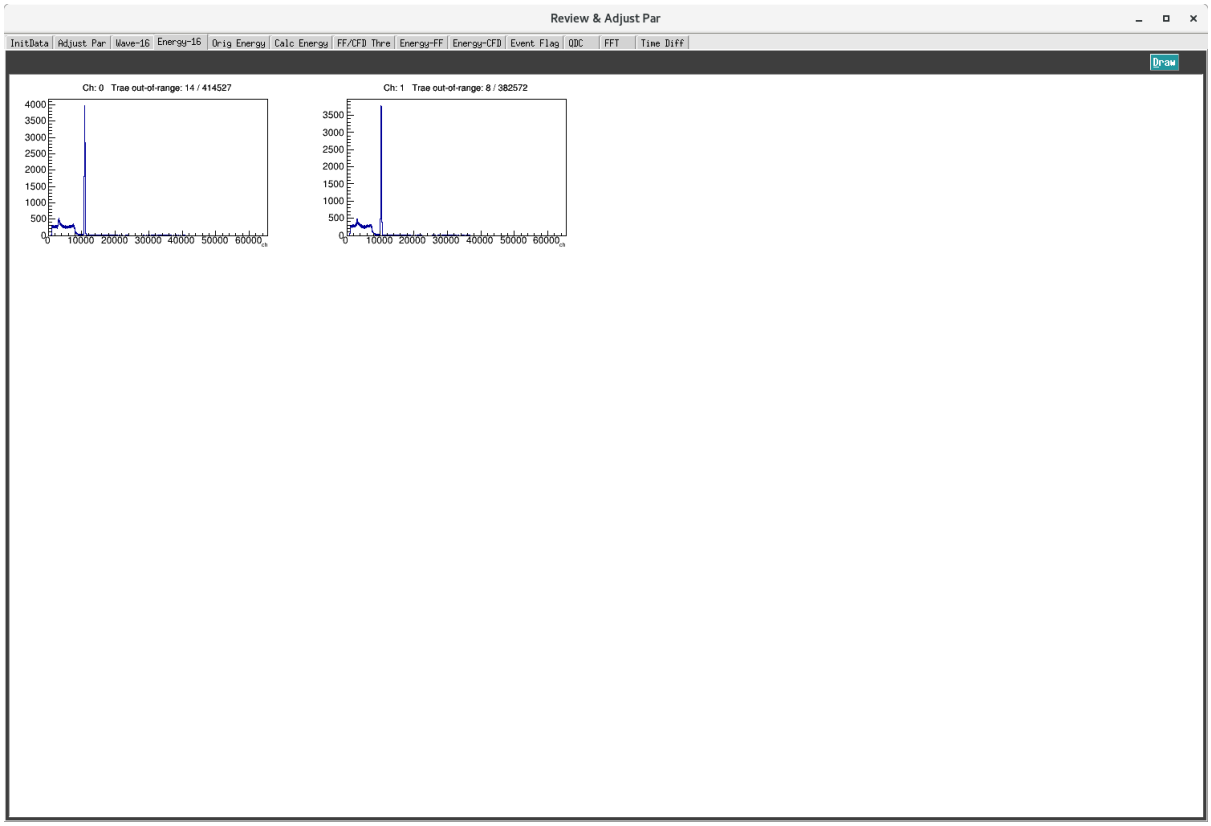


This page is used to view 16 channels of raw waveforms, filter waveforms, thresholds, and so on. Users can choose to display multiple waveforms in **Wave/Slow Filter/Fast filter/Thres/CFD/CFD Thres** simultaneously.

Through this page, the user can quickly check whether the waveforms of all channels of the module are normal and whether the parameter settings are reasonable. Click the button **Draw** once to display the next waveform for all channels.

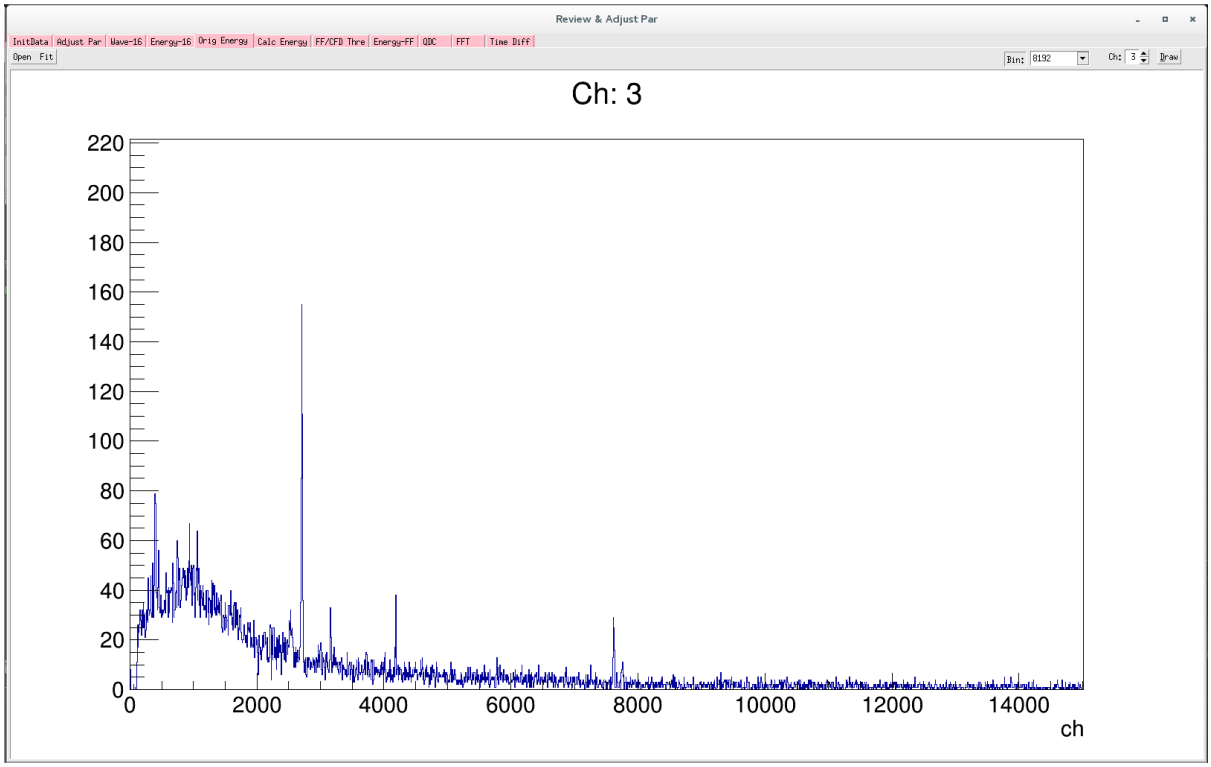
It should be noted that when the acquired waveform pre-trigger trace length is greater than 3 x EFRT + EFFT, the Slow Filter waveform on this page is correct.

7.6.4 Energy-16

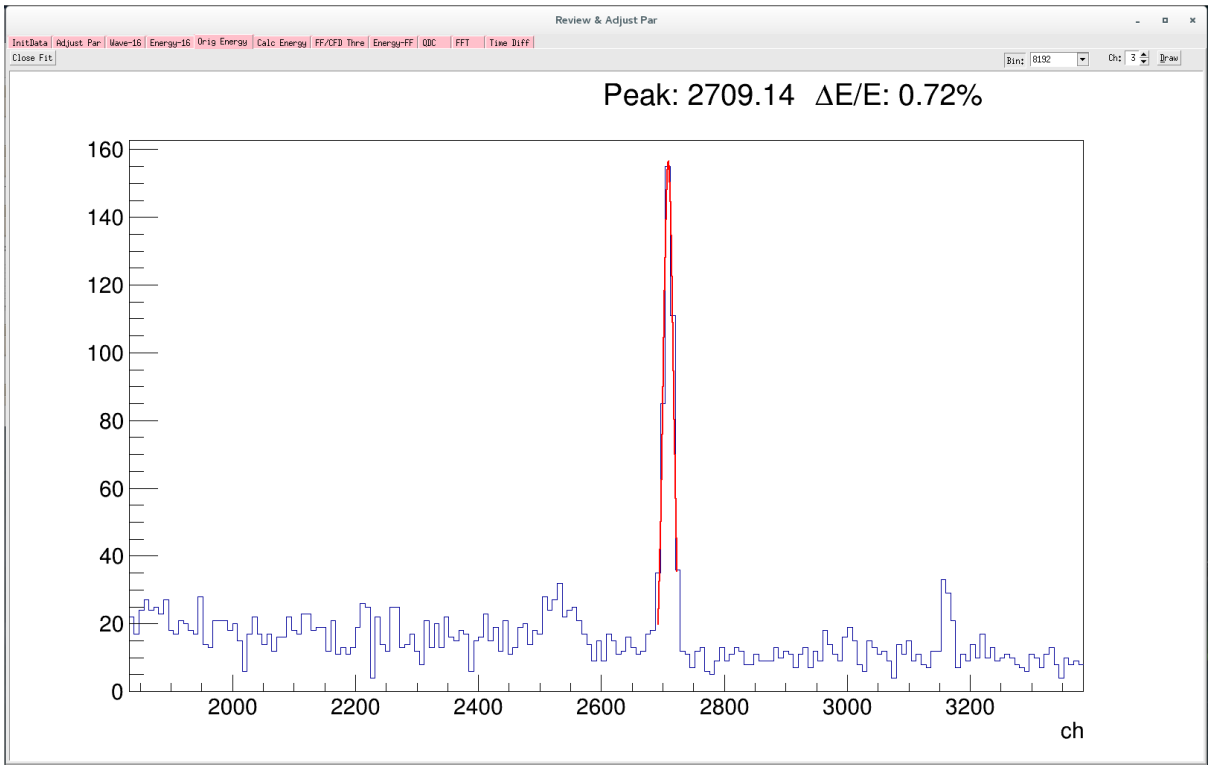


This interface is used to view the 1D spectrum of 16 channels simultaneously. Click the button *Draw* in the top right corner.

7.6.5 Orig Energy



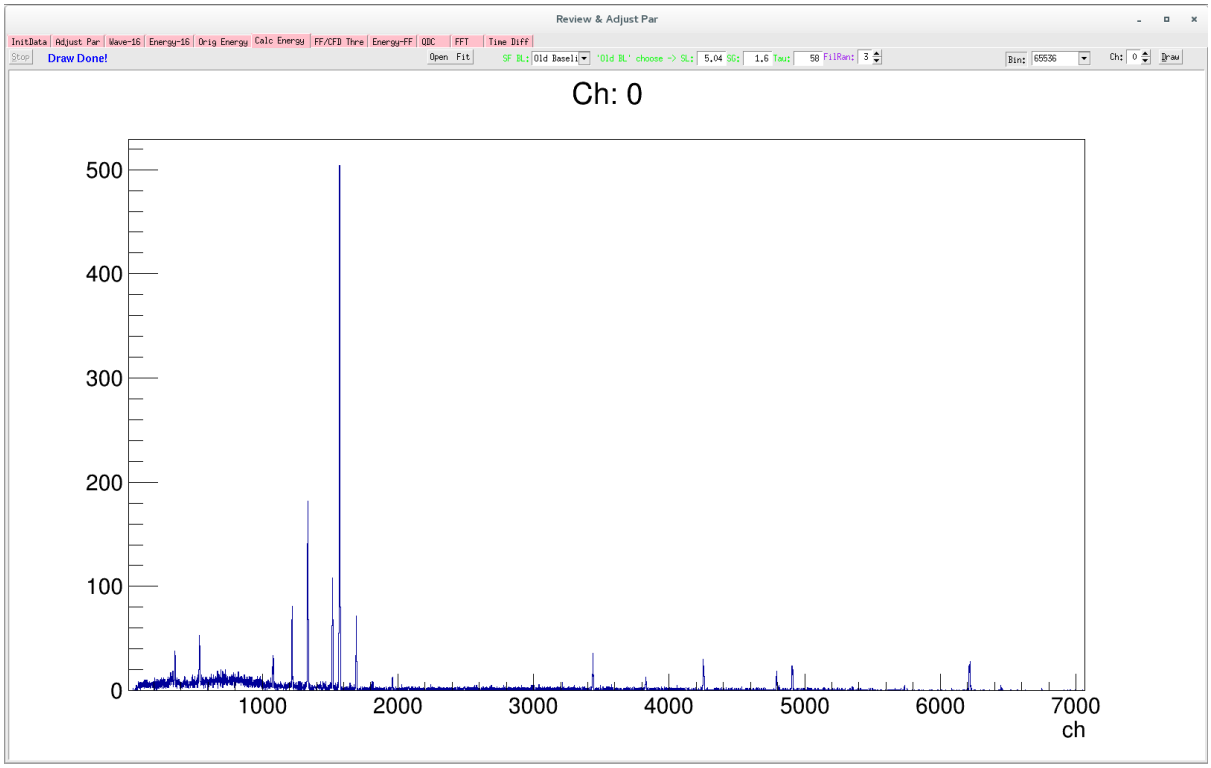
This page is used to quickly view the energy spectrum of a channel. The user selects the number of bins of the spectrum, which represents how many points we can dididе 0-65536 into. Select to view the channel and then press the *Draw* button.



The *Open Fit* button in the upper left corner is used for fast Gaussian fitting to see energy resolution. Click the button to turn on the fitting mode, and click the button again to turn it off. Move the mouse to the blue line of the histogram and the mouse cross will become a triangular arrow. The mouse of the triangle arrow clicks on two positions in the

histogram, and the interval between the two points is the fitting interval, then the energy resolution can be viewed.

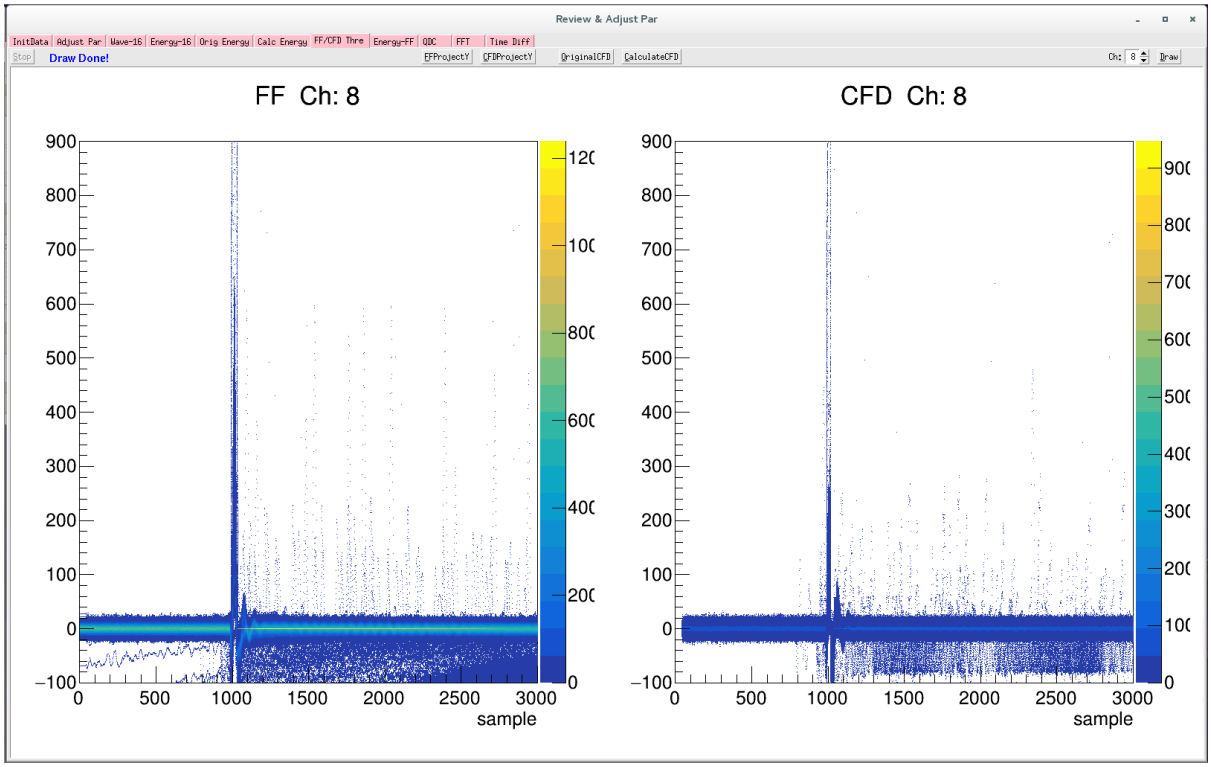
7.6.6 Calc Energy



This page recalculates the energy using the acquired waveform. Like the **Adjust Par** page, the SF BL algorithm can choose either the Calculate algorithm or the Old Baseline algorithm.

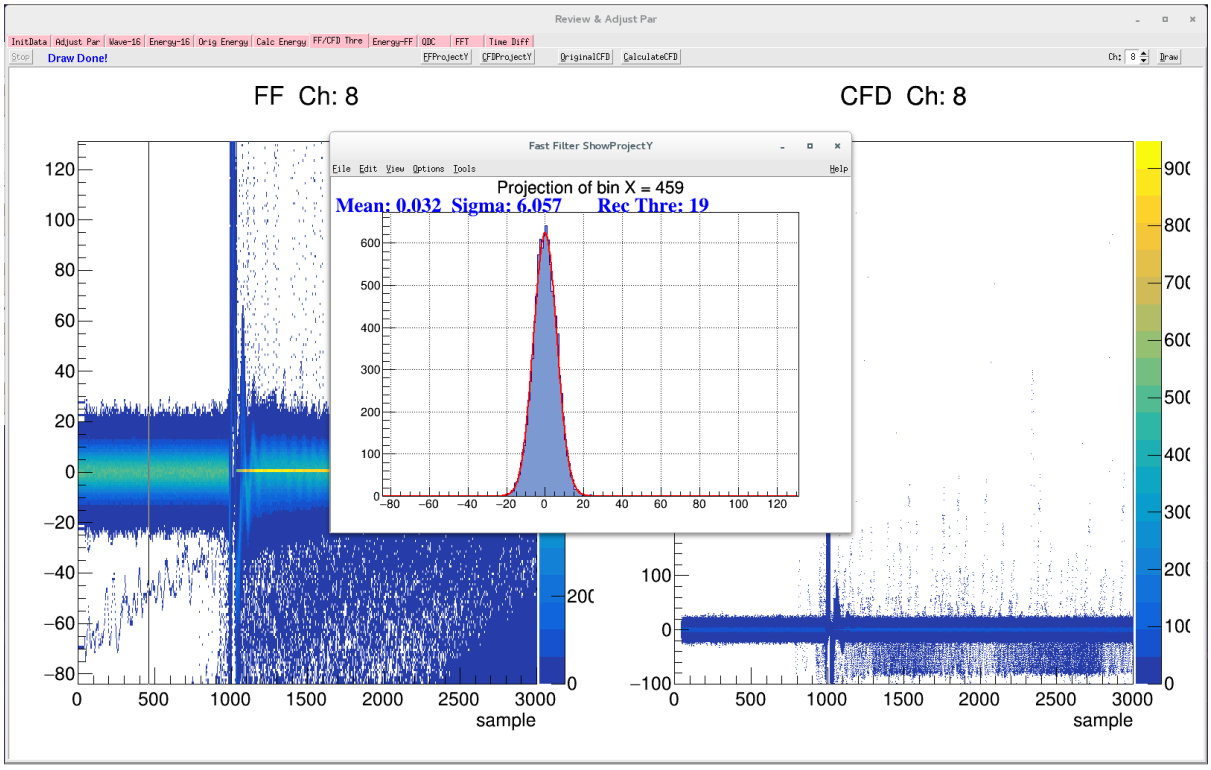
The fast filter and energy filter parameters used to calculate the energy adopt the setting parameters of the module. The user needs to select the number of bins that the energy about 65536 channels are divided into. You can select 1024/2048/4096/8192/16384/32768/65536. Select the calculated channel and then press the button *Draw* to start the calculation, the upper left corner will display the progress of the calculation, or press the button *Stop* to terminate the calculation early. When the calculation is terminated, the spectrum will be displayed on the artboard.

7.6.7 FF/CFD Thre



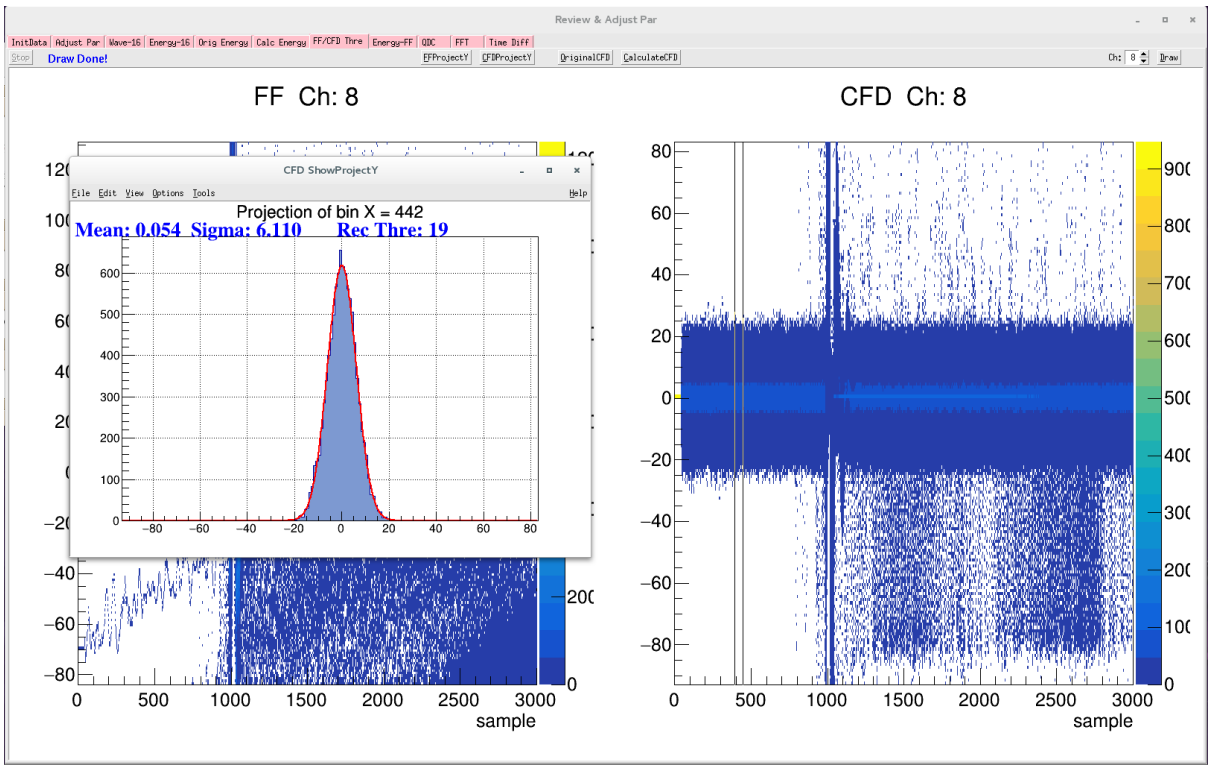
This interface is used for the accumulation of fast filter waveforms and cfd filter waveforms. The user selects the channel to view and presses the *Draw* button to start the calculation. The top left corner of the page can monitor the progress from time to time, or press the *Stop* button to terminate the calculation early. The calculation ends as shown in the figure above.

The upper buttons *FFProjectY*, *CFDProjectY*, *OriginalCFD*, and *CalculateCFD* respectively pop up the sub-painters.

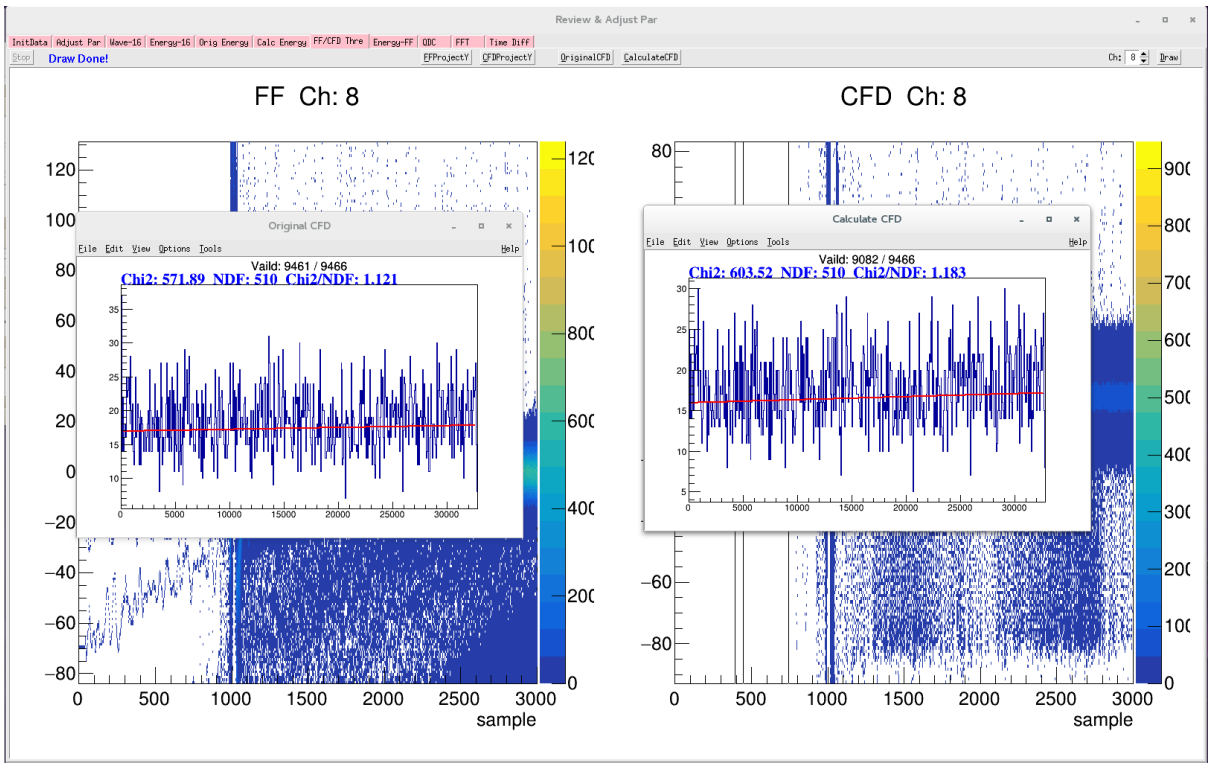


Click the button *FFProjectY* to open the view of the fast filter projection. Click again to close the function. When

the function is turned on, place the mouse on the 2D map and move the mouse left and right. The Fast Filter ShowProjectY sub-panel displays the projection distribution of the position pointed by the mouse. This distribution before triggering also characterizes the level of noise.

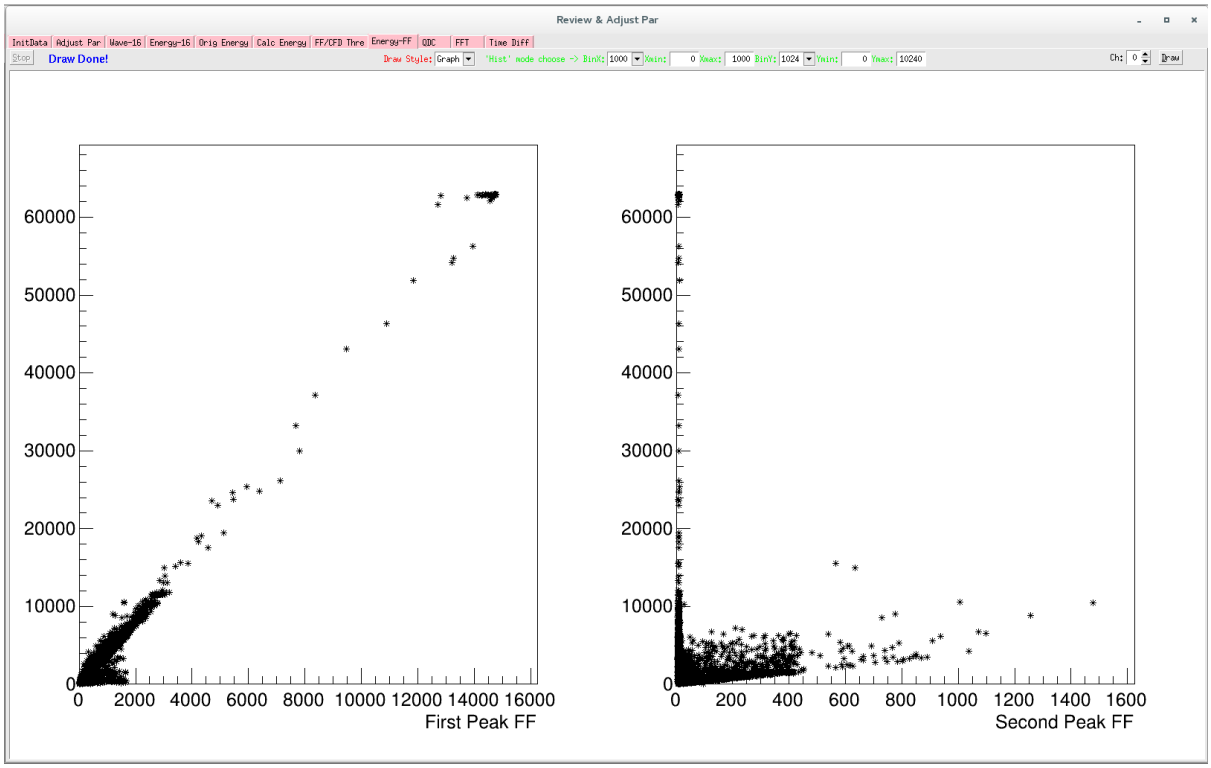


Similarly, the button *CFDProjectY* function is as shown in the figure above.



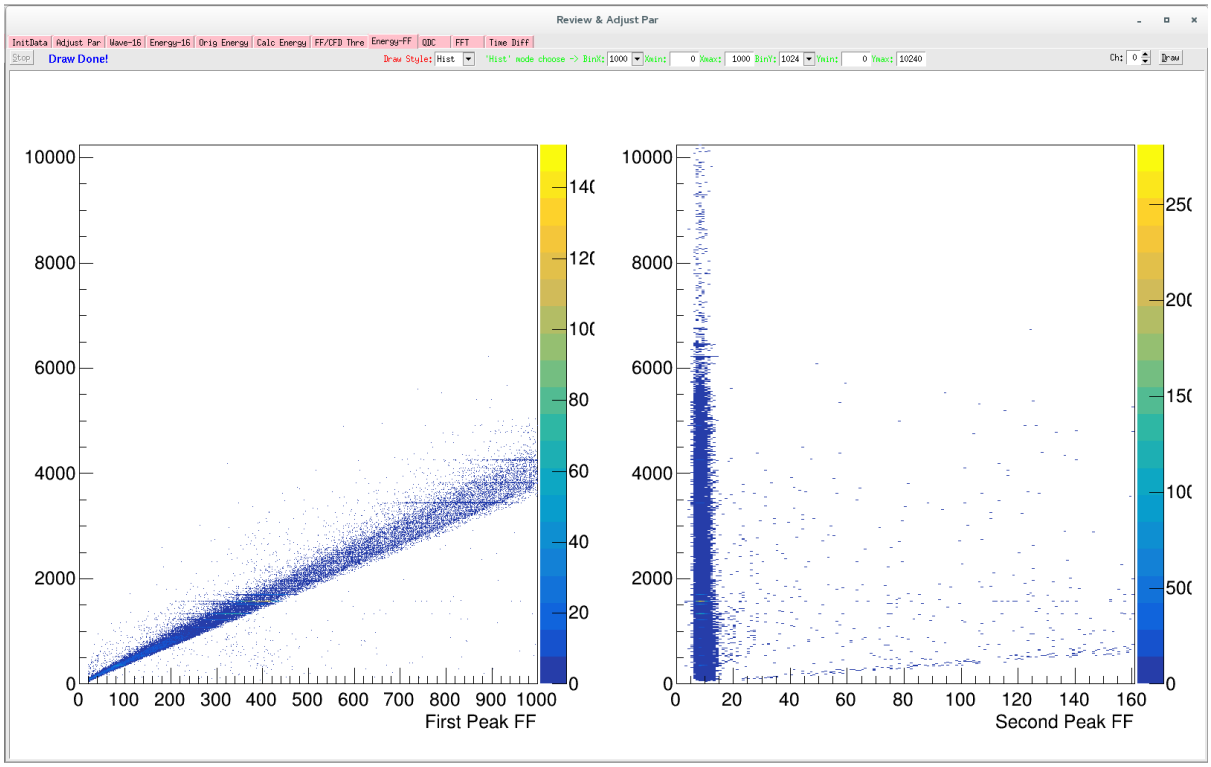
Click the button *OriginalCFD* to display the distribution of CFD values in the raw data on the left. Click the button *CalculateCFD* to display the result of the offline waveform calculation on the right, and the parameters used for the calculation are the current ones. For a suitable CFD parameter setting, the CFD should be evenly distributed.

7.6.8 Energy-FF



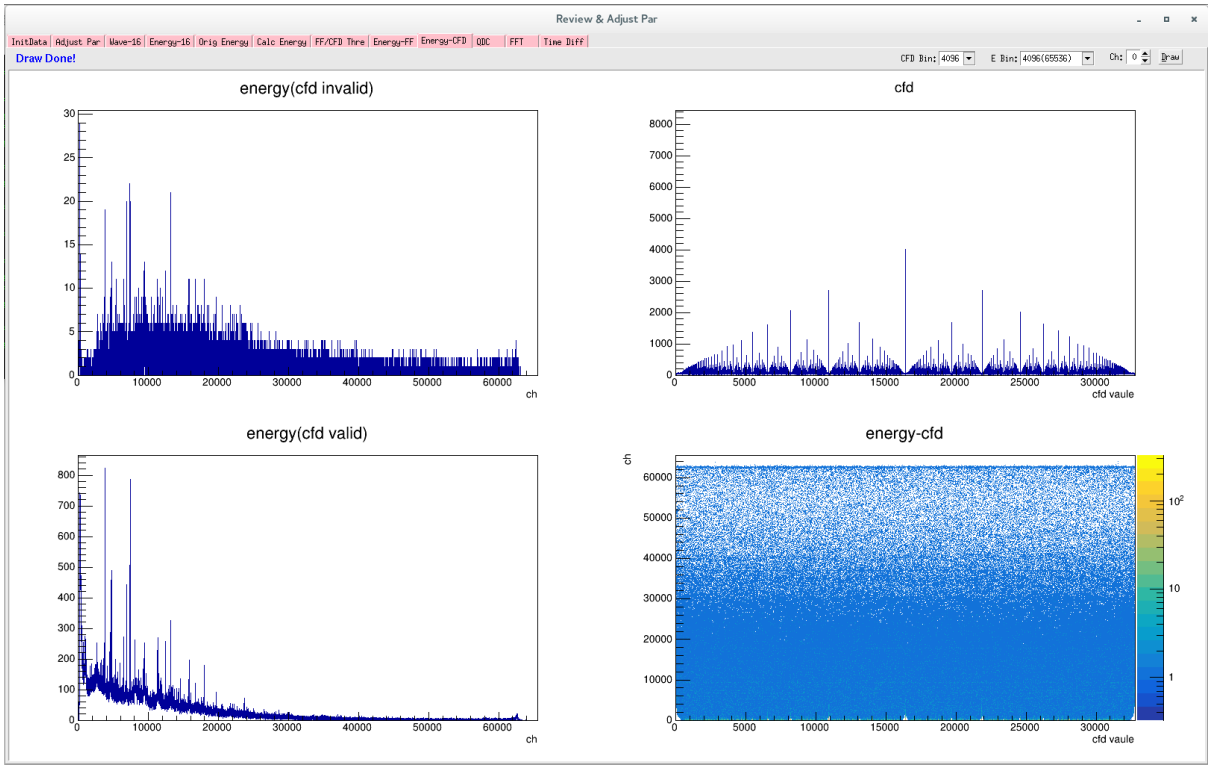
This interface is a two-dimensional map of the energy and fast filter peak heights used to determine the appropriate threshold. The left picture is a two-dimensional correlation between energy and fast filter, which should have a good linear relationship. The right picture shows the two-dimensional correlation between the energy and the remaining maximum value of the part of the fast filer where the trapezoid is thrown away, which characterizes the noise level. And the energy should be uncorrelated with this value.

Firstly, for **Draw Style** we choose **Graph**, which is the 2D scatter plot mode. Select the channel you want to view, then press the *Draw* button to start the calculation, in the top left corner the progress from time to time can be monitored, or you can press the *Stop* button to terminate the calculation early. The calculation ends as shown in the figure above.



The 2D scatter plot does not visually show the density distribution of the displayed data points, so for **Draw Style** we select the **Hist mode**. Select the bins or ranges of the X and Y axis, and then press the *Draw* button to start the calculation. The result is shown in the figure above, and the right figure reflects the level of noise.

7.6.9 Energy-CFD



- The upper left picture shows the energy spectrum when cfd is invalid.
- The lower left picture shows the energy spectrum when cfd is valid.
- The upper right picture shows the CFD spectrum.
- The lower right picture is a two-dimensional map of energy and CFD.

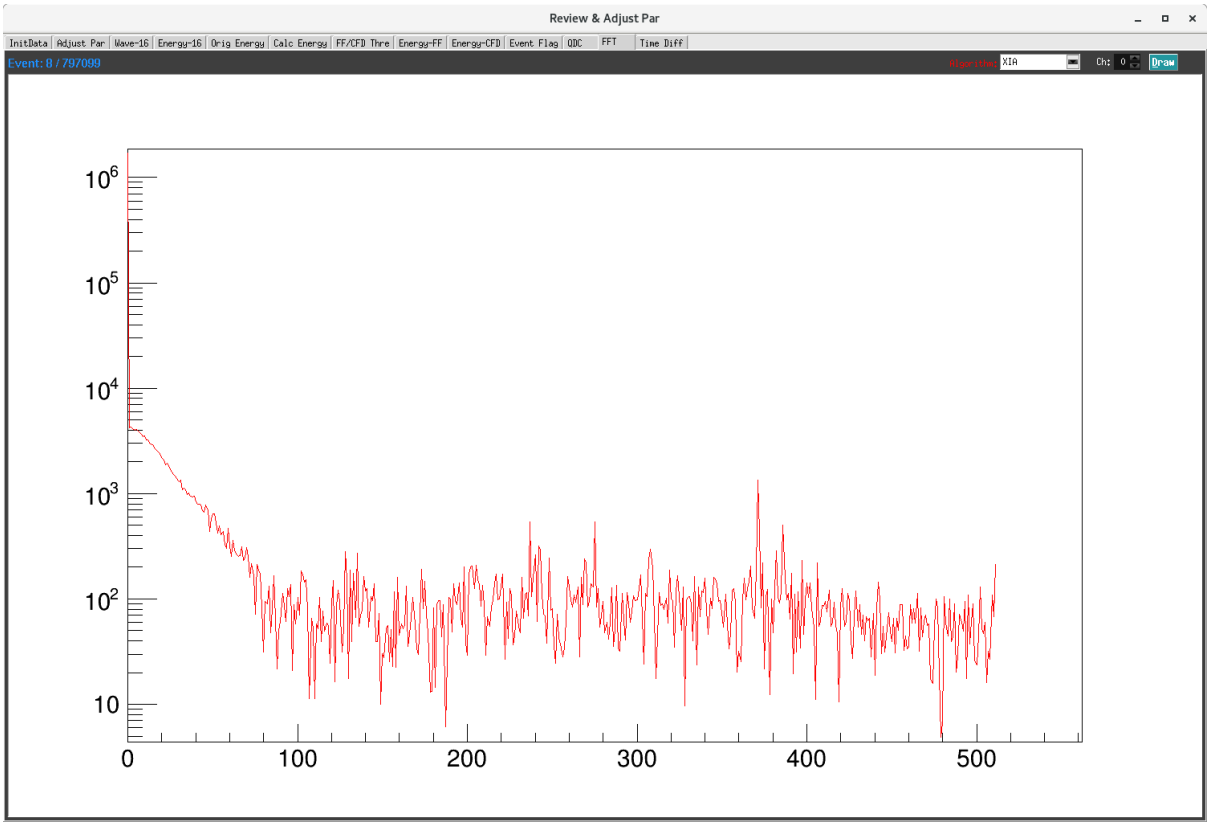
首先选择能量与 CFD 二维关联图中 bin 数。其中 CFD 分 bin 可选择 4096，2084，1024；能量可选择 bin 数与道址范围。之后选择查看通道，然后按 Draw 按钮开始进入计算。

7.6.10 QDC

to do not completed

QDC TODO

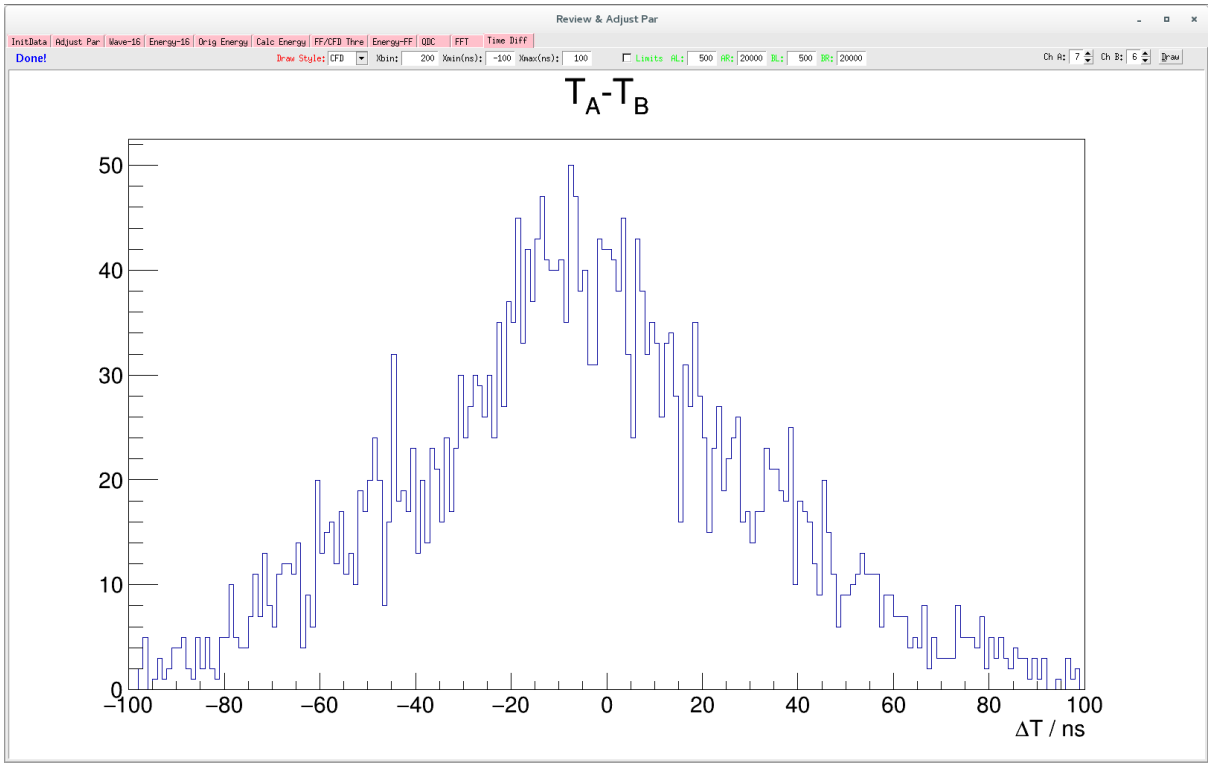
7.6.11 FFT



This interface is used to quickly view the Fourier transform of the waveform. Users can choose different algorithms, such as **XIA, fftw3, CAEN(HANNING), CAEN(HAMMING), CAEN(BLACKMAN), CAEN(RECT)**. Select the channel you want to view. Then press the *Draw* button, and each time the button is clicked, the next result is displayed.

the ADC trace display also includes the option to view a FFT of the acquired trace. This is useful to diagnose noise contributions.

7.6.12 Time Diff



This interface is used to quickly view the time resolution of the two signals. The user can choose to view the time difference between the two signals of the zero crossing of the CFD algorithm or the time difference between two signals of the fast filter over threshold. **Xbin** represents the bin number of the abscissa, **Xmin** represents the minimum value of the abscissa, and **Xmax** represents the maximum value of the abscissa. With Ch A, Ch B we can select the two channels we want to view. Then press the *Draw* button.

The option **Limits** selection turns on the energy range constraint. After selecting this option, the following four parameters, AL, AR, BL, and BR, take effect, which respectively represent the left and right ranges of the Ch A/B energy address, and only events with energy falling in this interval are filled into the histogram. The **Orig Energy** page allows the user to select the appropriate energy channel interval.

7.6.13 Simulation

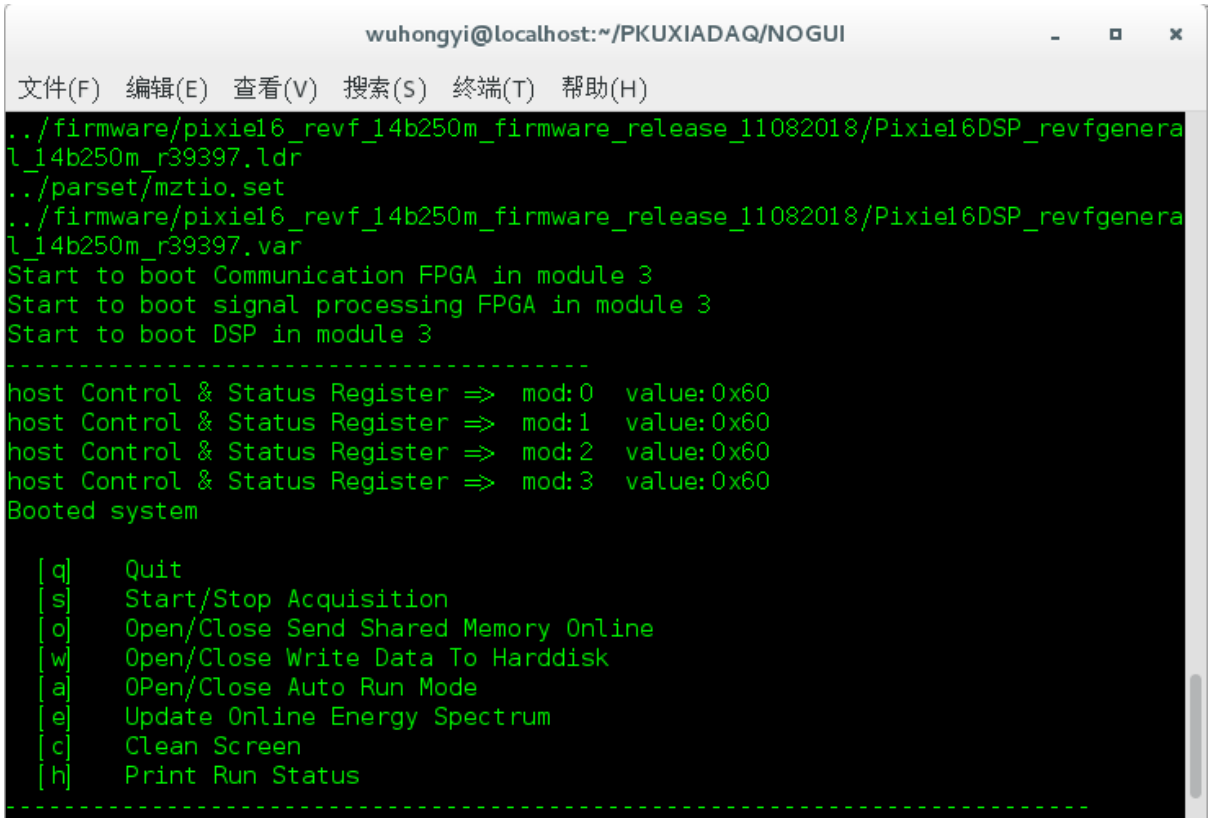
Different types of detection and different signal-to-noise ratio waveforms are generated by the model to assist the user in learning parameter optimization adjustment.

CHAPTER 8

NOGUI

8.1 Interface

When the program is started, the firmware will be automatically loaded for initialization. After the initialization is completed, the following interface appears



The following are the functions of all commands.

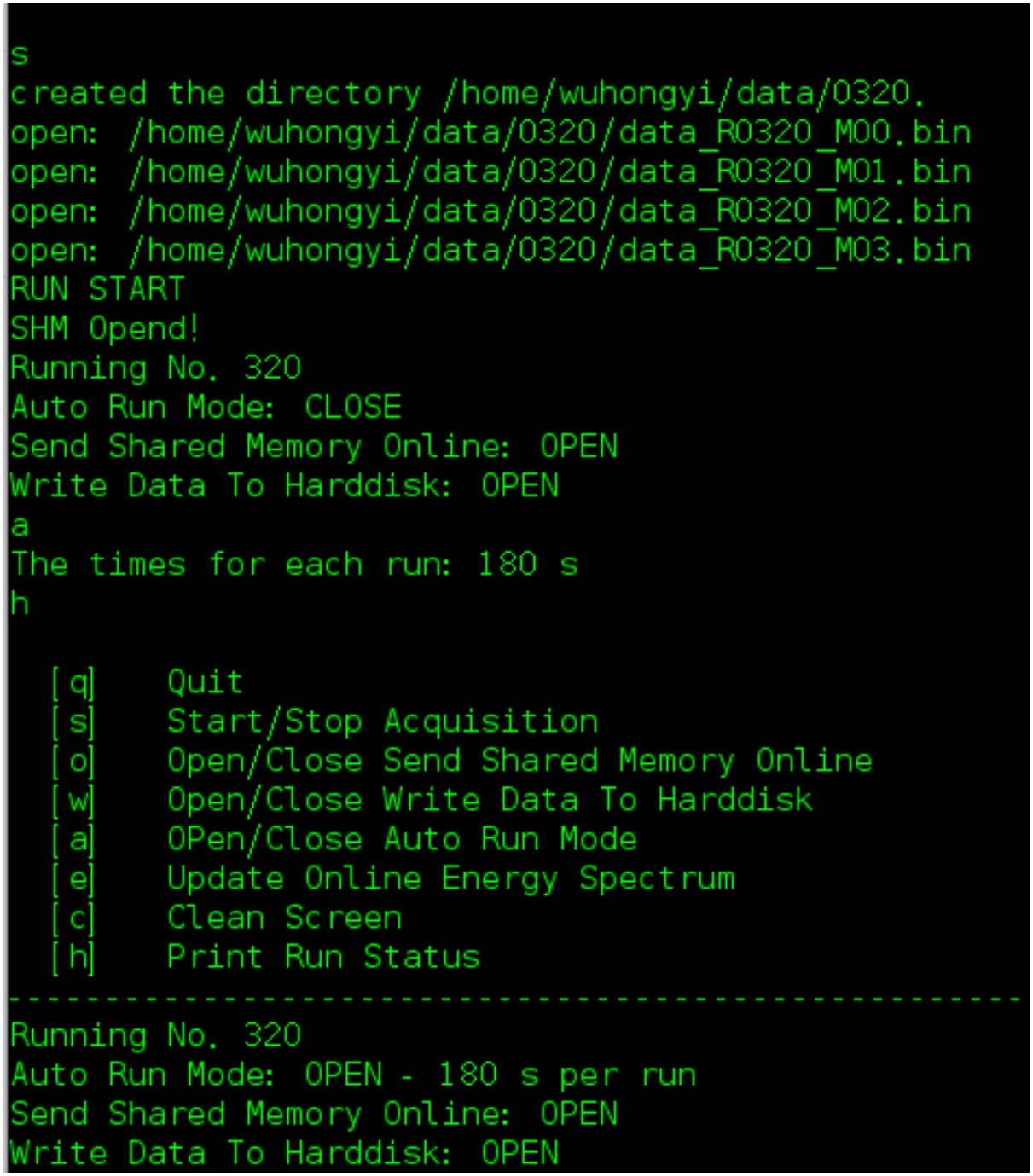
```
[q]   Exit the program
[s]   Controls the start and stop of the acquisition
```

(下页继续)

(续上页)

```
[o]   Controls the opening and closing of online shared memory
[w]   Control data write to the hard disk on and off
[a]   Controls the automatic operation mode on and off
[e]   Refreshing the spectrum of online monitoring
[c]   Clear the screen to display this command prompt
[h]   Output operation control parameters
```

The figure below is a typical running interface information, which can clearly see the current operating mode.



8.2 auto run

When the automatic operation mode is turned on, it will automatically switch to the next round according to the time set by the user.

The time parameter is set in the file parset/cfgPixie16.txt

```
# Only use in NOGUI, unit: second
AutoRunModeTimes   180
```

CHAPTER 9

Online Stattics

9.1 Interface

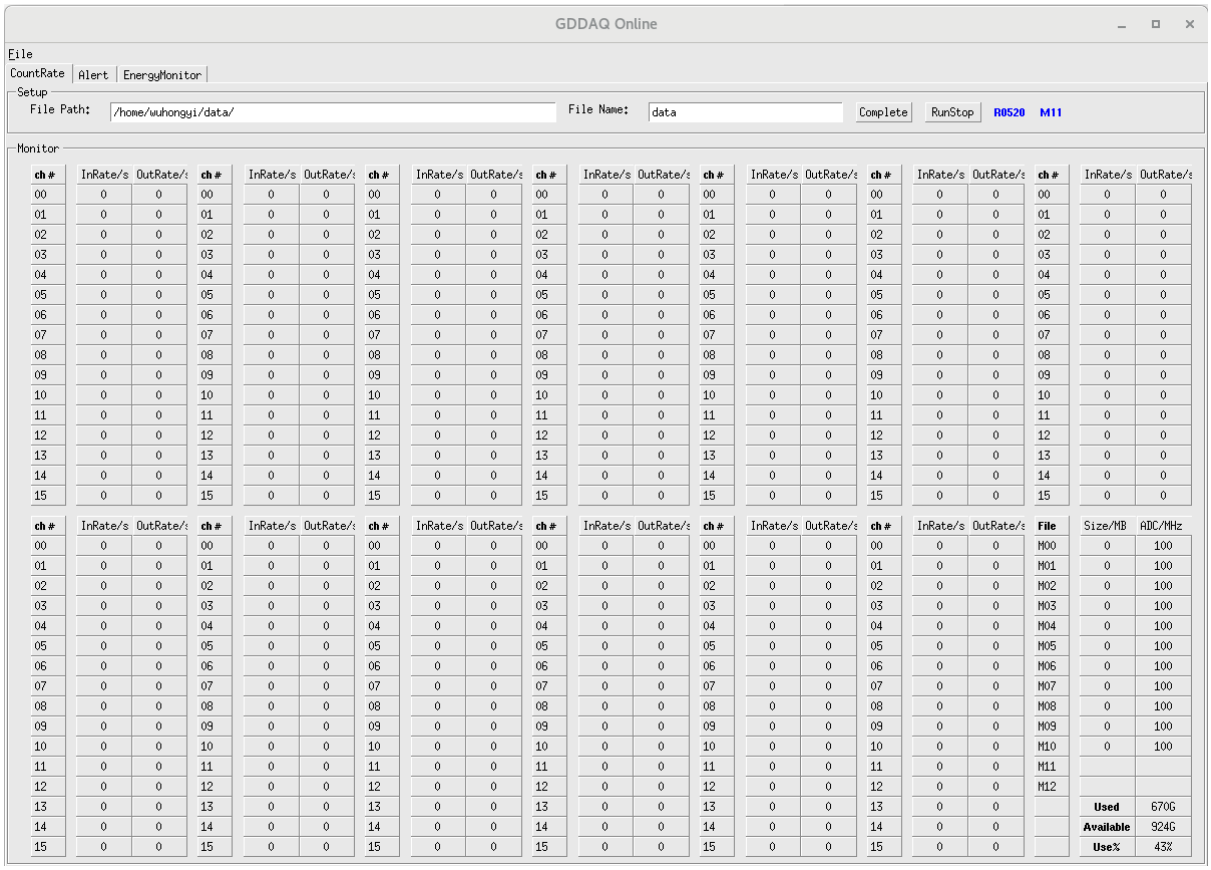
Modify the file **PixieOnline.config** in **OnlineStattics**, where the first line is the original binary file storage path and the second line is the file name. The two-line parameters is file name. There are used to monitor the real-time size and disk usage of each file.

Open the online monitoring main interface by executing the following command:

```
./online
```

Check the binary file path and file name for any problems. If there is no problem, click the button **Complete**, then click **RunStart** to enable online monitoring, and online monitoring will be refreshed every 3 seconds. The trigger rate of each channel and the actual event output rate of each channel can be monitored in real time.

The monitoring interface is as follows:


GDDAQ Online
File
CountRate
Alert
EnergyMonitor
Setup
File Path:
/home/wuhongyi/data/
File Name:
data
Complete
RunStop
R0520
M11
Monitor
ch #
InRate/s
OutRate/s
File
Size/MB
ADC/MHz
Used
670G
Available
924G
Use%
43%

CHAPTER 10

MakeEvent

本转换程序的使用前提，插件必须从第一个插槽开始，中间不留空插槽。

MakeEvent 程序用来快速将数据组装成与传统 VME 获取数据类似的结构，方便实验时的初步物理分析，最终的物理分析不能以本程序产生的数据为基准。

用户首先需要修改 **UesrDefine.hh** 文件中的定义

```
#define OUTFILEPATH "/home/wuhongyi/data/"
#define RAWFILEPATH "/home/wuhongyi/data/"
#define RAWFILENAME "data"


// 设置插件个数
#define BOARDNUMBER 5
```

用户需要修改：

- 原始 ROOT 文件的路径
- 生成的事件结构 ROOT 文件的存放路径
- 文件名
- 使用采集卡个数

修改之后执行以下命令编译程序：

```
make clean
make
```

编译成功之后将生成一个可执行文件 **event**，程序运行方式：

```
./event  [RunNnumber]  [windows]
```

其中 *[RuNnumber]* 为想要转换的文件运行编号，*[windows]* 为事件的时间窗，单位为 ns。

ROOT File Branch：

- sr: 采样率，该事件中该通道数值不为 0 表示探测到信号。
- adc: 能量

- outofr: 标记是否超模数转换的量程
- qdc: QDC 的八段积分
- tdc: 时间
- cfd: cfd 数值
- cfdft: 标记 CFD 数值是否有效
- cfds: 仅适用于 250/500 MHz 采集卡，cfd source

TODO 这里添加一个 Branch 截图。。。

CHAPTER 11

Front Panel

On the front panel of each Pixie-16 module, there are 16 analog signal input connectors, one LVDS I/O port, five digital I/O connectors as well as three LEDs near the bottom of the front panel. In addition, a sticker showing Pixie-16 model number (e.g., P16L-250-14, meaning the 14-bit, 250 MHz variant of the Pixie-16) is affixed to the top handle of the front panel, and another sticker indicating the serial number of the Pixie-16 module (e.g., S/N 1100) is placed at the bottom handle of the front panel.

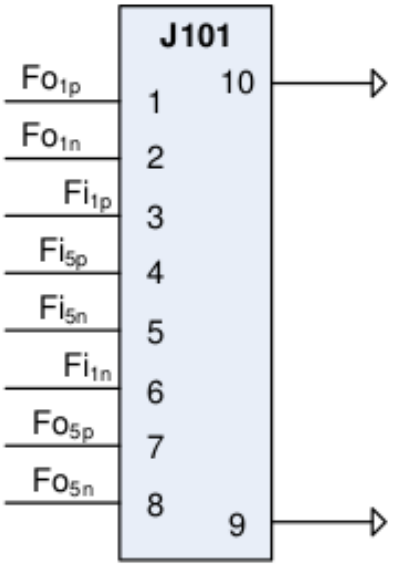
11.1 Analog Signal Input Connectors(all revisions)

- **Connector Labels**
  - 0 to 15 for 16 channels
- **Connector Type**
  - SMB Jack

Each Pixie-16 module accepts 16 analog input signals, and each input connector is a SMB Jack (male contact) connector.

11.2 LVDS I/O Port (all revisions)

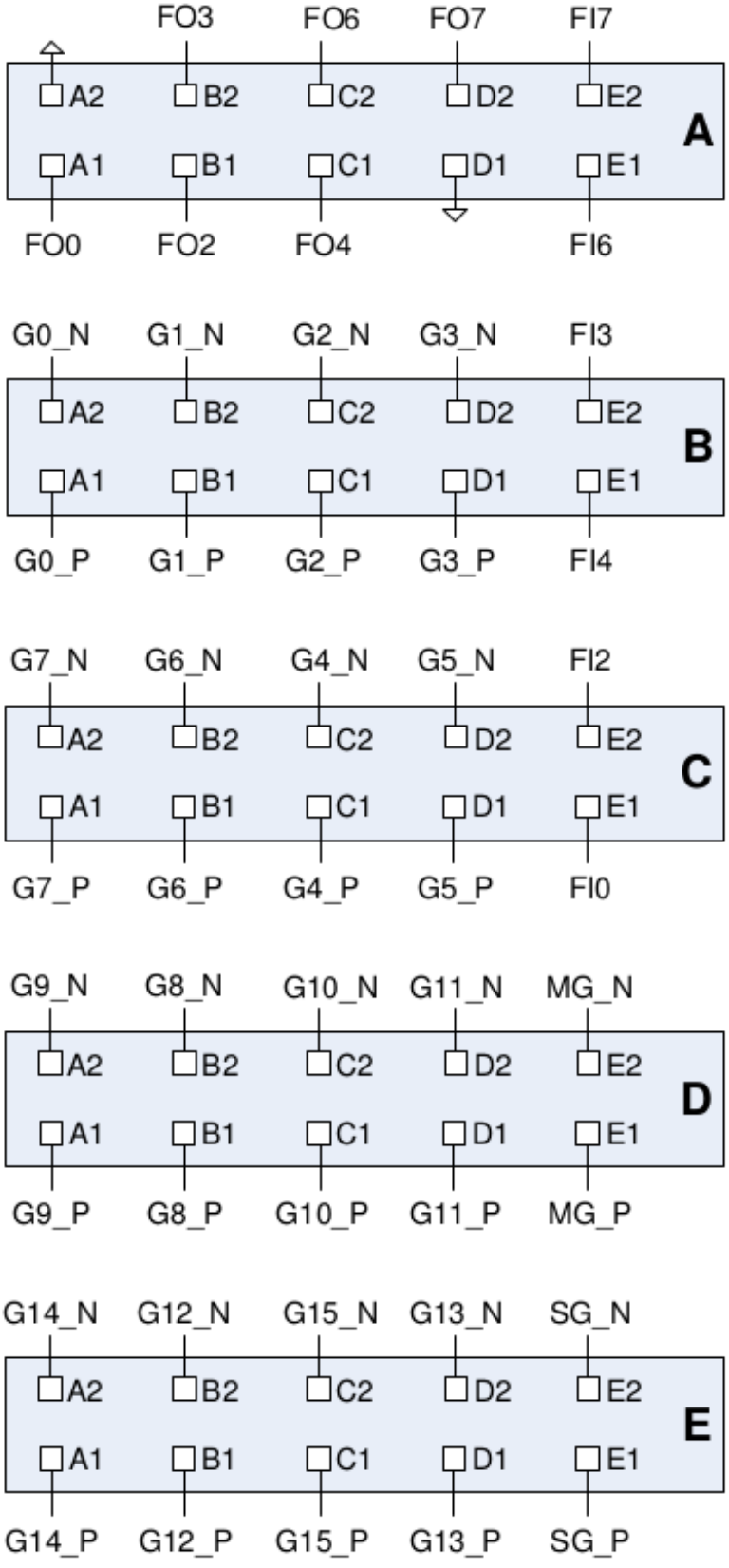
Connected Signals	4 LVDS pairs (Fo_{1p}/Fo_{1n}, Fi_{1p}/Fi_{1n}, Fi_{5p}/Fi_{5n}, Fo_{5p}/Fo_{5n}, see below for pin layout)
Signal Direction	Input or Output, software configurable
Cable Type	Cat 5 or Cat 6 (the same ones used for Ethernet)



Each Pixie-16 module is equipped with one LVDS I/O port on its front panel. LVDS stands for low voltage differential signaling. The LVDS I/O connector is a RJ45 connector, which implies that the same Cat 5 or Cat 6 Ethernet cables can be used to connect signals to or from this I/O port. However, no Ethernet connectivity is available through this Pixie-16 I/O port.

Four differential signal pairs, i.e., between pin-pairs Fo_{1p} / Fo_{1n}, Fi_{1p} / Fi_{1n}, Fi_{5p} / Fi_{5n}, and Fo_{5p} / Fo_{5n}, are available from this I/O port. Each pair can be configured as either an input or output signal.

11.3 Digital I/O Connectors(Rev. F only)



The Pixie-16 Rev. F modules are equipped with five har-link® connectors on its front panel which act as digital I/O connectors. The 2mm pitch har-link® connector from HARTING is designed for high speed data transfer with

rates up to 2 Gbit/s. Its EMI shielding, shown in Figure *2mm pitch har-link® connector from HARTING*, guarantees excellent performance in EM-polluted environment.

Each har-link® connector has 2 rows with 5 pins on each row, and is labelled using one of the five letters in red color font, from A to E. The signals connected to each pin of these five connectors are shown in Table *Rev. F Module's Digital I/O Connectors*.

Connector Type	har-link® (HARTING, 2mm pin spacing)
FI_0, FI_2, FI_3, FI_4, FI_6, FI_7	TTL digital input signals (max. 5V)
FO_0, FO_2, FO_3, FO_4, FO_6, FO_7	Digital outputs for test/debug purpose (TTL 5V)
Gx_P/Gx_N (x=0-15)	Channel Gate Inputs (0-15 for 16 channels) (LVDS format)
MG_P/MG_N	Module Gate Input (LVDS format)
SG_P/SG_N	Spare Gate Input (LVDS format)

Among them, FI0, FI2, FI3, FI4, FI6, FI7 are six TTL digital input signals. They can be signals like global fast trigger, global validation trigger,external clock, run inhibit, etc. The specific usage of each input pin is determined by the specific firmware that is downloaded to the Pixie-16 module (see Table 1-9 for input signals supported by the standard firmware). The six digital output signals, FO0, FO2, FO3, FO4, FO6, FO7, which are connected to six test output pins on the System FPGA of the Pixie-16, can be used to assist a user in the process of system setup. These test pins are connected to various internal signals of the Pixie-16 to provide insight of the current status of the system.

The Channel Gate Inputs (0-15 for 16 channels) are LVDS format input signals which independently gate the data acquisition of each of the 16 channels of a Pixie-16 module.

The Channel Gate signal is level sensitive signal, i.e., when the level of the Channel Gate Signal is logic high(1), the gate signal is effective; when the level of the Channel Gate Signal is logic low(0), the gate signal is not in use. In normal cases, the Channel Gate Signal is set up to veto the data acquisition in a given channel, i.e., at the time of the arrival of fast trigger in that channel, if the Channel Gate Signal is logic high(1), that fast trigger is discarded since it is vetoed. However, this type of logic can be reversed through setting corresponding registers in the FPGA via software. In such cases, the Channel Gate Signal is set up to validate the data acquisition in a given channel, i.e.,

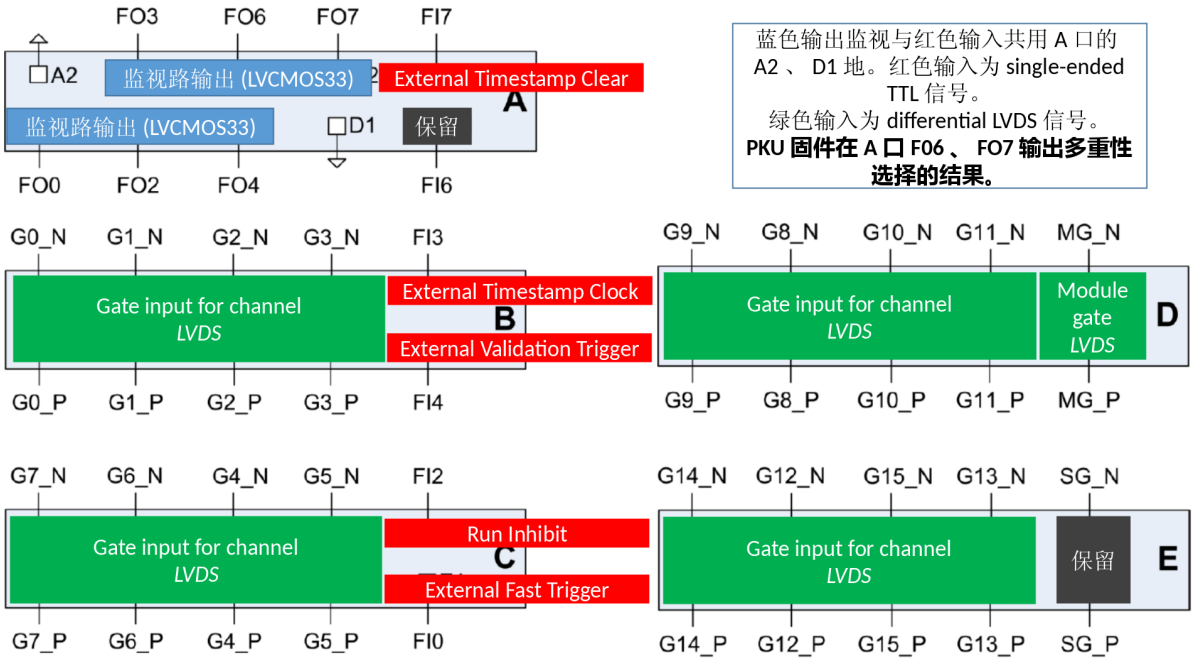
at the time of fast trigger in that channel, only if the Channel Gate Signal is logic high(1) will that fast trigger be accepted to have the event recorded.

The Module Gate Input is a LVDS format signal that gates the data acquisition in all 16 channels of a Pixie-16 module. It is also a level sensitive signal, i.e., when the level of the Module Gate Signal is logic high(1), the gate signal is effective; when the level of the Module Gate Signal is logic low(0), the gate signal is not in use. In normal cases, the Module Gate Signal is set up to veto the data acquisition in all 16 channels, i.e., at the time of the arrival of fast trigger in any of the 16 channels, if the Module Gate Signal is logic high(1), that fast trigger of that channel is discarded since it is vetoed. However, this type of logic can be reversed through setting corresponding registers in the FPGA via software.

In such cases, the Module Gate Signal is set up to validate the data acquisition in all 16 channels, i.e., at the time of fast trigger in any of the 16 channel, only if the Module Gate Signal is logic high(1) will that fast trigger of that channel be accepted to have the event recorded.

The Spare Gate Input is a LVDS format signal that is reserved for special applications.

Such applications typically require development of custom firmware to support special functionalities of the Pixie-16 system.



11.4 Front Panel LEDs (all revisions)

Near the bottom of the Pixie-16 front panel, there are three LEDs. They are labelled as RUN, I/O, and ERR, respectively, from left to right. They correspond to three different colors, **green, yellow, and red**, respectively.

LED Name	Color	Function
RUN	Green	ON when run is in progress, and OFF if run is stopped or not started yet
I/O	Yellow	Flashing when there is I/O activity on the PCI bus between the Pixie-16 module and host computer
ERR	Red	ON when there is no more space in the External FIFO for storage of list mode event data, and OFF when there is sufficient space to store at least one more list mode event data **(ON does not indicate any actual error condition. Rather, it simply indicates the External FIFO's FULL condition)**

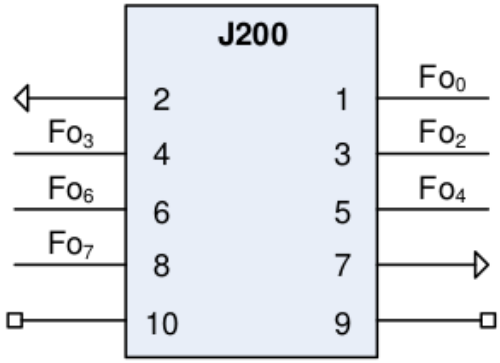
The **RUN LED** will be turned on when a run in the Pixie-16 module is in progress, and will be turned off when the run is stopped or not started yet.

The **I/O LED** will blink when there is I/O activity on the PCI bus between the Pixie-16 module and host computer.

The **ERR LED** is, in fact, not to signal any error condition in the Pixie-16 module. Instead, it is used to indicate whether or not the External FIFO of the Pixie-16 module is full. It will be ON when there is no more space in the External FIFO for storage of list mode event data, and OFF when there is sufficient space to store at least one more list mode event data. When the External FIFO is full, no more list mode event data can be written into it until the host software reads out part of the data in the External FIFO through the PCI bus.

11.5 3.3V I/O Connector(Rev. D only)

Connector Type	Single-ended, 2mm pin spacing
FO_0, FO_2, FO_3, FO_4, FO_6, FO_7	Digital outputs for test/debug purpose (3.3V)

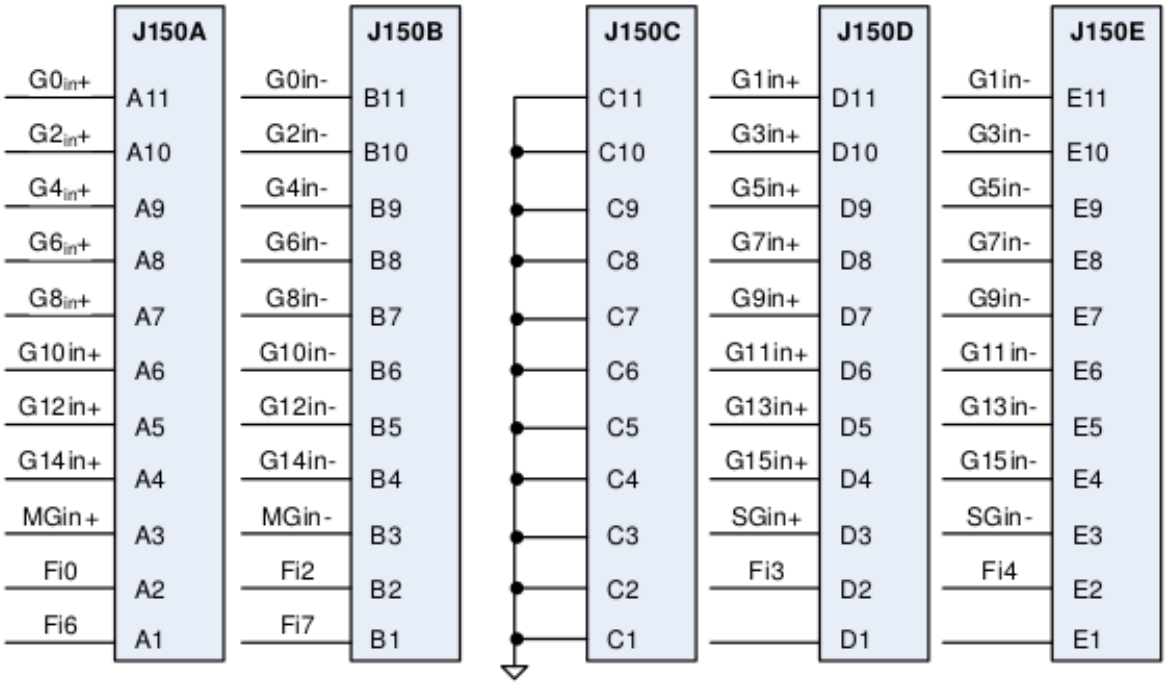


On Rev. D Pixie-16 modules, between analog input SMB connectors for channel 7 and channel 8, respectively, is the 3.3V I/O Connector(J200). It has 10 single-ended pins with 2mm spacing. Pins #1, 3, 4, 5, 6, and 8 are connected to six digital output signals from the System FPGA of the Pixie-16 module, i.e. FO0, FO2, FO3, FO4, FO6, FO7, mainly for the purpose of testing and debugging. Pins #2 and 7 are ground pins, and pins #9 and 10 are not in use.

11.6 GATE Inputs(Rev. D only)

Connector Type	Amphenol FCI® 55 Position Header, 2mm pin spacing
FI_0, FI_2, FI_3, FI_4, FI_6, FI_7	TTL digital input signals (max. 5V)
$Gx_{in}+/Gx_{in}-$ (x=0-15)	Channel Gate Inputs (0-15 for 16 channels) (LVDS format)
$MG_{in}+/MG_{in}-$	Module Gate Input (LVDS format)
$SG_{in}+/SG_{in}-$	Spare Gate Input (LVDS format)

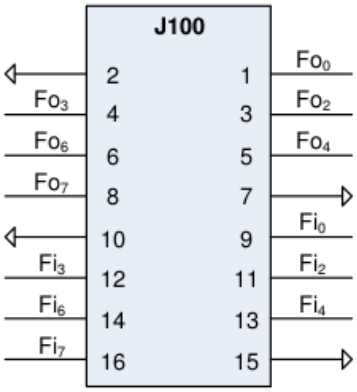
On Rev. D Pixie-16 modules, between analog input SMB connectors for channel 11 and channel 12, respectively, is the GATE INPUTS connector. This connector is an Amphenol FCI® 55 Position Header with 2mm pin spacing. The layout of these 55 pins is shown in Figure *Rev. D Module's GATE INPUTS Connector*. The 11 pins from the middle pin column (J150C) are all tied to the Ground. Among the first 8 rows of the GATE INPUTS connector, each differential pair of pins from the A/B columns(J150A/J150B) or the D/E columns (J150D/J150E) corresponds to one channel's GATE INPUT, which has the LVDS format, e.g. Gxin+/Gxin-(x=0-15). Differential pair of pins at J150A3/J150B3 is the Module Gate Input signal, MGin+/MGin-. Channel Gate Input signal can be used to veto or validate that given channel's own trigger. Module Gate Input signal works on the whole module level, i.e. it can be used to veto or validate all 16 channels' own trigger of that given module. Differential pair of pins at J150D3/J150E3 is the Spare Gate Input signal, SGin+/SGin-. Spare Gate Input signal can be used for special applications which require a custom firmware.



On Rev. D Pixie-16 modules, the TTL digital input signals (max. 5V), i.e. FI0, FI2, FI3, FI4, FI6, FI7, are distributed among the bottom two rows of the GATE INPUTS Connector, as illustrated in Figure *Rev. D Module's GATE INPUTS Connector*.

11.7 3.3V I/O Connector(Rev. B and C only)

Connector Type	Single-ended, 2mm pin spacing
FI_0, FI_2, FI_3, FI_4, FI_6, FI_7	TTL digital input signals (max. 5V)
FO_0, FO_2, FO_3, FO_4, FO_6, FO_7	Digital outputs for test/debug purpose (3.3V)



On Rev. B and C Pixie-16 modules, between analog input SMB connectors for channel 11 and channel 12, respectively, is the 3.3V I/O Connector(J100). It has 16 single-ended pins with 2mm spacing. Pins #1, 3, 4, 5, 6, and 8 are connected to six digital output signals from the System FPGA of the Pixie-16 module, i.e. FO0, FO2, FO3, FO4, FO6, FO7, mainly for the purpose of testing and debugging. Pins #2, 7, 10 and 15 are ground pins. Pins #9, 11, 12, 13, 14 and 16 are connected to the six TTL digital input signals (max. 5V), i.e. FI0, FI2, FI3, FI4, FI6, FI7.

11.8 Digital Signals in Standard Firmware(all revisions)

The standard firmware of the Pixie-16 supports input and output of digital signals through its front panel I/O connectors, which were discussed earlier.

TTL digital input signals	Connected signals in standard firmware	Direction	Description
FI_0	EXT_FASTTRIG	Input	External fast trigger signal
FI_2	INHIBIT	Input	Run inhibit signal
FI_3	EXT_TS_CLK	Input	External timestamp clock signal
FI_4	EXT_VALIDTRIG	Input	External validation signal
FI_6	not used		
FI_7	EXT_TS_CLR	Input	External timestamp clear signal

Table *TTL Digital Input Signals* shows the five TTL digital input signals supported by the Pixie-16 standard firmware.

Among them, the signals **EXT_TS_CLK** and **EXT_TS_CLR** are used for external timestamping in the Pixie-16, i.e. the Pixie-16 accepting an external clock signal(the frequency of this external clock is not recommended to exceed about 20 MHz in order to avoid the clock signal integrity issue), counting such clock signal with a 48-bit counter, and outputting such counter value to the list mode data stream when an event trigger occurs.

The external timestamping is useful for synchronizing the Pixie-16 data acquisition system with another data acquisition system through correlating the external timestamps of the events recorded by both systems.

The **INHIBIT** signal is used by an external system to inhibit the data acquisition run in a Pixie-16 system when synchronization requirement is enabled in the Pixie-16 modules. It is a level sensitive signal, i.e. when the **INHIBIT**

signal is at the logic high level, the run in the Pixie-16 won't start. Only when the **INHIBIT** signal goes to the logic low level will the run start in the Pixie-16. During the run, if the **INHIBIT** signal returns to the logic high level, the run will be aborted.

The **EXT_FASTTRIG** signal is the external fast trigger signal, which can be used to replace the local fast trigger for recording events in the Pixie-16 modules. The **EXT_VALIDTRIG** signal is the external validation signal, which can be used to validate events in the Pixie-16 modules.

Connector J101 Pins	Connected signals in standard firmware	Direction	Description
Fo_{1p}/Fo_{1n}	not used		
Fi_{1p}/Fi_{1n}	LVDS_VALIDTRIG	Input	External validation trigger signal in LVDS format
Fi_{5p}/Fi_{5n}	LVDS_FASTTRIG	Input	External fast trigger signal in LVDS format
Fo_{5p}/Fo_{5n}	SYNC_LVDS_FP	Output	Pixie-16 synchronization output signal in LVDS format (to synchronize with other DAQ systems)

Table *Connector J101 LVDS I/O Port Signals* shows the Pixie-16 connector J101 LVDS I/O port signals. This J101 LVDS I/O port can use the regular Ethernet cable for connection but it does not have Ethernet connectivity. Among the four LVDS pairs available from this J101 port, one pair is currently not in use, two pairs are used for input and one pair is used for output. The **LVDS_VALIDTRIG** is the external validation trigger signal in LVDS format, and the **LVDS_FASTTRIG** is the external fast trigger signal in LVDS format. The **SYNC_LVDS_FP** is an output signal from the Pixie-16 module to indicate to external data acquisition systems the synchronization status of the Pixie-16 system so that both data acquisition systems can be synchronized.

TTL digital output signals	Connected signals in standard firmware		Direction	Description	
FO_0	FTRIG_DELAY	FTRIG_DELAY	Output	Delayed local fast trigger of one of the 16 channels	Delayed local fast trigger of one of the 16 channels
FO_2	FTRIG_VAL	VETO_CE	Output	Validated, delayed local fast trigger one of the 16 channels	Stretched veto trigger of one of the 16 channels
FO_3	ETRIG_CE	LDPMFULL	Output	Stretched external global validation trigger of one of the 16 channels	Module level dual port memory (DPM) full status flag
FO_4	CHANTRIG_CE	SDPMFULL	Output	Stretched channel validation trigger of one of the 16 channels	System level dual port memory (DPM) full status flag
FO_6	FTIN_OR	FTIN_OR	Output	OR of 16 local fast triggers	OR of 16 local fast triggers
FO_7	TEST_SEL	TEST_SEL	Output	Selected test signal	Selected test signal

Table *TTL Digital Output Signals* lists the six Pixie-16 TTL digital output signals. Two groups of six output signals can be chosen through software settings (see bits [14:12] and [19:16] of TrigConfig0). The last output signal TEST_SEL

can be further selected through software settings. More details about these signals will be provided in later sections of this manual.

CHAPTER 12

Logic

12.1 Logic function

Whether each event of a certain signal is recorded depends on:

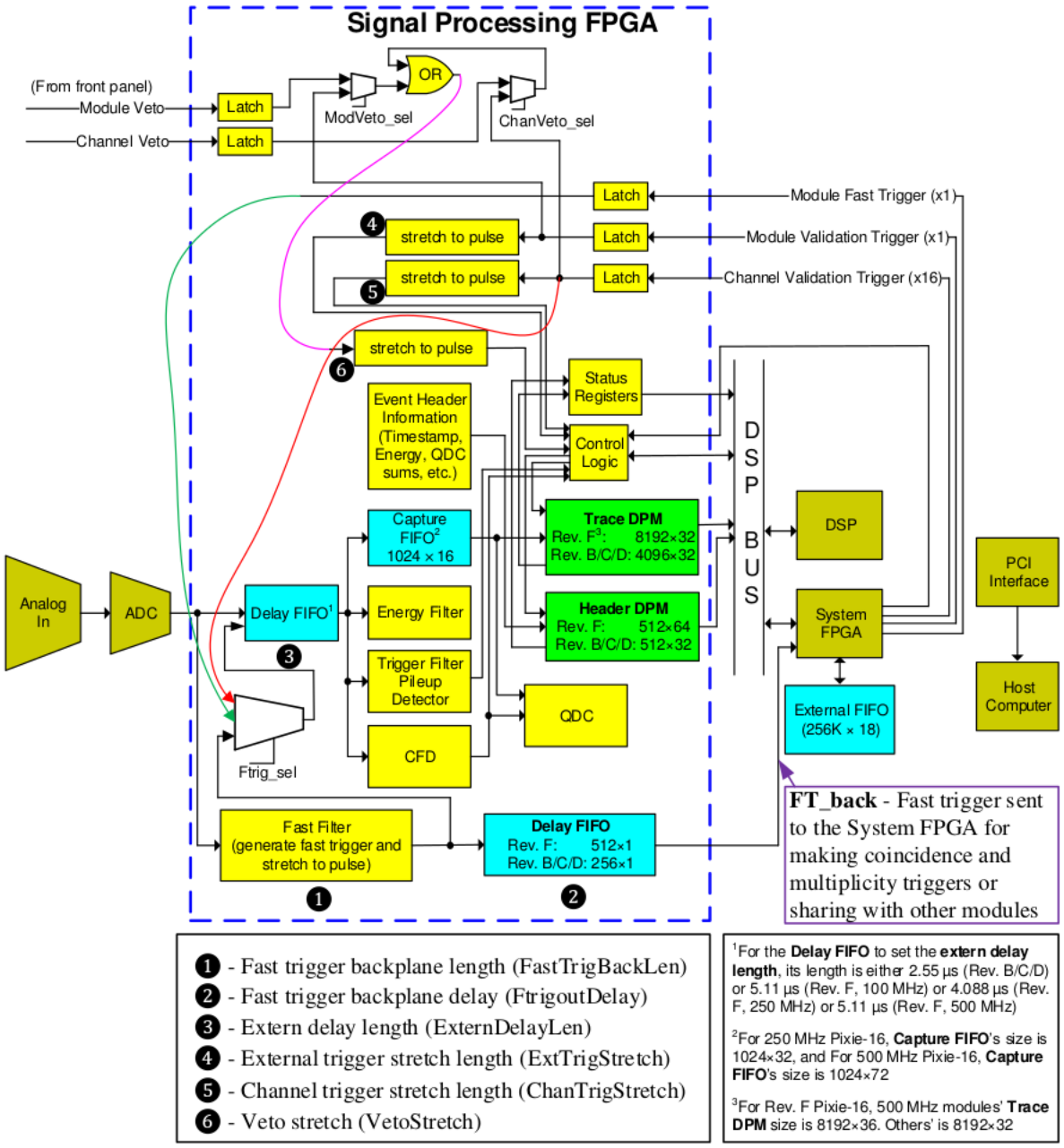
- Fast trigger select (first trigger)
- Control logic (second trigger)

Fast trigger select：

- Local fast filter
- Channel validation trigger
- Module fast trigger

Control logic：

- Module validation trigger
- Channel validation trigger
- Veto
- Pileup
- …



As shown in Figure, the incoming analog pulse is first digitized by the ADC and then enters the signal processing circuitries in the Signal Processing FPGA, each of which processes ADC data from 4 channels of a Pixie-16 module.

The digitized data stream is first fed into two branches: a fast filter generating fast triggers to be sent to the System FPGA and a Delay FIFO which could be used to compensate for the delay between fast triggers and the external triggers.

The digitized data stream passing through the Delay FIFO is then branched into four parts:

- energy filter which samples energy running sums at the PeakSample time;
- trigger filter which detects pulse and performs pileup inspection;
- capture FIFO which delays the ADC data according to the trace delay parameter value before the ADC data is streamed into the Trace Dual Port Memory (DPM) when a valid pulse is detected;
- CFD circuitry where a CFD trigger is generated to trigger the computation of QDC sums, latch timestamps and record traces.

The Control Logic in the signal processing FPGA utilizes the local fast trigger, CFD trigger, veto and external triggers to determine whether and when to stream waveform data into the Trace DPM and to write event information into

the Header DPM. The DSP polls the status of the DPMs through the Status Registers and moves event data into the External FIFO through the DSP bus and the System FPGA.

Trigger Stretch Lengths

- **External trigger stretch** is used to stretch the module validation trigger pulse.
- **Channel trigger stretch** is used to stretch the channel validation trigger pulse.
- **Veto stretch** is used to stretch the veto pulse for this channel.
- **Fast trigger backplane length** is used to stretch the fast trigger pulse to be sent to the System FPGA, where this fast trigger can be sent to the backplane to be shared with other modules, or can be used for making coincidence or multiplicity triggers.

FIFO Delays

- **External delay length** is used to delay the incoming ADC waveform and the local fast trigger in order to compensate for the delayed arrival of the external trigger pulses, e.g., module validation trigger, channel validation trigger, etc.
- **Fast trigger backplane delay** is used to delay the fast trigger pulse before it is sent to the System FPGA for sharing with other modules through the backplane or making coincidence or multiplicity triggers.

Parameters	Range	
	100 or 500 MHz	250 MHz
External trigger stretch	0.01 – 40.95 µs	0.008 – 32.76 µs
Channel trigger stretch	0.01 – 40.95 µs	0.008 – 32.76 µs
Veto stretch	0.01 – 40.95 µs	0.008 – 32.76 µs
Fast trigger backplane length	0.01 – 40.95 µs	0.008 – 32.76 µs
External delay length	0 – 2.55 µs (Rev. B/C/D) 0 – 5.11 µs (Rev. F)	0 – 4.088 µs
Fast trigger backplane delay	0 – 2.55 µs (Rev. B/C/D) 0 – 5.11 µs (Rev. F)	0 – 4.088 µs

12.2 Module Fast Trigger(for trigger)

Module fast trigger has four options：

- **Ext_FastTrig_In(From this module)**
  - Ext_FastTrig_Sel(Front panel TTL input)
  - Int_FastTrig_Sgl(Local fast trigger of a specified channel in this module)
  - FTIN_Or(The OR of local fast trigger of all channel in this module)
  - LVDS_FastTrig_FP(Front panel RJ45 port input)
  - ChanTrig_Sel(The valid trigger of a specified channel of the module)(Share a setting with module validation trigger)
- FT_LocalCrate_BP(Trigger sent by the specified module in this crate)
- FT_In_BP(Trigger sent by the specified module on the specified crate in multiple crates)
- FT_WiredOr(OR of local fast trigger of all modules in this crate)

12.3 Module Validation Trigger(for control logic)

Module validation trigger has five options：

- **Ext_ValidTrig_In(From this module)**
  - Ext_ValidTrig_Sel(Front panel TTL input)
  - Int_ValidTrig_Sgl(Local fast trigger of a specified channel in this module)
  - FTIN_Or(The OR of local fast trigger of all channel in this module)
  - LVDS_ValidTrig_FP(Front panel RJ45 port input)
  - ChanTrig_Sel(The valid trigger of a specified channel of the module)(Share a setting with module fast trigger)
- ET_LocalCrate_BP(Trigger sent by the specified module in this crate)
- ET_In_BP(Trigger sent by the specified module on the specified crate in multiple crates)
- ET_WiredOr(OR of local fast trigger of all modules in this crate)
- Front panel module GATE input LVDS signal

12.4 Channel Validation Trigger(for trigger/control logic)

Channel validation trigger has five options：

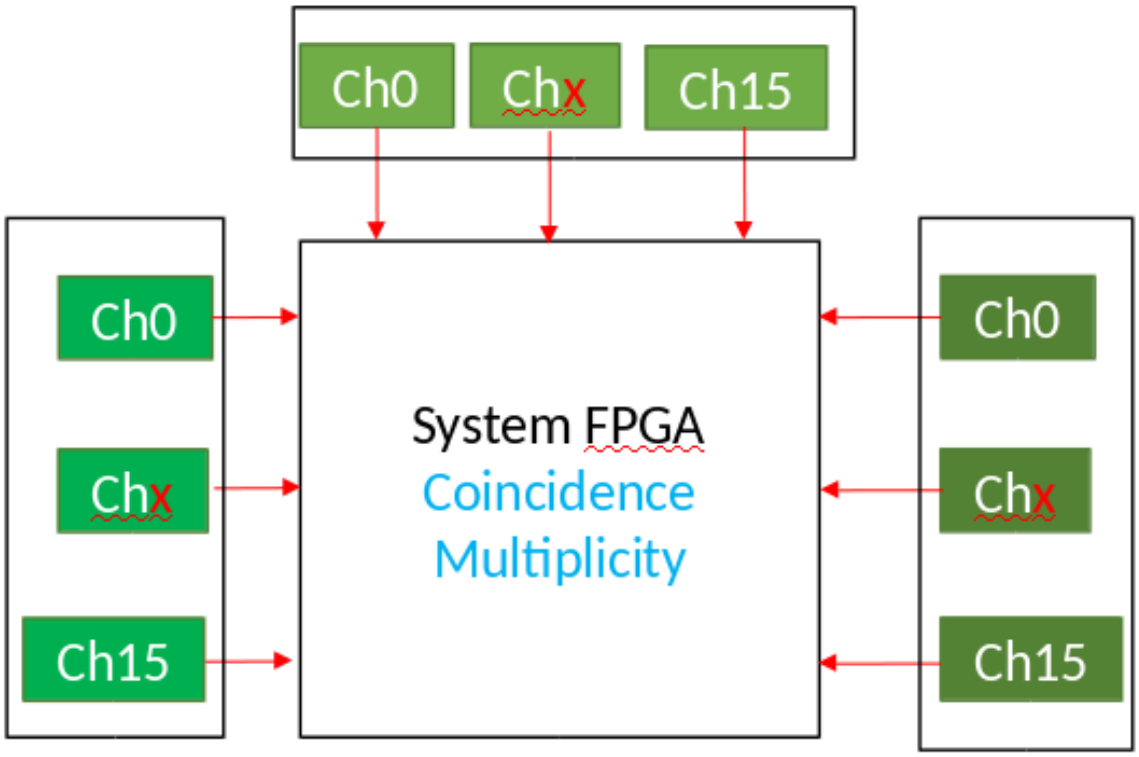
- Independent setting for each channel, from multiplicity
- Independent setting for each channel, from coincidence
- One setting for every 4 channels, derived from the left, its own and right modules FT
- One setting for every 4 channels, from the coincidence between its own FT and Ext_FastTrig_In
- Independent setting for each channe, the front panel channel GATE inputs the LVDS signal (shares one input port with the front panel Veto)

12.5 Veto

OR from ModuleVeto and ChannelVeto:

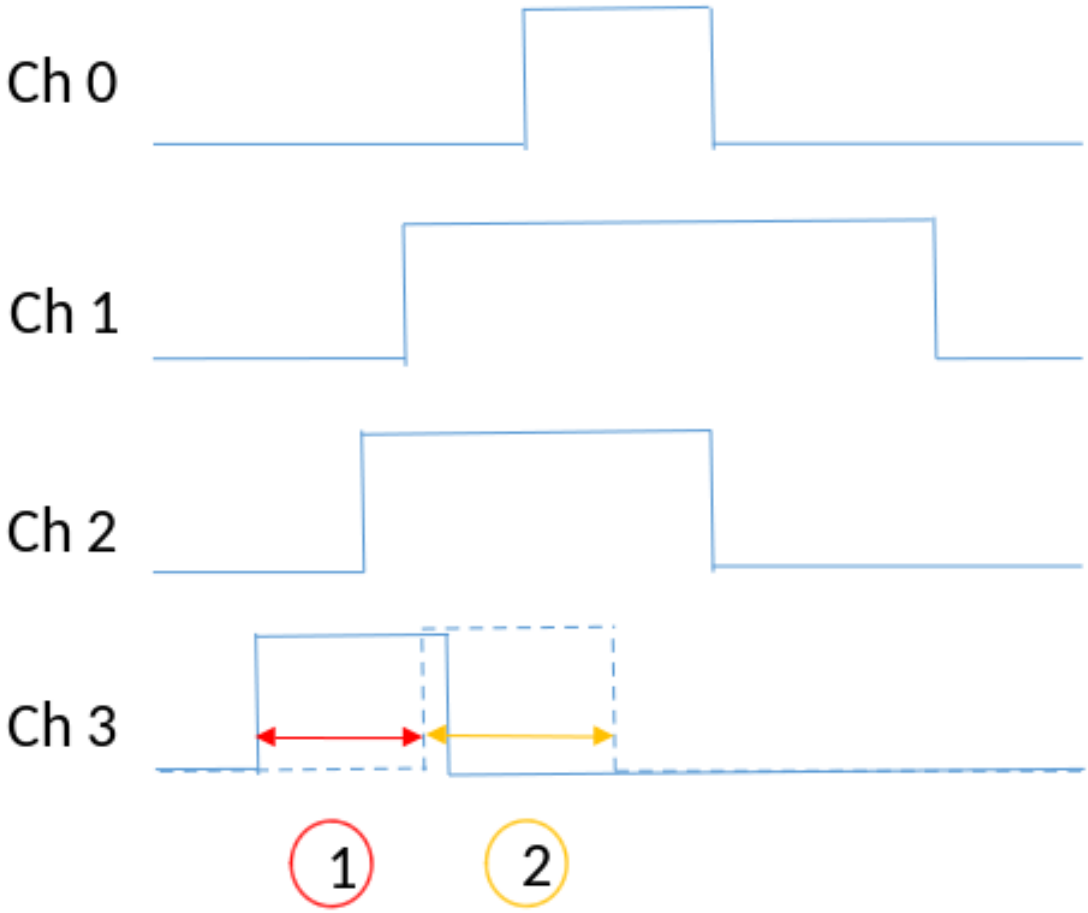
- **ModuleVeto source has two options：**
  - Module Validation Trigger
  - Front panel Module Gate
- **ChannelVote source has two options：**
  - Channel Validtion Trigger
  - Front panel Gate input for channel (Share one input port with the front panel Channel validation trigger)

12.6 System FPGA（coincidence/multiplicity）



Multiplicity: For the channel set up, the left neighbor module, its own module, and the right-side module have a total of 48 channels, and you can choose the number of channel to participate in multiply trigger.

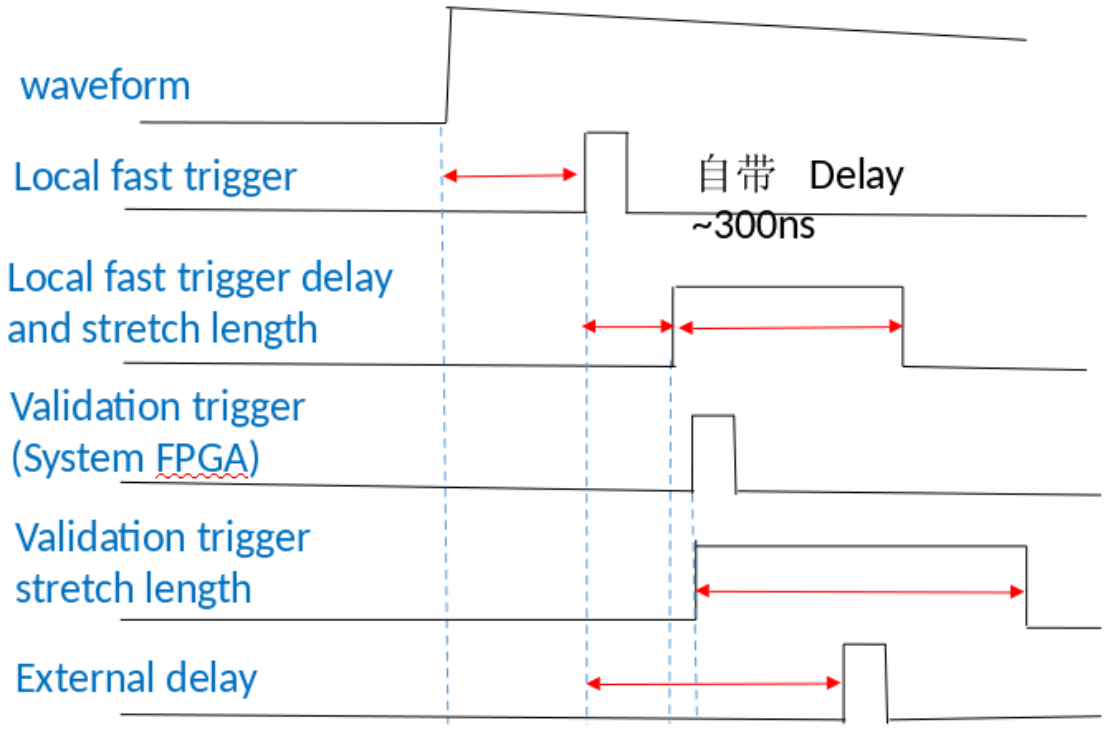
Coincidence: For the channel set, the left neighbor module, its own module, and the right neighbor module, each module match the set criteria to give a coincidence trigger.



It takes about 100 ns for the fast filter trigger of other modules to pass to the module through the crate backplane. Therefore, by adjusting the gate width and delay, it is guaranteed to coincidence and multiply trigger.

- Fast trigger stretch length: Set fast trigger gate width,
- fast trigger delay length: Set fast trigger delay。

Control logic (module/channel validation trigger)

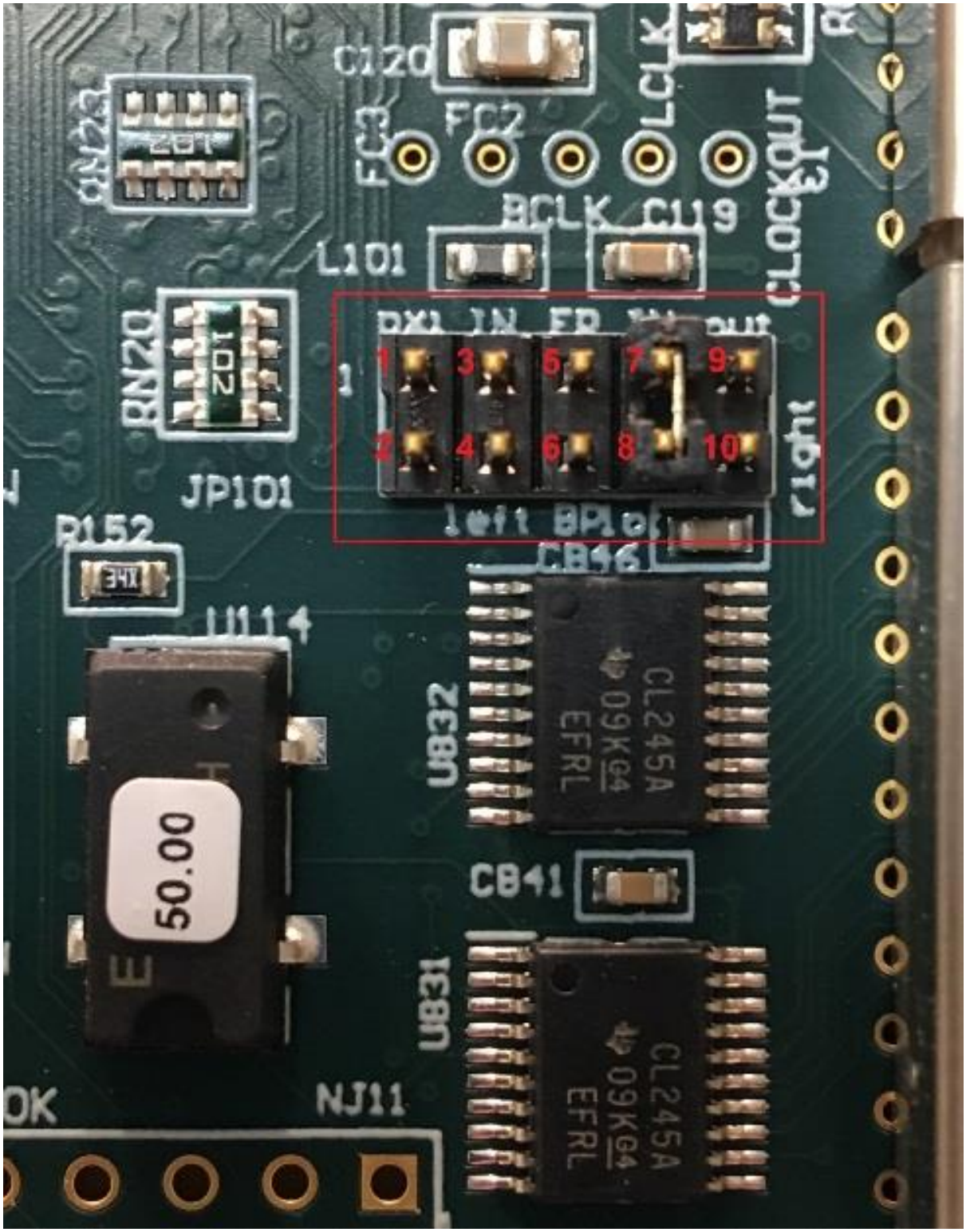


Particular attention should be paid to the fact that it takes about 100 ns for the signal to be transmitted through the backplane.

CHAPTER 13

Multiple Modules Synchronously

When many Pixie-16 modules are operated together as a system, it may be required to synchronize clocks and timers between them and to distribute triggers across modules. It will also be necessary to ensure that runs are started and stopped synchronously in all modules. All these signals are distributed through the PXI backplane of the Pixie-16 crate.

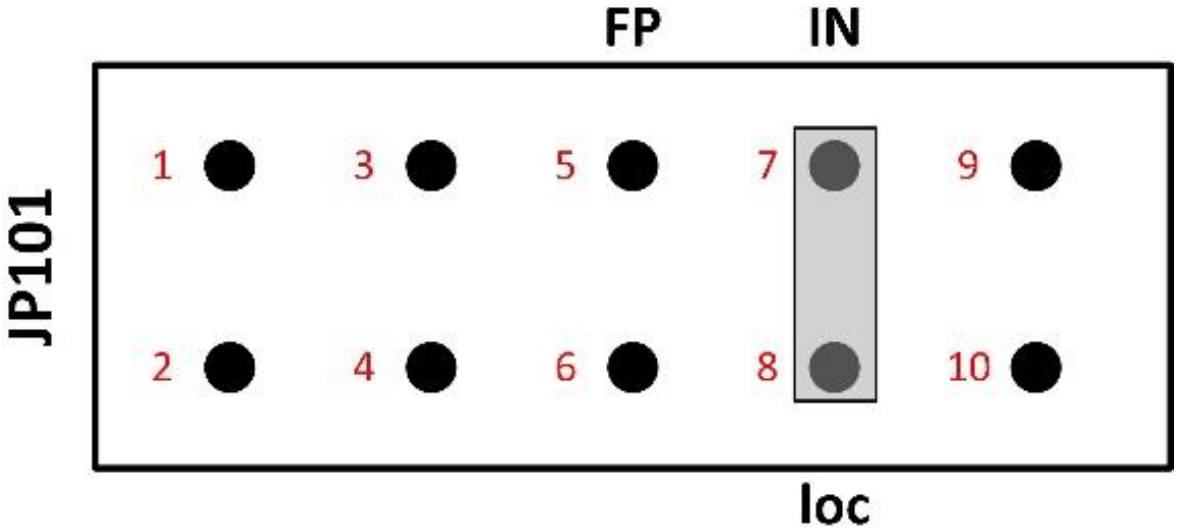


In a multi-module system there will be one clock master and a number of clock slaves or repeaters. The clock function of a module can be selected by setting shunts on Jumper JP101 near the bottom right corner of the board. The 10-pin Jumper JP101 is shown in the picture on the top with those pins labelled in red color. Shunts are provided to connect pins that are appropriate for each chosen clock distribution mode. Four clock distribution modes, individual clock mode, PXI clock mode, daisy-chained clock mode, and multi- crate clock mode, are described below.

警告： In 250 MHz or 500 MHz Pixie-16 modules, the frequency of signal processing clock in the FPGA has been divided down to either 125 MHz or 100 MHz, respectively, for more practical implementation of the design.

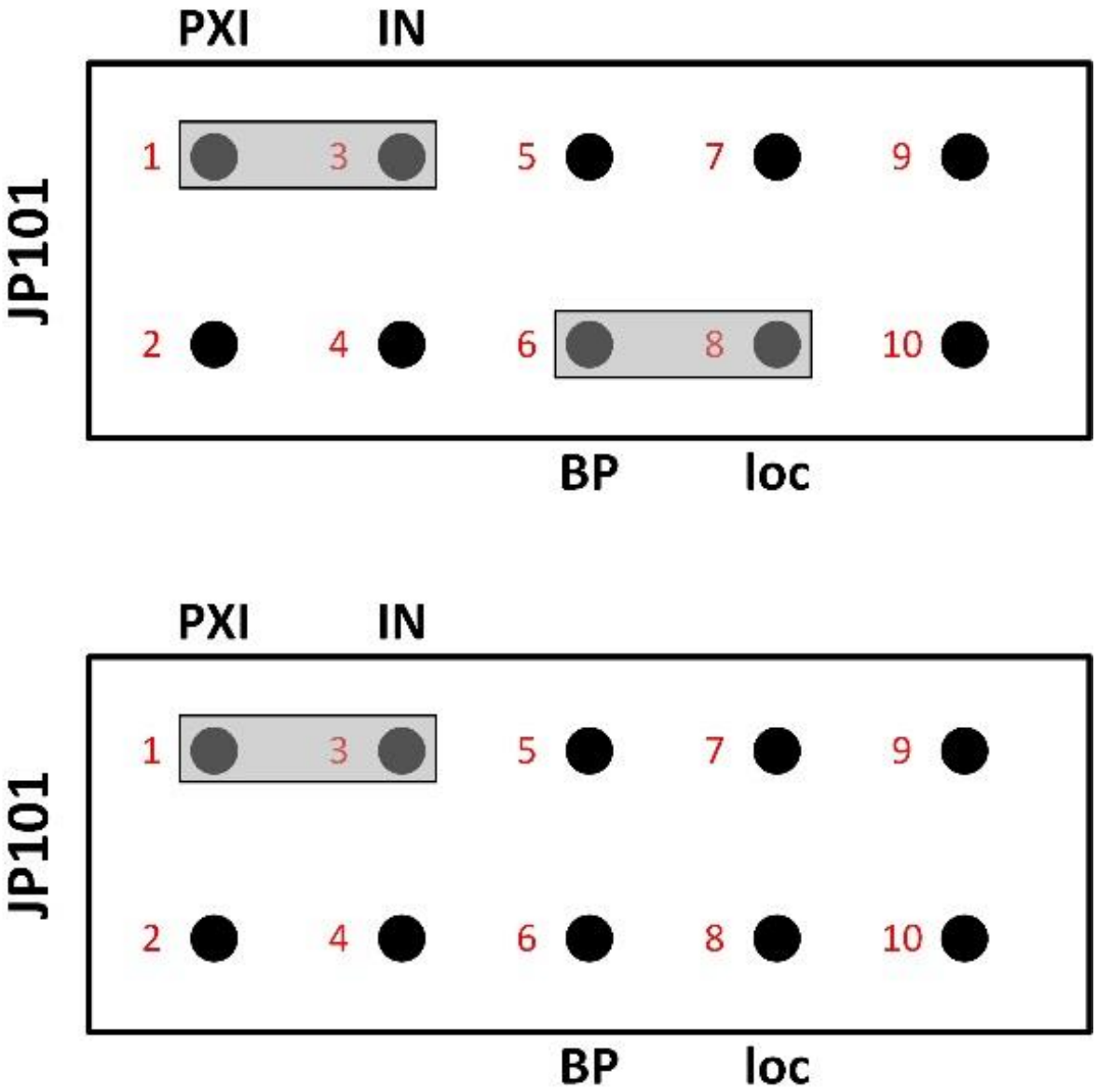
That division might result in different clock phase and thus different timestamp offset for each channel within a given 250 MHz or 500 MHz Pixie-16 module whenever the module is reinitialized. Calibration might be needed to quantify the different timestamp offset for each channel.

13.1 Individual Clock Mode



If only one Pixie-16 module is used in the system, or if clocks between modules do not need to be synchronized, the module(s) should be set into individual clock mode. Connect pin 7 of JP101 (the clock input) with a shunt to pin 8 (loc – IN). This will use the 50 MHz local crystal oscillator of the Pixie-16 module as the clock source.

13.2 PXI Clock Mode



Top: PXI clock master (slot 2)
Bottom: PXI clock recipient (slots 3-14)

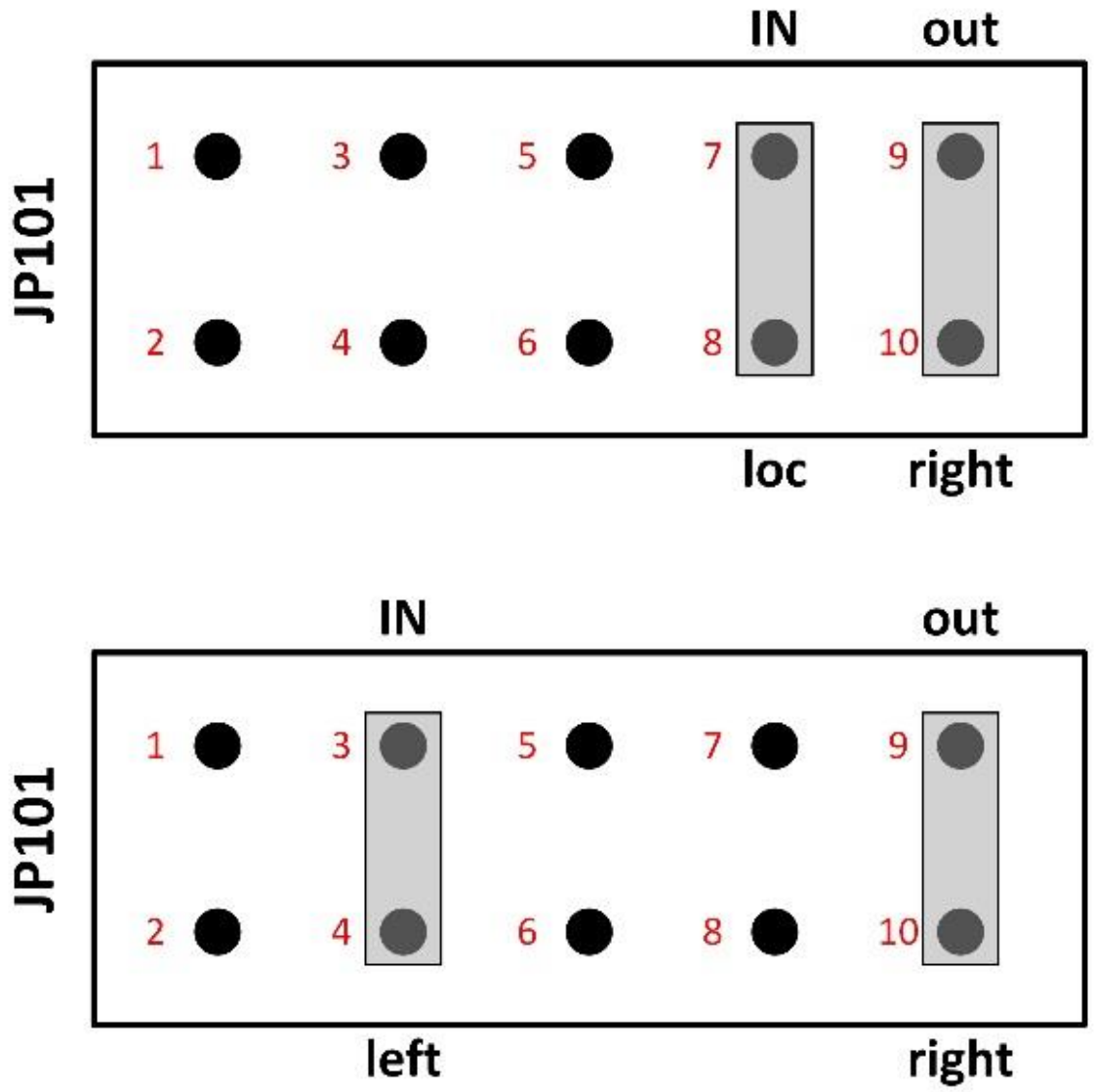
The preferred way to distribute clocks among multiple Pixie-16 modules is to use the PXI clock distributed on the backplane. This clock is by default generated on the backplane and is a 10MHz clock signal, which is then repeated by a fan out buffer and connected to each crate slot by a dedicated line with minimum skew(equal trace length to each slot). Although the 10MHz is too slow to be a useful clock for the Pixie-16, it can be overridden by a local clock signal from a Pixie-16 module that is installed in slot 2 through proper shunt settings on the JP101.

A Pixie-16 module can be configured to be the PXI clock master in slot 2 by connecting pins 6 and 8 (loc – BP) of the JP101. All modules, including the clock master, should be set to receive the PXI clock by connecting pin 1 and 3 on JP101 (PXI – IN). In this way, the 50 MHz clock from the Pixie-16 clock master is distributed to all Pixie-16

modules through the backplane with nearly identical clock phase.

One other advantage of the PXI clock mode over the daisy-chained clock mode, which will be discussed next, is that except for the Pixie-16 master module, which has to be installed in slot 2, other Pixie-16 slave modules can be installed in any other slot of the Pixie-16 crate. In contrast, when the daisy-chained clock mode is used, all Pixie-16 modules have to be installed next to each other, i.e. no gap is allowed between modules.

13.3 Daisy-chained Clock Mode

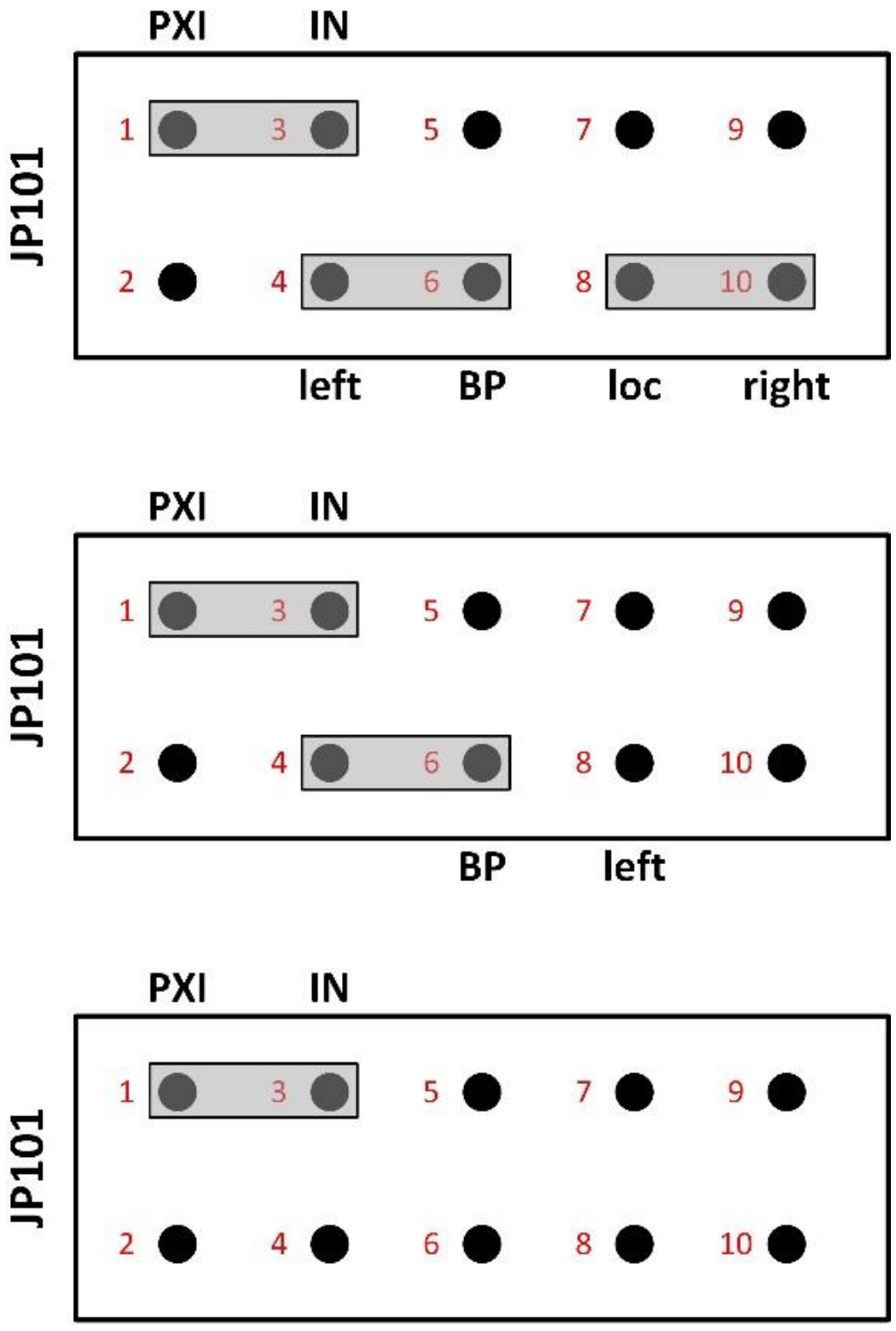


Top: Daisy-chain clock master (leftmost module)
Bottom: Daisy-chain clock repeater

A further option for clock distribution is to daisy-chain the clocks from one module to the other, with each module repeating the clock signal and transmitting it to the neighbor on the right. This requires one master module, located

in the leftmost slot of the group of Pixie-16 modules. The master module uses its local crystal oscillator as the input and sends its output to the right (loc – IN, out – right). Other Pixie-16 modules in the crate should be configured as clock repeaters by using the signal from the left neighbor as the input and sending its output to the right (left – IN, out – right). However, as mentioned earlier, there must be no slot gap between modules.

13.4 Multi-Crate Clock Mode



Top: System Director Module
Middle: Crate Master Module
Bottom: Regular Module

In multi- crate systems, a global clock signal can be distributed among these crates using dedicated trigger and clock distribution cards, i.e. the Pixie-16 Rear I/O trigger modules, which are available from XIA.

An example of clock distribution between two crates is illustrated below.

13.4.1 Installation of Pixie-16 Modules

Multiple Pixie-16 modules can be installed in two 14-slot Pixie-16 crates, #1 and #2. For clock distribution purpose, crate #1 is called the Master crate, where the system-wide global clock for all Pixie-16 modules is originated, and crate #2 is called the Slave crate, which receives the global clock from the Master crate.

The Pixie-16 module installed in slot 2 of the Master crate is designated as the System Director Module, whose local 50 MHz crystal oscillator acts as the source of the system-wide global clock. The distribution of the clock signal from the System Director Module to all Pixie-16 modules in the 2-crate system is done through the Pixie-16 Rear I/O trigger modules.

The Pixie-16 module installed in slot 2 of the Slave crate is called the crate Master module, which is responsible for receiving the global clock from the Master crate and sending such clock to all modules in that crate through length-matched traces on the backplane. The System Director Module is also responsible for sending the global clock to all modules in the Master crate. Therefore, it is also a crate Master module. Other modules in these two crates are regular modules. Table shows the different types of modules in a 2-crate system.

Crate #	**1**				
Slot #	**2**	**3**	...	**13**	**14**
Module	System Director Module / Crate Master Module	Regular Module	...	Regular Module	Regular Module
Crate #	**2**				
Slot #	**2**	**3**	...	**13**	**14**
Module	Crate Master Module	Regular Module	...	Regular Module	Regular Module

13.4.2 Clock Jumper (JP101) Settings on the Pixie-16 Modules

For all Pixie-16 modules in a 2-crate system to use the same global clock signal, the clock jumper (JP101) in all modules should be set according to Table *Clock Jumper JP101 Settings in a 2-crate System* and Figure *multi-crate clock mode*.

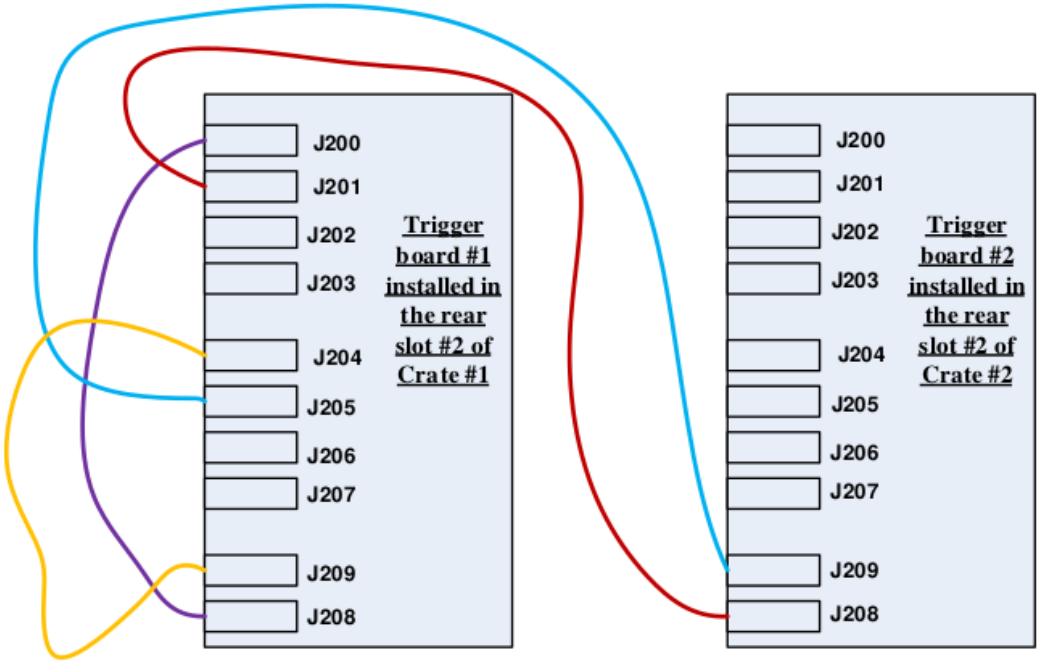
System Director Module	Connect pins 1 and 3, 4 and 6, 8 and 10.
Crate Master Module	Connect pins 1 and 3, 4 and 6.
Regular Module	Connect pins 1 and 3.

13.4.3 Cable Connections for Pixie-16 Rear I/O Trigger Modules

The Pixie-16 Rear I/O trigger modules are installed at the rear side of each crate where a 6U card cage is installed. Figure *rear I/O trigger modules* shows a Pixie-16 Rear I/O trigger module is installed directly behind either the Director or the Master module, respectively, to share clock, triggers, and run start or stop synchronization signals among multiple Pixie-16 crates. The rear of the backplane has connectors J3, J4 and J5, but it does not have J1 and J2, since it does not need to use CompactPCI or PXI communication.

Typically the first slot at the rear of the backplane with J3, J4, J5 connectors installed is the slot where the Pixie-16 Rear I/O trigger module should be installed. While installing the module, please ensure the alignment of top and bottom rails with the trigger module to avoid damage to the backplane pins.

Figure *Cable connections between two Pixie-16 rear I/O trigger modules* shows the cable connections between two Pixie-16 rear I/O trigger modules that are installed in two separate crates. All connection cables are Category 5 or 6 Ethernet cables and shall have the same length to minimize clock phase difference between Pixie-16 modules in the two crates.



13.4.4 Jumper Settings on the Pixie-16 Rear I/O Trigger Modules

Trigger module #1 is installed in the rear slot #2 of crate #1. As mentioned earlier, the rear slot #2 is located at the back of the crate and is at the direct opposite side of the front slot #2 of the crate. Care should be taken when installing the trigger module into the rear slot #2 by avoiding bending any pins of the rear side of the backplane, since that could cause the 3.3V pin to be shorted to neighboring ground pin and thus damage the whole backplane.

Please note pin numbering for all jumpers on the trigger module is counted from right to left when facing the top side of the module, i.e. the backplane connectors J3 to J5 are on the left (only exception is JP1, which is in vertical orientation and should be counted from bottom to top). A tiny ‘1’ label is painted on the right hand side of the jumpers, indicating pin 1. Figure *Pin numbering for the jumpers on the Pixie-16 rear I/O trigger module* shows the pin ‘1’ in red boxes.

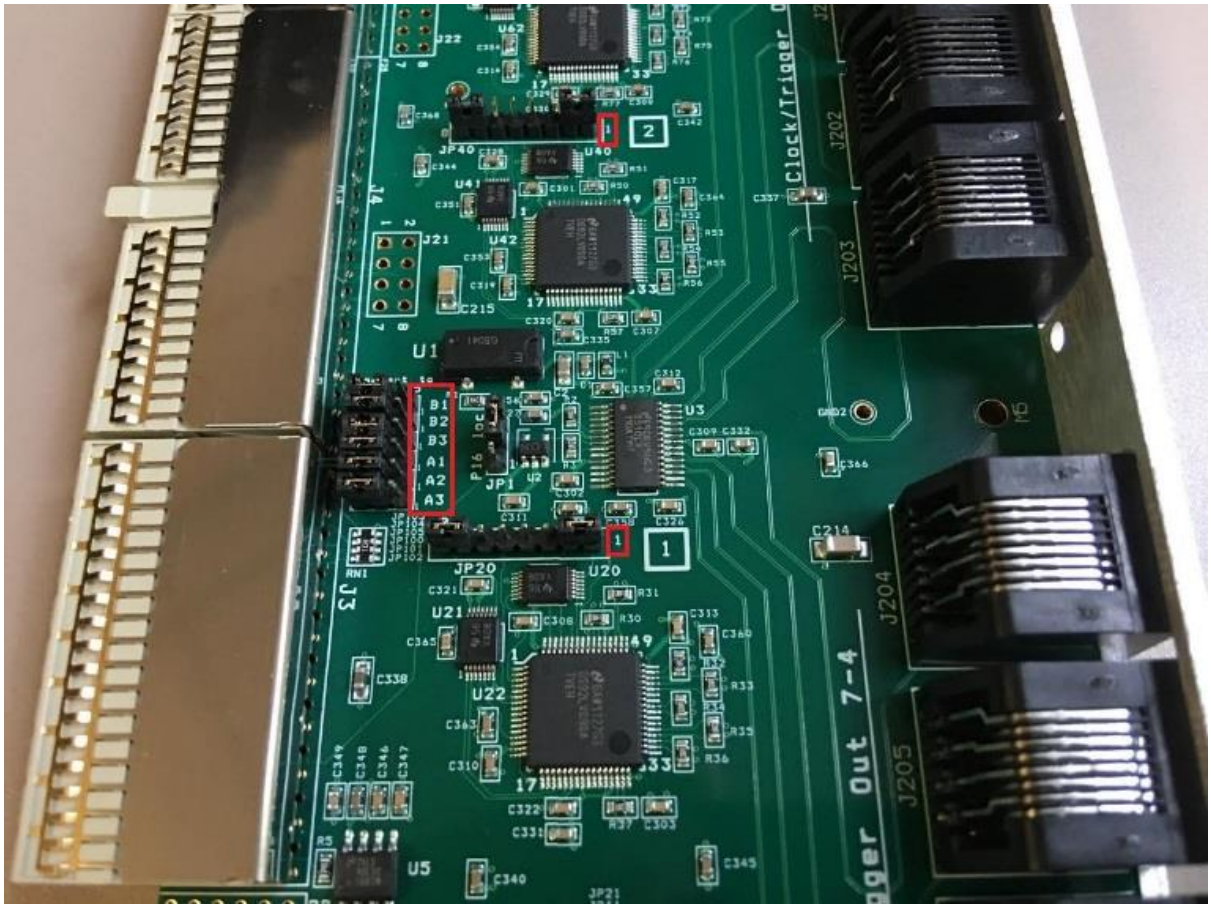
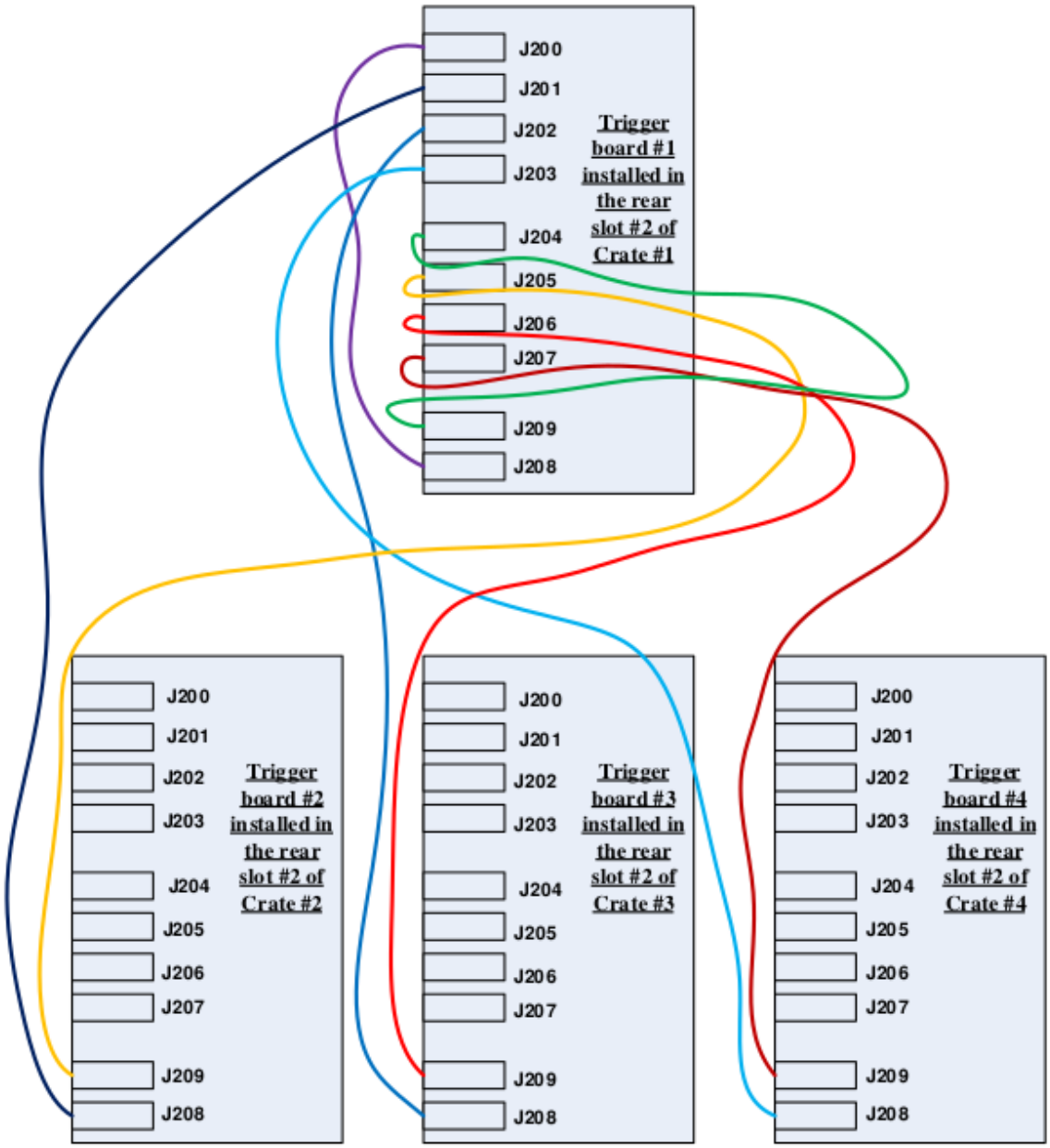


Table *Rear I/O Trigger Module #1' s Jumper Settings* shows the jumper settings of the Pixie-16 rear I/O trigger module #1 in a 2-crate system.

JP1	Connect pins 1 and 2 for "P16"
JP20	Connect pins 2 and 3, 6 and 7
JP40	Connect pins 2 and 3, 6 and 7
JP60	Connect pins 1 and 2, 7 and 8
JP21	Connect pins 2 and 3
JP41	Connect pins 2 and 3
JP61	Connect pins 1 and 2
JP100	Connect pins 2 and 3 (connect to J4)
JP101	Connect pins 2 and 3 (connect to J4)
JP102	Connect pins 2 and 3 (connect to J4)
JP103	Connect pins 2 and 3 (connect to J4)
JP104	Connect pins 2 and 3 (connect to J4)
JP105	Connect pins 2 and 3 (connect to J4)

Trigger module #2 is installed in the rear slot #2 of crate #2. Table *Rear I/O Trigger Module #2' s Jumper Settings* shows the jumper settings of the Pixie-16 rear I/O trigger module #2 in a 2-crate system.

JP1	Connect pins 2 and 3 for “loc”
JP20	Connect pins 1 and 2, 7 and 8
JP40	Connect pins 1 and 2, 7 and 8
JP60	Connect pins 2 and 3, 6 and 7
JP21	Don’t connect any pin
JP41	Don’t connect any pin
JP61	Connect pins 1 and 2
JP100	Connect pins 2 and 3 (connect to J4)
JP101	Connect pins 2 and 3 (connect to J4)
JP102	Connect pins 2 and 3 (connect to J4)
JP103	Connect pins 2 and 3 (connect to J4)
JP104	Connect pins 2 and 3 (connect to J4)
JP105	Connect pins 2 and 3 (connect to J4)



Please note, if there are a total of four crates, the cable connections among those four Pixie-16 rear I/O trigger modules that are installed in those four separate crates should follow the connection methods shown in Figure. For the jumper settings on the Pixie-16 rear I/O trigger modules, trigger module #1 and #2 should use the same jumper settings as those in the trigger module #1 and #2 of the 2-crate system, respectively, whereas trigger module #3 and #4 should use the same jumper settings as those in trigger module #2.

CHAPTER 14

Update CPLD

If you encountered issues with booting the Pixie-16 modules when you were using a newer and more powerful computer.

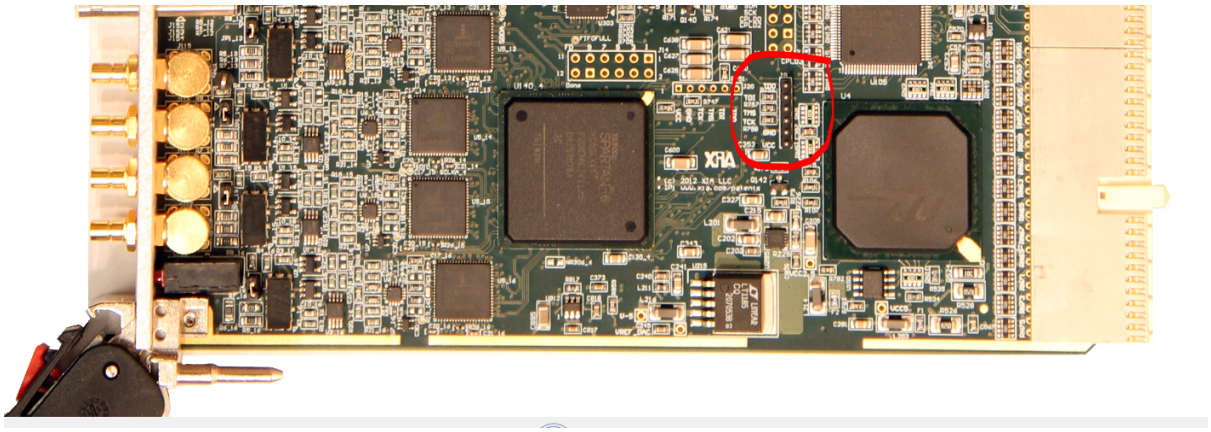
The root cause is that XIA API software replies on delays in the software to ensure proper delays when downloading the firmware bitstream to the Pixie-16 modules.

However, as the computer gets faster and faster with multi-cores, such delays can no longer be guaranteed in the software. So you had to update XIA CPLD firmware to ensure such delay is met using hardware methods.

Please note this CPLD code should be used to update all of your Pixie-16 modules, 100 MHz, 250 MHz or 500 MHz.

14.1 6 pin JTAG connector

You should now try to locate the 6-pin vertical Pixie-16 CPLD JTAG header J8 (circled in red in the screenshot below).



The next step is to identify each of the 6 pins of the Pixie-16 CPLD JTAG header J8. When facing the top side of the Pixie-16 board, the 6 pins of the J8 connector can be identified as follows from top to bottom: **TDO**, **TDI**, **TMS**, **TCK**, **GND**, and **VCC**. It is very important to match these pins with the colored cables from the green PCB connector that is plugged into the Xilinx Platform USB programmer.

One issue here is that the Pixie-16 CPLD JTAG header J8 has 2 mm pin spacing, and the colored cables from the green PCB connector of the Xilinx programmer might have too large header to be plugged into the J8 pins directly. In that case, an adapter might be needed to convert those colored cable headers to 2 mm pin spacing first.

14.2 update the CPLD

Turn on your Pixie16 crate and the LED on the Xilinx Platform Cable USB should turn green. If the LED does not turn green check you cable connection. The grey INIT cable is purposely left unconnected. Your start menu should now include the folder Xilinx ISE Design Suite 14.7. Run the 32 bit or 64 bit version of "iMPACT". Click on **File->Initialize Chain** and the program should find the **xc2c256 CPLD**. Click on **"No"** in the window **"Auto Assign Configuration Files Query"**, then click on **"Cancel"**. Left click on the Xilinx part in the window once, then right click and select **"Assign New Configuration Files"** and **select the new "jed" file**. Left click on the the Xilinx part, right click and then click on **"Program"**. A device programming properties window appears. Ensure that only **"Verify"** and **"Erase before Programming"** are checked, then click on "OK". The program should now say **"program succeeded"** in blue at the bottom of the window. The CPLD is now reprogrammed and the Pixie16 is ready to be tested.

I would suggest testing out your first board before reprogramming the CPLDs on all of your boards. Once you are in the batch mode of reprogramming and you have the cable connected (the USB programmer LED is green), you only need to click on "Program" to reprogram CPLD on your next Pixie16.

CHAPTER 15

Applications

This section introduces some experimental application cases, test results, and so on.

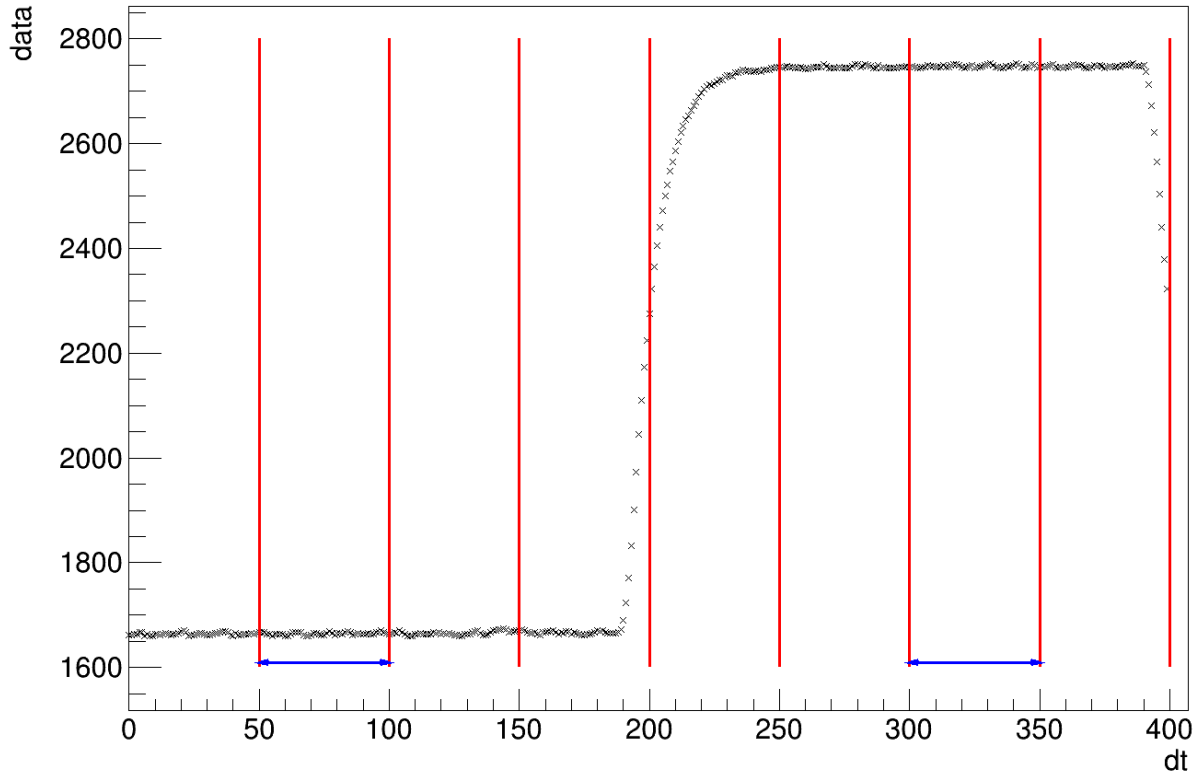
CHAPTER 16

TOF measurement in 100M

The time-of-flight measurements is a basic requirement in nuclear physics experiments. In the digital acquisition system, the intrinsic time resolution of the 100M module is about 100-200 ps, the 250M module is about 40ps, and the 500M module is about 20ps. When the sampling rate is not enough to directly measure the time-of-flight, the TOF information needs to be converted to the TAC signal first, and the time information is expressed by the amplitude of the TAC. The following shows how to achieve high-precision time-of-flight measurements with the 100M module.

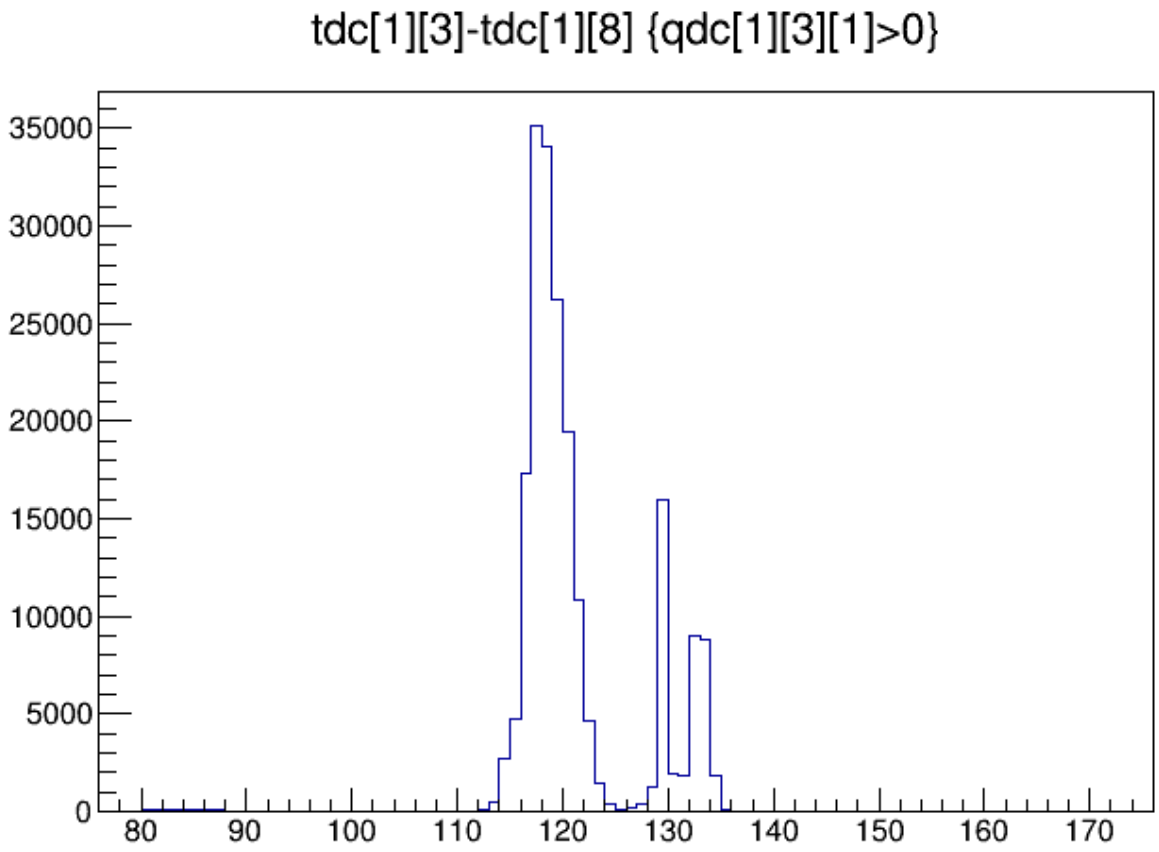
16.1 TAC signal characteristics

- The TOF signal is converted to a wave with a width of 2 us as shown in the figure below, and its height indicates the time-of-flight.
- XIA slow filter calculation energy is suitable for single exponential falling waveform, the TAC signal cannot give accurate height value by energy filter algorithm
- **Therefore, the QDC method is used here. Trigger point before 2 us is the starting point of QDC integration, and each 5**
  - pre-trigger 2 us, all QDCs gate width 0.5 us
- **Using the integral value difference of the two integral windows indicated by the blue arrow as the time-of-flight**
  - TOF = QDC[6]-QDC[1] (QDC[0], QDC[1], $\cdots$,QDC[7])
- The above parameters are a set of parameters we measured, which gives a time resolution comparable to CAEN V1290. This set of parameters avoids the rising portion of the waveform and the reflection peak near the sampling point 150.

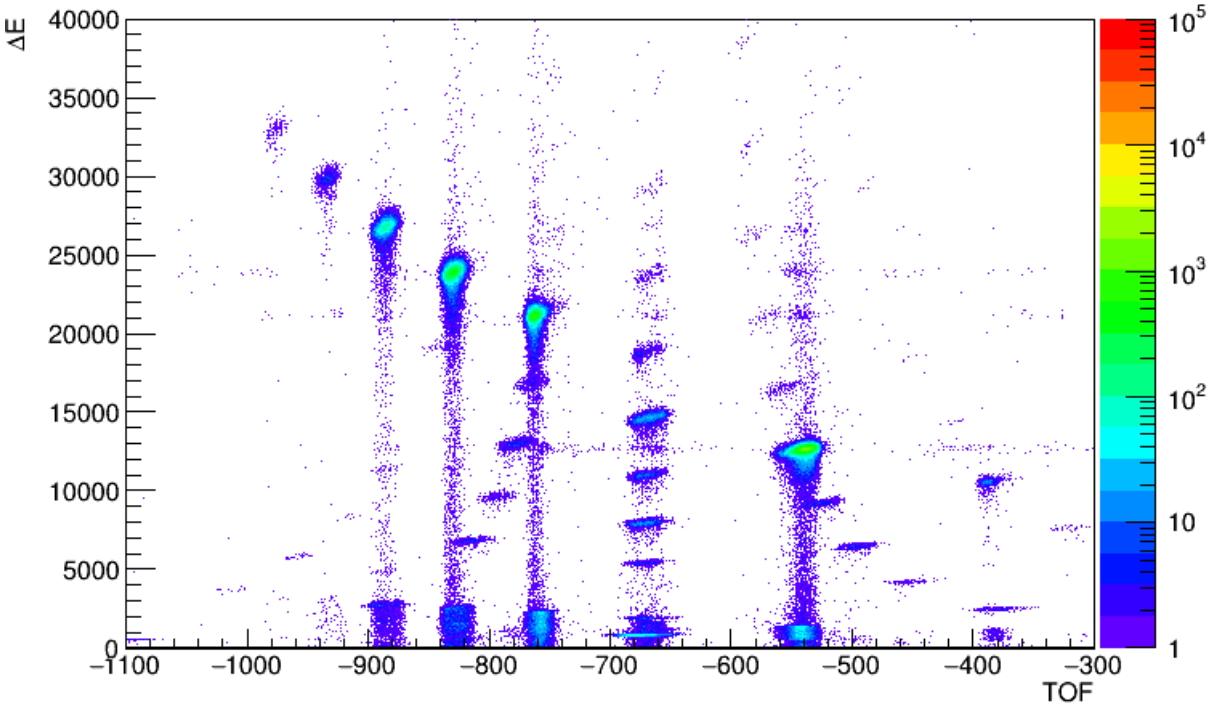


16.2 Experimental results(2017 RIBLL)

The following figure shows the time difference between the TAC signal and the silicon detector (10 ns/channel). We can find that there is a delay in the TAC signal output, and this delay is related to the TAC range we selected. Please pay attention to the time difference when reconstructing the event.



The figure below shows the experimentally measured TOF-DeltaE spectrum. The time-of-flight is derived from two thin plastic scintillation detectors, and DeltaE is from a silicon detector.



CHAPTER 17

Time Resolution

17.1 Pulse Generator

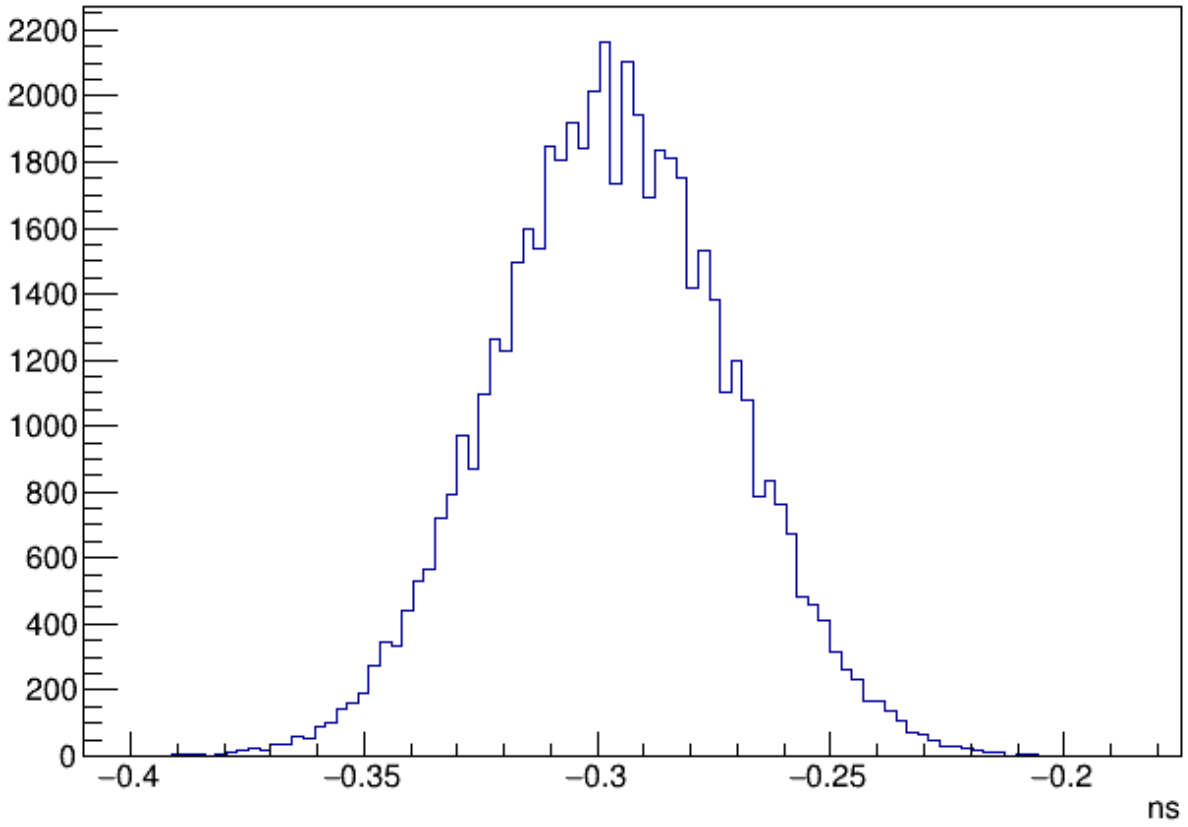
- **Model PB-5**
  - https://www.berkeleynucleonics.com/model-pb-5

17.2 100M module

- Width: 100ns
- Rise time: 0.1 us
- Fall time: 100 us
- Rate: 1.0 kHz
- Delay: 250ns
- Ampl: 10.0 V
- Polarity: Pos
- Pulse Top: Flat
- Atten: 1X
- PB-5 Pulse: ON

The pulse generator generates a signal that is split into two through the splitter.

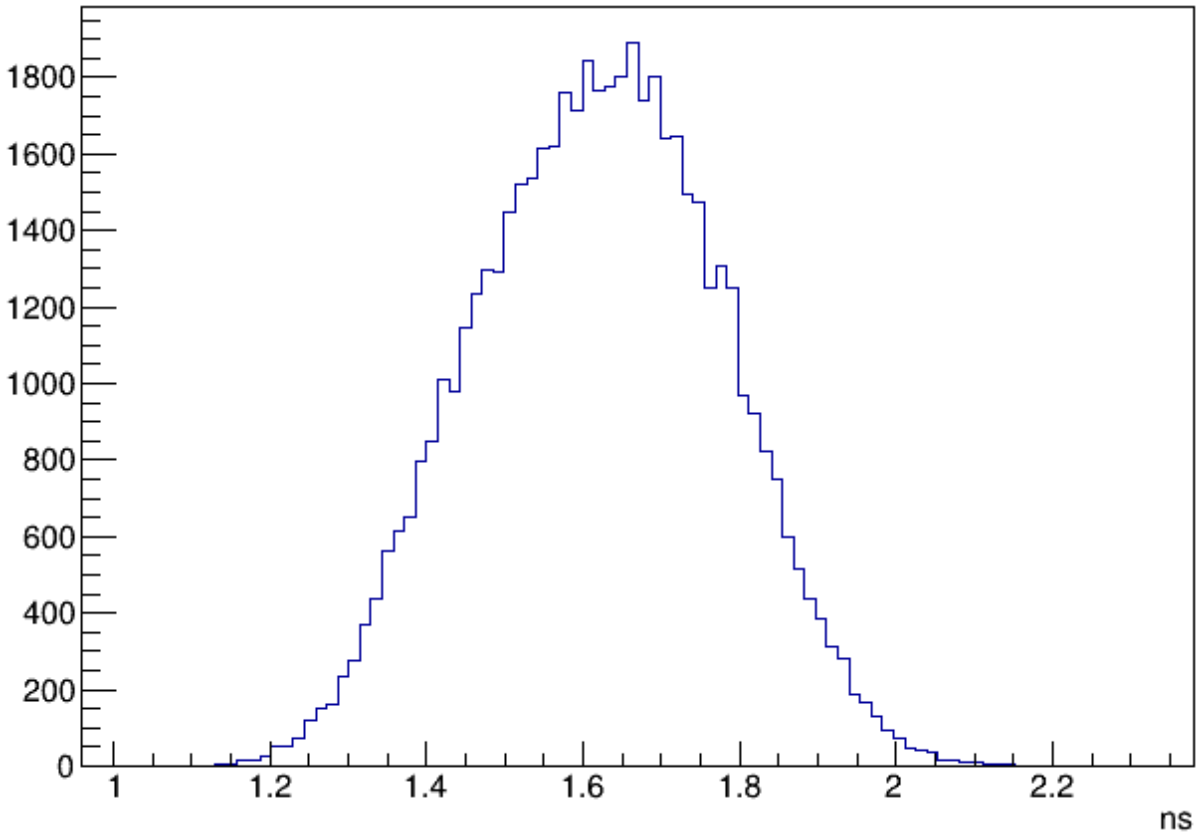
17.2.1 2 channel in one module



FWHM ~ 60ps

17.2.2 2 channel in one crate

17.2.3 2 channel in different crate



FWHM ~ 360 ps

17.3 250M module

17.3.1 2 channel in one module

17.3.2 2 channel in one crate

17.3.3 2 channel in different crate

17.4 100M & 250M module

17.4.1 2 channel in one crate

17.4.2 2 channel in different crate

CHAPTER 18

Recommended Parameters

The parameters of this page are for reference only.

18.1 TAC

18.1.1 100M

- **fast filter**
  - FL:
  - GF:
- **slow filter**
  - SL:
  - SG:
  - Range: 1
  - TAU:
- **cfd filter**
  - dealy:
  - scale:

18.1.2 250M

- **fast filter**
  - FL:
  - GF:
- **slow filter**
  - SL:
  - SG:

  - Range: 1
  - TAU:
- **cfd filter**
  - dealy:
  - scale:

18.1.3 500M

- **fast filter**
  - FL:
  - GF:
- **slow filter**
  - SL:
  - SG:
  - Range: 1
  - TAU:
- **cfd filter**
  - dealy:
  - scale:

18.2 NIM signal

18.2.1 100M

- **fast filter**
  - FL:
  - GF:
- **slow filter**
  - SL:
  - SG:
  - Range: 1
  - TAU:
- **cfd filter**
  - dealy:
  - scale:

18.2.2 250M

- **fast filter**
  - FL:
  - GF:
- **slow filter**
  - SL:
  - SG:
  - Range: 1
  - TAU:
- **cfd filter**
  - dealy:
  - scale:

18.2.3 500M

- **fast filter**
  - FL:
  - GF:
- **slow filter**
  - SL:
  - SG:
  - Range: 1
  - TAU:
- **cfd filter**
  - dealy:
  - scale:

18.3 HPGe

18.3.1 100M

- **fast filter**
  - FL: 0.1
  - GF: 0.1
- **slow filter**
  - SL: 5.04
  - SG: 1.2/1.6
  - Range: 3
  - TAU: Based on actual measurements

- **cfd filter**
  - dealy: 0.02
  - scale: 5

18.3.2 250M

- **fast filter**
  - FL: 0.13
  - GF: 0.13
- **slow filter**
  - SL: 5.04
  - SG: 1.2
  - Range: 3
  - TAU: Based on actual measurements
- **cfd filter**
  - dealy: 0.08
  - scale: 0

18.4 BGO

18.4.1 100M

- **fast filter**
  - FL: 0.06
  - GF: 0.0
- **slow filter**
  - SL:
  - SG:
  - Range: 1
  - TAU: Based on actual measurements
- **cfd filter**
  - dealy: 0.08
  - scale: 0

18.5 Si

18.5.1 100M

- **fast filter**
  - FL: 0.1
  - GF: 0.0
- **slow filter**
  - SL: 3.04
  - SG: 0.24
  - Range: 2
  - TAU: Based on actual measurements

18.6 LaBr3

18.6.1 250M

- **fast filter**
  - FL: 0.08
  - GF: 0.016
- **slow filter**
  - SL: 0.144/0.128
  - SG: 0.048
  - Range: 1
  - TAU: Based on actual measurements(0.023)
- **cfd filter**
  - dealy: 0.024
  - scale: 0

18.6.2 500M

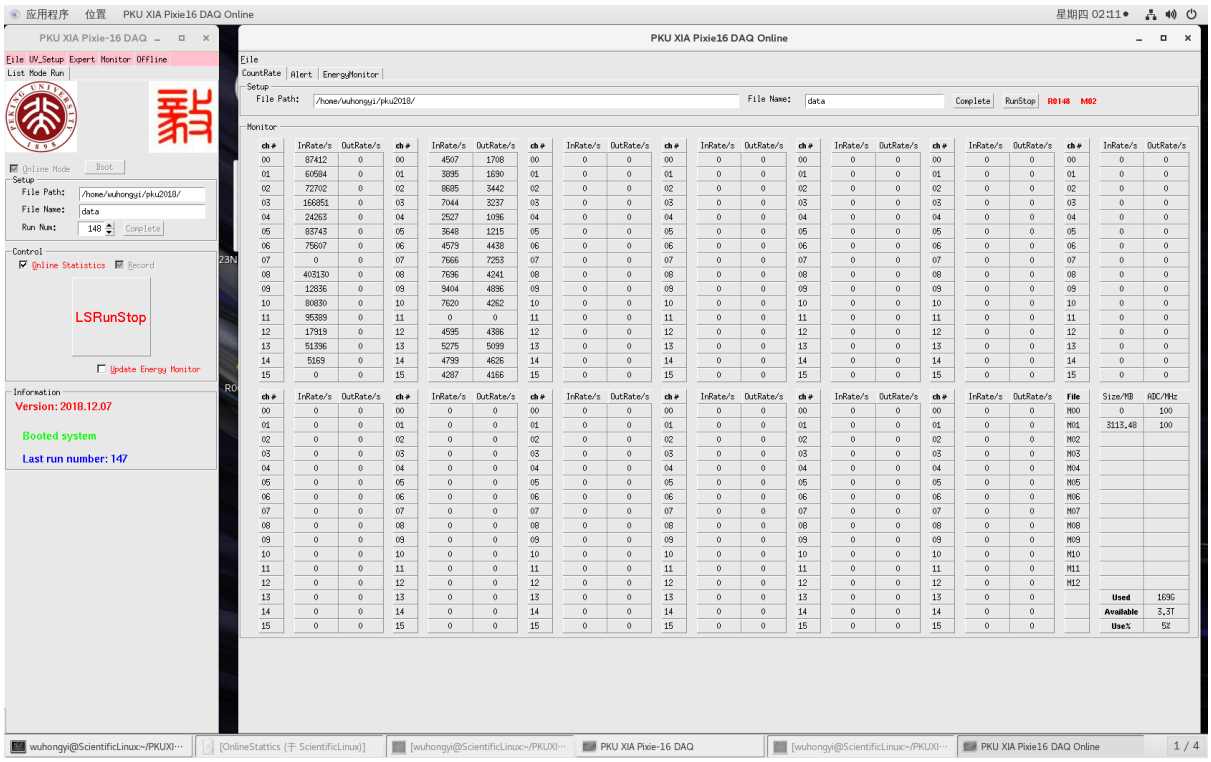
- **fast filter**
  - FL: 0.02
  - GF: 0.0
- **slow filter**
  - SL: 0.140
  - SG: 0.06
  - Range: 1
  - TAU: Based on actual measurements(0.023)

CHAPTER 19

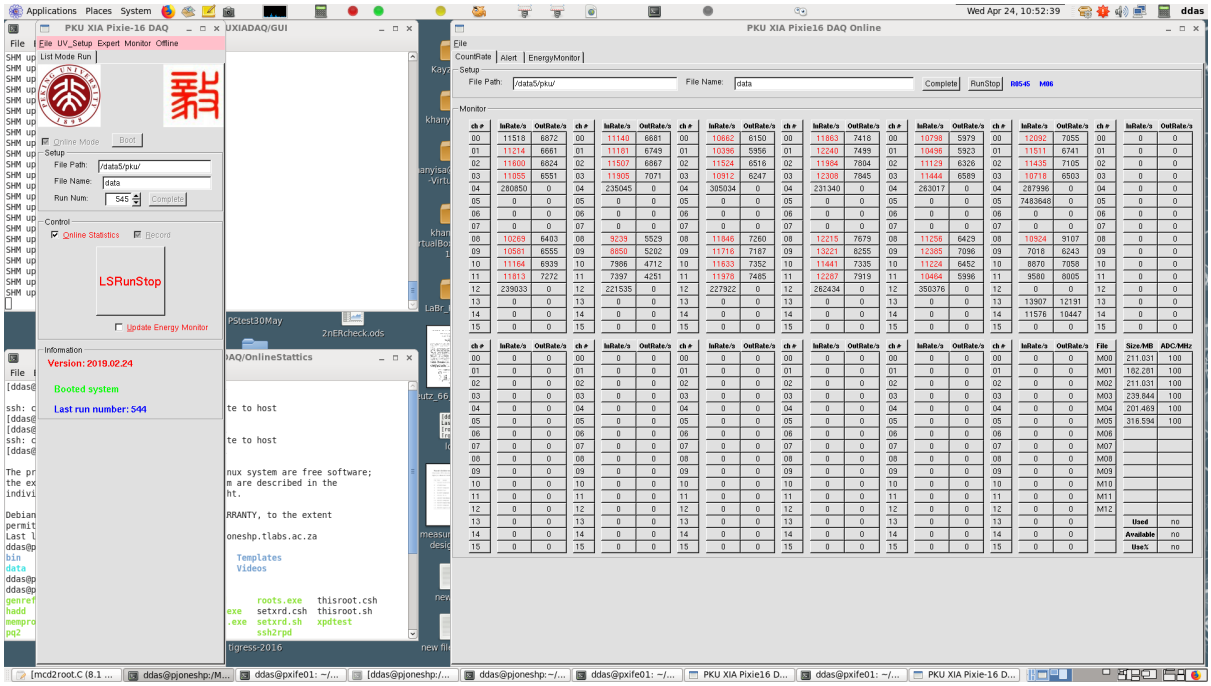
In Beam Gamma

19.1 Control Interface

The experimental control interface is shown below

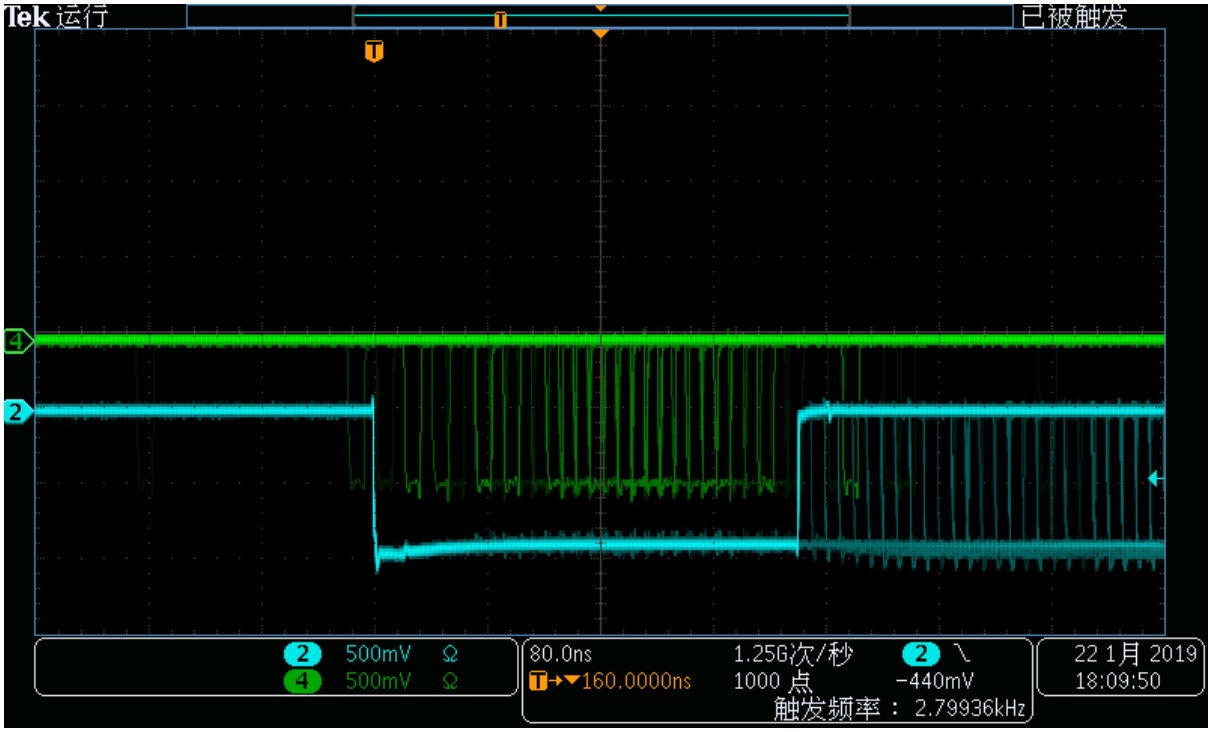


5 HPGe + 2 Clovers + 2LEPS

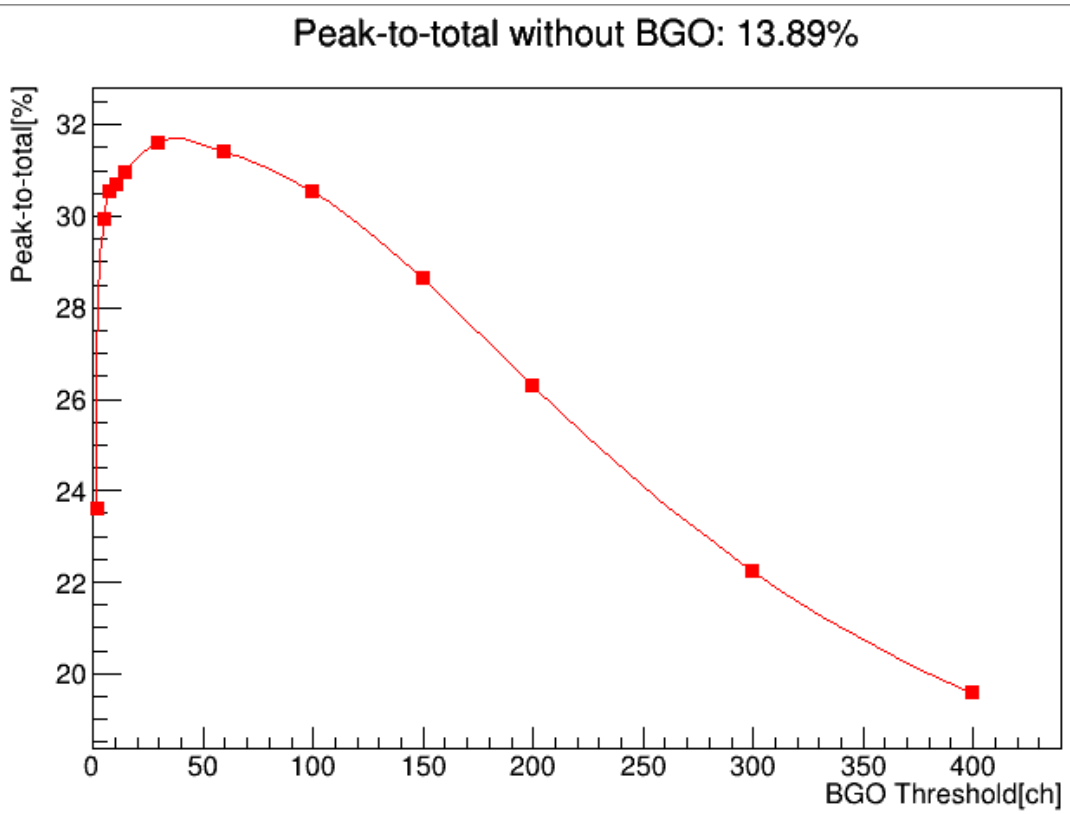


11 Clovers + 4 channel LEPS + 2 LEPS

19.2 Veto gate width



19.3 Peak-to-total



CHAPTER 20

Developer’s Guide

This chapter introduces the basic principles of Pixie-16 API functions.

Provide users with the possibility of program development based on our DAQ.

CHAPTER 21

PIXIE-16 API

It from **Pixie-16 Programmer's Manual** *Version 3.06, September 13, 2019*

21.1 XIA API

21.1.1 Pixie16AcquireADCTrace

Acquire ADC traces in single or multiple modules

```
// Use this function to acquire ADC traces from Pixie-16 modules. Specify the␣
↪module using ModNum which starts counting at 0. If ModNum is set to be less than␣
↪the total number of modules in the system, only the module specified by ModNum␣
↪will have its ADC traces acquired. But if ModNum is equal to the total number of␣
↪modules in the system, then all modules in the system will have their ADC traces␣
↪acquired.
// After the successful return of this function, the DSP's internal memory will be␣
↪filled with ADC trace data. A user's application software should then call the␣
↪function Pixie16ReadSglChanADCTrace to read the ADC trace data out to the host␣
↪computer, channel by channel.
int Pixie16AcquireADCTrace (
      unsigned short ModNum );// module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Failed to start run**
  - Reboot the modules
- **-3 Acquiring ADC traces timed out**
  - Reboot the modules

Usage example

```
unsigned short ModNum;
int retval;

// assume we want to acquire ADC trace from module 0
ModNum = 0;

// acquire the trace
retval = Pixie16AcquireADCTrace (ModNum);
if(retval < 0)
{
// error handling
}
```

21.1.2 Pixie16AcquireBaselines

Acquire baselines from a module

```
// Use this function to acquire baselines from Pixie-16 modules. Specify the␣
↪module using ModNum which starts counting at 0. If ModNum is set to be less than␣
↪the total number of modules in the system, only the module specified by ModNum␣
↪will have its baselines acquired. But if ModNum is set to be equal to the total␣
↪number of modules in the system, then all modules in the system will have their␣
↪baselines acquired.
// After the successful return of this function, the DSP' s internal memory will be␣
↪filled with baselines data. Users should then call the function␣
↪Pixie16ReadSglChanBaselines to read the baselines data out to the host computer,␣
↪channel by channel.
int Pixie16AcquireBaselines (
      unsigned short ModNum );// module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Failed to start the GET_BASELINES run**
  - Reboot the modules
- **-3 GET_BASELINES run timed out**
  - Reboot the modules

Usage example

```
unsigned short ModNum;
int retval;

// assume we want to acquire baselines for module 0
ModNum = 0;

// acquire the trace
retval = Pixie16AcquireBaselines (ModNum);
if(retval < 0)
{
// error handling
}
```

21.1.3 Pixie16AdjustOffsets

Adjust DC-offsets in single or multiple modules

```
// Use this function to adjust the DC-offsets of Pixie-16 modules. Specify the␣
↪module using ModNum which starts counting at 0. If ModNum is set to be less than␣
↪the total number of modules in the system, only the module specified by ModNum␣
↪will have its DC-offsets adjusted. But if ModNum is set to be equal to the total␣
↪number of modules in the system, then all modules in the system will have their␣
↪DC-offsets adjusted.
// After the DC-offset levels have been adjusted, the baseline level of the␣
↪digitized input signals will be determined by the DSP parameter BaselinePercent.␣
↪For instance, if BaselinePercent is set to 10(%), the baseline level of the␣
↪input signals will be ~409 on the 12-bit ADC scale (minimum: 0; maximum: 4095).
// The main purpose of this function is to ensure the input signals fall within␣
↪the voltage range of the ADCs so that all input signals can be digitized by the␣
↪ADCs properly.
int Pixie16AdjustOffsets (
      unsigned short ModNum ); // module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Check if ModNum is valid
- **-2 Failed to start the ADJUST_OFFSETS run**
  - Reboot the module
- **-3 ADJUST_OFFSETS run timed out**
  - Check error message log file Pixie16msg.txt

Usage example

```
int retval;
unsigned short ModNum;

// set to the first module
ModNum = 0;

// adjust dc-offsets
retval = Pixie16AdjustOffsets(ModNum);
if(retval < 0)
{
// error handling
}
```

21.1.4 Pixie16BLcutFinder

Compute new Baseline Cut values of a module

```
// Use this function to find the Baseline Cut value for one channel of a Pixie-16␣
↪module. The baseline cut value is then downloaded to the DSP, where baselines␣
↪are captured and averaged over time. The cut value would prevent a bad baseline␣
↪value from being used in the averaging process, i.e., if a baseline value is␣
↪outside the baseline cut range, it will not be used for computing the baseline␣
↪average. Averaging baselines over time improves energy resolution measurement.
// ModNum is the module number which starts counting at 0. ChanNum is the channel␣
↪number which starts counting at 0.
```

(下页继续)

(续上页)

```
int Pixie16BLcutFinder (
      unsigned short ModNum, // module number
      unsigned short ChanNum,// channel number
      unsigned int *BLcut ); // returned BLcut value
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Check if ModNum is valid
- **-2 Failed to collect baselines**
  - Reboot the module
- **-3 Failed to read baselines from the data memory**
  - Check error message log file Pixie16msg.txt

Usage example

```
int retval;
unsigned short ModNum, ChanNum;
unsigned int BLcut;

// set to the first module
ModNum = 0;

// set to the first channel
ChanNum = 0;

// find the BLcut value
retval = Pixie16BLcutFinder(ModNum, ChanNum, &BLcut);
if(retval < 0)
{
// error handling
}
```

21.1.5 Pixie16BootModule

Boot modules so that they can be set up for data taking

```
// Use this function to boot Pixie-16 modules so that they can be set up for data␣
↪taking. The function downloads to the Pixie-16 modules the communication (or␣
↪system) FPGA configurations, signal processing FPGA configurations, trigger FPGA␣
↪configurations (Revision A modules only), executable code for the digital signal␣
↪processor (DSP), and DSP parameters.
// The FPGA configurations consist of a fixed number of words depending on the␣
↪hardware mounted on the modules; the DSP codes have a length which depends on␣
↪the actual compiled code; and the set of DSP parameters always consists of 1280␣
↪32-bit words for each module. The host software has to make the names of those␣
↪boot data files on the hard disk available to the boot function.
// ModNum is the module number which starts counting at 0. If ModNum is set to be␣
↪less than the total number of modules in the system, only the module specified␣
↪by ModNum will be booted. But if ModNum is equal to the total number of modules␣
↪in the system, e.g. there are 5 modules in the chassis and ModNum = 5, then all␣
↪modules in the system will be booted.
// The boot pattern is a bit mask (shown below) indicating which on-board chip␣
↪will be booted. Under normal circumstances, all on-board chips should be booted,␣
↪i.e. the boot pattern would be 0x7F. For Rev-B, C, D, F modules, bit 1, i.e.
```

(下页继续)

```
"Boot trigger FPGA" , will be ignored even if that bit is set to 1.
```

(续上页)

```
// bit 0: Boot communication FPGA, All Pixie-16 Revisions
// bit 1: Boot trigger FPGA, Pixie-16 Revision A only
// bit 2: Boot signal processing FPGA, All Pixie-16 Revisions
// bit 3: Boot digital signal processor (DSP), All Pixie-16 Revisions
// bit 4: Download DSP I/O parameters, All Pixie-16 Revisions
// bit 5: Program on-board FPGAs, All Pixie-16 Revisions
// bit 6: Set on-board DACs, All Pixie-16 Revisions

int Pixie16BootModule (
      char *ComFPGAConfigFile,      // name of ComFPGA configuration file
      char *SPFPGAConfigFile,       // name of SPFPGA configuration file
      char *TrigFPGAConfigFile,     // name of trigger FPGA file
      char *DSPCodeFile,            // name of DSP code file
      char *DSPParFile,             // name of DSP parameter file
      char *DSPVarFile,             // name of DSP variable names file
      unsigned short ModNum,        // pixie-16 module number
      unsigned short BootPattern); // boot pattern bit mask
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Size of ComFPGAConfigFile is invalid**
  - Correct ComFPGAConfigFile
- **-3 Failed to boot Communication FPGA**
  - Check log file (Pixie16msg.txt)
- **-4 Failed to allocate memory to store data in ComFPGAConfigFile**
  - Close other programs or reboot the computer
- **-5 Failed to open ComFPGAConfigFile**
  - Correct ComFPGAConfigFile
- **-10 Size of SPFPGAConfigFile is invalid**
  - Correct SPFPGAConfigFile
- **-11 Failed to boot signal processing FPGA**
  - Check log file (Pixie16msg.txt)
- **-12 Failed to allocate memory to store data in SPFPGAConfigFile**
  - Close other programs or reboot the computer
- **-13 Failed to open SPFPGAConfigFile**
  - Correct SPFPGAConfigFile
- **-14 Failed to boot DSP**
  - Check log file (Pixie16msg.txt)
- **-15 Failed to allocate memory to store DSP executable code**
  - Close other programs or reboot the computer
- **-16 Failed to open DSPCodeFile**
  - Correct DSPCodeFile

- **-17 Size of DSPParFile is invalid**
  - Correct DSPParFile
- **-18 Failed to open DSPParFile**
  - Correct DSPParFile
- **-19 Can’t initialize DSP variable indices**
  - Correct DSPVarFile
- **-20 Can’t copy DSP variable indices**
  - Check log file (Pixie16msg.txt)
- **-21 Failed to program Fippi in a module**
  - Check log file (Pixie16msg.txt)
- **-22 Failed to set DACs in a module**
  - Check log file (Pixie16msg.txt)
- **-23 Failed to start RESET_ADC run in a module**
  - Check log file (Pixie16msg.txt)
- **-24 RESET_ADC run timed out in a module**
  - Check log file (Pixie16msg.txt)

Usage example

```
int retval;
char ComFPGAConfigFile[256], SPFPGAConfigFile[256];
char TrigFPGAConfigFile[256], DSPCodeFile[256], DSPParFile[256];
char DSPVarFile[256];
unsigned short ModNum, BootPattern;

// first, specify file names and paths for all boot data files
sprintf(ComFPGAConfigFile, "C:\\XIA\\Pixie16\\Firmware\\syspixie16.bin");
sprintf(TrigFPGAConfigFile, " ");
sprintf(SPFPGAConfigFile, "C:\\XIA\\Pixie16\\Firmware\\fippixie16.bin");
sprintf(DSPCodeFile, "C:\\XIA\\Pixie16\\DSP\\Pixie16DSP.ldr");
sprintf(DSPParFile, "C:\\XIA\\Pixie16\\Configuration\\default.set");
sprintf(DSPVarFile, "C:\\XIA\\Pixie16\\DSP\\Pixie16DSP.var");

// second, select the Pixie-16 module to boot. All modules in the system
// can be booted either one-by-one or all at once. The simplest way to
// boot all modules at once is to set ModNum to be equal to the total
// number of modules in the system. Here we assume there are total 5
// Pixie-16 modules.
ModNum = 5;

// third, specify the boot pattern. Normally, it should be 0x7F, i.e.
// all bits of the bit mask are selected.
BootPattern = 0x7F;

// finally, boot the selected modules
retval = Pixie16BootModule (ComFPGAConfigFile, SPFPGAConfigFile,
TrigFPGAConfigFile, DSPCodeFile, DSPParFile, DSPVarFile, ModNum,
BootPattern);
if(retval < 0)
{
// error handling
}
```

21.1.6 Pixie16CheckExternalFIFOStatus

Check status of external FIFO of a module

```
// Use this function to check the status of the external FIFO of a Pixie-16 module␣
↪while a list mode data acquisition run is in progress. The function returns the␣
↪number of words (32-bit) that the external FIFO currently has. If the number of␣
↪words is greater than a user-set threshold, function␣
↪Pixie16ReadDataFromExternalFIFO can then be used to read the data from the␣
↪external FIFO.
// The threshold can be set by the user to either minimize reading overhead or to␣
↪read data out of the FIFO as quickly as possible. The Pixie-16 API (pixie16app_
↪defs.h) has defined a threshold with value of 1024 for external FIFO read out␣
↪(EXTFIFO_READ_THRESH).
// *nFIFOWords returns the number of 32-bit words that the external FIFO currently␣
↪has.
// ModNum is the module number which starts counting at 0.
int Pixie16CheckExternalFIFOStatus (
      unsigned int *nFIFOWords,  // returned number of words in the FIFO
      unsigned short ModNum );   // module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum

Usage example

```
int retval;
unsigned int nFIFOWords;
unsigned short ModNum;

ModNum = 0;
// the first module
retval = Pixie16CheckExternalFIFOStatus (&nFIFOWords, ModNum);
```

21.1.7 Pixie16CheckRunStatus

Check status of a data acquisition run

```
// Use this function to check the run status of a Pixie-16 module while a list␣
↪mode data acquisition run is in progress. If the run is still in progress␣
↪continue polling.
// If the return code of this function indicates the run has finished, there might␣
↪still be some data in the external FIFO (Rev-B, C, D, F modules) that need to be␣
↪read out to the host computer. In addition, final run statistics and histogram␣
↪data are available for reading out too.
// In MCA histogram run mode, this function can also be called to check if the run␣
↪is still in progress even though it is normally self-terminating.
// ModNum is the module number which starts counting at 0.
int Pixie16CheckRunStatus (
      unsigned short ModNum ); // module number
```

Return values description and error handling

- **0 No run is in progress**
  - None

- **1 Run is still in progress**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum

Usage example

```
int retval;
unsigned short ModNum;

ModNum = 0;
// the first module
retval = Pixie16CheckRunStatus(ModNum);
if(retval < 0)
{
// invalid module number
}
else if(retval == 1)
{
// run is still in progress
}
else if(retval == 0)
{
// run has finished
}
```

21.1.8 Pixie16ComputeFastFiltersOffline

Compute fast filter response for offline analysis

```
// Use this function to compute fast filter responses of each event in the list␣
↪mode data file for offline analysis. The algorithm implemented in this offline␣
↪analysis function is equivalent to the one implemented in the Pixie-16 hardware.␣
↪So this function can be used to analyze how the leading edge fast trigger filter␣
↪and the CFD filter implemented in the hardware respond to the shape of recorded␣
↪traces. By analyzing the response of these filters, it is possible to optimize␣
↪the performance of the leading edge trigger or CFD trigger by adjusting the fast␣
↪filter and CFD parameters offline. Such optimized parameters can then be saved␣
↪to a settings file to be used for online data acquisition.
int Pixie16ComputeFastFiltersOffline (
      char *FileName, // the list mode data file name (with complete path)
      unsigned short ModuleNumber, // the module to be analyzed
      unsigned short ChannelNumber, // the channel to be analyzed
      unsigned int FileLocation, // the location of the trace in the file
      unsigned short RcdTraceLength,// recorded trace length
      unsigned short *RcdTrace, // recorded trace
      double *fastfilter, // fast filter response
      double *cfd ); // cfd response
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Null pointer RcdTrace**
  - Make sure RcdTrace has been allocated enough memory (maximum possible length is 32768, but can be shorter to match the actual trace length)
- **-2 Null pointer fastfilter**

  - Make sure fastfilter has been allocated enough memory (maximum possible length is 32768, but can be shorter to match the actual trace length)

- **-3 Null pointer cfd**
  - Make sure cfd has been allocated enough memory (maximum possible length is 32768, but can be shorter to match the actual trace length)
- **-4 Target module number is invalid**
  - Correct ModuleNumber
- **-5 Trace length is too short**
  - The length of recorded trace is too short for the offline analysis using the fast filter length (fast filter rise time and flat top). Either a different settings file with shorter fast filter length has to be used, or new traces with longer trace length have to be acquired
- **-6 Failed to open list mode data file**
  - Check the list mode file name and its path are correct

Usage example

```
int retval;
unsigned short ModuleNumber, ChannelNumber;
unsigned int FileLocation;
unsigned short RcdTraceLength;
unsigned short RcdTrace[1000];
double fastfilter[1000];
double cfd[1000];

char *FileName = {"C:\\XIA\\Pixie16\\PulseShape\\listmodedata.bin"};
ModuleNumber = 0;       //the first module
ChannelNumber = 0;      //the first channel
FileLocation = 16;      //the starting address of the trace in the list mode data␣
↪file (32-bit word address)
RcdTraceLength = 1000; //the length of the recorded trace

retval = Pixie16ComputeFastFiltersOffline (FileName, ModuleNumber,
ChannelNumber, FileLocation, RcdTrace, fastfilter, cfd);
if(retval < 0)
{
// error handling
}
```

21.1.9 Pixie16ComputeInputCountRate

Compute input count rate of a channel

```
// Use this function to calculate the input count rate on one channel of a Pixie-␣
↪16 module. This function does not communicate with Pixie-16 modules. Before␣
↪calling this function, another function, Pixie16ReadStatisticsFromModule, should␣
↪be called to read statistics data from the module.
// *Statistics is a pointer to an array whose size is exactly 448 unsigned integer␣
↪words (32-bit). The *Statistics array is filled with data from a Pixie-16 module␣
↪after calling function Pixie16ReadStatisticsFromModule. ModNum is the module␣
↪number which starts counting at 0. ChanNum is the channel number which starts␣
↪counting at 0.
double Pixie16ComputeInputCountRate (
      unsigned int *Statistics,// run statistics data
      unsigned short ModNum,   // module number
      unsigned short ChanNum );// channel number
```

Return values description and error handling

The return value is the input count rate in counts per second.

Usage example

```
double icr;
unsigned int Statistics[448];
unsigned short ModNum, ChanNum;
int retval;

ModNum = 0;  // the first module
ChanNum = 0; // the first channel

// first call Pixie16ReadStatisticsFromModule to get the statistics data
retval = Pixie16ReadStatisticsFromModule (Statistics, ModNum);
if(retval < 0)
{
// error handling
}

// compute input count rate
icr = Pixie16ComputeInputCountRate (Statistics, ModNum, ChanNum);
```

21.1.10 Pixie16ComputeLiveTime

Compute live time that a channel accumulated in a run

```
// Use this function to calculate the live time that one channel of a Pixie-16␣
↪module has spent on data acquisition. This function does not communicate with␣
↪Pixie-16 modules. Before calling this function, another function,␣
↪Pixie16ReadStatisticsFromModule, should be called to read statistics data from␣
↪the module.
// *Statistics is a pointer to an array whose size is exactly 448 unsigned integer␣
↪words (32-bit). The *Statistics array is filled with data from a Pixie-16 module␣
↪after calling function Pixie16ReadStatisticsFromModule. ModNum is the module␣
↪number which starts counting at 0. ChanNum is the channel number which starts␣
↪counting at 0.
double Pixie16ComputeLiveTime (
      unsigned int *Statistics, // run statistics data
      unsigned short ModNum,    // module number
      unsigned short ChanNum ); // channel number
```

Return values description and error handling

The return value is the live time in seconds.

Usage example

```
double livetime;
unsigned int Statistics[448];
unsigned short ModNum, ChanNum;
int retval;

ModNum = 0; // the first module
ChanNum = 0;// the first channel

// first call Pixie16ReadStatisticsFromModule to get the statistics data
retval = Pixie16ReadStatisticsFromModule (Statistics, ModNum);
if(retval < 0)
{
// error handling
```

(下页继续)

(续上页)

```
}

// compute live time
livetime = Pixie16ComputeLiveTime (Statistics, ModNum, ChanNum);
```

21.1.11 Pixie16ComputeOutputCountRate

Compute output count rate of a channel

```
// Use this function to calculate the output count rate on one channel of a Pixie-
↪16 module. This function does not communicate with Pixie-16 modules. Before␣
↪calling this function, another function, Pixie16ReadStatisticsFromModule, should␣
↪be called to read statistics data from the module.
// *Statistics is a pointer to an array whose size is exactly 448 unsigned integer␣
↪words (32-bit). The *Statistics array is filled with data from a Pixie-16 module␣
↪after calling function Pixie16ReadStatisticsFromModule. ModNum is the module␣
↪number which starts counting at 0. ChanNum is the channel number which starts␣
↪counting at 0.
double Pixie16ComputeOutputCountRate (
      unsigned int *Statistics, // run statistics data
      unsigned short ModNum,    // module number
      unsigned short ChanNum ); // channel number
```

Return values description and error handling

The return value is the output count rate in counts per second.

Usage example

```
double ocr;
unsigned int Statistics[448];
unsigned short ModNum, ChanNum;
int retval;

ModNum = 0;  // the first module
ChanNum = 0; // the first channel

// first call Pixie16ReadStatisticsFromModule to get the statistics data
retval = Pixie16ReadStatisticsFromModule(Statistics, ModNum);
if(retval < 0)
{
// error handling
}

// compute output count rate
ocr = Pixie16ComputeOutputCountRate(Statistics, ModNum, ChanNum);
```

21.1.12 Pixie16ComputeProcessedEvents

Compute number of events processed by a module

```
// Use this function to calculate the number of events that have been processed by␣
↪a Pixie-16 module during a data acquisition run. This function is only used by␣
↪Rev-A modules. This function does not communicate with Pixie-16 modules. Before␣
↪calling this function, another function, Pixie16ReadStatisticsFromModule, should␣
↪be called to read statistics data from the module first.
// *Statistics is a pointer to an array whose size is exactly 448 unsigned integer␣
↪words (32-bit). The *Statistics array is filled with data from a Pixie-16 module␣
↪after calling function Pixie16ReadStatisticsFromModule. ModNum is the module␣
↪number which starts counting at 0. ChanNum is the channel number which starts␣
↪counting at 0.
```

(下页继续)

(续上页)

```
double Pixie16ComputeProcessedEvents (
      unsigned long *Statistics,// run statistics data
      unsigned short ModNum );  // module number
```

Return values description and error handling

The return value is the number of processed events.

Usage example

```
double NumEvents;
unsigned long Statistics[448];
unsigned short ModNum;
int retval;

ModNum = 0;// the first module

// first call Pixie16ReadStatisticsFromModule to get the statistics data
retval = Pixie16ReadStatisticsFromModule(Statistics, ModNum);
if(retval < 0)
{
// error handling
}

// compute number of processed events
NumEvents = Pixie16ComputeProcessedEvents(Statistics, ModNum);
```

21.1.13 Pixie16ComputeRealTime

Compute real time that a module accumulated in a run

```
// Use this function to calculate the real time that a Pixie-16 module has spent␣
↪on data acquisition. This function does not communicate with Pixie-16 modules.␣
↪Before calling this function, another function, Pixie16ReadStatisticsFromModule,␣
↪should be called to read statistics data from the module.
// *Statistics is a pointer to an array whose size is exactly 448 unsigned integer␣
↪words (32-bit). The *Statistics array is filled with data from a Pixie-16 module␣
↪after calling function Pixie16ReadStatisticsFromModule. ModNum is the module␣
↪number which starts counting at 0. ChanNum is the channel number which starts␣
↪counting at 0.
double Pixie16ComputeRealTime (
      unsigned int *Statistics,// run statistics data
      unsigned short ModNum ); // module number
```

Return values description and error handling

The return value is the real time in seconds.

Usage example

```
double realtime;
unsigned int Statistics[448];
unsigned short ModNum;
int retval;
ModNum = 0; // the first module

// first call Pixie16ReadStatisticsFromModule to get the statistics data
retval = Pixie16ReadStatisticsFromModule(Statistics, ModNum);
if(retval < 0)
{
// error handling
```

(下页继续)

(续上页)

```
}

// compute real time
realtime = Pixie16ComputeRealTime(Statistics, ModNum);
```

21.1.14 Pixie16ComputeSlowFiltersOffline

Compute slow filter response for offline analysis

```
// Use this function to compute slow filter responses of each event in the list␣
↪mode data file for offline analysis. The algorithm implemented in this offline␣
↪analysis function is equivalent to the one implemented in the Pixie-16 hardware.␣
↪So this function can be used to analyze how the slow filter implemented in the␣
↪hardware for computing event energy responds to the shape of recorded traces. By␣
↪analyzing the responses of these filters, it is possible to optimize the␣
↪performance of the slow filter by adjusting its parameters offline. Such␣
↪optimized parameters can then be saved to a settings file to be used for online␣
↪data acquisition.
int Pixie16ComputeSlowFiltersOffline (
      char *FileName, // the list mode data file name (with complete path)
      unsigned short ModuleNumber, // the module to be analyzed
      unsigned short ChannelNumber, // the channel to be analyzed
      unsigned int FileLocation, // the location of the trace in the file
      unsigned short RcdTraceLength,// recorded trace length
      unsigned short *RcdTrace,// recorded trace
      double *slowfilter);// slow filter response
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Null pointer RcdTrace**
  - Make sure RcdTrace has been allocated enough memory (maximum possible length is 32768, but can be shorter to match the actual trace length)
- **-2 Null pointer slowfilter**
  - Make sure slowfilter has been allocated enough memory (maximum possible length is 32768, but can be shorter to match the actual trace length)
- **-3 Target module number is invalid**
  - Correct ModuleNumber
- **-4 Trace length is too short**
  - The length of recorded trace is too short for the offline analysis using the slow filter length (slow filter rise time and flat top). Either a different settings file with shorter slow filter length has to be used, or new traces with longer trace length have to be acquired
- **-5 Failed to open list mode data file**
  - Check the list mode file name and its path are correct

Usage example

```
int retval;
unsigned short ModuleNumber, ChannelNumber;
unsigned int FileLocation;
unsigned short RcdTraceLength;
unsigned short RcdTrace[1000];
```

(下页继续)

(续上页)

```
double slowfilter[1000];

char *FileName = {"C:\\XIA\\Pixie16\\PulseShape\\listmodedata.bin"};

ModuleNumber = 0;      // the first module
ChannelNumber = 0;     // the first channel
FileLocation = 16;     // the starting address of the trace in the list mode data␣
↪file (32-bit word address)
RcdTraceLength = 1000;// the length of the recorded trace

retval = Pixie16ComputeSlowFiltersOffline (FileName, ModuleNumber,
ChannelNumber, FileLocation, RcdTrace, slowfilter);
if(retval < 0)
{
// error handling
}
```

21.1.15 Pixie16ControlTaskRun

Execute special control tasks

```
// Use this function to execute special control tasks. This may include␣
↪programming the Fippi or setting the DACs after downloading DSP parameters.
// ModNum is the module number which starts counting at 0. ChanNum is the channel␣
↪number which starts counting at 0.
int Pixie16ControlTaskRun (
      unsigned short ModNum,      // module number
      unsigned short ControlTask,// control task run type
      unsigned int Max_Poll );    // Timeout control in unit of ms
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 The control task run failed**
  - Reboot the module

Usage example

```
int retval;
unsigned short ControlTask, ModNum;
unsigned int Max_Poll;

ModNum = 0;             // this is the first module
ControlTask = SET_DACS;// constant SET_DACS defined in header file
Max_Poll = 10000;       // give the run 10 seconds to finish

retval = Pixie16ControlTaskRun (ControlTask, ModNum, Max_Poll);
if(retval < 0)
{
// Error handling
}
```

21.1.16 Pixie16CopyDSPParameters

Copy DSP parameters from a module to others

```
// Use this function to copy DSP parameters from one module to the others that are␣
↪installed in the system.
// BitMask is bit pattern which designates which items should be copied from the␣
↪source module to the destination module(s).
// Bit 0: Filter (energy and trigger)
// Bit 1: Analog signal conditioning (polarity, dc-offset, gain/attenuation)
// Bit 2: Histogram control (minimum energy, binning factor)
// Bit 3: Decay time
// Bit 4: Pulse shape analysis (trace length and trace delay)
// Bit 5: Baseline control (baseline cut, baseline percentage)
// Bit 7: Channel CSRA register (good channel, trigger enabled, etc.)
// Bit 8: CFD trigger (CFD delay, scaling factor)
// Bit 9: Trigger stretch lengths (veto, external trigger, etc.)
// Bit 10: FIFO delays (analog input delay, fast trigger output delay, etc.)
// Bit 11: Multiplicity (bit masks, thresholds, etc.)
// Bit 12: QDC (QDC sum lengths)
// SourceModule and SourceChannel is the module and channel number of the source␣
↪of the DSP parameters which are to be copied to other modules and channels.
// DestinationMask is an array which indicates the channel and module whose␣
↪settings will be copied from the source channel and module. For instance, if␣
↪there are 5 modules (total 80 channels) in the system, DestinationMask would be␣
↪defined as DestinationMask[80], where DestinationMask[0] to DestinationMask[15]␣
↪would select channel 0 to 15 of module 0, DestinationMask[16] to␣
↪DestinationMask[31] would select channel 0 to 15 of module 1, and so on. If a␣
↪given channel i is to be copied, then DestinationMask[i] should be set to 1,␣
↪otherwise, it should be set to 0.
int Pixie16CopyDSPParameters (
      unsigned short BitMask,           // copy items bit mask
      unsigned short SourceModule,      // source module
      unsigned short SourceChannel,     // source channel
      unsigned short *DestinationMask );// destination bit mask
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Failed to program Fippi in a module**
  - Reboot the modules
- **-2 Failed to set DACs in a module**
  - Reboot the modules

Usage example

```
int retval;
unsigned short BitMask;             //copy items bit mask
unsigned short SourceModule;        //source module
unsigned short SourceChannel;       //source channel
unsigned short DestinationMask[384];//destination bit mask
unsigned short k;

BitMask = 0x1FF;  // copy everything
SourceModule = 0; // the first module
SourceChannel = 0;// the first channel

// assume there are 6 Pixie-16 modules in the system
for(k = 0; k < (6 * 16); k ++)
```

(下页继续)

(续上页)

```
{
DestinationMask[k] = 1; // copy to all channels of all modules
}

retval = Pixie16CopyDSPParameters (BitMask, SourceModule, SourceChannel,
DestinationMask);
if(retval < 0)
{
// error handling
}
```

21.1.17 Pixie16EMbufferIO

Transfer data between host and DSP external memory

```
// Use this function to directly read data from or write data to the on-board␣
↪external memory of a Pixie-16 module. The valid memory address is from 0x0 to␣
↪0x7FFFF (32-bit wide). Use Direction = 1 for read and Direction = 0 for write.
// The external memory is used to store the histogram data accumulated for each of␣
↪the 16 channels of a Pixie-16 module. Each channel has a fixed histogram length␣
↪of 32768 words (32-bit wide), and the placement of the histogram data in the␣
↪memory is in the same order of the channel number, i.e. channel 0 occupies␣
↪memory address 0x0 to 0x7FFF, channel 1 occupies 0x8000 to 0xFFFF, and so on.
// NOTE: another function Pixie16ReadHistogramFromModule can also be used to read␣
↪out the histograms except that it needs to be called channel by channel.
// ModNum is the module number which starts counting at 0.
int Pixie16EMbufferIO (
      unsigned int *Buffer,// buffer data
      unsigned int NumWords, // number of buffer words to read or write
      unsigned int Address,// buffer address
      unsigned short Direction, // I/O direction
      unsigned short ModNum ); //module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 null pointer for buffer data**
  - Correct pointer Buffer
- **-2 number of buffer words exceeds the limit**
  - Reduce the number of buffer words
- **-3 invalid DSP external memory address**
  - Use the valid address
- **-4 invalid I/O direction**
  - Use the valid direction
- **-5 invalid Pixie-16 module number**
  - Correct the ModNum
- **-6 I/O Failure**
  - Reboot the module

Usage example

```
int retval;
unsigned short Direction, ModNum;
unsigned int MCAData[32768], NumWords, Address;
NumWords = 32768;//to read out the MCA data from channel 0
ModNum = 0;     //the first module in the system
Address = 0x0;          //the starting memory address
Direction = 1;          //I/O direction is read

// read out the MCA data for Channel 0
retval = Pixie16EMbufferIO (MCAData, NumWords, Address, Direction, ModNum);
if(retval != 0)
{
// Error handling
}
```

21.1.18 Pixie16EndRun

Stop a data acquisition run

```
// Use this function to end a histogram run, or to force the end of a list mode␣
↪run. In a multi-module system, if all modules are running synchronously, only␣
↪one module needs to be addressed this way. It will immediately stop the run in␣
↪all other module in the system. ModNum is the module number which starts␣
↪counting at 0.
int Pixie16EndRun (
      unsigned short ModNum );// module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Failed to stop the run**
  - Reboot the module

Usage example

```
int retval;
unsigned short ModNum;
ModNum = 0;
// the first module
retval = Pixie16EndRun (ModNum);
if(retval < 0)
{
// error handling
}
```

21.1.19 Pixie16ExitSystem

Release user virtual addressees assigned to modules

```
// Use this function to release the user virtual addressees that are assigned to␣
↪Pixie-16 modules when these modules are initialized by function␣
↪Pixie16InitSystem. This function should be called before a user' s application␣
↪exits.
// If ModNum is set to less than the total number of modules in the system, only␣
↪the module specified by ModNum will be closed. But if ModNum is equal to the␣
↪total number of modules in the system, e.g. there are 5 modules in the chassis␣
↪and ModNum = 5, then all modules in the system will be closed altogether. Note␣
↪that the modules are counted starting at 0.
int Pixie16ExitSystem (
      unsigned short ModNum ); // Pixie-16 module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum (it should not be greater than the total number of modules in the system)
- **-2 Failed to close Pixie-16 modules**
  - Check error message log file Pixie16msg.txt

Usage example

```
int retval;
unsigned short ModNum, k;

// assume there are totally 5 modules in the system
// so close all 5 modules
ModNum = 5;
retval = Pixie16ExitSystem (ModNum);
if(retval < 0)
{
// error handling
}

// or module by module
for(k=0; k<5; k++)
{
  retval = Pixie16ExitSystem (k);
  if(retval < 0)
  {
  // error handling
  }
}
```

21.1.20 Pixie16GetEventsInfo

Get detailed events information from a data file

```
// Use this function to retrieve the detailed information (except waveforms) of␣
↪each event in the list mode data file for the designated module. ModNum is the␣
↪module number which starts counting at 0. ChanNum is the channel number which␣
↪starts counting at 0.
// Before calling this function to get the individual events information, another␣
↪function Pixie16GetModuleEvents should be called first to determine the number␣
↪of events that have been recorded for each module. If the number of events for a␣
↪given module is nEvents, a memory block *EventInformation should be allocated␣
↪with a length of (nEvents*68):
```

(下页继续)

(续上页)

```
// EventInformation = (unsigned int *)malloc(sizeof(unsigned int) * nEvents * 68);
// where 68 is the length of the information records of each event (energy,␣
↪timestamps, etc.) and has the following structure.
// EventInformation [0] Event number
// EventInformation [1] Channel number
// EventInformation [2] Slot number
// EventInformation [3] Crate number
// EventInformation [4] Header length
// EventInformation [5] Event length
// EventInformation [6] Finish code
// EventInformation [7] Event timestamp (lower 32-bit)
// EventInformation [8] Event timestamp (upper 16-bit)
// EventInformation [9] Event energy
// EventInformation [10] Trace length
// EventInformation [11] Trace location
// EventInformation [67:12] Not used
int Pixie16GetEventsInfo (
      char *FileName,// the list mode data file name (with complete path)
      unsigned int *EventsInformation,// to hold event information
      unsigned short ModuleNumber );// module to get events from
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Null pointer EventInformation**
  - Correct EventInformation
- **-2 Invalid Pixie-16 module number**
  - Correct ModuleNumber
- **-3 Failed to open list mode data file**
  - Correct file name and path

Usage example

```
int retval;
char *FileName = {"C:\\XIA\\Pixie16\\PulseShape\\listmodedata.bin"};
unsigned short ModuleNumber;
unsigned int *EventInformation;
unsigned int ModuleEvents[7]; // assume maximum number of modules is 7

retval = Pixie16GetModuleEvents (FileName, ModuleEvents);
if(retval < 0)
{
// error handling
}

ModuleNumber = 0;// the first module
EventInformation = (unsigned int *)malloc(sizeof(unsigned int) *
                   ModuleEvents[ModuleNumber] * 68);
retval = Pixie16GetEventsInfo(FileName, EventInformation, ModuleNumber);
if(retval < 0)
{
// error handling
}
```

21.1.21 Pixie16GetModuleEvents

Parse a list mode data file to get events information

```
// Use this function to parse the list mode events in the list mode data file. The␣
↪number of events for each module will be reported.
int Pixie16GetModuleEvents (
      char *FileName, // the list mode data file name (with complete path)
      unsigned int *ModuleEvents ); // receives number of events for modules
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Null pointer ModuleEvents**
  - Correct ModuleEvents
- **-2 Failed to open list mode data file**
  - Correct file name and path

Usage example

```
int retval;
char *FileName = {"C:\\XIA\\Pixie16\\PulseShape\\listmodedata.bin"};
unsigned int ModuleEvents[7];// assume maximum number of modules is 7

retval = Pixie16GetModuleEvents (FileName, ModuleEvents);
if(retval < 0)
{
// error handling
}
```

21.1.22 Pixie16IMbufferIO

Transfer data between host and DSP internal memory

```
// Use this function to directly transfer data between the host and the DSP␣
↪internal memory of a Pixie-16 module. ModNum is the module number which starts␣
↪counting at 0.
// The DSP internal memory is split into two blocks with address range 0x40000 to␣
↪0x4FFFF for the first block and address range 0x50000 to 0x5FFFF for the second␣
↪block. Within the first block, address range 0x40000 to 0x49FFF is reserved for␣
↪program memory and shouldn' t be accessed directly by the host computer. Address␣
↪range 0x4A000 to 0x4A4FF is used by the DSP I/O parameters which are stored in␣
↪the configuration files (.set files) in the host. Within this range, 0x4A000 to␣
↪0x4A33F can be both read and written, but 0x4A340 to 0x4A4FF can only be read␣
↪but not written. The remaining address range (0x4A500 to 4FFFF) in the first␣
↪block and the entire second block (0x50000 to 0x5FFFF) should only be read but␣
↪not written by the host. Use Direction = 1 for read and Direction = 0 for write.
int Pixie16IMbufferIO (
      unsigned int *Buffer,     //buffer data
      unsigned int NumWords,    //number of buffer words to transfer
      unsigned int Address,     //buffer address
      unsigned short Direction,//I/O direction
      unsigned short ModNum ); //module number
```

Return values description and error handling

- **0 Success**

  - None
- **-1 Null pointer for buffer data**
  - Correct pointer Buffer
- **-2 Number of buffer words exceeds the limit**
  - Reduce the number of buffer words
- **-3 Invalid DSP internal memory address**
  - Use the valid address
- **-4 Invalid I/O direction**
  - Use the valid direction
- **-5 Invalid Pixie-16 module number**
  - Correct the ModNum
- **-6 I/O Failure**
  - Reboot the module

Usage example

```
int retval;
unsigned short Direction, ModNum;
unsigned int DSPMemBlock1[65536], NumWords, Address;

NumWords = 65536; //to read out block 1 of the DSP internal memory
ModNum = 0;      //the first module in the system
Address = 0x50000;//the starting address for block 1
Direction = 1;           //I/O direction is read

// read out the whole block 1 of the DSP internal memory
retval = Pixie16IMbufferIO (DSPMemBlock1, NumWords, Address, Direction, ModNum);
if(retval != 0)
{
// Error handling
}
```

21.1.23 Pixie16InitSystem

Configure modules for communication in PXI chassis

```
// Use this function to configure the Pixie-16 modules in the PXI chassis.
// NumModules is the total number of Pixie-16 modules installed in the system.␣
↪PXISlotMap is the pointer to an array that must have at least as many entries as␣
↪there are Pixie-16 modules in the chassis.
// PXISlotMap serves as a simple mapping of the logical module number and the␣
↪physical slot number that the modules reside in. The logical module number runs␣
↪from 0. For instance, in a system with 5 Pixie-16 modules, these 5 modules may␣
↪occupy slots 3 through 7. The user must fill PXISlotMap as follows: PXISlotMap =
↪{3, 4, 5, 6, 7 ...} since module number 0 resides in slot number 3, etc. To find␣
↪out in which slot a module is located, any piece of subsequent code can use the␣
↪expression PXISlotMap[ModNum], where ModNum is the logic module number.
// OfflineMode is used to indicate to the API whether the system is running in␣
↪OFFLINE mode (1) or ONLINE mode (0). OFFLINE mode is useful for situations where␣
↪no Pixie-16 modules are present but users can still test their calls to the API␣
↪functions in their application software.
// This function must be called as the first step in the boot process. It makes␣
↪the modules known to the system and "opens" each module for communication.
```

(下页继续)

(续上页)

```
// The function relies on an initialization file (pxisys.ini) that contains␣
↪information about the Host PC' s PCI buses, including the slot enumeration scheme.
↪ XIA' s software distribution normally puts this file under the same folder as␣
↪Pixie-16 software installation folder. However, the user has the flexibility of␣
↪putting it in other folders by simply changing the definition of the string␣
↪PCISysIniFile in the header part of the file pixie16sys.c.
int Pixie16InitSystem (
      unsigned short NumModules,    //total number of Pixie-16 modules
      unsigned short *PXISlotMap,   //an array containing the slot number of each␣
↪Pixie-16 module
      unsigned short OfflineMode );//specify if using offline mode
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid total number of Pixie-16 modules**
  - Check if NumModules <= PRESET_MAX_MODULES(defined in pixie16app_defs.h)
- **-2 Null pointer PXISlotMap**
  - Correct PXISlotMap
- **-3 Failed to initialize system**
  - Check error message log file Pixie16msg.txt

Usage example

```
int retval;
unsigned short NumModules, PXISlotMap[8], OfflineMode;

// there are 5 modules in the system
NumModules = 5;

// specify the slot number for each module
PXISlotMap[0] = 2;
PXISlotMap[1] = 3;
PXISlotMap[2] = 4;
PXISlotMap[3] = 5;
PXISlotMap[4] = 6;

// running in online mode
OfflineMode = 0;

// configure the PXI slots in the chassis
retval = Pixie16InitSystem (NumModules, PXISlotMap, OfflineMode);
if(retval < 0)
{
// error handling
}
```

21.1.24 Pixie16LoadDSPParametersFromFile

Load DSP parameters to modules from a file

```
// Use this function to read DSP parameters from a settings file and then download␣
↪the settings to Pixie-16 modules that are installed in the system. Each module␣
↪has exactly 1280 DSP parameter values (32-bit unsigned integers), and depending␣
↪on the value of PRESET_MAX_MODULES (defined in pixie16app_defs.h), the settings␣
↪file should have exactly (1280 * PRESET_MAX_MODULES * 4) bytes when stored on␣
↪the computer hard drive.
int Pixie16LoadDSPParametersFromFile (
      char *FileName ); // DSP parameters file name (with complete path)
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Size of DSPParFile is invalid**
  - Correct DSPParFile
- **-2 Failed to program Fippi in a module**
  - Reboot the modules
- **-3 Failed to set DACs in a module**
  - Reboot the modules
- **-4 Failed to open the DSP parameters file**
  - Correct the DSP parameters file name

Usage example

```
int retval;
char *FileName = {"C:\\XIA\\Pixie16\\Configuration\\test.set"};

retval = Pixie16LoadDSPParametersFromFile (FileName);
if(retval < 0)
{
// error handling
}
```

21.1.25 Pixie16ProgramFippi

Program on-board signal processing FPGAs

```
// Use this function to program the on-board signal processing FPGAs of the Pixie-␣
↪16 modules. After the host computer has written the DSP parameters to the DSP␣
↪memory, the DSP needs to write some of these parameters to the FPGAs. This␣
↪function makes the DSP perform that action. ModNum is the module number which␣
↪starts counting at 0.
int Pixie16ProgramFippi (
      unsigned short ModNum);// module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Failed to start the PROGRAM_FIPPI run**
  - Reboot the module

- **-3 PROGRAM_FIPPI run timed out**
  - Reboot the module

Usage example

```
int retval;
unsigned short ModNum;
ModNum = 0;// the first module

retval = Pixie16ProgramFippi (ModNum);
if(retval < 0)
{
// error handling
}
```

21.1.26 Pixie16ReadCSR

Read Control & Status Register value from a module

```
// Use this function to read the host Control & Status Register (CSR) value. This␣
↪register is unrelated to the DSP parameters ModCSRA/B, ChanCSRA/B. It is used to␣
↪control the operation of the module and read directly by the host. Direct␣
↪reading or writing by the host is not recommended, for example use functions␣
↪like Pixie16CheckRunStatus to poll the active bit.
int Pixie16ReadCSR (
      unsigned short ModNum,// module number
      unsigned int *CSR );  // returned CSR value
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum

Usage example

```
unsigned short ModNum;
unsigned int CSR;
ModNum = 0;// the first module
Pixie16ReadCSR (ModNum, &CSR);
```

21.1.27 Pixie16ReadDataFromExternalFIFO

Read data from external FIFO of a module

```
// Use this function to read data from the external FIFO of a module.
// This function reads list mode data from the external FIFO of a Pixie-16 module.␣
↪The data are 32-bit unsigned integers. Normally, function␣
↪Pixie16CheckExternalFIFOStatus is called first to see how many words the␣
↪external FIFO currently has, and then this function is called to read the data␣
↪from the FIFO. ModNum is the module number which starts counting at 0.
int Pixie16ReadDataFromExternalFIFO (
      unsigned int *ExtFIFO_Data,// to receive the external FIFO data
      unsigned int nFIFOWords,// number of words to read from FIFO
      unsigned short ModNum );// module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Check ModNum
- **-2 Failed to read data from the external FIFO**
  - Check error message log file Pixie16msg.txt

Usage example

```
int retval;
unsigned int nFIFOWords, *ExtFIFO_Data;
unsigned short ModNum;

ModNum = 0;// the first module in the system
retval = Pixie16CheckExternalFIFOStatus (&nFIFOWords, ModNum);
if(retval < 0)
{
// Error handling
}

if(nFIFOWords > 0) // Check if there is data in the external FIFO
{
  ExtFIFO_Data =
  (unsigned int *)malloc(sizeof(unsigned int) * nFIFOWords);
  retval = Pixie16ReadDataFromExternalFIFO(ExtFIFO_Data,nFIFOWords,ModNum);
  if(retval != 0)
  {
  // Error handling
  }
}
```

21.1.28 Pixie16ReadHistogramFromFile

Read histogram data from a histogram data file

```
// Use this function to read histogram data from a histogram data file. Before␣
↪calling this function, the host code should allocate appropriate amount of␣
↪memory to store the histogram data. The default histogram length is 32768.␣
↪Histogram data are 32-bit unsigned integers.
// Specify the module using ModNum and the channel on the module using ChanNum.␣
↪Note that both the modules and channels are counted starting at 0.
int Pixie16ReadHistogramFromFile (
      char *FileName,// histogram file name (with complete path)
      unsigned int *Histogram,// histogram data
      unsigned int NumWords,// number of words to be read out
      unsigned short ModNum,// module number
      unsigned short ChanNum);// channel number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Failed to open the histogram data file**
  - Correct the histogram data file name
- **-2 No histogram data is available for this channel**
  - Change the ModNum and ChanNum

Usage example

```
int retval;
char *FileName = {"C:\\XIA\\Pixie16\\MCA\\histogramdata.bin"};
unsigned short ModNum, ChanNum;
unsigned int NumWords, Histogram[32768];
ModNum = 0;  // the first module
ChanNum = 0; // the first channel
NumWords = 32768;

retval = Pixie16ReadHistogramFromFile (FileName, Histogram, NumWords,
ModNum, ChanNum);
if(retval < 0)
{
// error handling
}
```

21.1.29 Pixie16ReadHistogramFromModule

Read histogram data from a module

```
// Use this function to read out the histogram data from a Pixie-16 module's␣
↪histogram memory. Before calling this function, the host code should allocate␣
↪appropriate amount of memory to store the histogram data. The default histogram␣
↪length is 32768. Histogram data are 32-bit unsigned integers.
// Specify the module using ModNum and the channel on the module using ChanNum.␣
↪Note that both the modules and channels are counted starting at 0.
int Pixie16ReadHistogramFromModule(
      unsigned int *Histogram,//histogram data
      unsigned int NumWords,//number of words to be read out
      unsigned short ModNum,//module number
      unsigned short ChanNum);//channel number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Invalid Pixie-16 channel number**
  - Correct ChanNum
- **-3 Failed to get the histogram data**
  - Reboot the module

Usage example

```
int retval;
unsigned short ModNum, ChanNum;
unsigned int NumWords, Histogram[32768];

ModNum = 0; // the first module
ChanNum = 0;// the first channel
NumWords = 32768;

retval = Pixie16ReadHistogramFromModule (Histogram, NumWords, ModNum, ChanNum);
if(retval < 0)
{
```

(下页继续)

(续上页)

```
// error handling
}
```

21.1.30 Pixie16ReadListModeTrace

Read trace data from a list mode data file

```
// Use this function to retrieve list mode trace from a list mode data file. It␣
↪uses the trace length and file location information obtained from function␣
↪Pixie16GetEventsInfo for the selected event.
int Pixie16ReadListModeTrace (
      char *FileName,            //list mode data file name
      unsigned short *Trace_Data,//list mode trace data (16-bit words)
      unsigned short NumWords,   //number of 16-bit words to be read out
      unsigned int FileLocation);//the location of the trace in the file
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Failed to open list mode data file**
  - Correct file name and path

Usage example

```
int retval;
char *FileName = {"C:\\XIA\\Pixie16\\PulseShape\\listmodedata.bin"};
unsigned short ModuleNumber, ChannelNumber;
unsigned int *EventInformation, FileLocation, EventNumber;
unsigned short *Trace_Data, NumWords;
unsigned int ModuleEvents[7]; // assume maximum number of modules is 7

retval = Pixie16GetModuleEvents (FileName, ModuleEvents);
if(retval < 0)
{
// error handling
}
ModuleNumber = 0;// the first module
EventInformation = (unsigned int *)malloc(sizeof(unsigned int) *␣
↪ModuleEvents[ModuleNumber]*68);

retval = Pixie16GetEventsInfo (FileName, EventInformation, ModuleNumber);
if(retval < 0)
{
// error handling
}

ChannelNumber = 0;// the first channel
EventNumber = 0;  // the first event

NumWords = (unsigned short)EventInformation[EventNumber*68 + 10] * 2;

FileLocation = EventInformation[EventNumber*68 + 11];
Trace_Data = (unsigned short *)malloc(sizeof(unsigned short) * NumWords);
retval = Pixie16ReadListModeTrace (FileName,Trace_Data,NumWords,FileLocation);
if(retval < 0)
{
// error handling
}
```

21.1.31 Pixie16ReadModuleInfo

Read information about a module stored in the EEPROM

```
// Use this function to read information stored on each module, including its␣
↪revision, serial number, ADC bits and sampling rate. This should be done after␣
↪initializing the PCI communication. Information from the module can be used to␣
↪select the appropriate firmware, DSP, and configuration parameters files before␣
↪booting the module. ModNum is the module number which starts counting at 0.
int Pixie16ReadModuleInfo (
      unsigned short ModNum,        //module number
      unsigned short *ModRev,       //returned module revision
      unsigned int *ModSerNum,      //returned module serial number
      unsigned short *ModADCBits,   //returned module ADC bits
      unsigned short *ModADCMSPS );//returned module ADC sampling rate
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie module number**
  - Check ModNum
- **-2 Failed to read from I2C serial EEPROM**
  - Check error message log file Pixie16msg.txt

Usage example

```
int retval;
unsigned short ModuleNumber;
unsigned short ModRev;
unsigned int ModSerNum;
unsigned short ModADCBits;
unsigned short ModADCMSPS;

retval = Pixie16ReadModuleInfo (ModuleNumber, &ModRev, &ModSerNum,
&ModADCBits, &ModADCMSPS);
if(retval < 0)
{
// error handling
}
```

21.1.32 Pixie16ReadSglChanADCTrace

Read ADC trace data from a channel in a module

```
// Use this function to read ADC trace data from a Pixie-16 module. Before calling␣
↪this function, another function Pixie16AcquireADCTrace should be called to fill␣
↪the DSP internal memory first. Also, the host code should allocate appropriate␣
↪amount of memory to store the trace data. The ADC trace data length for each␣
↪channel is 8192. Since the trace data are 16-bit unsigned integers (for hardware␣
↪variants with less than 16-bit ADCs only the lower 12-bit or 14-bit contain real␣
↪data), two consecutive 16-bit words are packed into one 32-bit word in the DSP␣
↪internal memory. So for each channel, 4096 32-bit words are read out first from␣
↪the DSP, and then each 32-bit word is unpacked to form two 16-bit words.
// Specify the module using ModNum and the channel on the module using ChanNum.␣
↪Note that both the modules and channels are counted starting at 0.
int Pixie16ReadSglChanADCTrace (
      unsigned short *Trace_Buffer, //trace data
```

(下页继续)

(续上页)

```
    unsigned int Trace_Length,//number of trace data words to read
    unsigned short ModNum,//module number
    unsigned short ChanNum );//channel number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Invalid Pixie-16 channel number**
  - Correct ChanNum
- **-3 Invalid trace length**
  - Correct Trace_Length
- **-4 Failed to allocate memory to store ADC traces**
  - Close other programs or reboot the computer
- **-5 Failed to read ADC traces**
  - Reboot the module

Usage example

```
unsigned short NumWords, ModNum, ChanNum;
int retval;
unsigned short ADCTrace[8192];
// assume we want to acquire ADC trace from channel 0 of module 0
ModNum = 0;
ChanNum = 0;

// number of ADC trace words is 8192
NumWords = 8192;

// acquire the trace
retval = Pixie16AcquireADCTrace(ModNum);
if(retval < 0)
{
// error handling
}

// read out the trace
retval = Pixie16ReadSglChanADCTrace(ADCTrace, NumWords, ModNum, ChanNum);
if(retval < 0)
{
// error handling
}
```

21.1.33 Pixie16ReadSglChanBaselines

Read baselines from a channel in a module

```
// Use this function to read baseline data from a Pixie-16 module. Before calling␣
↪this function, another function Pixie16AcquireBaselines should be called to fill␣
↪the DSP internal memory first. Also, the host code should allocate appropriate␣
↪amount of memory to store the baseline data. The number of baselines for each␣
↪channel is 3640. In the DSP internal memory, each baseline is a 32-bit IEEE␣
↪floating point number. After being read out to the host, this function will␣
↪convert each baseline data to a decimal number. In addition to baseline values,␣
↪timestamps corresponding to each baseline are also returned after this function␣
↪call.
// Specify the module using ModNum and the channel on the module using ChanNum.␣
↪Note that the modules and channels are counted starting at 0.
int Pixie16ReadSglChanBaselines(
      double *Baselines,       //returned baseline values
      double *TimeStamps,      //timestamps for each baseline value
      unsigned short NumBases, //number of baseline data words to read
      unsigned short ModNum,   //module number
      unsigned short ChanNum );//channel number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Requested number of baselines exceeded the limit**
  - Correct NumBases
- **-3 Failed to allocate memory to store baselines**
  - Close other programs or reboot computer
- **-4 Failed to read baselines**
  - Reboot the module

Usage example

```
unsigned short NumWords, ModNum, ChanNum;
int retval;
double Baselines[3640], TimeStamps[3640];

// assume we want to acquire baselines for channel 0 of module 0
ModNum = 0;
ChanNum = 0;

// number of baseline words is 3640
NumWords = 3640;

// acquire the baselines
retval = Pixie16AcquireBaselines (ModNum);
if(retval < 0)
{
// error handling
}

// read out the baselines
retval = Pixie16ReadSglChanBaselines (Baselines, TimeStamps, NumWords, ModNum,␣
↪ChanNum);
if(retval < 0)
{
```

(下页继续)

(续上页)

```
// error handling
}
```

21.1.34 Pixie16ReadSglChanPar

Read a CHANNEL level parameter from a module

```
// Use this function to read a channel parameter from a Pixie-16 module. ModNum is␣
↪the module number which starts counting at 0. ChanNum is the channel number␣
↪which starts counting at 0.
int Pixie16ReadSglChanPar (
      char *ChanParName,        // channel parameter name
      double *ChanParData,      // channel parameter value
      unsigned short ModNum,    // channel number
      unsigned short ChanNum );// module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Invalid Pixie-16 channel number**
  - Correct ChanNum
- **-3 Invalid channel parameter name**
  - Correct ChanParName

Usage example

```
int retval;
unsigned short ModNum, ChanNum;
double ChanParData;

// read energy filter rise time from module 0 channel 0
ModNum = 0;// this is the first module
ChanNum = 0;// the first channel

retval = Pixie16ReadSglChanPar ("ENERGY_RISETIME" , &ChanParData, ModNum, ChanNum);
if(retval < 0)
{
// Error handling
}
```

21.1.35 Pixie16ReadSglModPar

Read a MODULE level parameter from a module

```
// Use this function to read a module parameter from a Pixie-16 module. ModNum is␣
↪the module number which starts counting at 0.
int Pixie16ReadSglModPar (
      char *ModParName,         // module parameter name
      unsigned int *ModParData,// module parameter value
      unsigned short ModNum ); // module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Invalid module parameter name**
  - Correct ModParName

Usage example

```
int retval;
unsigned short ModNum;
unsigned int ModParData;

// Read SlowFilterRange in module 0
ModNum = 0;// this is the first module

retval = Pixie16ReadSglModPar ("SLOW_FILTER_RANGE" , &ModParData, ModNum);
if(retval < 0)
{
// Error handling
}
```

21.1.36 Pixie16ReadStatisticsFromModule

Read run statistics data from a module

```
// Use this function to read out statistics data from a Pixie-16 module. Before␣
↪calling this function, the host code should allocate appropriate amount of␣
↪memory to store the statistics data. The number of statistics data for each␣
↪module is fixed at 448. Statistics data are 32-bit unsigned integers.
// Specify the module using ModNum. Note that the modules are counted starting at␣
↪0.
int Pixie16ReadStatisticsFromModule (
      unsigned int *Statistics,// statistics data
      unsigned short ModNum ); // module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Failed to get the statistics data**
  - Reboot the module

Usage example

```
int retval;
unsigned short ModNum, ChanNum;
unsigned int Statistics[448];

ModNum = 0;// the first module
ChanNum = 0;// the first channel
```

(下页继续)

(续上页)

```
retval = Pixie16ReadStatisticsFromModule (Statistics, ModNum);
if(retval < 0)
{
// error handling
}
```

21.1.37 Pixie16RegisterIO

Read from or write to registers on a module

```
// Use this function to read data from or write data to a register in a Pixie-16␣
↪module.
// Specify the module using ModNum. Note that the modules are counted starting at␣
↪0.
int Pixie16RegisterIO (
      unsigned short ModNum,    //module number
      unsigned int address,     //register address
      unsigned short direction,//read or write
      unsigned int *value );    //holds or receives the data
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum

Usage example

```
int retval;
unsigned short ModNum;
unsigned int address, value;
ModNum = 0;// the first module
address = PCI_STOPRUN_REGADDR;// address of the register for ending run
value = 0;

retval = Pixie16RegisterIO(ModNum, address, MOD_WRITE, &value);
if(retval < 0)
{
// error handling
}
```

21.1.38 Pixie16SaveDSPParametersToFile

Read DSP parameters from modules and save to a file

```
// Use this function to save DSP parameters to a settings file. It will first read␣
↪the values of DSP parameters on each Pixie-16 module and then write them to the␣
↪settings file. Each module has exactly 1280 DSP parameter values (32-bit␣
↪unsigned integers), and depending on the value of PRESET_MAX_MODULES (defined in␣
↪pixie16app_defs.h), the settings file should have exactly (1280 * PRESET_MAX_
↪MODULES * 4) bytes when stored on the computer hard drive.
int Pixie16SaveDSPParametersToFile (
      char *FileName ); // DSP parameters file name (with complete path)
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Failed to read DSP parameter values from the Pixie-16 modules**
  - Reboot the modules
- **-2 Failed to open the DSP parameters file**
  - Correct the DSP parameters file name

Usage example

```
int retval;
char *FileName = {"C:\\XIA\\Pixie16\\Configuration\\test.set"};

retval = Pixie16SaveDSPParametersToFile (FileName);
if(retval < 0)
{
// error handling
}
```

21.1.39 Pixie16SaveExternalFIFODataToFile

Read data from external FIFO and save to a file

```
// Use this function to read data from the external FIFO of a module. ModNum is␣
↪the module number which starts counting at 0.
// This function first checks the status of the external FIFO of a Pixie-16 module,
↪ and if there are data in the external FIFO, this function then reads list mode␣
↪data (32-bit unsigned integers) from the external FIFO. So this function␣
↪essentially encapsulates both functions Pixie16CheckExternalFIFOStatus and␣
↪Pixie16ReadDataFromExternalFIFO within one function. The number of words that␣
↪are read from the external FIFO is recorded in variable *FIFOWords. The function␣
↪also expects setting the value of a variable called "EndOfRunRead" to indicate␣
↪whether this read is at the end of a run (1) or during the run (0). This is␣
↪necessary since the external FIFO needs special treatment when the host reads␣
↪the last few words from the external FIFO due to its pipelined structure.
int Pixie16SaveExternalFIFODataToFile (
      char *FileName,// list mode data file name
      unsigned int *nFIFOWords,// number of words read from ext. FIFO
      unsigned short ModNum,// module number
      unsigned short EndOfRunRead ); // indicator if this is end of run read
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Check ModNum
- **-2 Failed to allocate memory to store list mode data**
  - Check computer resources
- **-3 Failed to open list mode data file**
  - Check if file is protected
- **-4 Failed to read external FIFO status**
  - Check error message log file Pixie16msg.txt
- **-5 Failed to read data from external FIFO**

  - Check error message log file Pixie16msg.txt

Usage example

```
int retval;
unsigned int nFIFOWords;
unsigned short ModNum, EndOfRunRead;
ModNum = 0;// the first module in the system
EndOfRunRead = 0;// this is a read during the run

retval = Pixie16SaveExternalFIFODataToFile("listmodedata_mod0.bin", &nFIFOWords,␣
↪ModNum, EndOfRunRead);
if(retval < 0)
{
// Error handling
}
```

21.1.40 Pixie16SaveHistogramToFile

Read histogram data from a module and save to a file

```
// Use this function to read histogram data from a Pixie-16 module and save the␣
↪histogram data to a file with file name specified by the user: First this␣
↪function saves the histogram data to a binary file, and it then saves the␣
↪histogram data to an ASCII file with run statistics data appended to the end of␣
↪the ASCII file. Existing files will be overwritten. ModNum is the module number␣
↪which starts counting at 0.
int Pixie16SaveHistogramToFile (
      char *FileName,// histogram data file name (with complete path)
      unsigned short ModNum); // module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Failed to get histogram data from module**
  - Reboot the module
- **-3 Failed to allocate memory to store histogram data**
  - Close other programs or reboot computer
- **-4 Failed to open histogram data file**
  - Correct file name and path
- **-5 Failed to open mca ascii output file**
  - Correct file name and path
- **-6 Failed to allocate memory to store histogram data for ascii text file**
  - Close other programs or reboot computer
- **-7 Failed to read histogram data from file**
  - Check file name and path
- **-8 Failed to read run statistics data from module**
  - Reboot the module

Usage example

```
int retval;
char *FileName = {"C:\\XIA\\Pixie16\\MCA\\histogramdata.bin"};
unsigned short ModNum;
ModNum = 0;// the first module

retval = Pixie16SaveHistogramToFile (FileName, ModNum);
if(retval < 0)
{
// error handling
}
```

21.1.41 Pixie16SetDACs

Program on-board DACs

```
// Use this function to reprogram the on-board digital to analog converters (DAC)␣
↪of the Pixie-16 modules. In this operation the DSP uses data from the DSP␣
↪parameters that were previously downloaded. ModNum is the module number which␣
↪starts counting at 0.
int Pixie16SetDACs (
      unsigned short ModNum);// module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Failed to start the SET_DACS run**
  - Reboot the module
- **-3 SET_DACS run timed out**
  - Reboot the module

Usage example

```
int retval;
unsigned short ModNum;
ModNum = 0;// the first module

retval = Pixie16SetDACs (ModNum);
if(retval < 0)
{
// error handling
}
```

21.1.42 Pixie16StartHistogramRun

Start a MCA histogram mode data acquisition run

```
// Use this function to begin a data acquisition run that accumulates energy␣
↪histograms, one for each channel. It launches a data acquisition run in which␣
↪only energy information is preserved and histogrammed locally to each channel.
// Call this function for each Pixie-16 module in the system to initialize the run␣
↪in each module. Actual data acquisition will start synchronously in all modules␣
↪when the last module finished the initialization(requires the synchronization␣
↪parameter to be set). Histogram runs can be self-terminating when the elapsed␣
↪run time exceeds the preset run time, or the user can prematurely terminate the␣
↪run by calling Pixie16EndRun. On completion, final histogram and statistics data␣
↪will be available.
```

(下页继续)

(续上页)

```
// Use mode=NEW_RUN (=1) to erase histograms and statistics information before␣
↪launching the new run. Use mode=RESUME_RUN (=0) to resume an earlier run.
// ModNum is the module number which starts counting at 0. If ModNum is set to be␣
↪less than the total number of modules in the system, only the module specified␣
↪by ModNum will have its histogram run started. But if ModNum is set to be equal␣
↪to the total number of modules in the system, then all modules in the system␣
↪will have their runs started together.
int Pixie16StartHistogramRun (
      unsigned short ModNum,// module number
      unsigned short mode );// run mode
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct the ModNum
- **-2 Invalid run mode**
  - Correct the run mode
- **-3 Failed to start histogram run**
  - Reboot the module

Usage example

```
int retval;
unsigned short mode, ModNum;
double preset_run_time;
unsigned int ieee_preset_run_time;

mode = NEW_RUN;// to start a new run
// Assume there are 5 modules in the system
ModNum = 5;// start histogram run in all 5 modules

// Assume preset run time is 10 seconds
preset_run_time = 10.0;

// Convert preset run time to IEEE 32-bit floating point number
ieee_preset_run_time = Decimal2IEEEFloating (preset_run_time);

// Download the preset run time to the DSP
retval = Pixie16WriteSglModPar("HOST_RT_PRESET", ieee_preset_run_time, ModNum)
if(retval < 0)
{
// Error handling
}

// Start the histogram run
retval = Pixie16StartHistogramRun (ModNum, mode);
if(retval < 0)
{
// Error handling
}
```

21.1.43 Pixie16StartListModeRun

Start a list mode data acquisition run

```
// Use this function to start a list mode data acquisition run in Pixie-16 modules.
↪ List mode runs are used to collect data on an event-by-event basis, gathering␣
↪energies, timestamps, pulse shape analysis values, and waveforms for each event.␣
↪Runs will continue until the user terminates the run by calling function␣
↪Pixie16EndRun. To start the data acquisition this function has to be called for␣
↪every Pixie-16 module in the system. If all modules are to run synchronously,␣
↪the last module addressed will release all others and the acquisition starts␣
↪then. The first module to end the run will immediately stop the run in all other␣
↪modules if run synchronization has been set up among these modules.
// Use mode=NEW_RUN (=1) to erase histograms and statistics information before␣
↪launching the new run. Note that this will cause a startup delay of up to 1␣
↪millisecond. Use mode=RESUME_RUN (=0) to resume an earlier run. This mode has a␣
↪startup delay of only a few microseconds.
// There is only one list mode run type supported, that is, 0x100. However,␣
↪different output data options can be chosen by enabling or disabling different␣
↪CHANCSRA bits.
// ModNum is the module number which starts counting at 0. If ModNum is set to be␣
↪less than the total number of modules in the system, only the module specified␣
↪by ModNum will have its list mode run started. But if ModNum is set to equal to␣
↪the total number of modules in the system, then all modules in the system will␣
↪have their runs started together.
int Pixie16StartListModeRun (
      unsigned short ModNum, // module number
      unsigned short RunType,// run type
      unsigned short mode ); // run mode
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct the ModNum
- **-2 Invalid run mode**
  - Correct the run mode
- **-3 Failed to start list mode run**
  - Reboot the module
- **-4 Invalid run type**
  - Correct RunType

Usage example

```
int retval;
unsigned short mode, ModNum, RunType;

mode = NEW_RUN;// to start a new run
RunType = 0x100;// general purpose list mode run

// Assume there are 5 modules in the system
ModNum = 5;

// start list mode run in all 5 modules
retval = Pixie16StartListModeRun (ModNum, RunType, mode);
if(retval < 0)
{
// Error handling
}
```

21.1.44 Pixie16TauFinder

Find the exponential decay time of a channel

```
// Use this function to find the exponential decay time constants (Tau value) of␣
↪the detector or preamplifier signal that is connected to each of the 16 channels␣
↪of a Pixie-16 module. The 16 found Tau values are returned via pointer *Tau. A '-
↪1.0' Tau value for a channel means the Tau_Finder was not successful for such a␣
↪channel. ModNum is the module number which starts counting at 0.
void Pixie16TauFinder (
      unsigned short ModNum,// module number
      double *Tau );// Tau value
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct the ModNum
- **-2 Invalid Pixie-16 channel number**
  - Correct the ChanNum
- **-3 Failed to acquire ADC traces**
  - Reboot the module
- **-4 Failed to read ADC traces**
  - Reboot the module
- **-5 Failed to find sufficient number of pulses**
  - Increase input count rate

Usage example

```
int retval;
unsigned short ModNum;
double Tau[16];
ModNum = 0;// the first module

retval = Pixie16TauFinder(ModNum, Tau);
if(retval < 0)
{
// Error handling
}
```

21.1.45 Pixie16WriteCSR

Write to Control & Status Register in a module

```
// Use this function to write a value to the host Control & Status Register (CSR).␣
↪This register is unrelated to the DSP parameters ModCSRA/B, ChanCSRA/B. It is␣
↪used to control the operation of the module and read directly by the host.␣
↪Direct reading or writing by the host is not recommended, for example use␣
↪functions like Pixie16CheckRunStatus to poll the active bit. ModNum is the␣
↪module number which starts counting at 0.
void Pixie16WriteCSR (
      unsigned short ModNum,// module number
      unsigned int CSR );    // CSR value to write
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct the ModNum

Usage example

```
int retval;
unsigned short ModNum;
unsigned int CSR;

ModNum = 0;// the first module
retval = Pixie16ReadCSR(ModNum, &CSR);
if(retval < 0)
{
// Error handling
}

CSR = APP32_ClrBit(3, CSR);
retval = Pixie16WriteCSR (ModNum, CSR);
if(retval < 0)
{
// Error handling
}
```

21.1.46 Pixie16WriteSglChanPar

Write a CHANNEL level parameter to a module

```
// Use this function to write a channel parameter to a Pixie-16 module. ModNum is␣
↪the module number which starts counting at 0. ChanNum is the channel number␣
↪which starts counting at 0.
int Pixie16WriteSglChanPar (
      char *ChanParName,        //channel parameter name
      double ChanParData,       //channel parameter value
      unsigned short ModNum,    //channel number
      unsigned short ChanNum );//module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Invalid Pixie-16 channel number**
  - Correct ChanNum
- **-3 Invalid channel parameter name**
  - Correct ChanParName
- **-4 Programing Fippi failed downloading channel parameter**
  - Reboot the module
- **-5 Failed to find BLcut after downloading channel parameter**
  - Reboot the module

- **-6 SetDACs failed downloading channel parameter**
  - Reboot the module

Usage example

```
int retval;
unsigned short ModNum, ChanNum;
double ChanParData;
// Set energy filter rise time to 6.08 us for module 0 channel 0
ModNum = 0;// this is the first module
ChanNum = 0;// the first channel
ChanParData = 6.08; // energy filter rise time = 6.08 us

retval = Pixie16WriteSglChanPar ("ENERGY_RISETIME", ChanParData, ModNum, ChanNum);
if(retval < 0)
{
// Error handling
}
```

21.1.47 Pixie16WriteSglModPar

Write a MODULE level parameter to a module

```
// Use this function to write a module parameter to a Pixie-16 module. ModNum is␣
↪the module number which starts counting at 0.
int Pixie16WriteSglModPar (
      char *ModParName,       // module parameter name
      unsigned int ModParData,// module parameter value
      unsigned short ModNum );// module number
```

Return values description and error handling

- **0 Success**
  - None
- **-1 Invalid Pixie-16 module number**
  - Correct ModNum
- **-2 Invalid module parameter name**
  - Correct ModParName
- **-3 Failed to program Fippi after downloading module parameter**
  - Reboot the module
- **-4 Failed to find BLcut after downloading module parameter**
  - Reboot the module

Usage example

```
int retval;
unsigned short ModNum;
unsigned int ModParData;
// Set SlowFilterRange in module 0 to 4
ModNum = 0;// this is the first module
ModParData = 4;// SlowFilterRange = 4

retval = Pixie16WriteSglModPar ("SLOW_FILTER_RANGE", ModParData, ModNum);
if(retval < 0)
{
```

(下页继续)

(续上页)

```
// Error handling
}
```

21.2 PKU API

Add by Hongyi Wu

21.2.1 HongyiWuPixie16ComputeFastFiltersOffline

```
int HongyiWuPixie16ComputeFastFiltersOffline (
      char           *FileName,           // the list mode data file name (with␣
↪complete path)
      unsigned short ModuleNumber,        // the module whose events are to be␣
↪analyzed
      unsigned short ChannelNumber,       // the channel whose events are to be␣
↪analyzed
      unsigned int   FileLocation,        // the location of the trace in the file
      unsigned short RcdTraceLength,      // recorded trace length
      unsigned short *RcdTrace,           // recorded trace
      double         *fastfilter,         // fast filter response, the same scale␣
↪as the trigger threshold
      double         *cfd,                // cfd response, the same scale as the␣
↪CFD threshold
      double         *cfds );             // cfd response, the same scale as the␣
↪fast filter
```

21.2.2 HongyiWuPixie16ComputeSlowFiltersOffline

```
int HongyiWuPixie16ComputeSlowFiltersOffline (
      char           *FileName,           // the list mode data file name (with␣
↪complete path)
      unsigned short ModuleNumber,        // the module whose events are to be␣
↪analyzed
      unsigned short ChannelNumber,       // the channel whose events are to be␣
↪analyzed
      unsigned int   FileLocation,        // the location of the trace in the file
      unsigned short RcdTraceLength,      // recorded trace length
      unsigned short *RcdTrace,           // recorded trace
      double         *slowfilter );       // slow filter response
```

21.2.3 HongyiWuPixie16ComputeSlowFiltersOfflineAverageBaseline

```
int  HongyiWuPixie16ComputeSlowFiltersOfflineAverageBaseline (
      char           *FileName,           // the list mode data file name (with␣
↪complete path)
      unsigned short ModuleNumber,        // the module whose events are to be␣
↪analyzed
      unsigned short ChannelNumber,       // the channel whose events are to be␣
↪analyzed
      unsigned int   FileLocation,        // the location of the trace in the file
      unsigned short RcdTraceLength,      // recorded trace length
      unsigned short *RcdTrace,           // recorded trace
```

(下页继续)

(续上页)

```
      double          *slowfilter,        // slow filter response
      int             pointtobl );        // Average number of estimated waveform␣
↪baselines
```

21.2.4 HongyiWuPixie16ComputeSlowFiltersOfflineExtendBaseline

```
int HongyiWuPixie16ComputeSlowFiltersOfflineExtendBaseline (
      char            *FileName,          // the list mode data file name (with␣
↪complete path)
      unsigned short ModuleNumber,        // the module whose events are to be␣
↪analyzed
      unsigned short ChannelNumber,       // the channel whose events are to be␣
↪analyzed
      unsigned int    FileLocation,       // the location of the trace in the file
      unsigned short RcdTraceLength,      // recorded trace length
      unsigned short *RcdTrace,           // recorded trace
      double          *slowfilter,        // slow filter response
      unsigned int    bl,                 // The baseline calculated in the firmware
      double          sl,                 // SL used to calculate the baseline in␣
↪the firmware
      double          sg,                 // SG used to calculate the baseline in␣
↪the firmware
      double          tau,                // TAU used to calculate the baseline in␣
↪the firmware
      int             sfr,                // SlowFilterRange used to calculate the␣
↪baseline in the firmware
      int             pointtobl );        // Average number of estimated waveform␣
↪baselines
```

21.2.5 HongyiWuPixie16SetOfflineVariant

```
int HongyiWuPixie16SetOfflineVariant(unsigned short mod,unsigned short variant,
↪unsigned short bits,unsigned short samplerate);
```

CHAPTER 22

PKU Code

本节介绍程序的主要思路。

DOTO 需要补充框图帮助理解程序!!!

22.1 Decode

- decoder.cc
- **decoder.hh**
  - 读取二进制文件
- **main.cc**
  - 主程序
- Makefile
- r2root.cc
- **r2root.hh**
  - 保存 ROOT 文件
- **UserDefine.hh**
  - 用户定义参数

22.2 GUI

- Base.cc
- **Base.hh**
  - 子界面，基线、极性、增益、波形长度、数据记录等参数调节
- Cfd.cc
- **Cfd.hh**
  - 子界面，CFD 参数调节

- CopyPars.cc
- **CopyPars.hh**
  - 子界面，参数复制
- Csra.cc
- **Csra.hh**
  - 子界面，方便快速调节每通道的控制寄存器
- Decimation.cc
- **Decimation.hh**
  - 子界面，降频参数设置
- Detector.cc
- **Detector.hh**
  - 数据采集循环主体
- Energy.cc
- **Energy.hh**
  - 子界面，梯形参数调节界面
- ExpertMod.cc
- **ExpertMod.hh**
  - 子界面，采集卡模块参数设置
- Global.cc
- **Global.hh**
  - 全局函数
- HistXDT.cc
- **HistXDT.hh**
  - 子界面，设置记录的一维能谱的最小值、bin 宽及 DSP 抓波形时的部长
- LogicTrigger.cc
- **LogicTrigger.hh**
  - 子界面，逻辑参数调节
- main.cc
- MainFrame.cc
- **MainFrame.hh**
  - 主控制界面
- MainLinkdef.h
- Makefile
- Offline.cc
- **Offline.hh**
  - 离线分析主界面，离线分析功能代码
- OfflineData.cc
- **OfflineData.hh**
  - 离线分析读取文件数据

- pkuFFTW.cc
- **pkuFFTW.hh**
  - 基于 FFTW3 封装类
- Qdc.cc
- **Qdc.hh**
  - 子界面，用于 QDC 积分门窗的调节
- ReadChanStatus.cc
- **ReadChanStatus.hh**
  - 子界面，查看 DSP 中抓取的波形及 baseline
- Simulation.cc
- **Simulation.hh**
  - 未实现
- Table.cc
- **Table.hh**
  - 基类，用于参数调节界面
- TriggerFilter.cc
- **TriggerFilter.hh**
  - 子界面，fast filter 参数调节界面
- **wuReadData.hh**
  - 模版函数，用来读取输入卡

22.3 MakeEvent

- **main.cc**
  - 主程序
- Makefile
- sort.cc
- **sort.hh**
  - 事件组装
- **UserDefine.hh**
  - 用户定义参数

22.4 OnlineStattics

- Linkdef.hh
- **main.cc**
  - 主程序
- Makefile
- Online.cc

- **Online.hh**
  - 在线监视界面
- PixieOnline.config