目錄

README

Figure: PKU logo

English | 简体中文

版本

我们建议用户下载稳定版本

稳定版本

稳定版本 **Version:2018.11.30**

下载最新版本，请点击: PKUXIADAQ stable

网页版说明书请访问: English 简体中文

- markdown版说明书: README/
- 离线网页版说明书: docs/
- pdf版本说明书: README_en.pdf README_ch.pdf

准预览版本

程序下载请访问: PKUXIADAQ

网页版说明书请访问: English 简体中文

- 对本获取程序有任何的意见及建议(功能添加及改进)，欢迎给吴鸿毅(wuhongyi@qq.com)发邮件。
- 我们将会尽快完善软件的中英文使用说明书，当前主要以操作演示来讲解软件的使用。

关于

This manual applies only to XIA LLC Pixie-16

- 本程序由北京大学实验核物理组开发。
- 最早的图形界面程序是基于 NSCL DDAS Nscope 开发。
- 特别感谢 谭辉*(XIA LLC)* 对我们开发的支持。

技术指导:

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

软件主要开发者:

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

说明书主要撰写者:

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

本程序的开发得到以下单位的支持：

- XIA LLC
- 中国科学院近代物理研究所(IMP)
- 中国原子能科学研究院(CIAE)
- 香港大学(HKU)
- 山东大学(威海)(SDU)
- ...

本程序适用于 XIA Pixie16 系列采集卡，支持 100/250/500 MHz 采集卡(具体支持型号可查看图形软件中的File->About)，最大支持 8 个机箱同步运行，即 1600+ 路信号同时采集。本程序包要求使用 **CERN ROOT6** 版本。要求采用 **1920x1080** 及以上分辨率显示屏。

本程序的设计兼容 100/250/500 MHz 采集卡的混合使用，只需在 cfgPixie16.txt 添加相应采样率采集卡的固件位置即可，程序在线能够自动识别采集卡类型并加载相应的固件。当前我们只有100/250 MHz 14 bit 的采集卡，因此默认可运行该类型的采集卡，如要支持其它类型，请联系 XIA LLC 获取对应固件或者联系吴鸿毅(wuhongyi@qq.com)。

目录

用户使用程序包中包含以下文件/文件夹:

- Decode(将原始二进制数据转为 ROOT)
- docs(使用说明书，网页版)
- firmware(固件)
  - firmware/firmware.md(历史各版本固件说明)
- GUI(图形软件)
- MakeEvent(事件重构程序，可选)
- NOGUI(非图形软件。新版本升级中，暂时不可用)
- OnlineStattics(在线监视程序)

- parset(参数设置文件)
- PlxSdk.tar.gz(Plx9054 驱动)
- README(markdown 版本说明书)
- README.md(主页介绍)
- README.pdf(pdf 版本说明书)
- software(非标准驱动，吴鸿毅修改)
- TestTool(开发者测试工具，用户不需要！！！)

升级计划

- 当前基于 ROOT GUI 开发的主控制界面复杂度高，用户修改难度大。其它用户不容易基于其发展适合自己的版本。
- 我们也在开发基于网页控制的获取在线/离线分析程序:
  - ZeroMQ
  - FastCGI
  - JSROOT
  - web
  - ...

 updated: 2018-11-30 20:58:21

程序安装

本程序安装要求

- CERN ROOT 6
  - GCC >= 4.8
- FFTW3

本程序测试过的系统包括 Scientific Linux 7.2/7.3/7.4

软件安装步骤

- 删除个人目录下的老版本PKUXIADAQ文件夹
- 将本程序包解压缩到个人目录中($HOME)
- 设置环境变量
- 编译Plx9054驱动
- 编译pixie16驱动API(该API被吴鸿毅修改过，非官方标准驱动)
- 编译图形化获取软件
- 编译在线监视程序
- 编译数据转换程序
- 编译事件重构程序(可选)

```
##设置环境变量

#在 .bashrc 文件中添加
export PLX_SDK_DIR=$HOME/PKUXIADAQ/PlxSdk

# 将 PKUXIADAQ.tar.gz(或者PKUXIADAQ-master.tar.gz) 放置到 /home 下的个人目录下，即 ~/ 位置
tar -zxvf PKUXIADAQ.tar.gz  #解压缩
或者
tar -zxvf PKUXIADAQ-master.tar.gz
mv PKUXIADAQ-master  PKUXIADAQ

#得到 PKUXIADAQ 目录
```

```
##编译Plx9054驱动

#打开新终端
cd ~
cd PKUXIADAQ/
rm -rf PlxSdk #删除可能存在的未删除驱动，如果没有该目录则不用执行该行命令
tar -zxvf PlxSdk.tar.gz
cd PlxSdk/PlxApi/
make clean
make
#成功后你将会看到 Library "Library/PlxApi.a" built successfully

cd ../Samples/ApiTest/
make clean
make
#成功后你将会看到 Application "App/ApiTest" built successfully

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

```
#成功后你将会看到 Driver "Plx9054/Plx9054.ko" built sucessfully
```

```
## Compile pixie16
##编译pixie16

cd ~
cd PKUXIADAQ/software/
make clean
make

#只要没报错，并且该文件夹内生成libPixie16App.a libPixie16Sys.a
```

```
##编译图形化获取软件

#修改设置参数
cd ~
cd PKUXIADAQ/parset/


#修改cfgPixie16.txt文件。
#其中CrateID 后面的数值表示机箱编号，该值允许0-15。如果单机箱则随意设置(一般就采用默认的0)，如果多个机箱同步运行务必让每个机箱的该编号设置为不同的数值。
#SettingPars 后面为参数设置文件，写入要采用的参数配置文件即可。
#ModuleSlot 后面第一个数值表示插件个数，如果有3个插件则为3。之后的数字未为每个插件在机箱的插槽位置（插槽位置从2开始计数），有三个插件则之后分别为2 3 4。
#参数 ModuleSampingRate与ModuleBits 只对离线模式生效，当主界面采用Offline模式初始化时则读取该参数。

#修改Run.config文件，该文件中第一行为原始数据存放路径，第二行为文件名。
#修改RunNumber文件，该文件中的数值为运行的run number。

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

```
##编译在线监视程序

cd ~
cd PKUXIADAQ/OnlineStattics/

#修改 PixieOnline.config 文件中的参数
#第一行为获取数据文件存放路径
#第二行为获取文件名

make clean
make
```

```
## 编译数据转换程序

cd ~
cd PKUXIADAQ/Decode/

#修改 UserDefine.hh，按照程序中的说明修改即可

make clean
make
```

```
## 编译事件重构程序

cd ~
```

```
cd PKUXIADAQ/MakeEvent/

#修改 UserDefine.hh，按照程序中的说明修改即可

make clean
make
```

程序使用说明

- 开机机箱后重启电脑(电脑必须晚于机箱开启)
- 开启机箱后ROOT权限下加载Plx9054驱动
- 正常获取

```
## ROOT权限下加载Plx9054驱动

cd ~
cd PKUXIADAQ/PlxSdk/Bin/
su #输入ROOT密码
./Plx_load 9054
#将会看到加载成功的提示
exit  #退出ROOT权限
```

```
##启动图形界面程序

cd ~
cd ~/PKUXIADAQ/GUI
./pku


#将会弹出图形化界面
#可选择 Online/Offline Mode 然后按 Boot 初始化
#等待初始化成功后，可修改输出数据文件路径，文件名，run number。按 Complete 按钮确认。
#此时 LSRunStart 按钮变为可操作。即可开始按Start，之后第二次按即为Stop。
#Online Statistics选项选择表示发送在线统计
#Update Energy Monitor每选择一次则从插件内部读取一次能谱信息并发送给在线程序（频繁选择会影响获取）
```

```
##启动在线监视程序

cd ~
cd PKUXIADAQ/OnlineStattics/
./online


#将会弹出图形化界面
#查看上面的原始数据文件夹路径、文件名是否正确。按 Complete 确认。
#按 RunStart开始启动监视，每3秒更新一次每路的输入率、输出率。(开启机箱后第一次启用该程序需要在获取开启之后)
#监视界面右下角有对写入硬盘使用量的监视。

#EnergyMonitor页面用来查看能谱。由于插件内部寄存器大小限制，该能谱与实际能谱道址范围存在差别。
```

```
##执行数据转换程序

cd ~
cd PKUXIADAQ/Decode/

#在上一轮获取结束之后，我们便可将上一轮数据转为ROOT文件
./decode xxx
# xxx 为运行 run number
```

 updated: 2018-11-05 15:22:07

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

> **[info] IMPORTANT**
>
> 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.
>
> **[danger] IMPORTANT**
>
> 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!

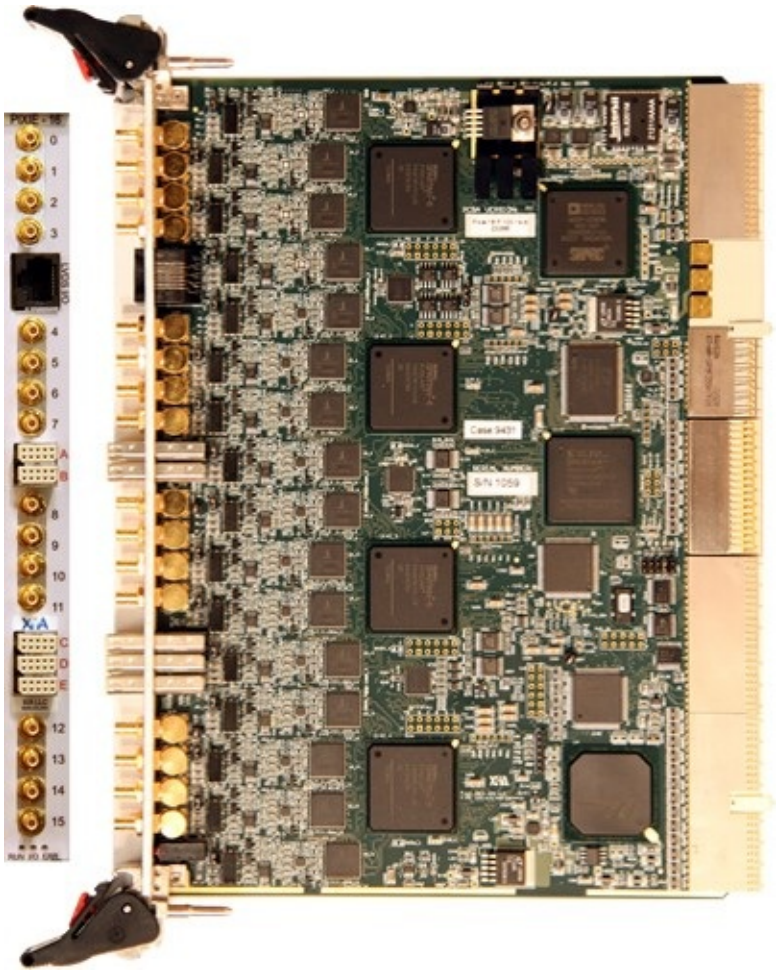


Figure: pixie16

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.

Output 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

以下为可选择输出

If trace recording is enabled, trace data will immediately follow the last word of the event header. Since raw ADC data points are 14-bit number, two 14-bit numbers are packed into one 32-bit word,as shown below. 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.

- CAEN 插件在采集波形时，一个插件所有通道每个事件波形采集长度只能设置成相同的。
- XIA 插件可以每个通道单独设置数据长度。例如：ch-0 设置采3000个点，ch-1 设置采2000个点，ch-2 设置采5000个点，ch-3 设置成不采集波形……。另外，还可每个通道选择是否记录baseline sum、QDC、external timestamps。
- XIA还可设置对pile up事件的处理。例如pile up事件记录波形，其它事件则不记录波形。意味着一个channel 不同事件其数据长度也不一定相等。
- 在读取数据时候，CAEN 插件每次拿到的数据都是完整事件的数据。 而 XIA 插件 FIFO 的读指针、写指针是独立的、同时进行的，所以每次我们要取数据时候都需要先查询下当前 FIFO 有多少数据，然后再指定该次读取多少数据。因为FIFO读、写是独立的，因而查询到的数据量是该时刻拥有的数据量，这个数据量基本都不是完整事件的数据量（还有部分是当前事件正在写入的）。每个事件除了有 4 words 的 header 及后面紧跟着的数据外并没有事件标记符，当然这样设计最大的好处就是速度快。

基于以上事实，两个公司的产品对用户写时时监视获取率、在线监视程序等的实现方式存在较大差异。

- CAEN： 由于 CAEN 一个插件上每个通道的采样数相同，因而他们每个事件的长度是一致的。在拿到数据时即可存文件或者 decode 出每个事件的信息用来统计时时获取速率，也可将数据发送到在线程序。
- XIA： 由于其上每个通道数据长度可单独设置，这是 XIA 插件的灵活性。又 FIFO 查询到的数据量是当前时刻的数据量，这样每次取到的数据都不是完整事件的数据。
- 对于多个插件的系统，获取时候一直循环读取每一个插件上的数据，如果将所有插件的数据存在同一个文件内，将造成数据的错乱。如果每个插件的数据单独存在一个文件内，那么下一次读取的该插件的数据中开头的不完整事件的数据刚好接上一次最后一个不完整事件的数据，可解决这个问题。
- 由于每次拿到的不是完整数据，因此也不好 decode 当前数据来获得每个事件的信息（从开始就decode记录每次数据的不完整事件的情况也可解决该问题，但是这样decode需要额外的时间）。但是 XIA 内部 DSP上统计着信号输入量、获取接收量、活时等信息。可频繁调用函数读取 DSP 上的统计信息来监视时时获取速率。但是频繁调用该函数会对获取的 I/O 有一点影响。几秒钟调用一次该函数，这个影响基本可以忽略。
- 由于每次拿到的不是完整数据，因此共享内存的在线监视也不容易实现，如若要实现，也不是没办法（从开始就decode记录每次数据的不完整事件的情况也可解决该问题，但是这样decode需要额外的时间）。
- 下面给出一些困难的解决思路： 牺牲XIA插件的灵活性，对一个插件设置成每个通道采集波形长度相同（这样就与CAEN一样了），因为每个通道的每个事件数据的长度相等，读取数据时候查询到当前的数据量之后，然后只读取事件长度整数倍的最大数据，这样每次拿到的都是完整的事件数据了，可参照CAEN的实现方式。
- 对采波形时候，可采用“伪”在线监视获取情况，即获取文件每隔一段时间保存一次，用上一轮的数据作为“在线”数据。

 updated: 2018-11-05 15:22:28

机箱

远程控制

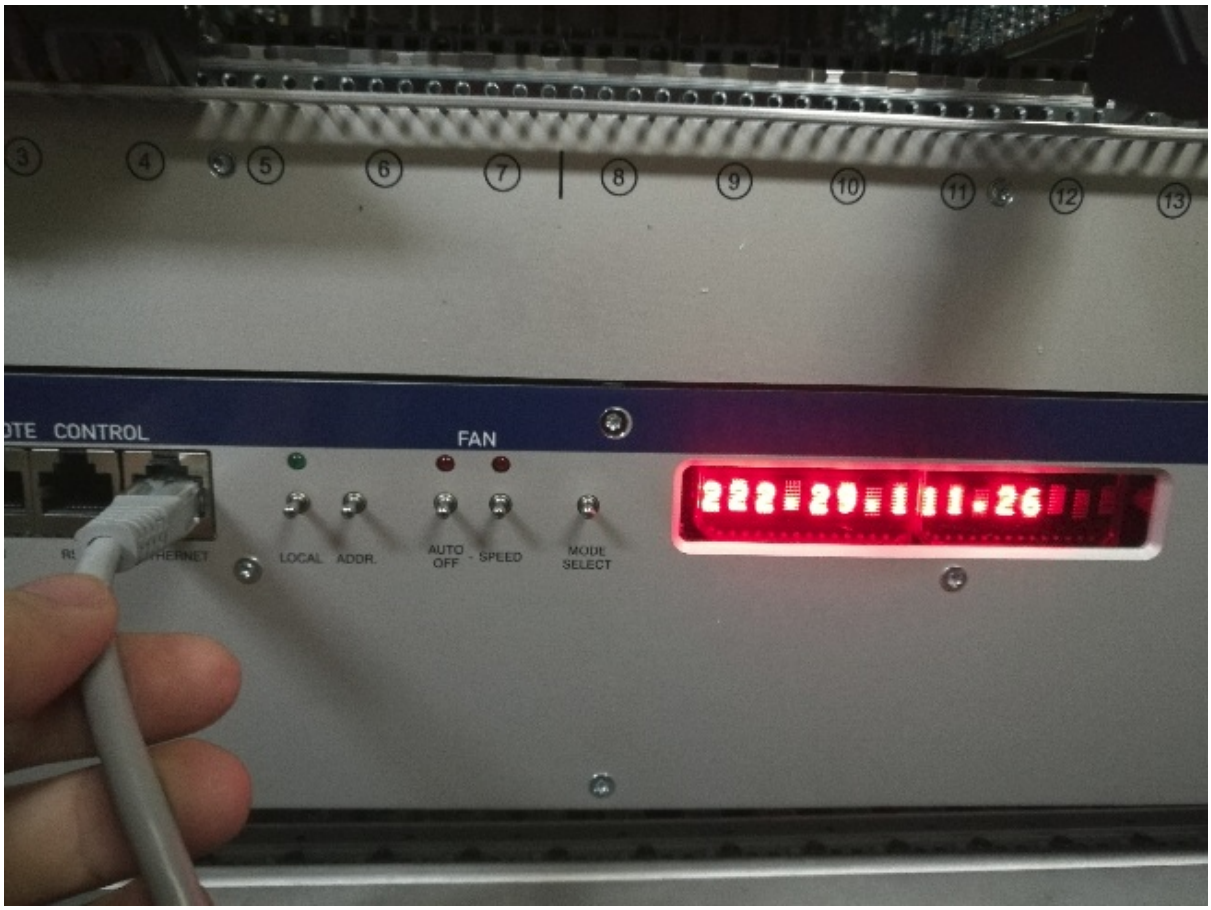


Figure: connect and IP

如上图中将机箱网口接入网线，右侧显示屏中会快速闪过所分配的 IP，例如这里 IP 为 222.29.111.26。如果没看清该 IP，则拔出网线重新连接。

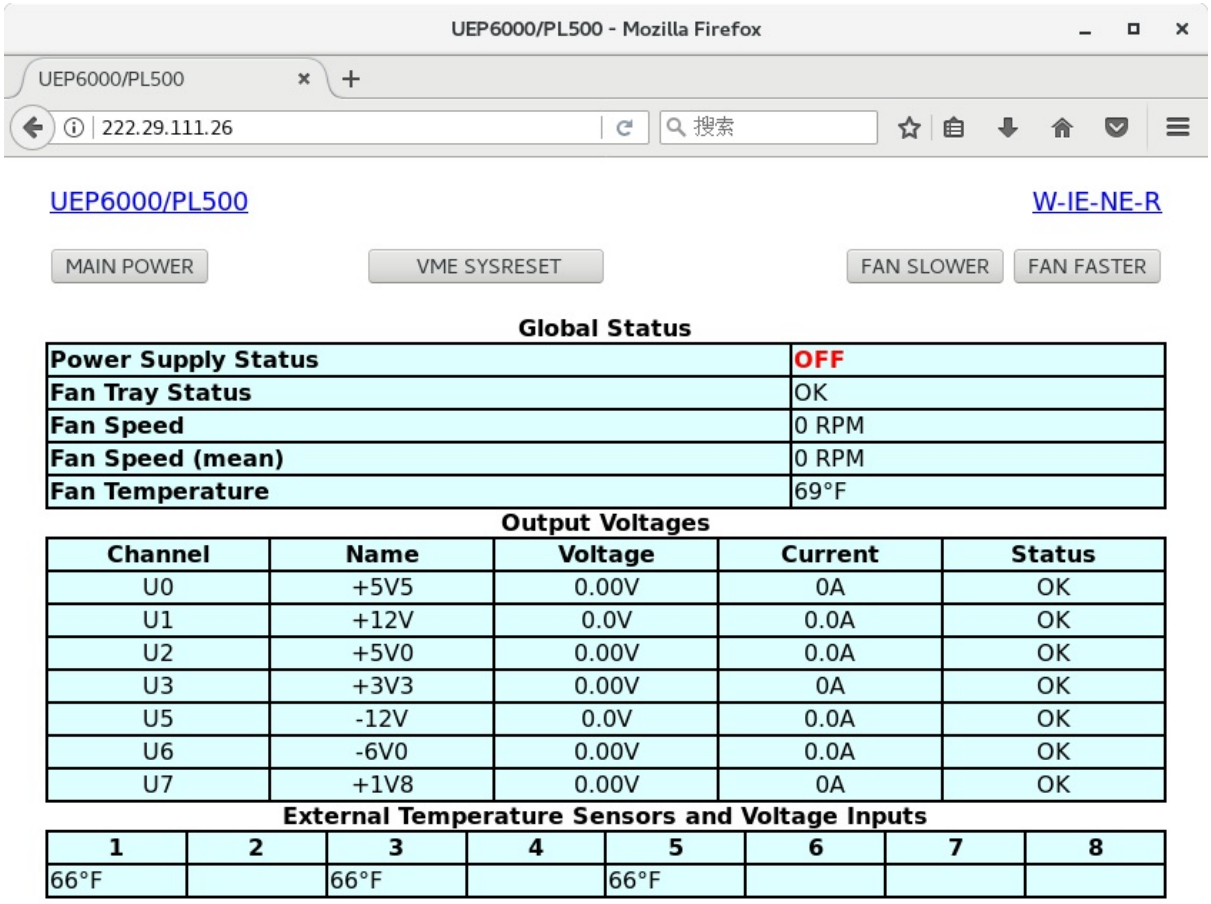


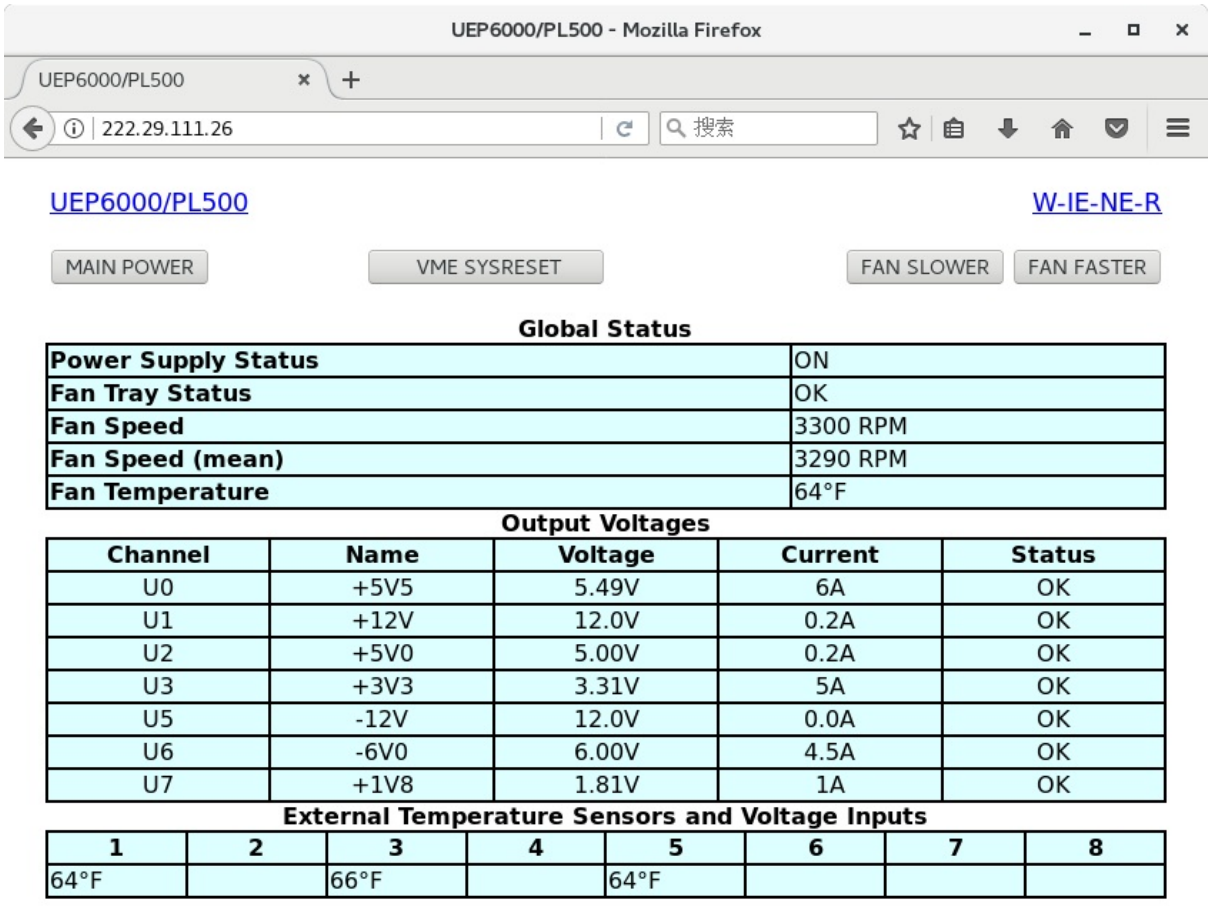
Figure: open control page

如上图，在浏览器中输入 IP，则可进入该控制页面。图中展示的是机箱关闭状态。

Figure: Login

按钮 **MAIN POWER** 用来控制机箱开启及关闭，首次点击会弹出以上登陆框。

输入用户名 "private"，默认密码 "private"



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			

Figure: status

上图为机箱开启后的监视的参数。

其中，右上角的按钮 **FAN SLOWER** 和 **FAN FASTER** 用来调节风散的转速。

插槽

机箱共有 14 个插槽，底下分别标有数字 1-14。其中插槽 1 为控制器插槽，2-14 为采集卡插槽。

 updated: 2018-11-05 15:37:01

FIRMWARE

北京大学定制固件

在标准固件的基础上添加了以下功能：

- 100MHz 14 bit(pixie16_revfpku_14b100m_dsp_update_05082018)
  - MultiplicityMaskHigh[31]=0和1 时候前面版均能输出 multiplicity 结果。
  - 当计算的能量为负数时，该值设置为 0。
  - pileup 事件能量保留，不设置为 0。
  - 在记录波形模式下，waveform 的buffer满了的时候，插件不busy，header继续记录，该情况下，输出的数据没有波形。
  - 在采集波形时候，增加了降频输出的功能，采取的策略为可选择1，1/2，1/4，1/8，1/16，1/32，1/64，1/128频率的输出，即多少个点保留一个点。保留的点是平均后的值。
- 250MHz 14bit(pixie16_revf_14b250m_firmware_release_04222018)
  - 前面板多重性MultiplicityMaskHigh[31]=0和1 时候前面版均能输出 multiplicity 结果。
  - 当计算的能量为负数时，该值设置为 0。
  - pileup 事件能量保留，不设置为 0。

 updated: 2018-11-03 15:08:17

数据解码

Decode 程序用来将同一轮数据不同采集卡采集的数据转为一个 **ROOT** 文件。用户的物理分析以本程序产生的 **ROOT** 文件为基准。

用户首先需要修改 **UesrDefine.hh** 文件中的定义

```
#define ROOTFILEPATH "/home/wuhongyi/data/"  //要生成ROOT文件的路径
#define RAWFILEPATH "/home/wuhongyi/data/"   //原始二进制文件的路径
#define RAWFILENAME "data"                   //原始文件的文件名


#define Crate0
#define Crate0num 5   //该机箱中使用插件数
const int Crate0SamplingRate[Crate0num] = {100,100,100,250,250};//分别指定每个插件的采样率  100/250/500三种采样率  0为跳过该插件
```

用户需要修改：

- 原始二进制文件存放目录
- 生成 ROOT 文件存放目录
- 文件名
- 机箱中使用采集卡个数
- 每个采集卡对应的采样频率，如果采样频率设置为0，则忽略该采集卡的数据

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 **decode**，程序运行方式：

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

其中 *[RuNnumber]* 为想要转换的文件运行编号。

例如：

```
./decade  3
```

ROOT File Branch：

- sr(short): sample rate，100/250/500，This value is specified in UesrDefine.hh. / 该数值由 UesrDefine.hh 中指定
- pileup(bool): 堆积标记。
- outofr(bool): 是否超量程标记。
- cid(short): 机箱编号
- sid(short): 采集卡所在插槽编号
- ch(short): 采集卡通道
- evte(unsigned short): 梯形算法的能量
- ts(long int 64 bit): 时间戳
- ets(long int 64 bit): 外部时间戳
- cfd(short): cfd 数值

- cfdft(bool): cfd 数值是否有效
- cfds(short):cfd source，仅适用于 250/500 MHz 采集卡
- trae(unsigned int): 能量梯形上升段积分
- leae(unsigned int): 能量梯形下降段积分
- gape(unsigned int): 能量梯形平台段积分
- base(double): 能量梯形算法的基线
- qs(unsigned int): 八个QDC面积的积分
- ltra(unsigned short): 波形采集点数
- data(unsigned short): 波形数据
- dt(unsigned short): 为了方便直接查看每个波形，添加了一个数值从0 - N-1 的数组
- nevt(unsigned int): 该事件在本文件中的编号

下图展示一个文件中的 Branch 定义：

```
[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/0                                               *
*Entries :  1123666 : Total  Size=    1140233 bytes  File Size  =      30956 *
*Baskets :      175 : Basket Size=      51200 bytes  Compression=  36.71     *
*.............................................................................*
*Br    2 :outofr    : outofr/0                                               *
*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/0                                                *
*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     *
*.............................................................................*
```

每轮数据转换结束，本文件夹内均会生成一个 **txt** 文件，该文件统计了采集卡每个通道的以下信息：

- Mod: 采集卡标记，从0开始
- Channel: 采集卡通道标记，0 - 15
- OutOfRange: 信号幅度超出模数转换模块范围的事件数
- Pileup: 标记为堆积的事件数
- CfdForcedTrigger: cfd强制触发事件数（cfd未过阈值）
- Energy->0: 计算梯形能量小于 0 的事件数（计算结果小于 0 的直接被标记为0了）
- WaveformCount: 记录波形的事件数
- TotalEvent: 总的输出事件数

 updated: 2018-11-05 16:30:42

图形交互界面

配置好 **parset** 内参数文件
进入 GUI 目录，执行以下命令即可弹出主控制界面

```
./pku
```

主控制界面


PKU XIA Pixie-16 DAQ
File
UV_Setup
Expert
Monitor
Offline
List Mode Run
Online Mode
Boot
Setup
File Path:
/home/wuhongyi/data/
File Name:
data
Run Num:
1061
Complete
Control
Online Statistics
LSRunStart
Update Energy Monitor
Information
Version: 2018.05.27
Booted system
Last run number: 1060

Figure: MainFrame

主界面最上方是File、UV_Setup、Expert、Monitor、Offiline五个下拉栏。里面的子菜单如下所示：

- File
  - Exit
  - About
- UV_Setup
  - Base Setup
  - Trigger Filter
  - Energy
  - CFD
  - QDC
  - Decimation
  - Copy Pars
  - Save2File
- Expert
  - Module Variables
  - CSRA
  - Logic Set
- Monitor
  - Hist & XDT
  - Trace & Baseline
- Offiline
  - Adjust Par
  - Simulation(暂未实现)

开启获取主界面之后，选择 **Online Mode** 选项表示在线模式，需要连接机箱，该模式下可使用所有的功能（包括离线分析功能）。不选择**Online Mode** 选项则表示开启离线模式，可设置、修改获取参数，分析已采集波形。

选择、或者不选择 **Online Mode** 选项之后，按 **Boot** 按钮开启初始化过程，可看到最下方 *Information* 栏目中的状态变化。

系统初始化成功后，再次确认 *Setup* 栏中的获取文件存放路径、文件名、文件编号是否有问题，如果有问题则直接修改，确认之后按 **Complete**。

确认 *Setup* 栏中的信息之后， *Control* 栏中的主按钮 **LSRunStart** 则开启，此时点击该按钮，获取则开启，按钮状态更改为 **LSRunStop**，再次点击该按钮，获取结束，运行的 *Run Num* 号码自动加一。再次点击 **LSRunStart** 开启下轮获取。

当前，获取之前可通过最上方的下拉栏里面的子菜单来调节、修改参数。获取数据时请勿操作 *Control*栏之外的所有选项。

Control 栏内的 **Online Statistics** 选项开启则获取每 **3 s** 向 *OnineStatistics* 程序发送时时每路信号的输入率、输出率信息。

点击 **Update Energy Monitor** 选项一次，则将所有采集卡内部寄存器中每路的一维能谱发送到 **Online Statistics** 程序，发送该信息会导致一定的死时间，请不要频繁点击该选项。

 updated: 2018-11-05 16:34:02

File toolbar drop-downs

本下拉栏内容没有实际用途。

 updated: 2018-11-05 16:35:12

About

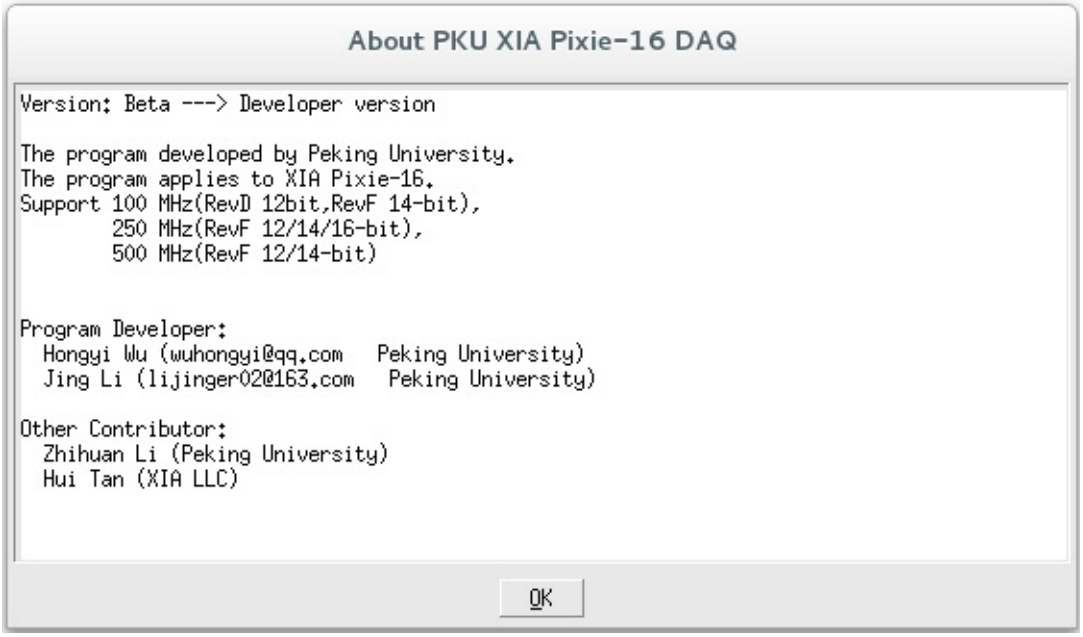


Figure: About

软件开发者介绍。之后将会添加主程序基本操作说明。

 updated: 2018-11-05 16:35:24

UV_Setup

本下拉栏中调节内容为基础内容，任何使用Pixie16获取系统的人员都应该熟悉并掌握其调节技巧。

 updated: 2018-11-05 16:35:39

Base Setup

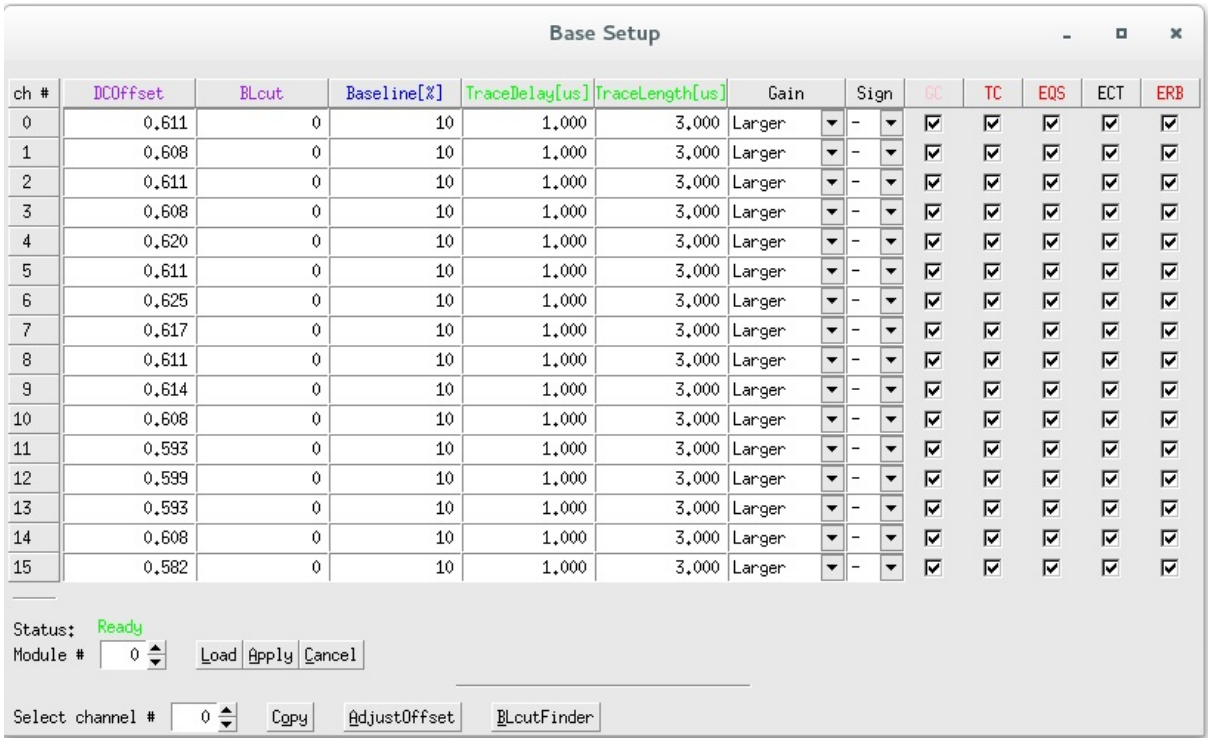


Figure: Base Setup

界面下方的 Status 显示为绿色的 **Ready** 时表示可操作该界面，否则需要等待。

界面中 Module 后面的参数用来选择调节的采集卡模块， **Load** 表示读取该采集卡的参数数值， **Apply** 表示将界面中的数值写入采集卡。

界面下方的 Select Channel 后面的参数表示选择用来将界面上该通道的参数复制给其它通道，点击后面 **Copy** 完成复制，然后需要 **Apply** 来将参数写入采集卡。

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

- 选项 *Gain* 为增益调节，用户可选择 Larger 或者 Small 档，具体每个采集卡这两档所对应的增益参数用户可自行测试或者咨询厂家。
- 选项 *Sign* 选择输入信号的极性，输入正信号选择 "+"，输入负信号则选择 "-"。
- 选项 *GC* 表示是否记录该通道数据，选择表示记录该通道数据，不选择表示不记录。
- 选项 *ECT* 选择表示开启CFD触发功能。否则，则采用快梯形的前沿甄别。

红色的*TC*，*EQS*，*ERB*用来选择输出哪些原始数据：

- 选项 *TC* 选择表示记录波形，此时*TraceDelay*、*TraceLength*生效，不选择则表示不记录波形。
- 选项 *EQS* 选择表示记录八个QDC的积分，不选择则不记录。
- 选项 *ERB* 选择表示记录能量梯形的三部分面积积分及梯形计算的基线数值。

绿色的*TraceDelay*、*TraceLength*为输出数据的点数，该参数除以采集卡的标称采样率即为波形实际输出数据点数：

- *TraceDelay* 表示触发前的采集波形长度。
- *TraceLength* 表示整个波形采集长度。 需要特别说明的是采用降频模式时，实际波形长度为 TraceDelay x 2^N / TraceLength x 2^N（N 为降频参数）

蓝色的 *Baseline* 用来调节基线位置，通过电压的补偿将基线调节到用户预期的位置：

- *Baseline* 可调节范围为 0 - 100，表示波形基线落在满量程的位置百分比。例如垂直精度 14 bit采集卡，该参数设置为10意味着降基线补偿调节到满量程 16384 道的 10% 左右即 1638 附近。

紫色的 *DCOffset*、*BLcut* 用户不需要修改，采用自动调节参数即可。本子菜单中修改了 *Baseline*、*Gain*、*Sign* 之后，需要按最下方的 **AdjustOffset**，之后再按**BLcutFinder** 来自动调节这两个参数。

Important note

> **[info] 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.

> **[info] 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).

 updated: 2018-11-05 16:38:00

Trigger Filter

GUI

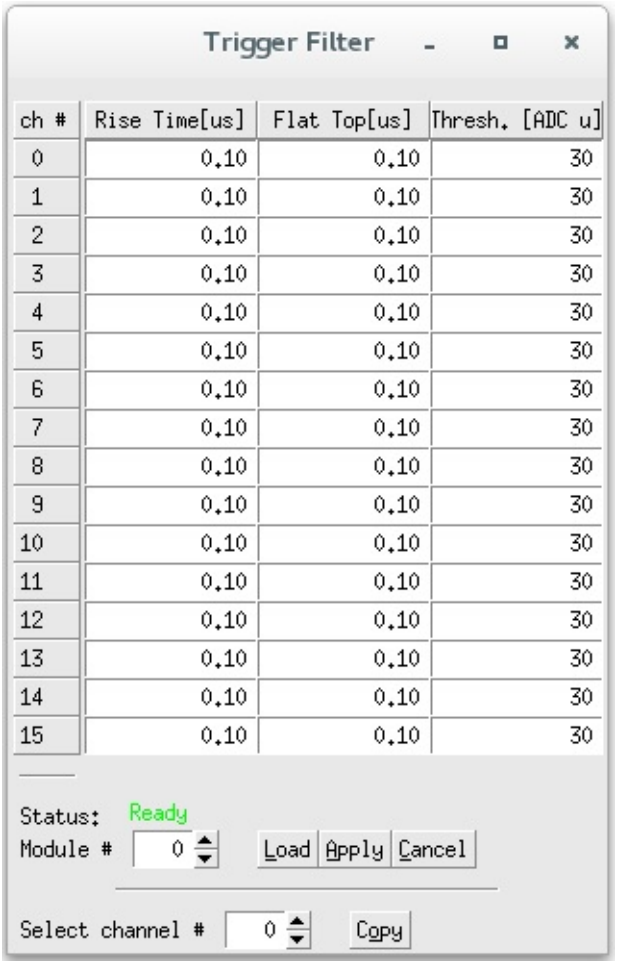


Figure: Trigger Filter

界面下方的 Status 显示为绿色的 **Ready** 时表示可操作该界面，否则需要等待。底下按钮的操作同上。

- 参数 *Rise Time* 。。。
- 参数 *Flat Top* 。。。
- 参数 *Thresh.* 表示阈值，该数值的设置是相对 fast filter 波形。

TODO 这里需要给出不同探测器的推荐值

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** .

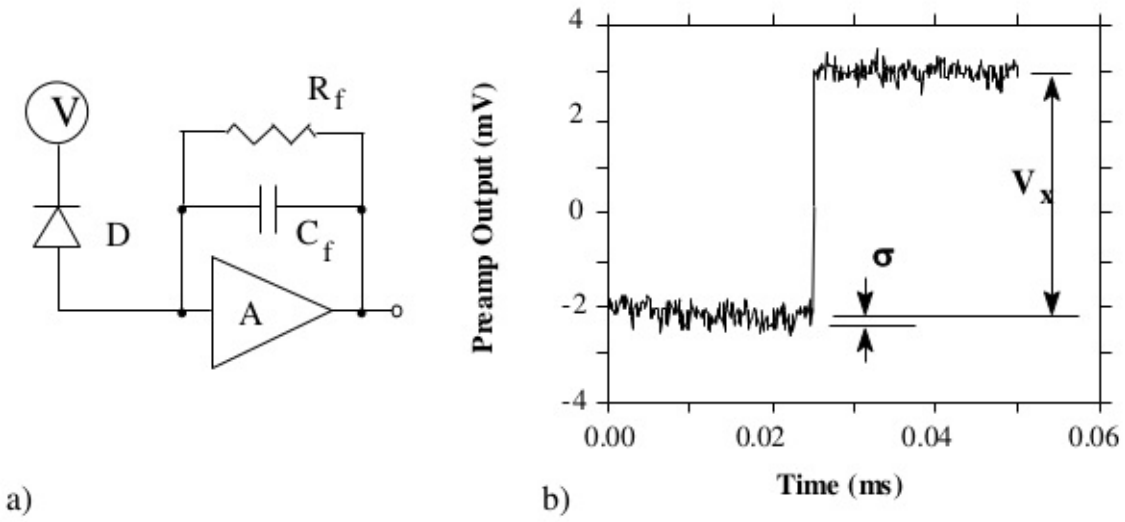


Figure: (a) Charge sensitive preamplifier with RC feedback; (b) Output on absorption of a gamma ray

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 Figure *Digitized version of the data of Figure (b) in the step region*, 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:

Where the values of the weighting constants W i 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.

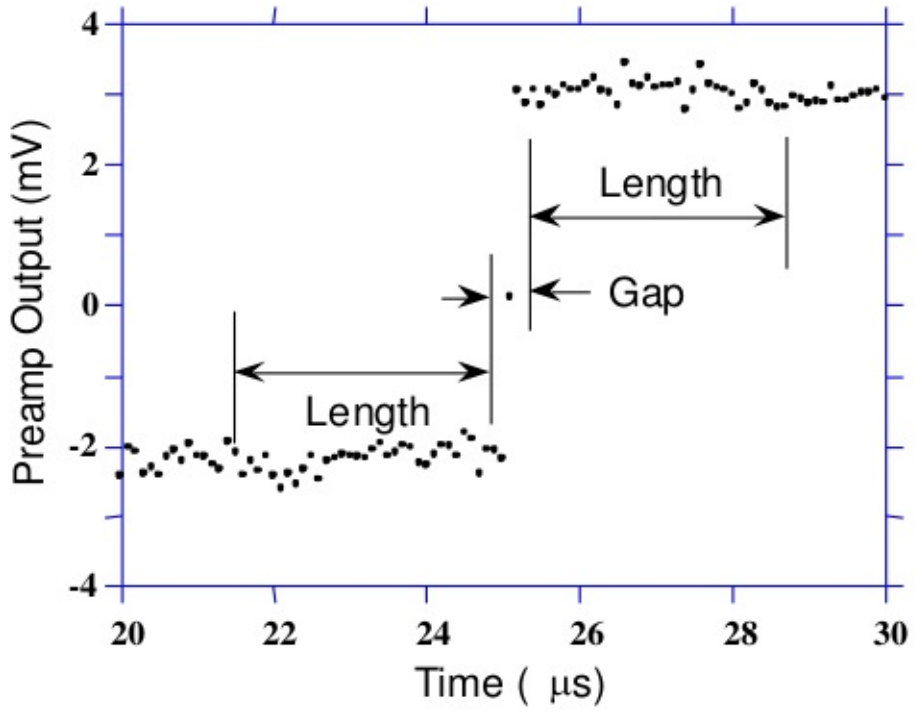


Figure: Digitized version of the data of Figure (b) in the step region

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

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

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.

 updated: 2018-11-05 16:38:42

Energy

GUI

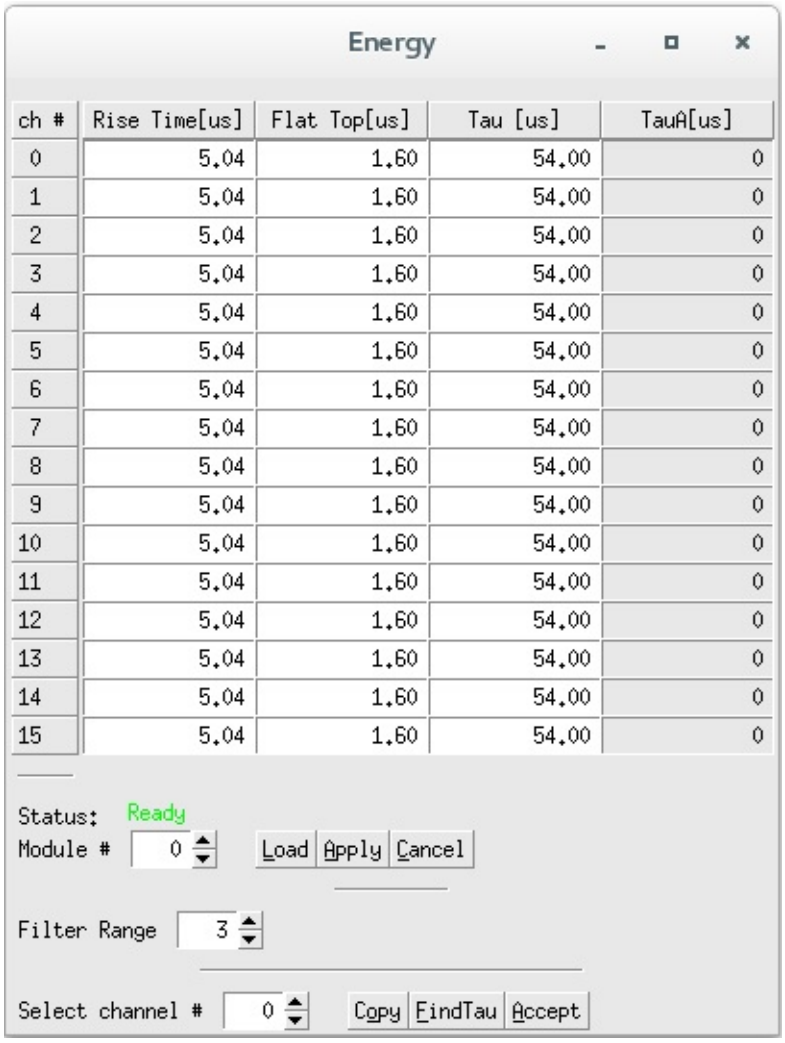


Figure: Energy

界面下方的 Status 显示为绿色的 **Ready** 时表示可操作该界面，否则需要等待。底下按钮的操作同上。

- 参数 Rise Time，请参考*Trapezoidal Filtering*部分
- 参数 Flat Top，请参考*Trapezoidal Filtering*部分
- 参数 Tau，请参考*Baselines and Preamp. Decay Times*部分
- 参数 filter range，请参考*Filter Range*部分

TODO 这里应该补充不同探测器的推荐值

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.

> **[danger] 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*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

Figure: Filter clock decimations and filter time granularity for 100 MHz or 500 MHz

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

Figure: Filter clock decimations and filter time granularity for 250 MHz

Trapezoidal Filtering

From this point onward, only trapezoidal filtering will be considered as it is implemented in a Pixie-16 module according to Equation . 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.

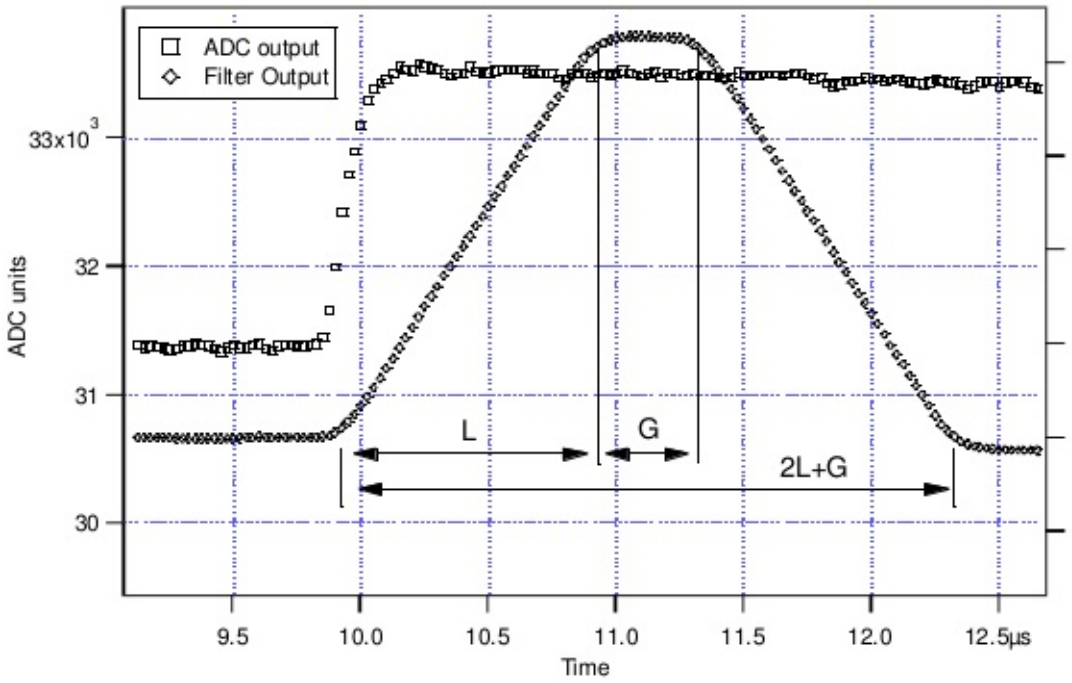


Figure: Trapezoidal filtering of a preamplifier step with L=1µs and G=0.4µs

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 time because it has a longer filtering time.
  - 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.

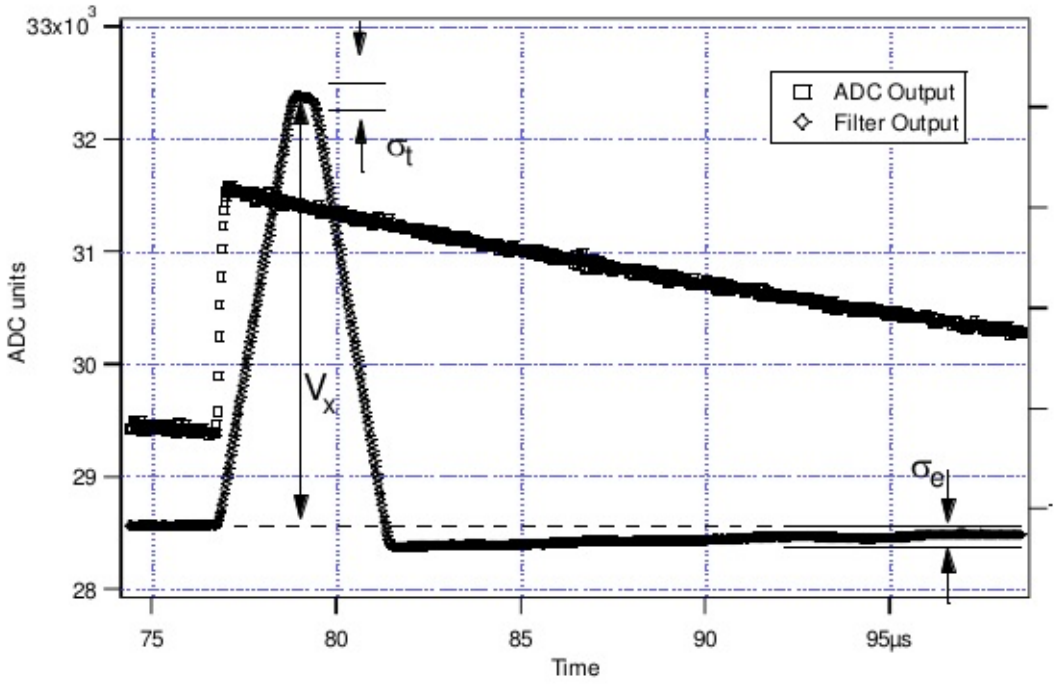


Figure: A gamma-ray event displayed over a longer time period to show baseline noise and the effect of preamplifier decay time

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 adds in quadrature with the electronic noise, so that the total noise in measuring Vx is found from:

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 , 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 , 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 amma-ray detected and use these values to construct a spectrum.

> **[info] info**
>
> 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 Figure *Peak detection and sampling*. 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.

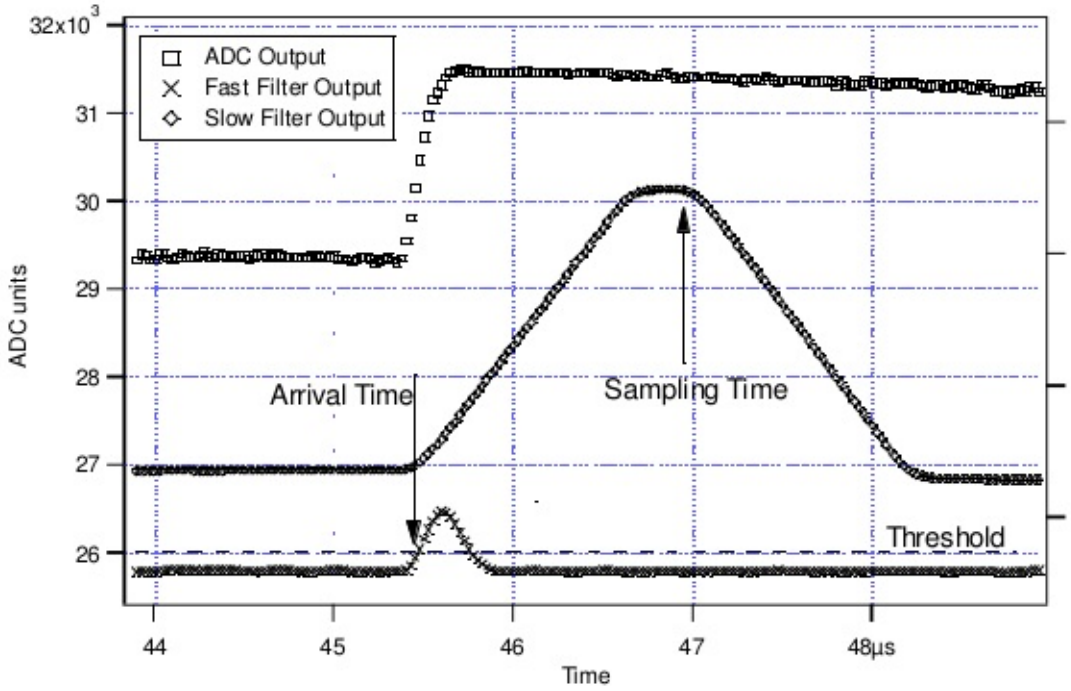


Figure: Peak detection and sampling

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, P**EAKSAMP** 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.

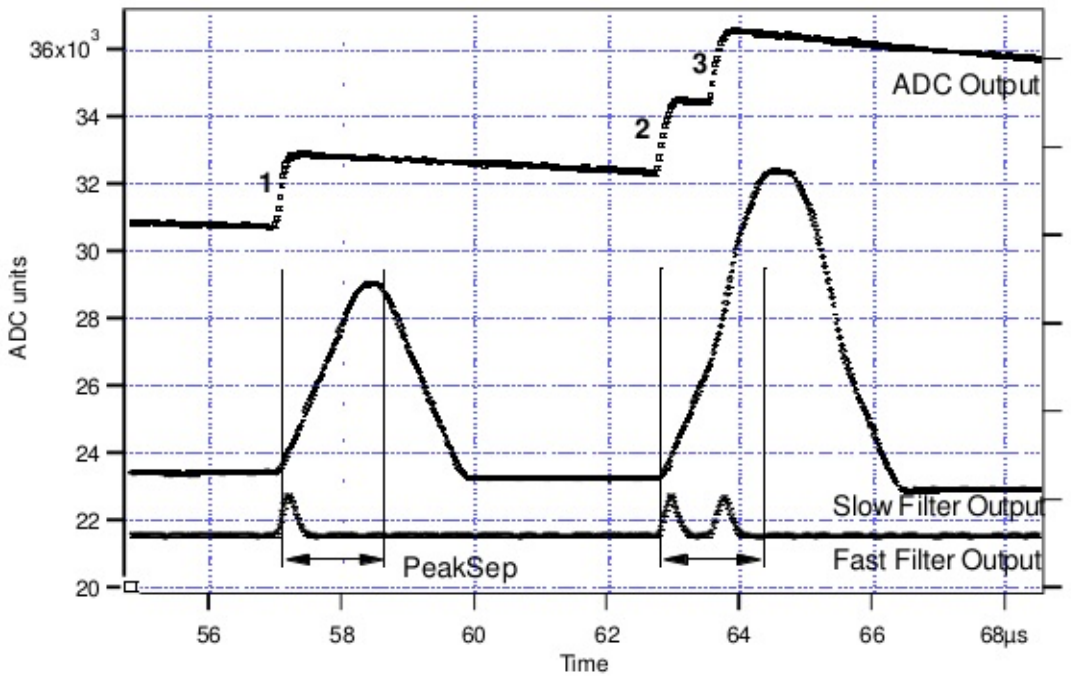


Figure: A sequence of 3 gamma-ray pulses separated by various intervals to show the origin of pileup and demonstrate how it is detected by the Pixie module

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.

 updated: 2018-11-05 16:56:26

CFD

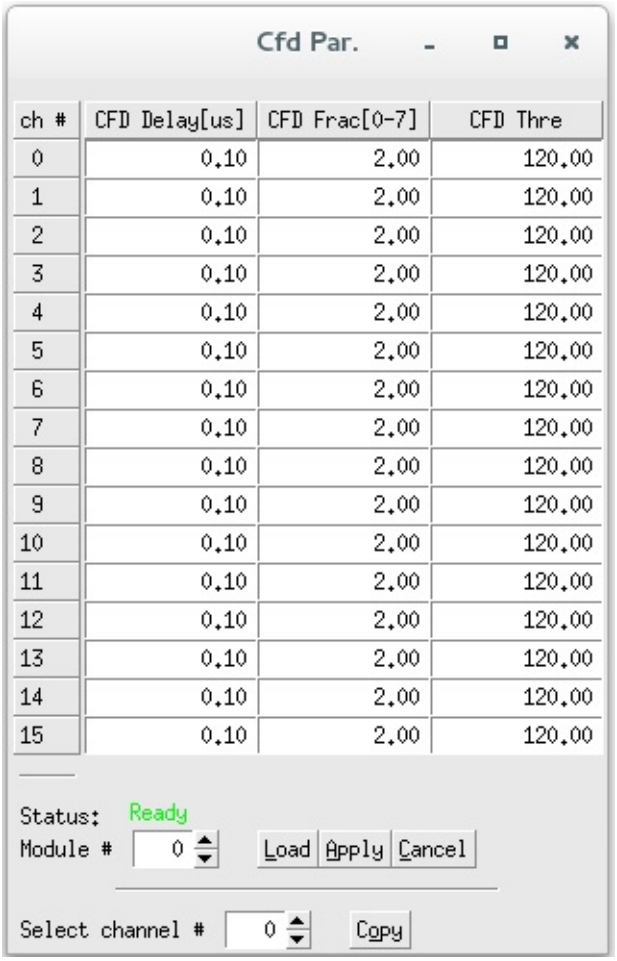


Figure: CFD

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

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 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.

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.

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

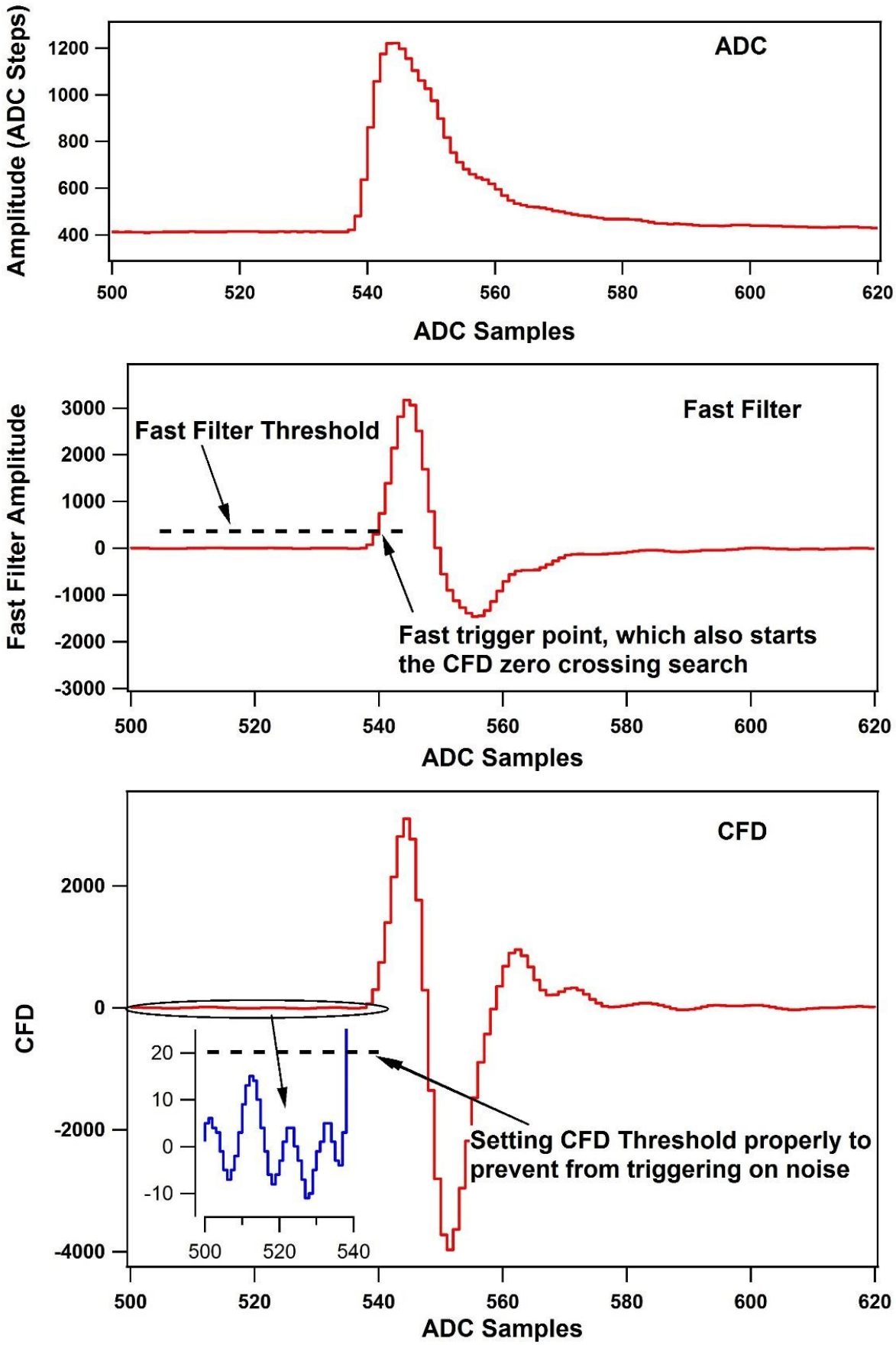


Figure shows a sample ADC trace, its fast filter response and its CFD response, respectively.

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

Figure: Corresponding DSP Parameters for the CFD Parameters

> **[info] 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.
>
> **[info] 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:

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

Figure: Fixed CFD Parameter Values for 500 MHz Pixie-16 Modules

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.

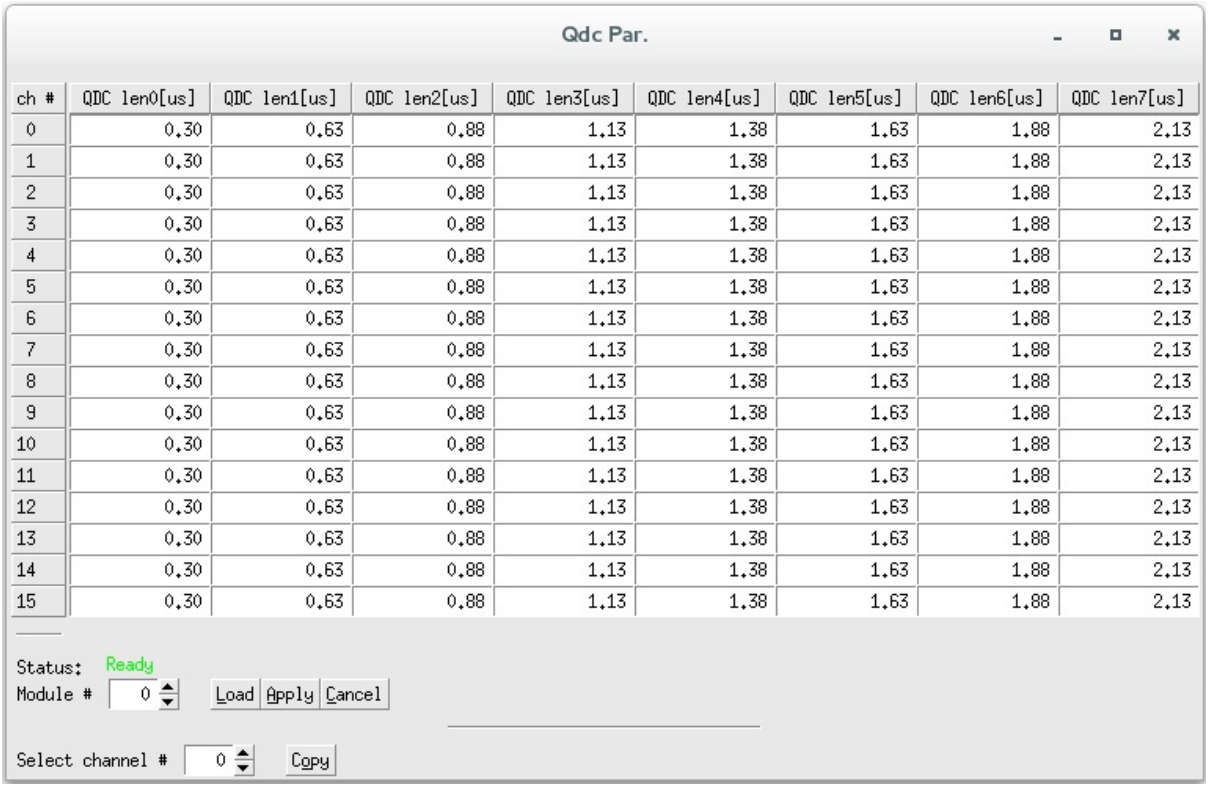
CFD Trigger Source [2:0]	Zero Crossing Point (ZCP) Location
000	ZCP occurred between the 5th ADC sample of the previous 5-sample group and the 1st ADC sample of the current 5-sample group
001	ZCP occurred between the 1th ADC sample of the current 5-sample group and the 2nd ADC sample of the current 5-sample 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

Figure: Meanings of the CFD Trigger Source for 500 MHz Pixie-16 Modules

The CFD fractional time is given as follows:

 updated: 2018-11-03 15:08:17

QDC



ch #	QDC len0[us]	QDC len1[us]	QDC len2[us]	QDC len3[us]	QDC len4[us]	QDC len5[us]	QDC len6[us]	QDC len7[us]
0	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
1	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
2	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
3	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
4	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
5	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
6	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
7	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
8	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
9	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
10	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
11	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
12	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
13	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
14	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13
15	0.30	0.63	0.88	1.13	1.38	1.63	1.88	2.13

Figure: QDC

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.

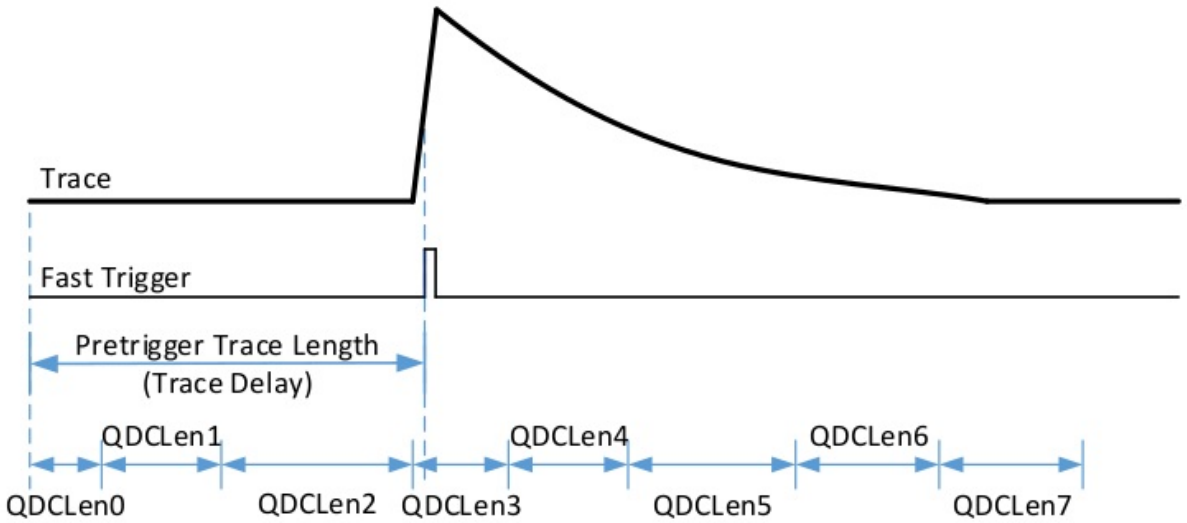


Figure: The 8 QDC sums of a triggered event

 updated: 2018-11-03 15:08:17

Decimation

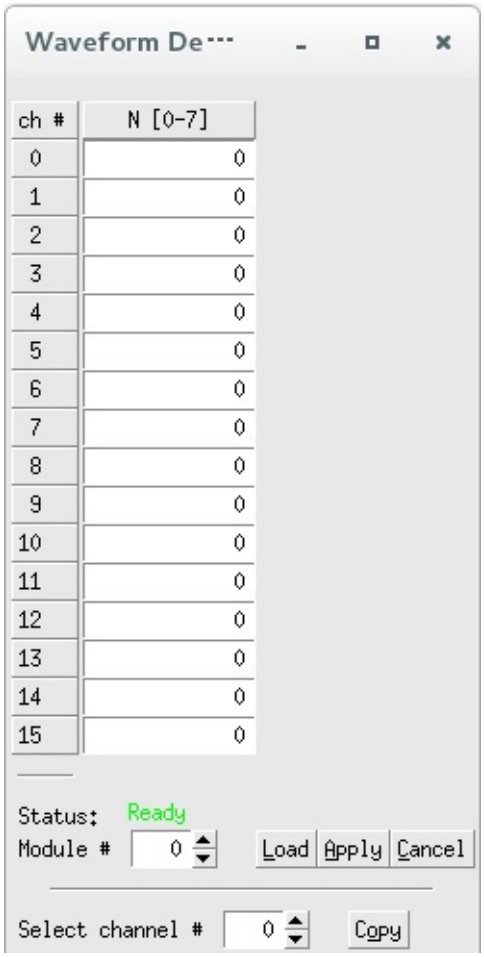


Figure: Decimation

。。**TODO**。。

 updated: 2018-11-03 15:08:17

Copy Pars

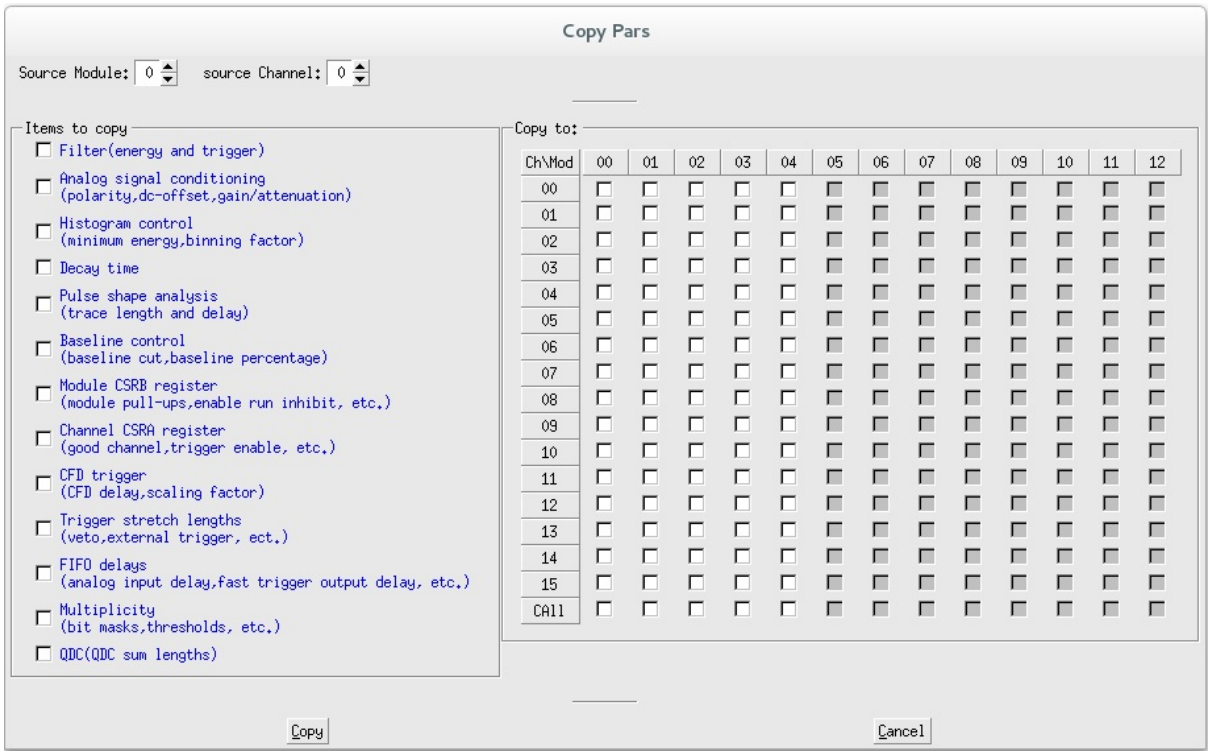


Figure: Copy Pars

。。**TODO**。。

 updated: 2018-11-03 15:08:17

Save2File

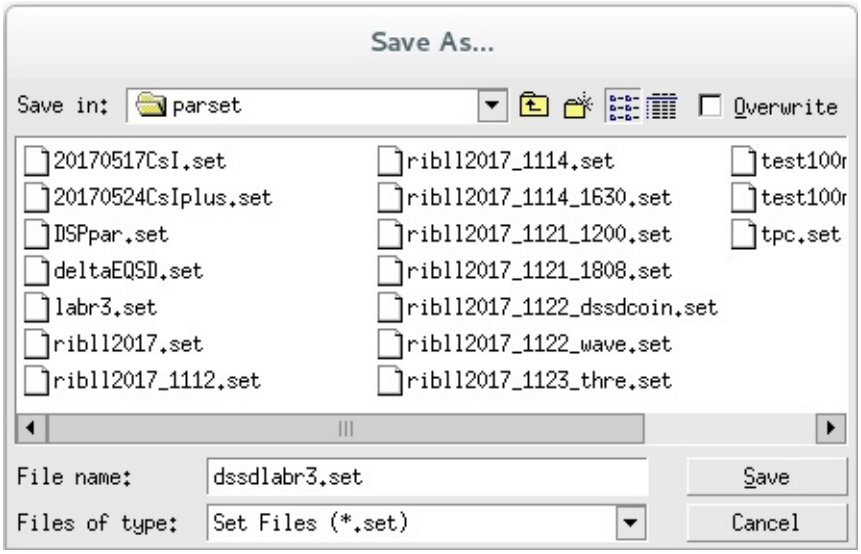


Figure: Save2File

。。**TODO**。。

 updated: 2018-11-03 15:08:17

Expert

本下拉栏中调节内容为高阶内容，需要对获取逻辑有一定基础的人学习掌握。

 updated: 2018-11-05 16:57:19

Module Variables

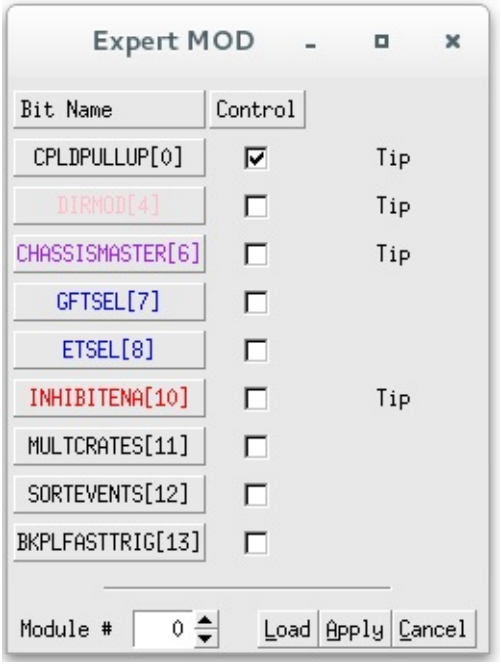


Figure: Module Variables

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.

> **[info] 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)**

 updated: 2018-11-03 15:08:17

CSRA

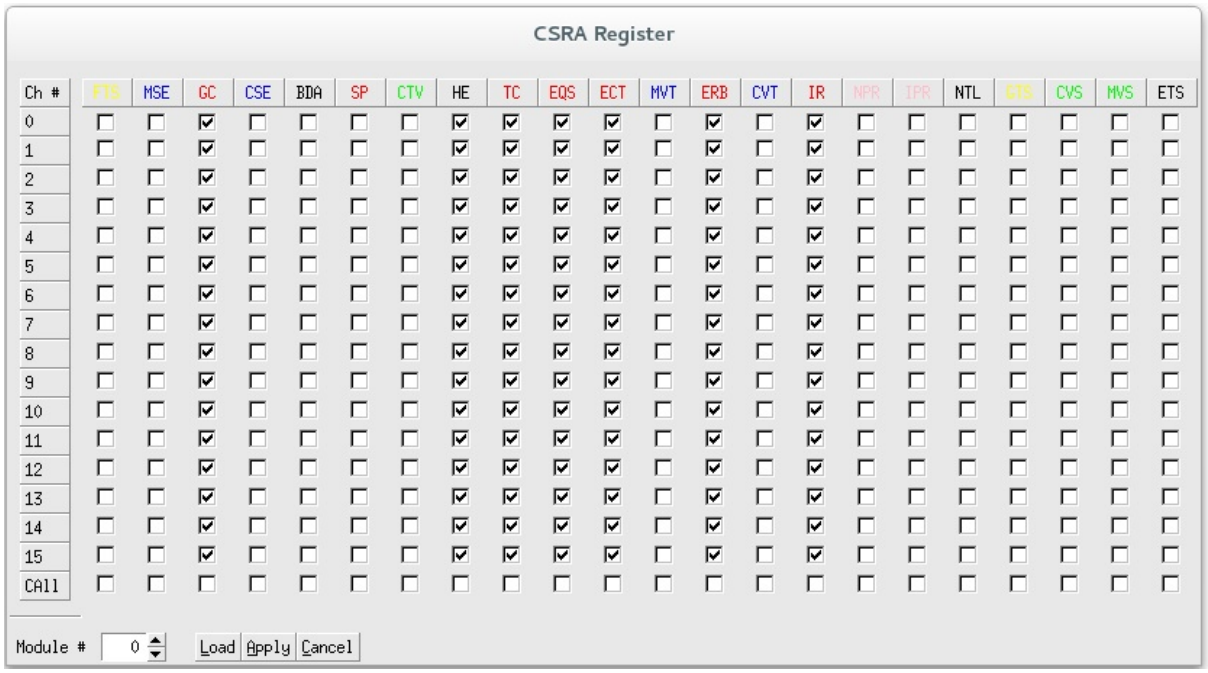


Figure: CSRA

- 黄色 FTS、GTS 组合来选择 channel fast trigger：
  - 两个均不选时为 local fast trigger
  - 选择 FTS 时为 latched module fast trigger
  - FTS 不选、GTS 选上时为 latched channel validation trigger
- 蓝色 MSE、CSE、MVT、CVT 用来选择 module/channel validation trigger：
  - MVT 为是否开启 module validation trigger
  - CVT 为是否开启 channel validation trigger
  - MSE选择 module validation trigger 来源于 System FPGA 还是前面板 module GATE
  - CSE选择 channel validation trigger 来源于 System FPGA 还是前面板 channel GATE
- 粉红色 NPR、IPR 组合选择 pileup 事件的处理：
  - 两个均不选时则记录所有事件，堆积事件能量值为无效
  - NPR 选择 IPR 不选时不记录堆积事件
  - NPR 不选 IPR 选择时堆积事件记录波形、不堆积时候不记录波形
  - 两个均选择时只记录堆积事件
- 绿色CTV、CVS、MVS用来选择module/channel veto：
  - MVS 选择 module veto 来源于前面板 module GATE 还是 module validation trigger
  - CVS 选择 channel veto 来源于前面板 channel GATE 还是 channel validation trigger
  - CTV 为是否开启 channel trigger veto
- 红色为基础设置
  - 黑色 NTL 是否保留超出量程的波形
  - 黑色 ETS 是否记录外部时钟的数据
- 剩余的BDA不选，HE不管

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

 updated: 2018-11-05 16:58:47

Logic Set

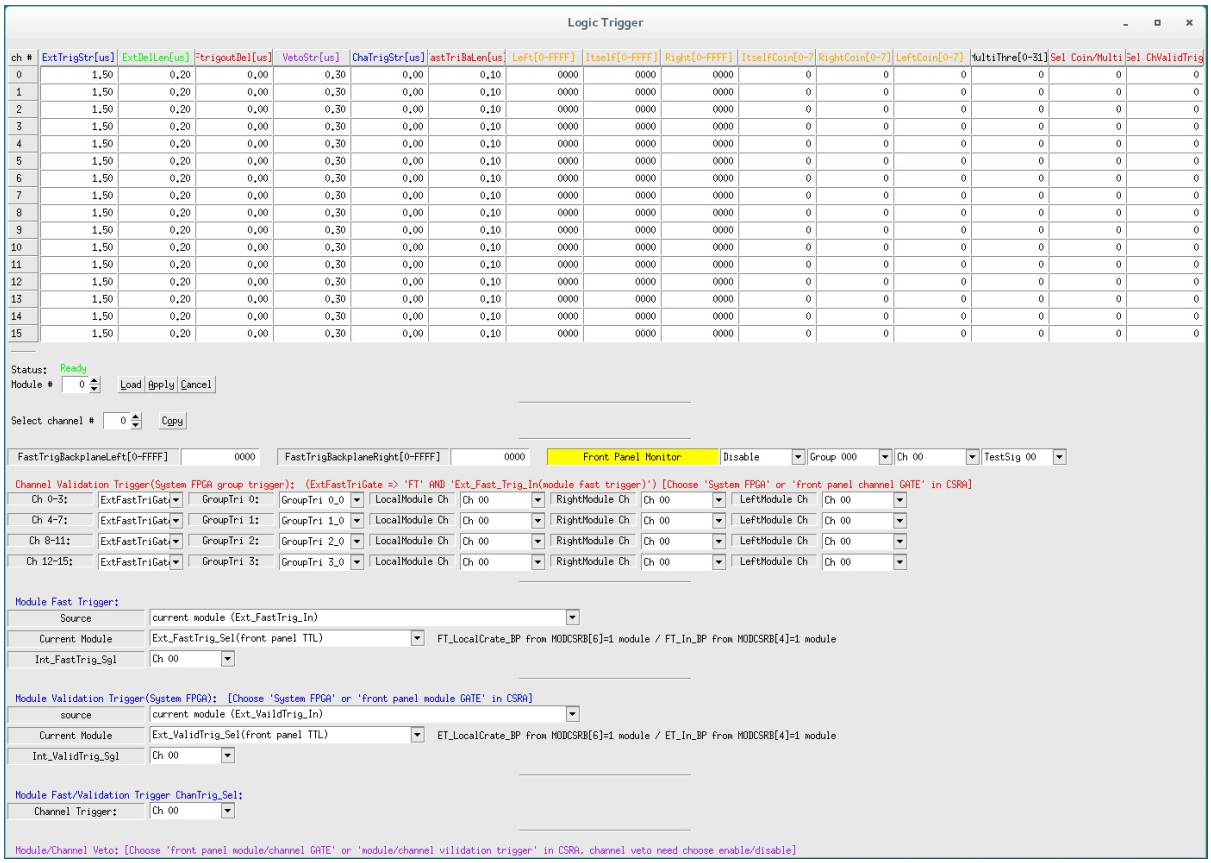


Figure: Logic Set

TODO

			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

Figure: stretch length

 updated: 2018-11-03 15:08:17

Monitor

本下拉栏中调节内容为监视波形噪声水平、基线分布等。

 updated: 2018-11-05 16:59:03

Hist & XDT

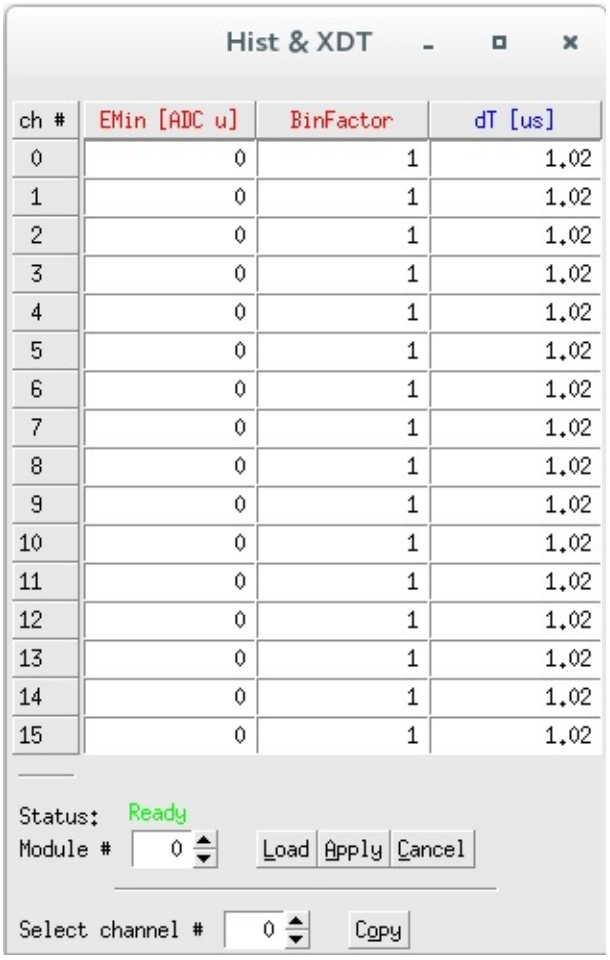


Figure: Hist & XDT

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 $2^{binning factor}$. 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.

 updated: 2018-11-03 15:08:17

Trace & Baseline

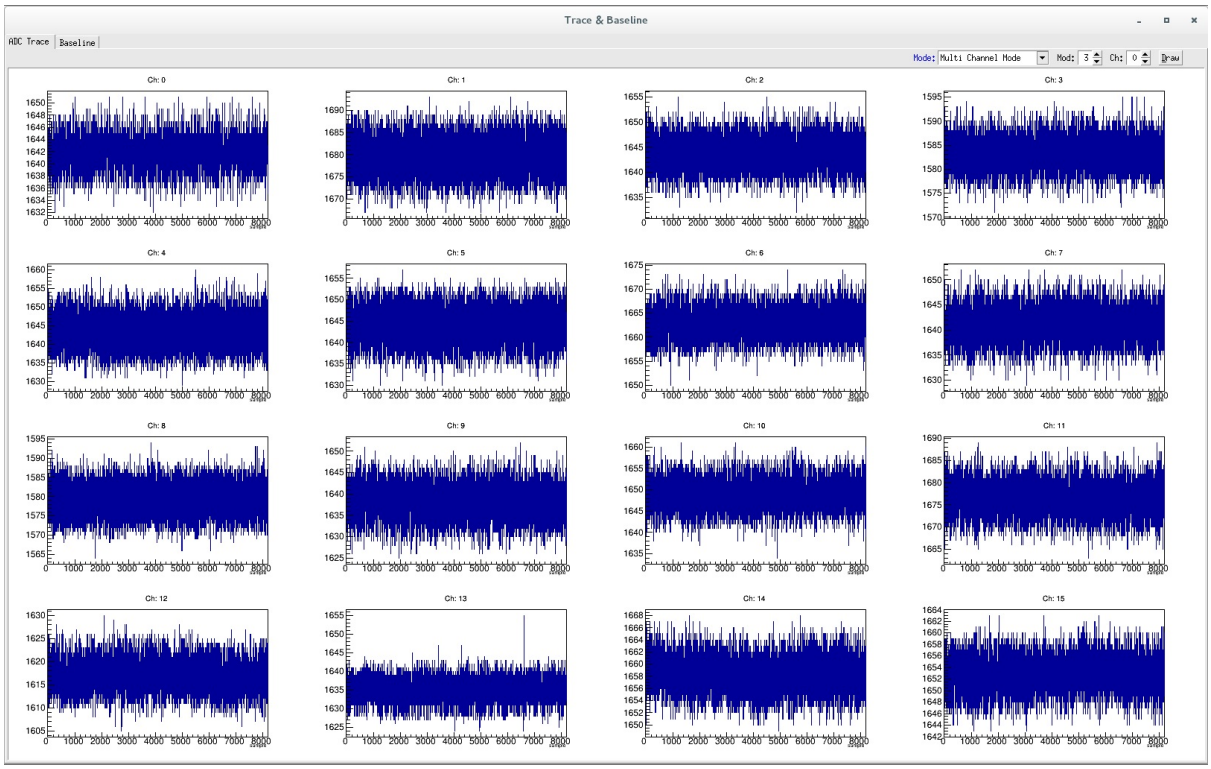


Figure: Trace & Baseline

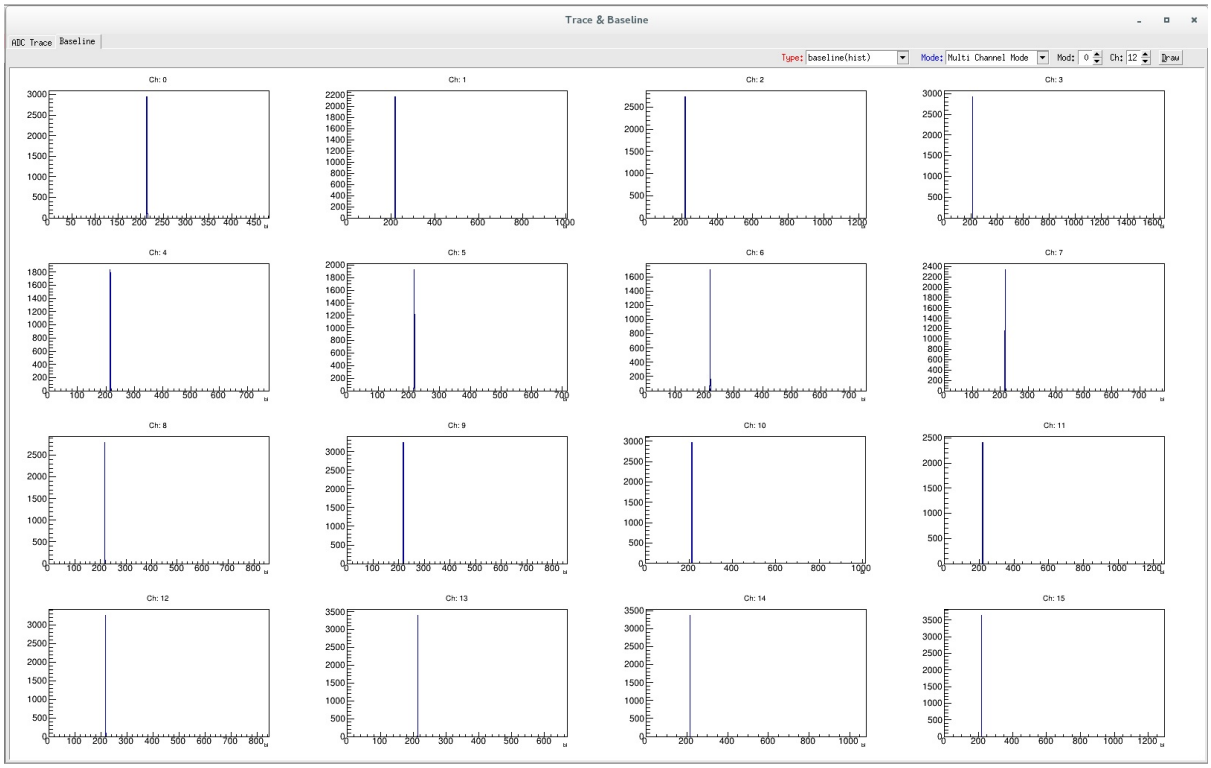


Figure: Trace & Baseline

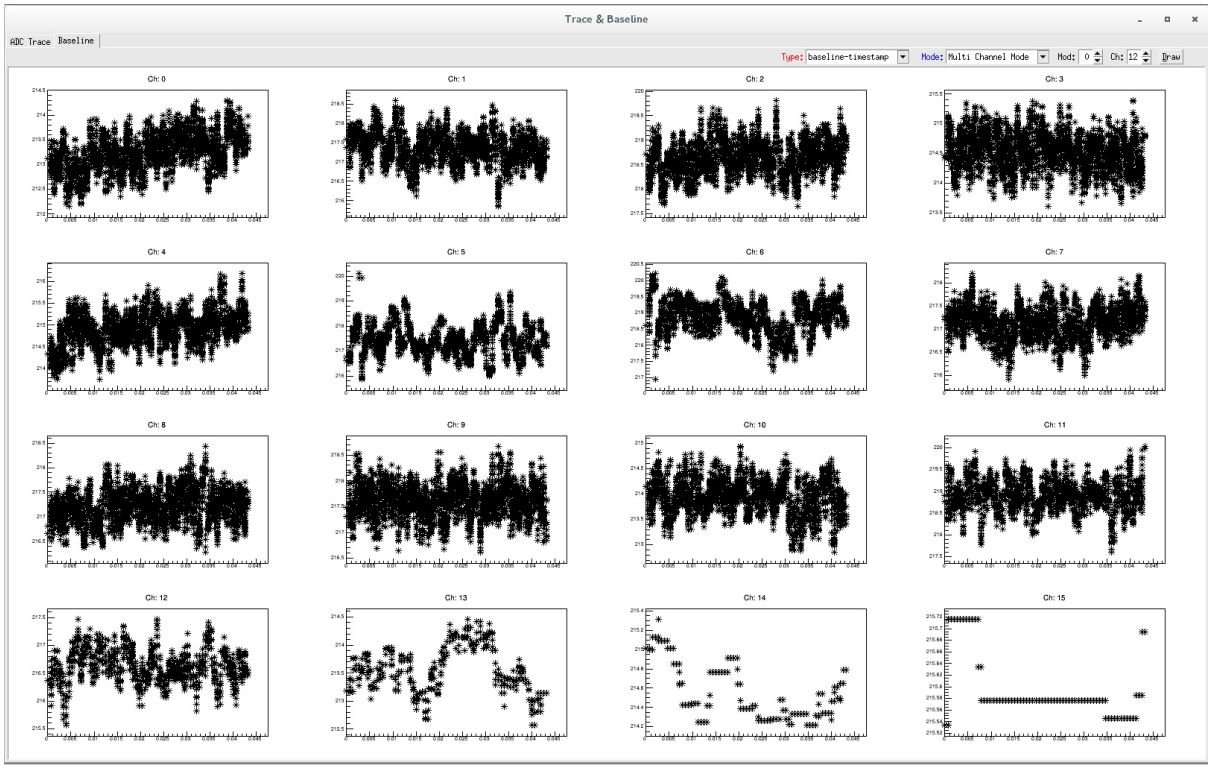


Figure: Trace & Baseline

。。TODO。。

 updated: 2018-11-03 15:08:17

Offiline

本下拉栏中为离线参数优化调节。

InitData

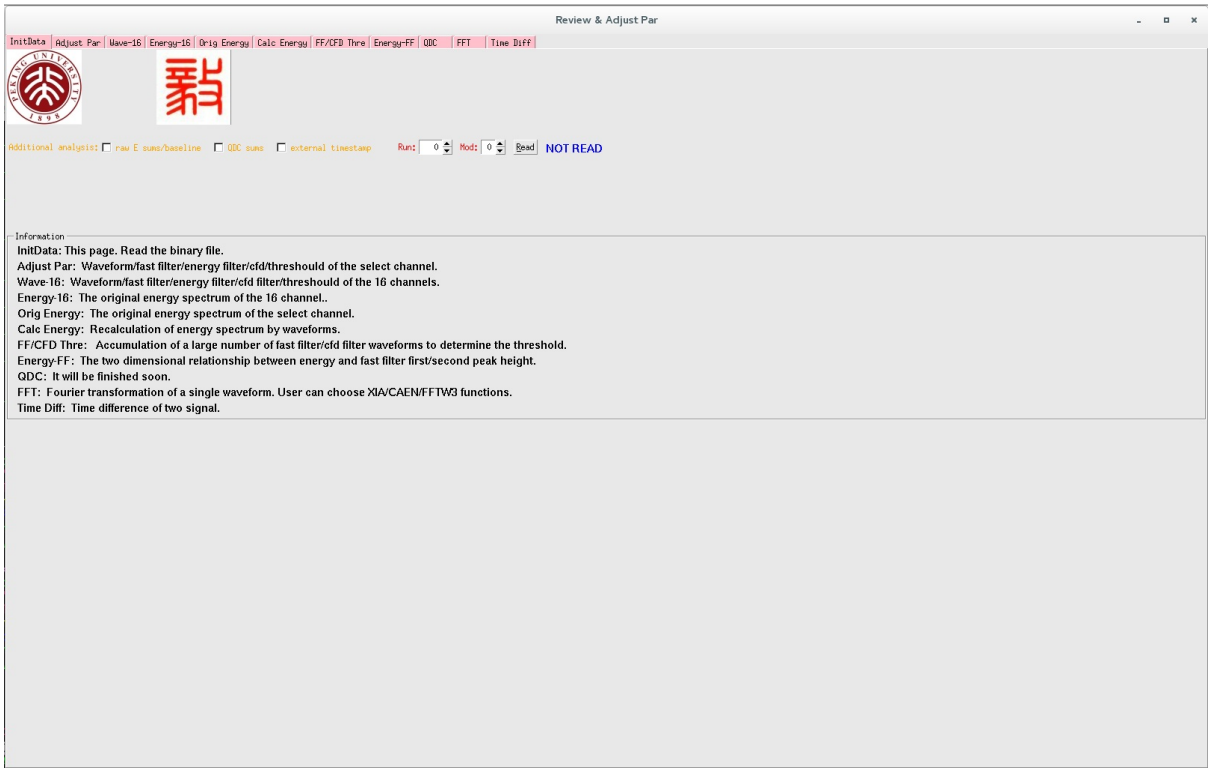


Figure: Adjust Par

- Run 选择要读取的文件运行编号，Mod 选择要读取第几个采集卡，按钮 Read 将文件主要信息(道址、能量、波形位置等)载入内存。
- Additional analysis: 三个选项中，选择表示读取该文件数据到内存中时包括该信息。只有读取了该数据，才能启用一些分析方法。但是前提是数据采集时候需要记录该信息。

- 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.

 updated: 2018-11-05 17:00:08

Adjust Par

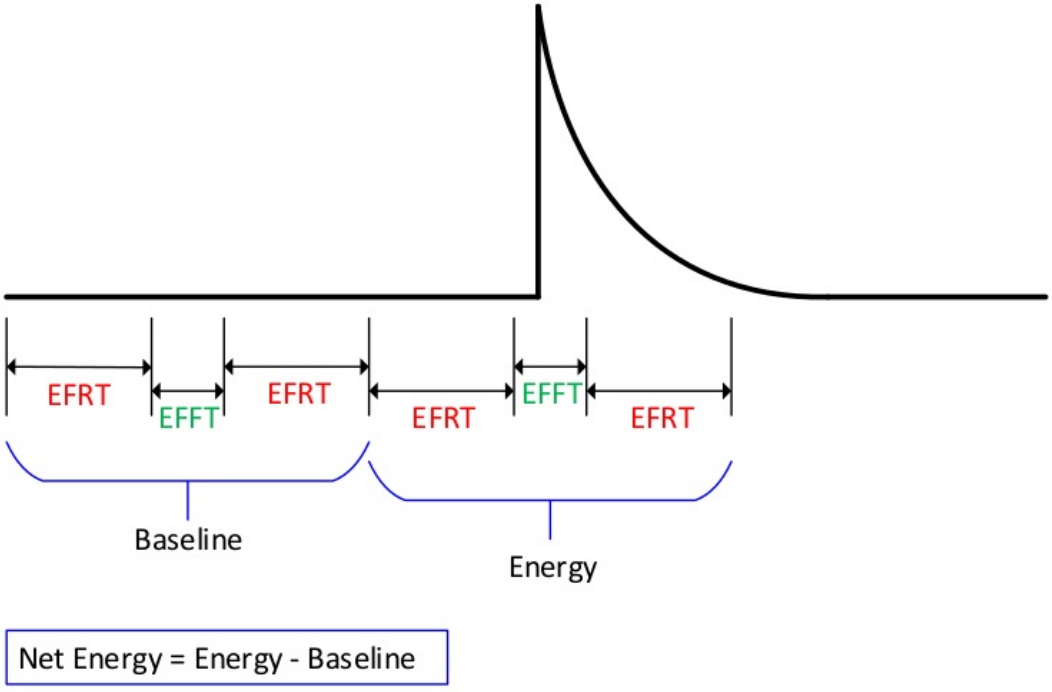


Figure: offlineapi

要通过采集的波形离线计算fast filter、slow filter cfd 曲线，对采集的波形有以下要求。如上图中，计算的能量是算法的能量与算法的基线的差，要得到正确的梯形，那么前提是前面有足够长的点来计算基线。

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 *(2* EFRT + EFFT)
- Pre-trigger trace (Trace-delay) length > (3 * EFRT + EFFT)

当然，这只是计算梯形的一个方法，如果我们记录了每个事件的能量梯形的基线，并且采用pre-trigger部分点的平均值作为波形左侧的无限延伸，那么就不受 Pre-trigger trace length > (3 x EFRT + EFFT) 条件的限制了。下面的页面中，当采用 Old Baseline 方法来计算能量梯形时，有个前提是 pre-trigger trace length 至少需要有 200 个点，因为波形左侧延伸采用前 200 个点来平均。

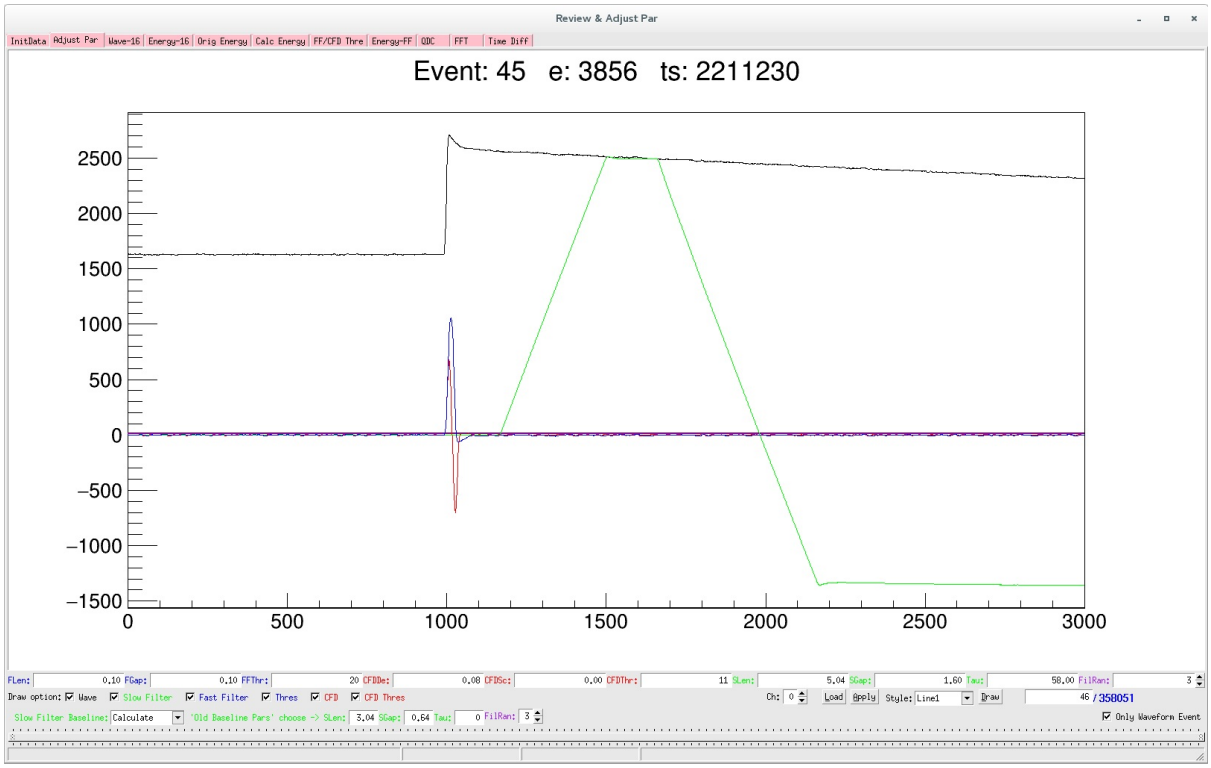


Figure: Adjustpars3

当采集的波形 pre-trigger trace length > 3 x EFRT + EFFT 时，pre-trigger trace 提供足够多的点来计算基线，SF BL 算法可选择 Calculate，否则需要选择 Old Baseline 算法。选择Old Baseline 算法的前提时记录数据的时候， 选择开启记录 梯形的baseline，并且 InitData 页面的 raw E sums/baseline 选项开启。当选择 Old Baseline 算法时，之后的四个选项参数生效，该四个参数为该数据采集时候所用的能量梯形的参数。

上图中绿色曲线为典型的不满足 pre-trigger trace length > 3 x EFRT + EFFT 时，采用的 Calculate 算法造成的结果。图中显示 pre-trigger trace length 为 10 us，EFRT 为 5.04 us，EFFT 为 1.60 us。

此时，应该采用下图所示的 Old Baseline 算法。

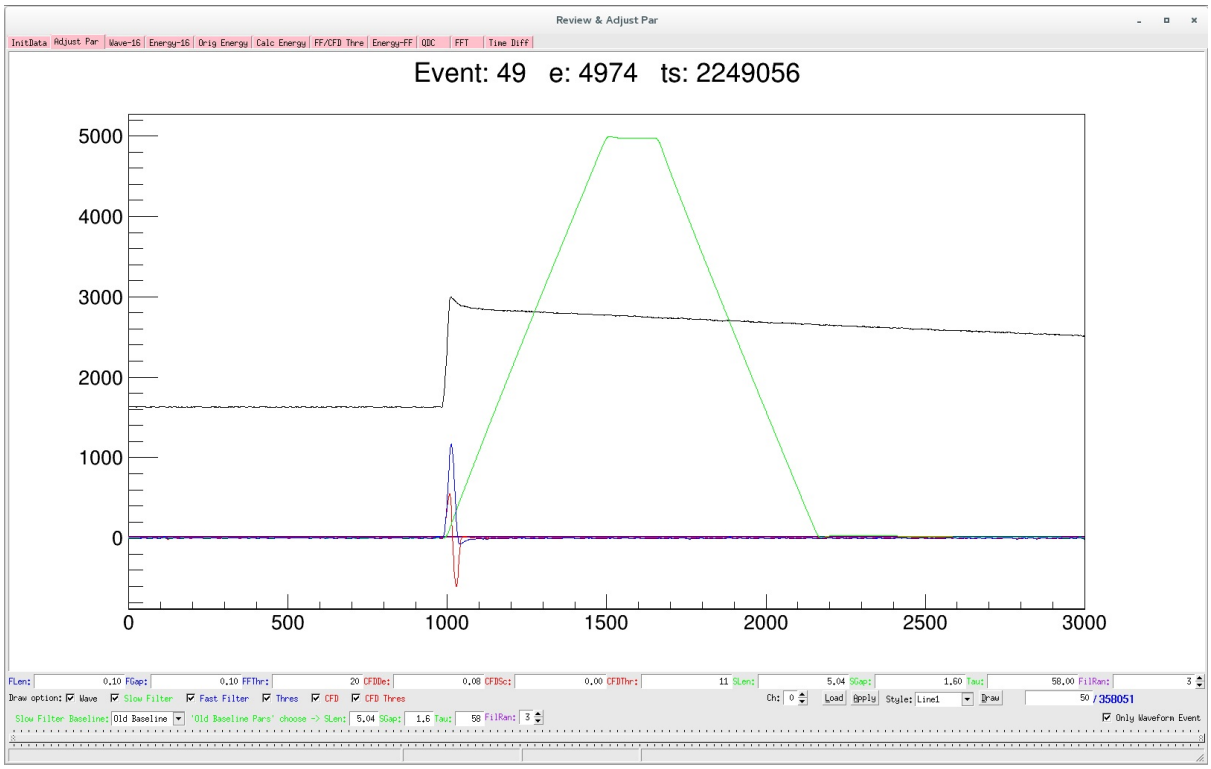


Figure: Adjustpars1

用户可选择查看波形的通道，按钮 Load 可读取并显示当前的参数设置情况，当修改以上的参数时候，需要按 Apply 按钮使之生效。按钮 Draw 用来显示下一个该通道的事件波形。

用户可选择同时显示 Wave / Slow Filter/ Fast filter / Thres / CFD / CFD Thres 中的多个波形。或者选择曲线的绘画样式。

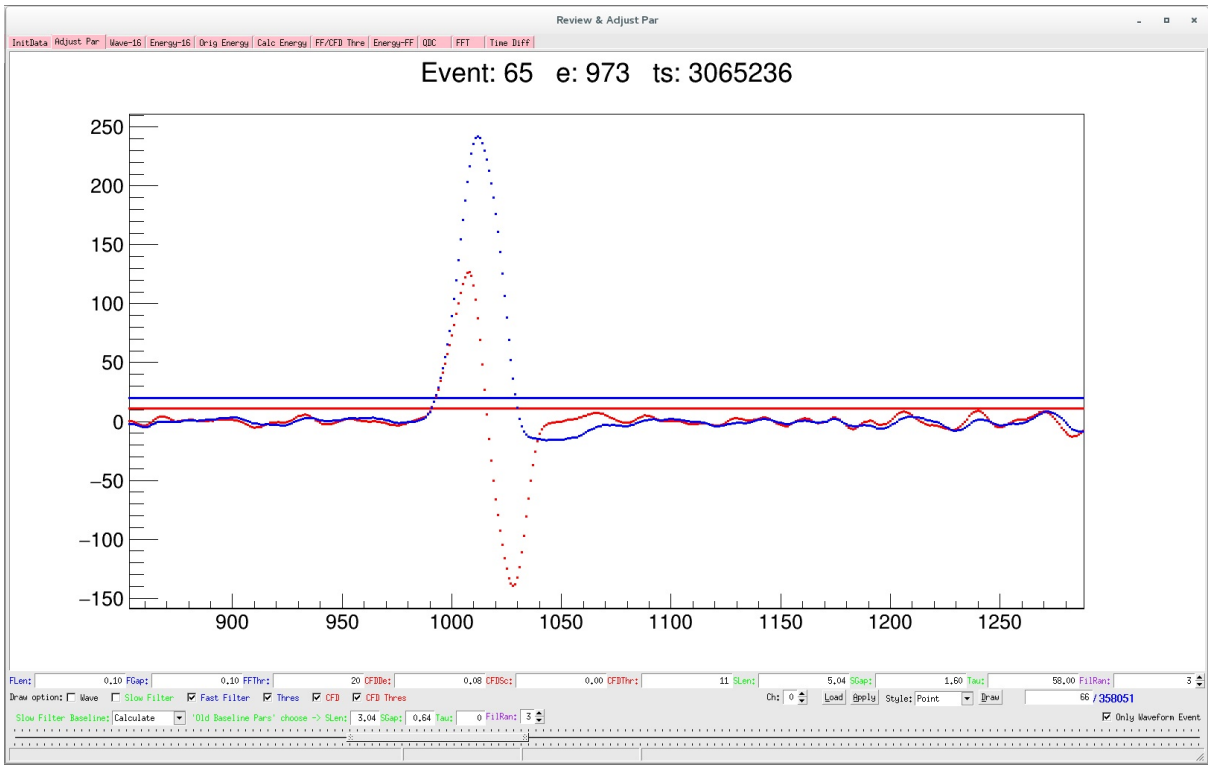


Figure: Adjustpars2

上图展示了显示 fast filter、Thres、CFD、CFD Thres 四个波形，图样采用点显示方式。最低端的水平条两端可以拖动，用户可拉动来控制波形横坐标的显示范围，如图中显示800 - 1300 的点。该情况下，点击 Draw 按钮，将会保持该指定的坐标范围。

 updated: 2018-11-05 17:02:09

Wave-16

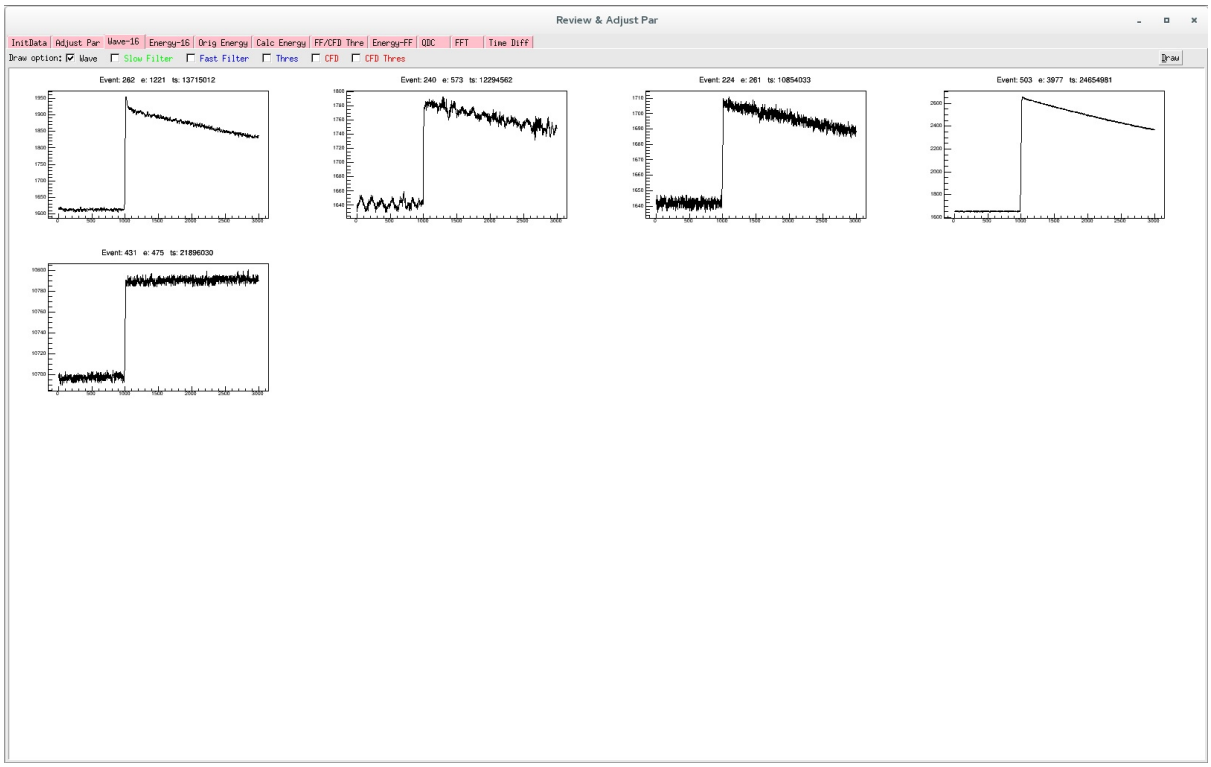


Figure: Waveform16

该页面用于同时查看16通道的原始波形、filter 波形，阈值等。用户可选择同时显示 Wave / Slow Filter/ Fast filter / Thres / CFD / CFD Thres 中的多个波形。

用户可通过该页面，快速查看该采集卡所有通道的波形是否正常，参数设置是否合理。点击按钮 Draw 一次，则显示所有通道下一个波形。

需要注意的一点，本页面的 *Slow Filter* 波形需要在采集的波形 *pre-trigger trace* 长度大于 *3 x EFRT + EFFT* 时才是正确的。

 updated: 2018-11-05 17:04:50

Energy-16

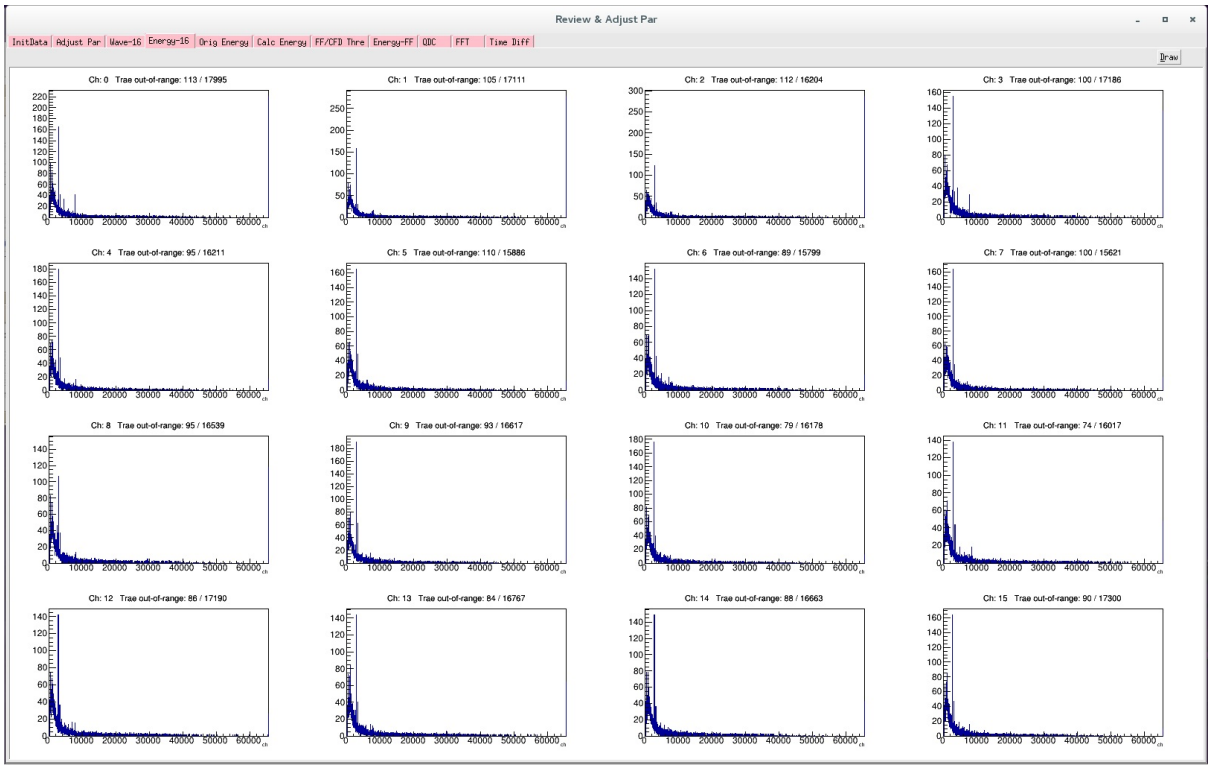


Figure: Energy16

该界面用于同时查看 16 通道的一维能谱。点击右上角的按钮 Draw 即可。

 updated: 2018-11-05 17:04:59

Orig Energy

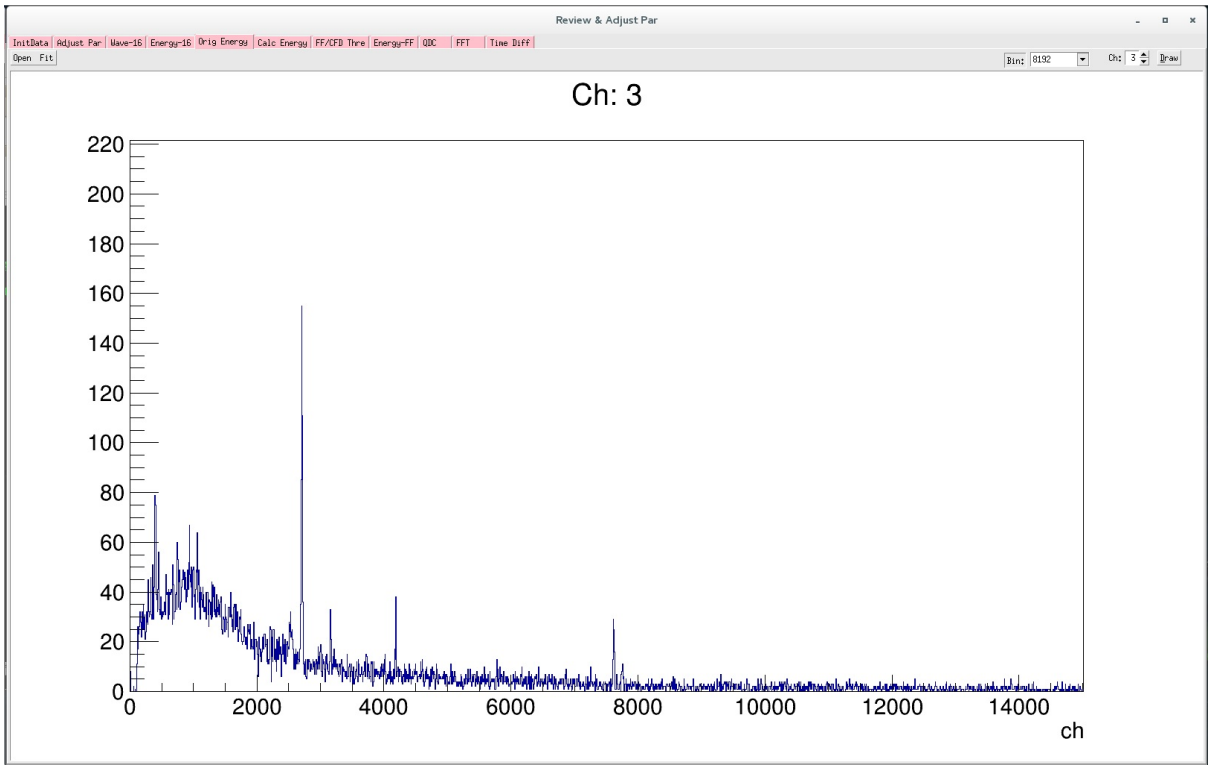


Figure: OrigEnergy

该页面用来快速查看某通道的能谱。用户选择能谱的分 Bin 数，该数值表示将 0 - 65536 道分成多少份。选择查看通道。然后按 Draw 按钮即可。

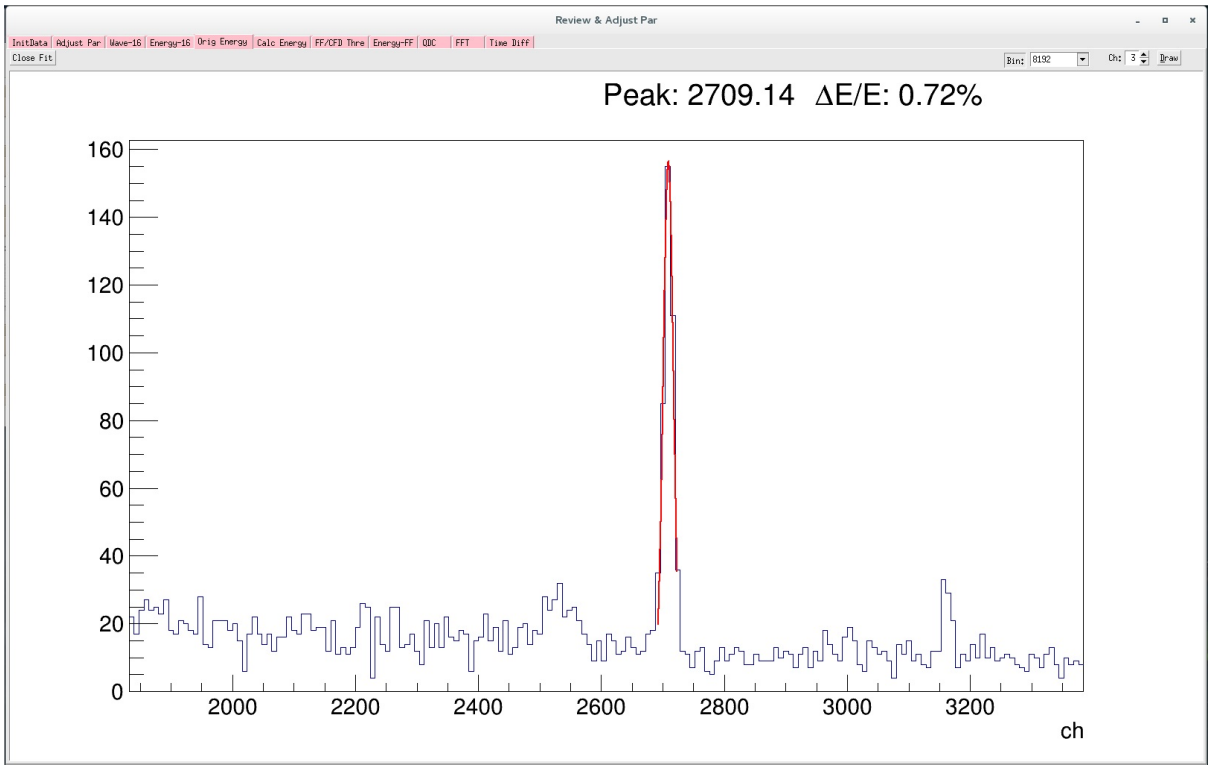


Figure: OrigEnergyFit

左上角的 Open Fit 按钮用来快速高斯拟合看能量分辨。点击按钮，开启拟合模式，再次点击按钮则关闭该功能。将鼠标移动到直方图的蓝线上，鼠标十字将会变成三角箭头。三角箭头的鼠标点击直方图中的两个位置，两点所在区间即为拟合区间，则可查看能量分辨。

 updated: 2018-11-05 17:05:18

Calc Energy

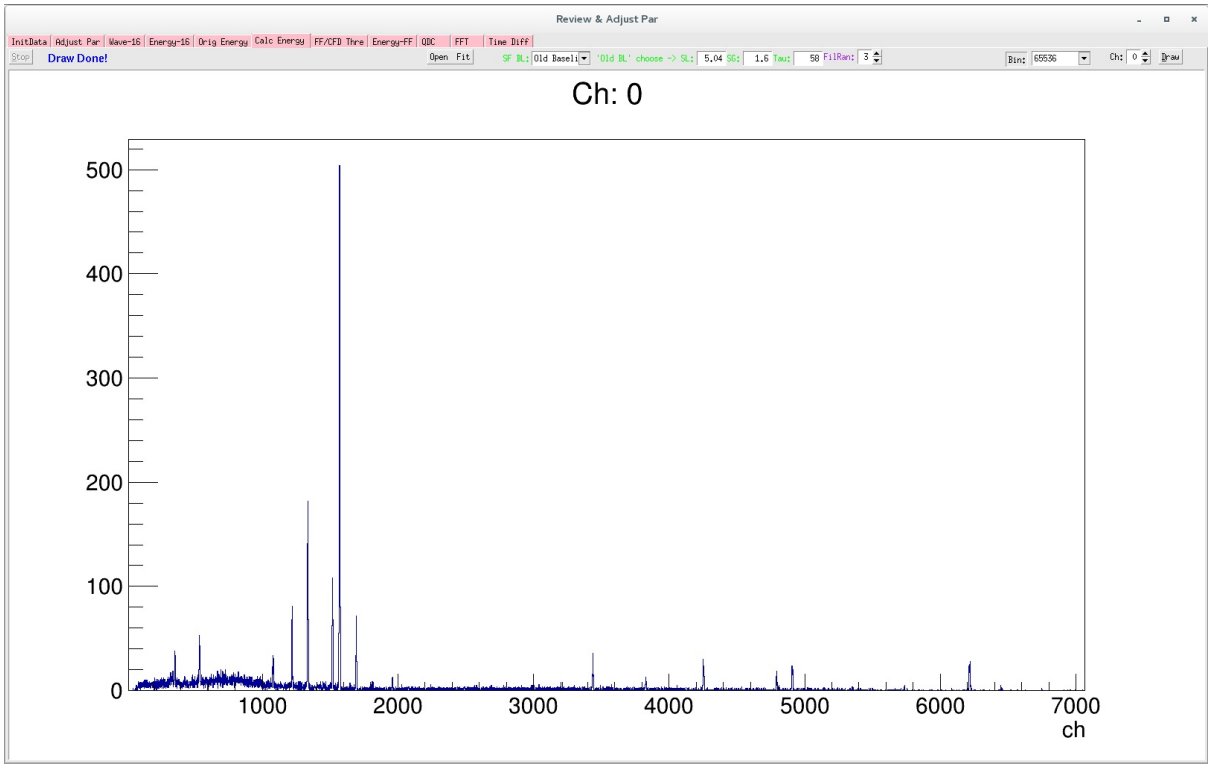


Figure: CalcEnergy

该页面利用采集的波形重新计算能量。同 Adjust Par 页面一样，SF BL 算法可选择Calculate算法或者 Old Baseline 算法。

计算能量采用的fast filter、energy filter参数采用采集卡的设置参数，用户需要选择能量 0-65536 分成多少个 bin，可选择 1024/2048/4096/8192/16384/32768/65536，选择计算的通道，然后按按钮 Draw即开始计算，左上角将会显示计算的进度，也可以按按钮 Stop 提前终止计算。当计算终止时，画板上将显示能谱。

 updated: 2018-11-05 17:05:52

FF/CFD Thre

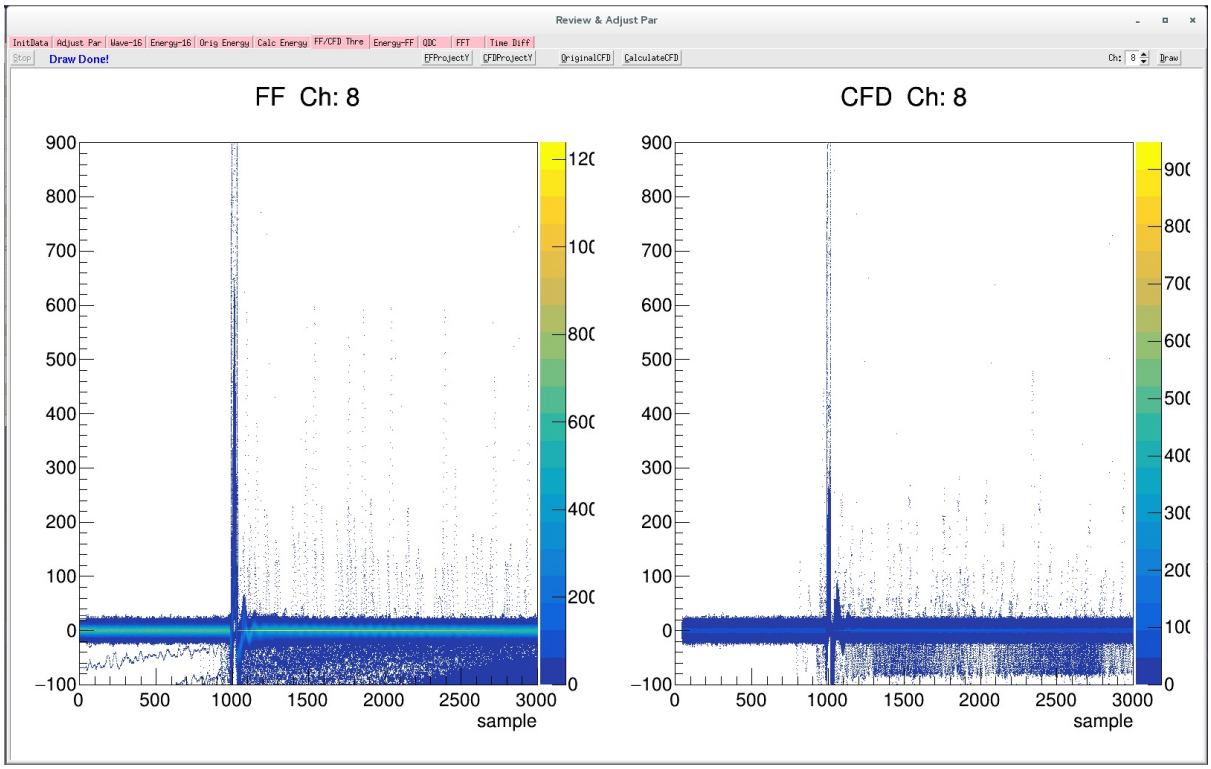


Figure: FFCFDThre

该界面用于 fast filter波形、cfd filter波形的累加。用户选择查看通道，然后按 Draw 按钮则开始进入计算，左上角可时时监视进度，也可按 Stop 按钮提前终止计算。计算结束得到如上图所示。

上方按钮 FFProjectY、CFDProjectY、OriginalCFD、CalculateCFD 分别可弹出子画板。

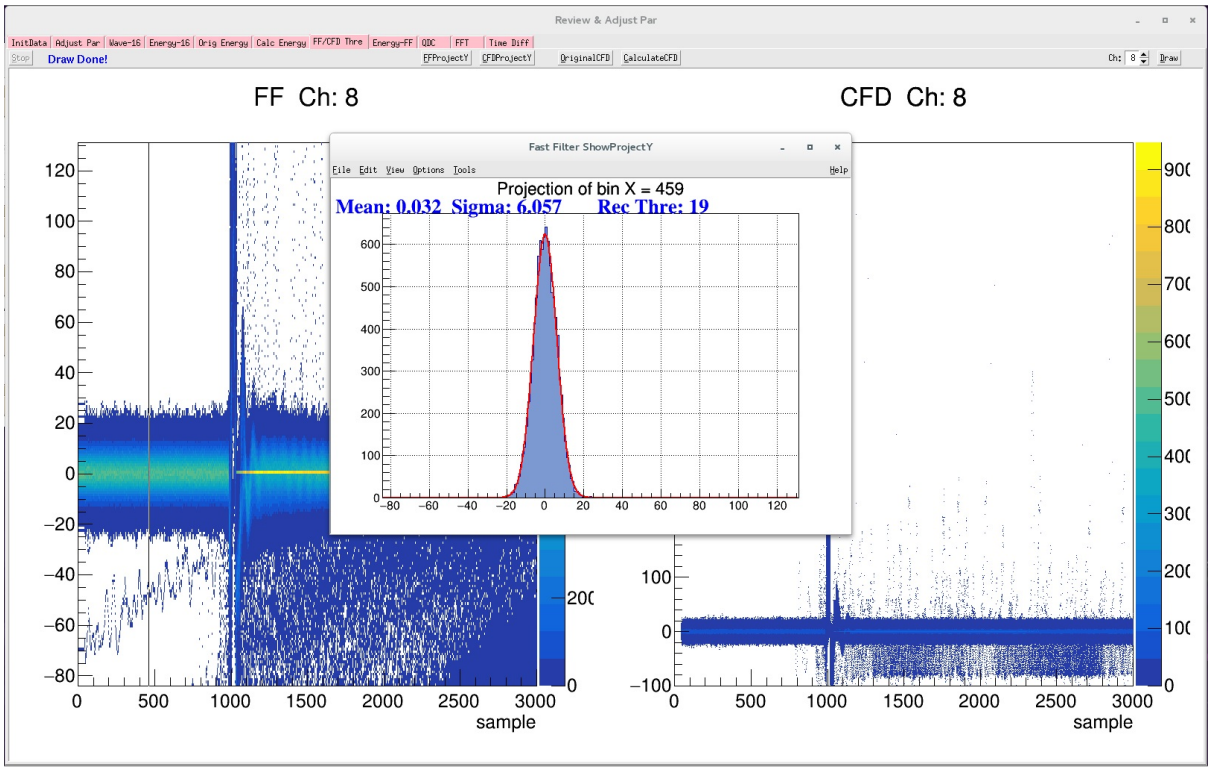


Figure: FFCFDThreFFProjectY

点击按钮 FFProjectY，则开启查看fast filter投影图，再次点击则关闭该功能。开启功能时，将鼠标放在二维图上，左右移动鼠标， Fast Filter ShowProjectY 子画板则显示鼠标指向的该位置的投影分布。触发前的该分布，也表征噪声的水平。

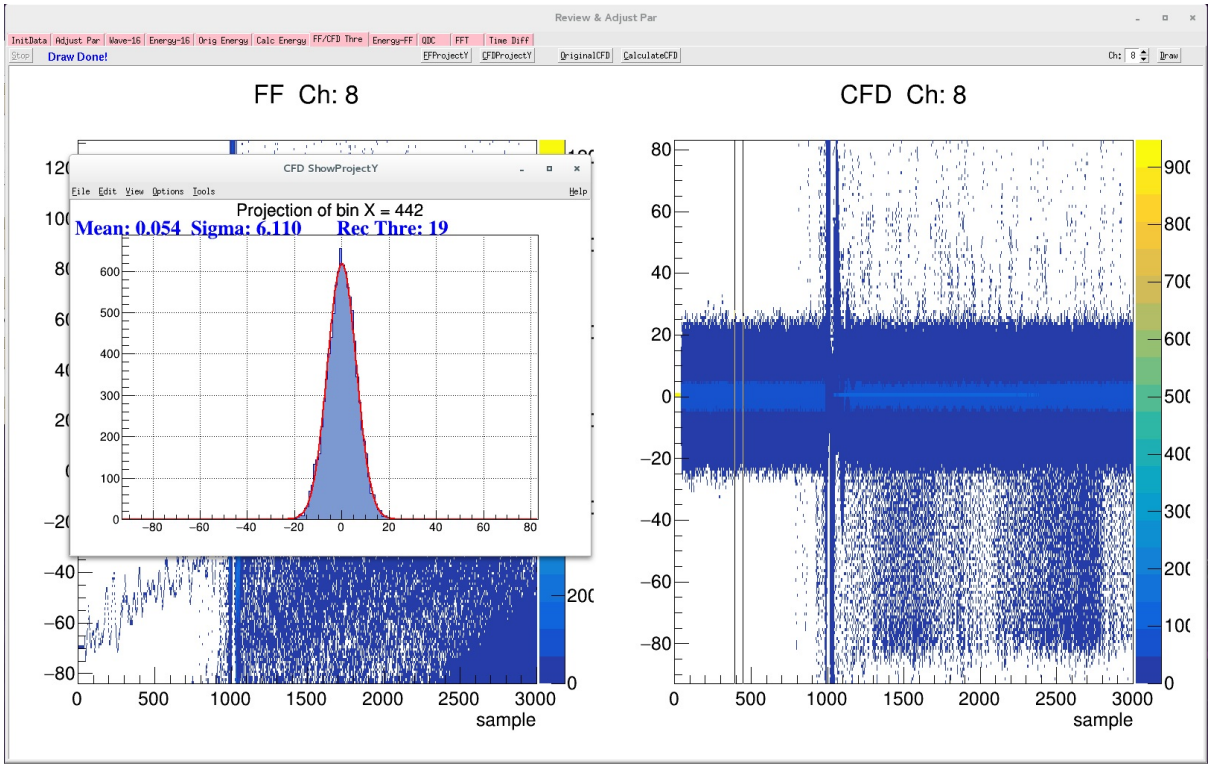


Figure: FFCFDThreCFDProjectY

同理，按钮 CFDProjectY 功能如上图所示。

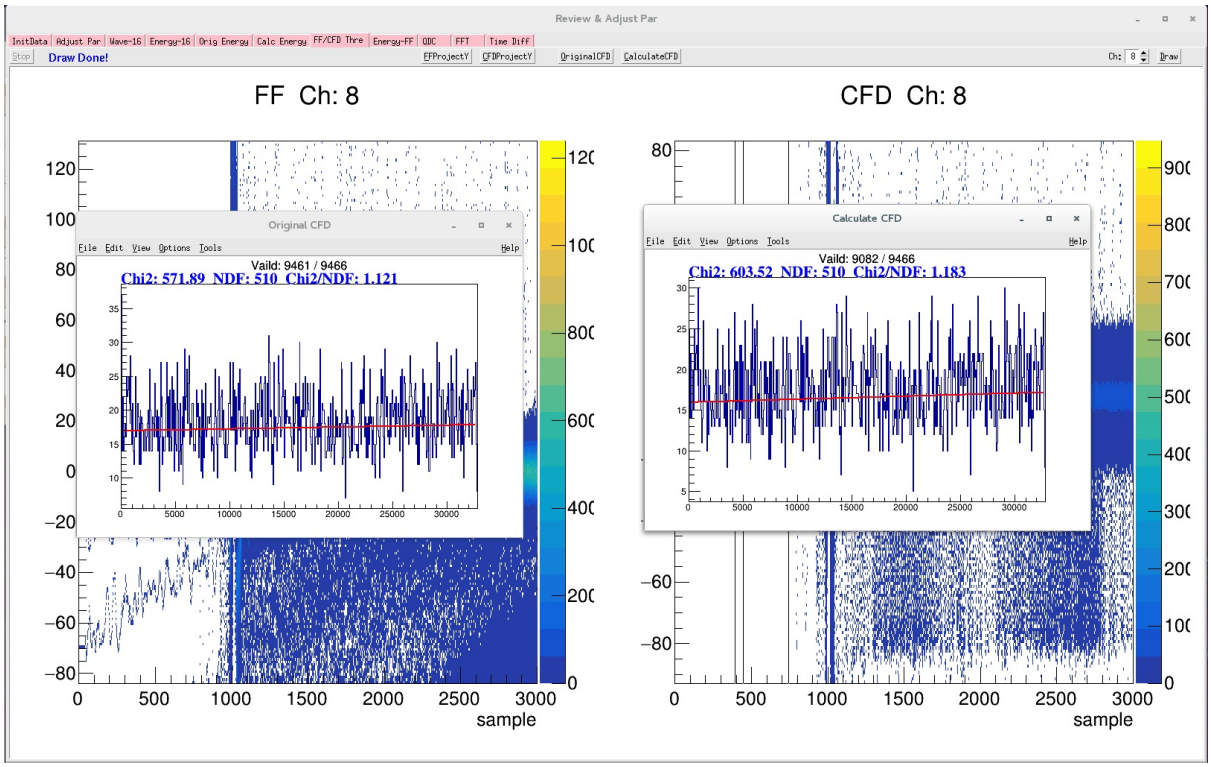


Figure: FFCFDThreCFD

点击按钮 OriginalCFD，则展示左图中原始数据中 CFD 数值的分布。点击按钮 CalculateCFD，则展示右图中通过离线波形计算的结果，计算所用参数为当前的参数。对于一个合适的 CFD 参数设置，该CFD分布该是平均分布的。

 updated: 2018-11-05 17:07:08

Energy-FF

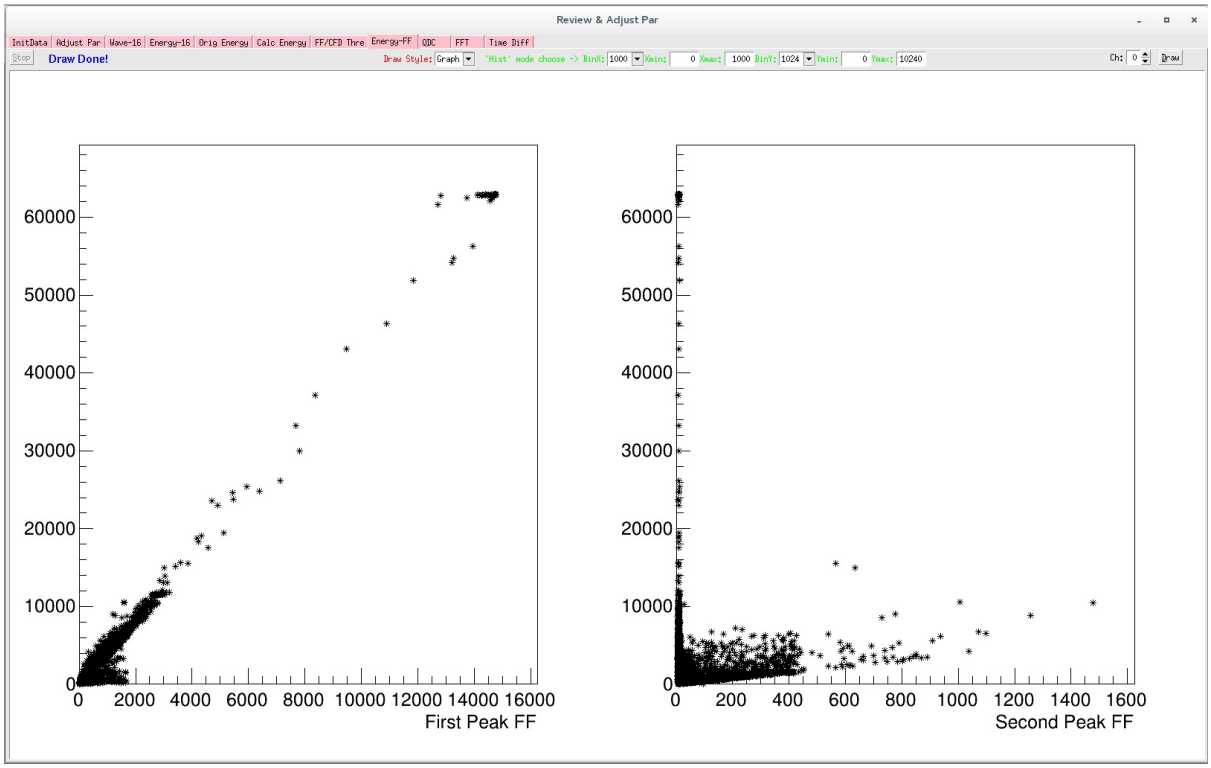


Figure: EnergyFFGraph

该界面是能量与 fast filter 峰高的二维关联图。用于确定合适的阈值。左图是能量与 fast filter的二维关联，它们应该有个较好的线性关系，右图为能量与 fast filer 中抛除梯形部分剩余中最大值的二维关联，抛除梯形部分剩余分布的最大值表征噪声水平，能量跟该值应该是没有关联的。

首先 Draw Style 选择 Graph，即二维散点图模式。选择查看通道，然后按 Draw 按钮则开始进入计算，左上角可时时监视进度，也可按 Stop 按钮提前终止计算。计算结束得到如上图所示。

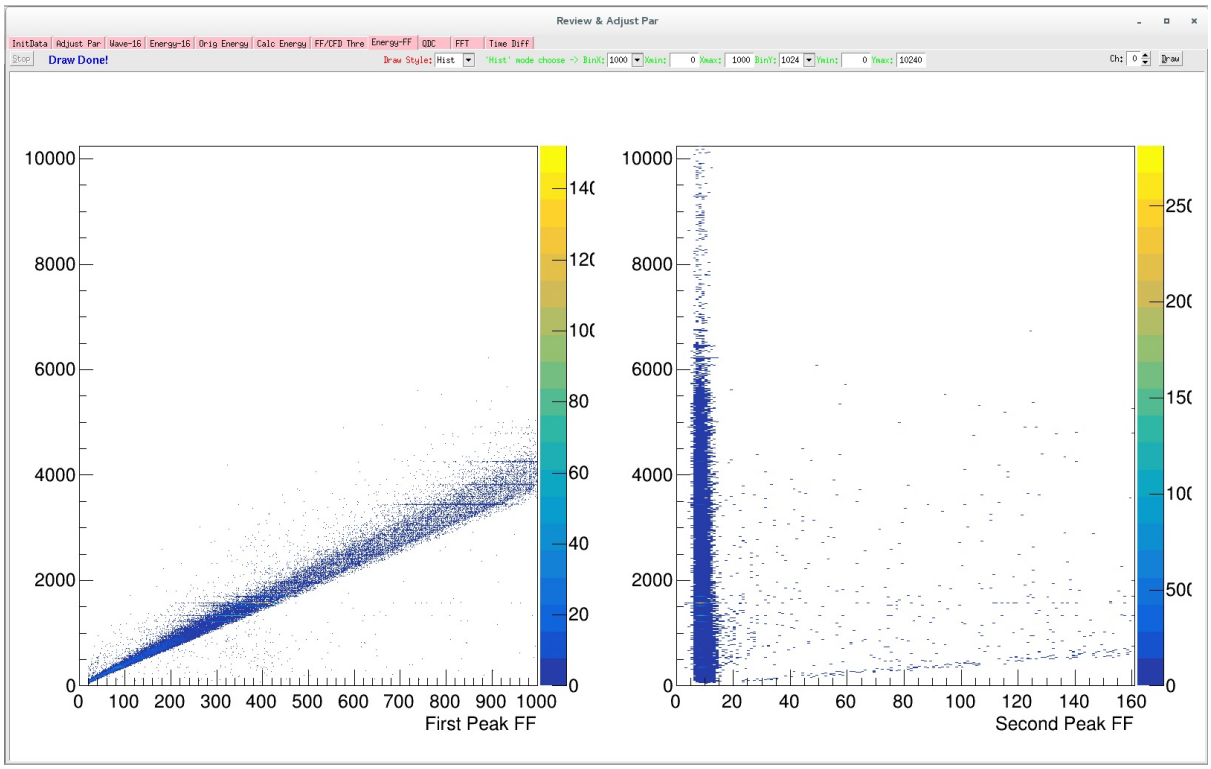


Figure: EnergyFFHist

二维散点图并不能很直观显示展示数据点的密度分布，因此 Draw Style 选择 Hist 模式，选择 X、Y轴的分 bin 数即范围，然后同样按 Draw 按钮开始计算。结果如上图所示，右图反映了噪声的水平。

 updated: 2018-11-05 17:07:28

QDC

to do not completed

QDC TODO 功能未完成

 updated: 2018-11-03 15:08:17

FFT

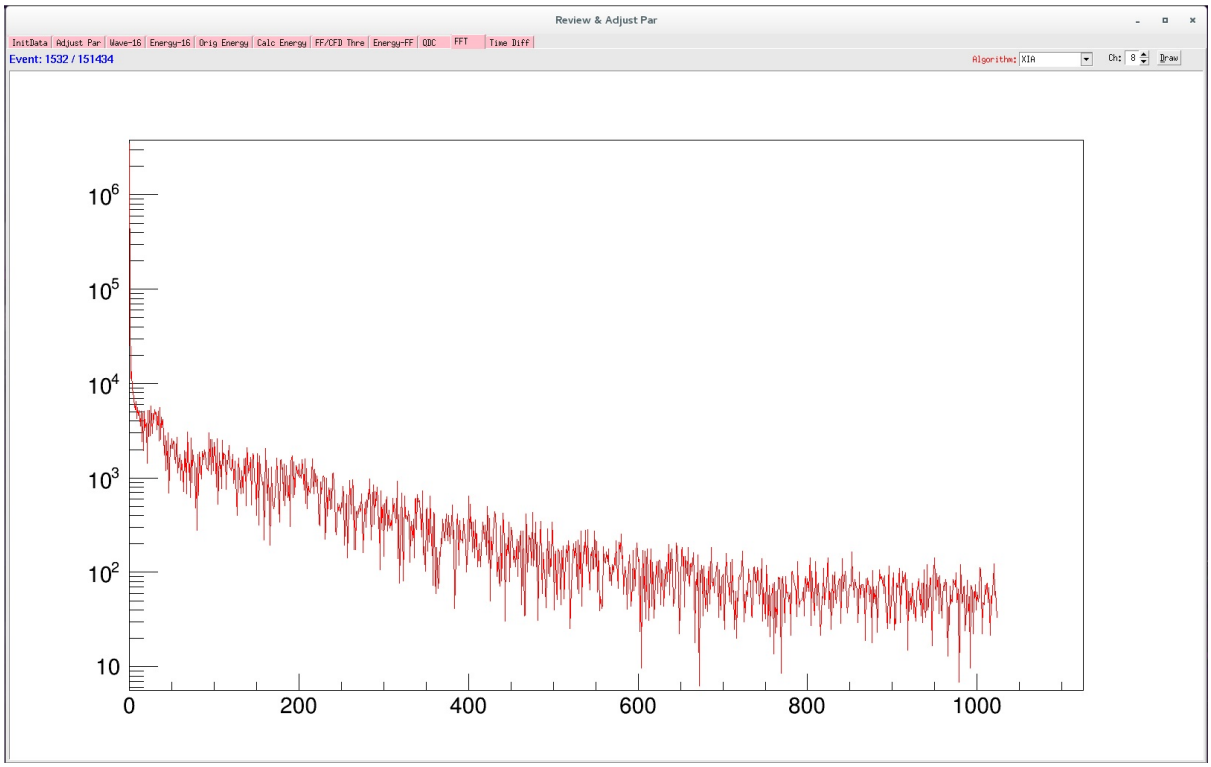


Figure: FFT

该界面用于快速查看波形的傅立叶变换。用户可以选择不同的算法，例如 XIA、fftw3、CAEN(HANNING)、CAEN(HAMMING)、CAEN(BLACKMAN)、CAEN(RECT)。选择查看通道。然后按 Draw 按钮即可，每点击一次该按钮，则显示下一个结果。

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

 updated: 2018-11-05 17:09:27

Time Diff

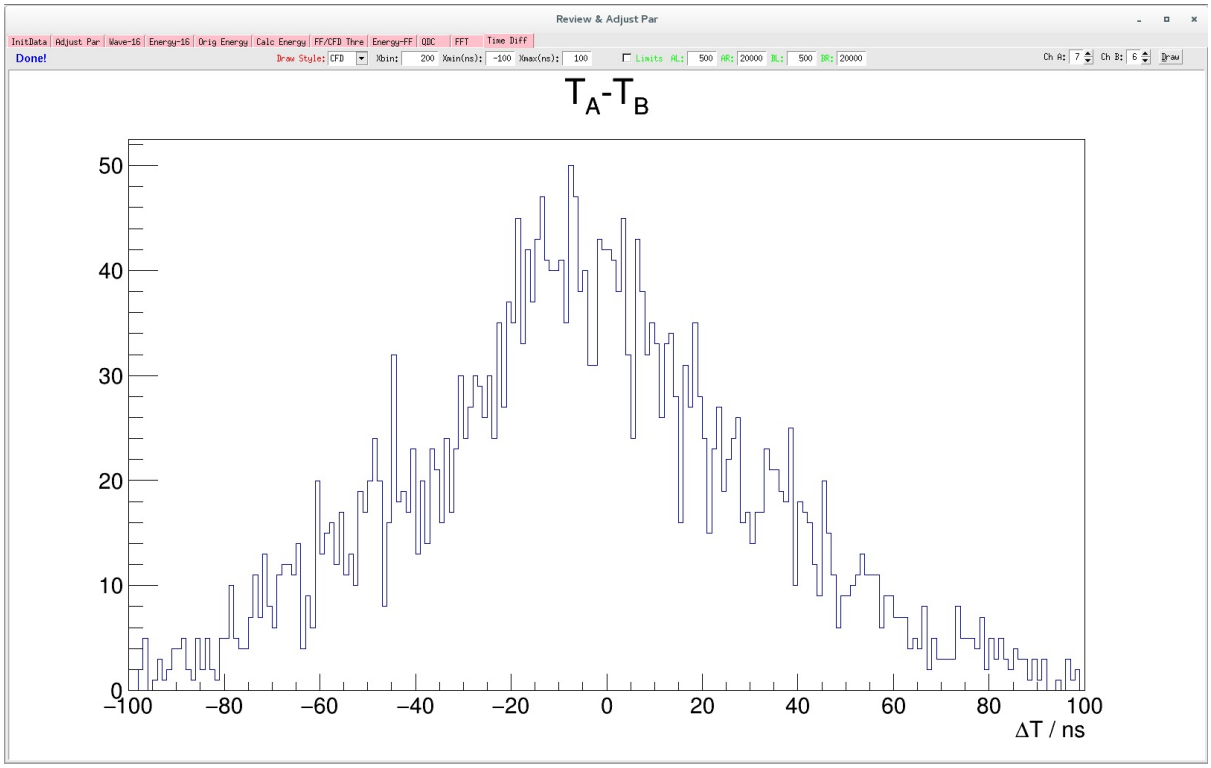


Figure: Time Diff

该界面用于快速查看两路信号的时间分辨。用户可以选择查看 CFD 算法过零点的时间差或者 fast filter 过阈值的时间差。Xbin 表示横坐标分 bin 数，Xmin 表示横坐标的最小值，Xmax 表示横坐标的最大值。通过 Ch A、Ch B 来选择想要查看的两个通道。然后按 Draw 按钮即可。

选项 Limits 选择则开启能量范围约束。选择该选项后，之后的四个参数 AL、AR、BL、BR才生效，其分别表示 Ch A/B 能量道址的左右范围，只有能量落在这个区间的事件才填充到直方图中。用户可通过 Orig Energy 页面来选择合适的能量道址区间。

 updated: 2018-11-05 17:10:00

Simulation(暂未实现)

通过模型产生不同类型探测、不同信噪比的波形，辅助使用者学习参数优化调节的。

 updated: 2018-11-05 17:00:34

Online Stattics

修改 **OnlineStattics** 中的文件 **PixieOnline.config**，其中第一行为原始二进制文件存放路径，第二行为文件名。通过该两行参数来监视每个文件时时大小及硬盘占用量。

通过执行以下命令，开启在线监视主界面：

```
./online
```

检查二进制文件路径、文件名是否有问题，如果没问题则点击按钮 **Complete**，之后点击 **RunStart**则开启在线监视，在线监视每 3 秒刷新一次。可时时监视每路的触发率、每路的实际事件输出率。

监视界面如下：

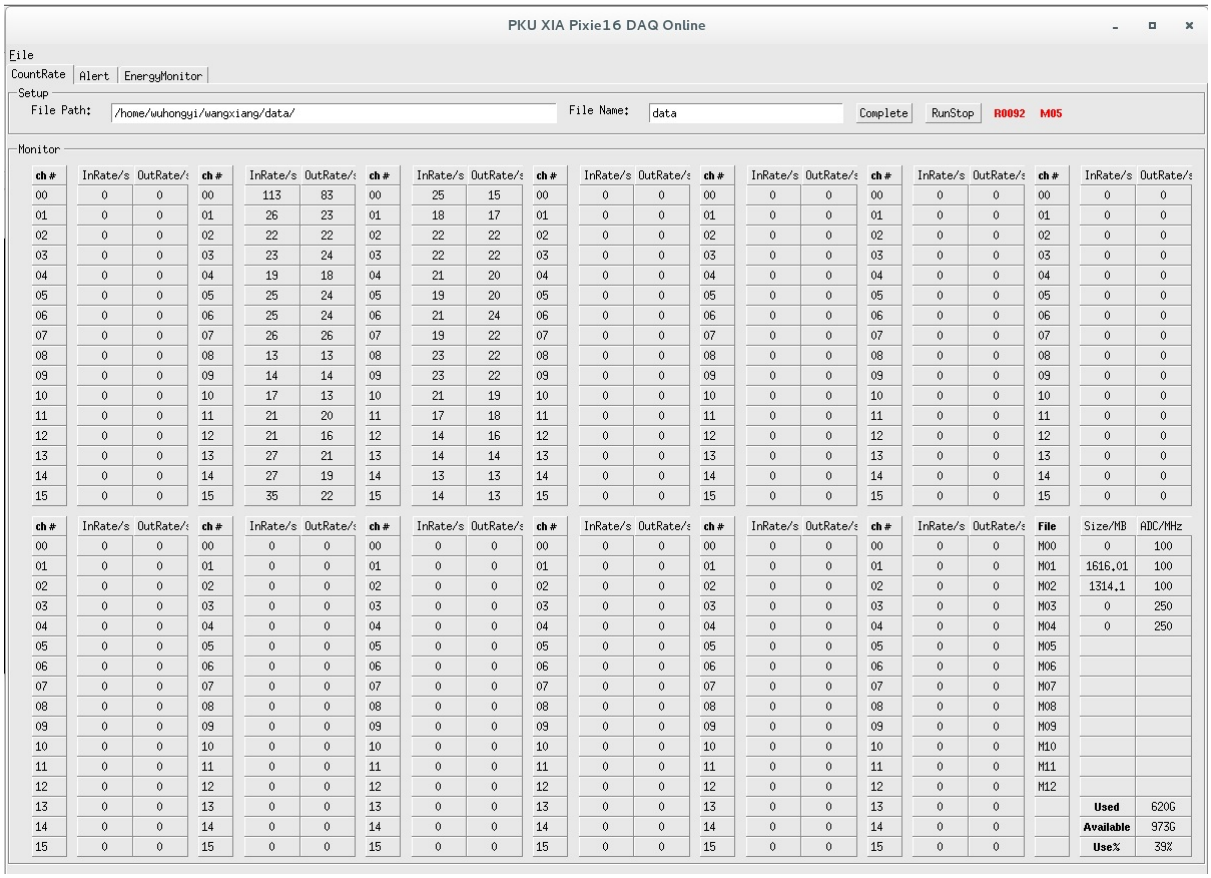


Figure: OnlineMainFrame

 updated: 2018-11-05 17:10:51

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。。。

 updated: 2018-11-03 15:08:17

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.

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.

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)

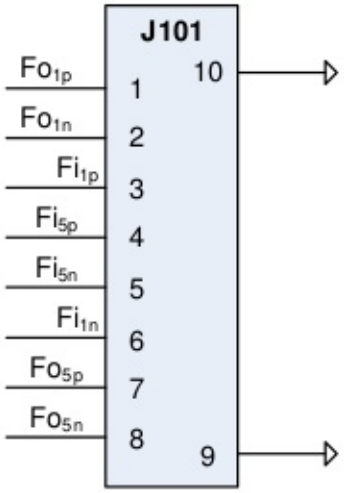


Figure: LVDS I/O Port

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.

Digital I/O Connectors(Rev. F only)

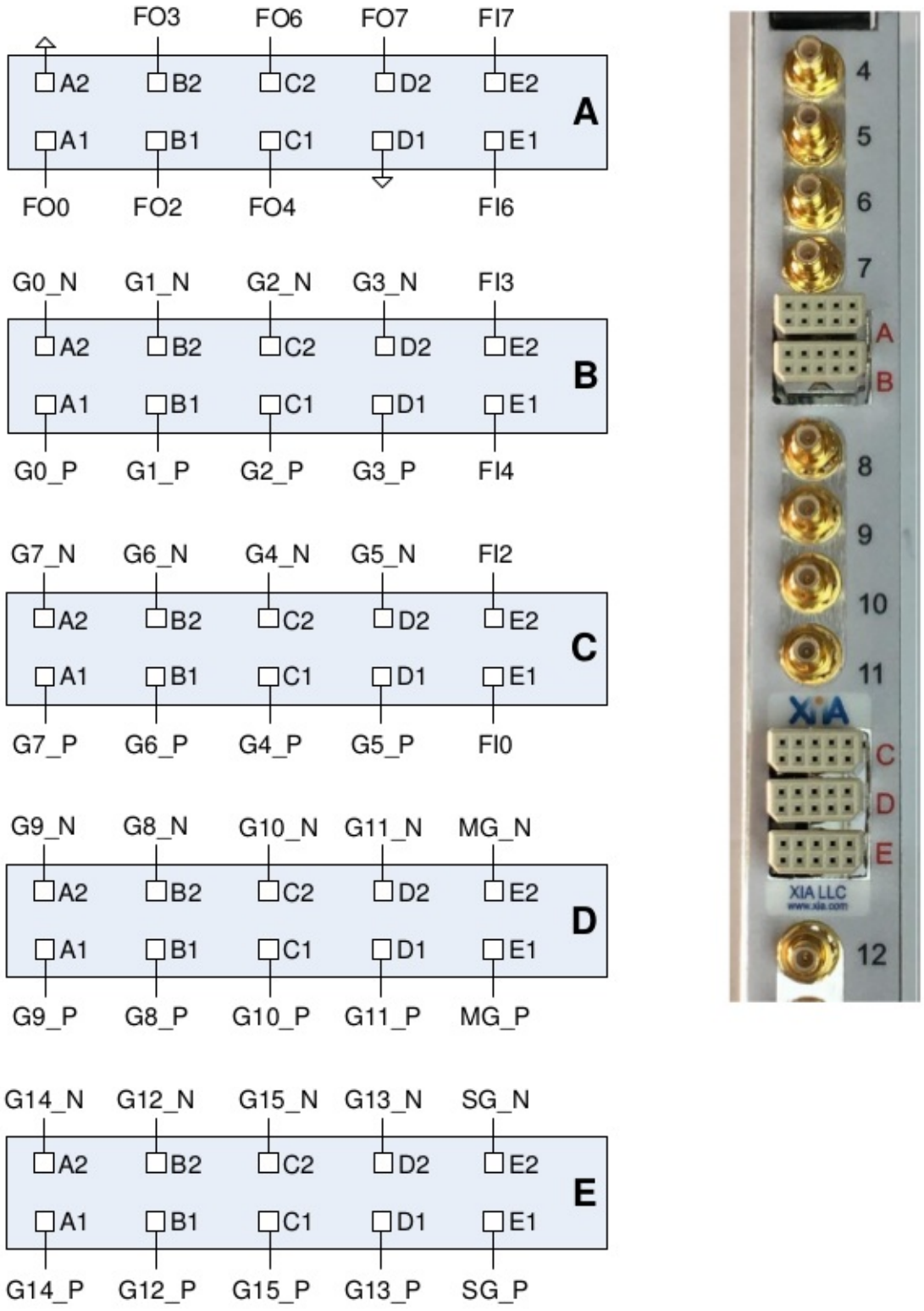


Figure: Digital I/O Connectors A, B, C, D and E

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.

Figure: 2mm pitch har-link connector from HARTING

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)

Figure: Rev. F Module's Digital I/O Connectors

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.

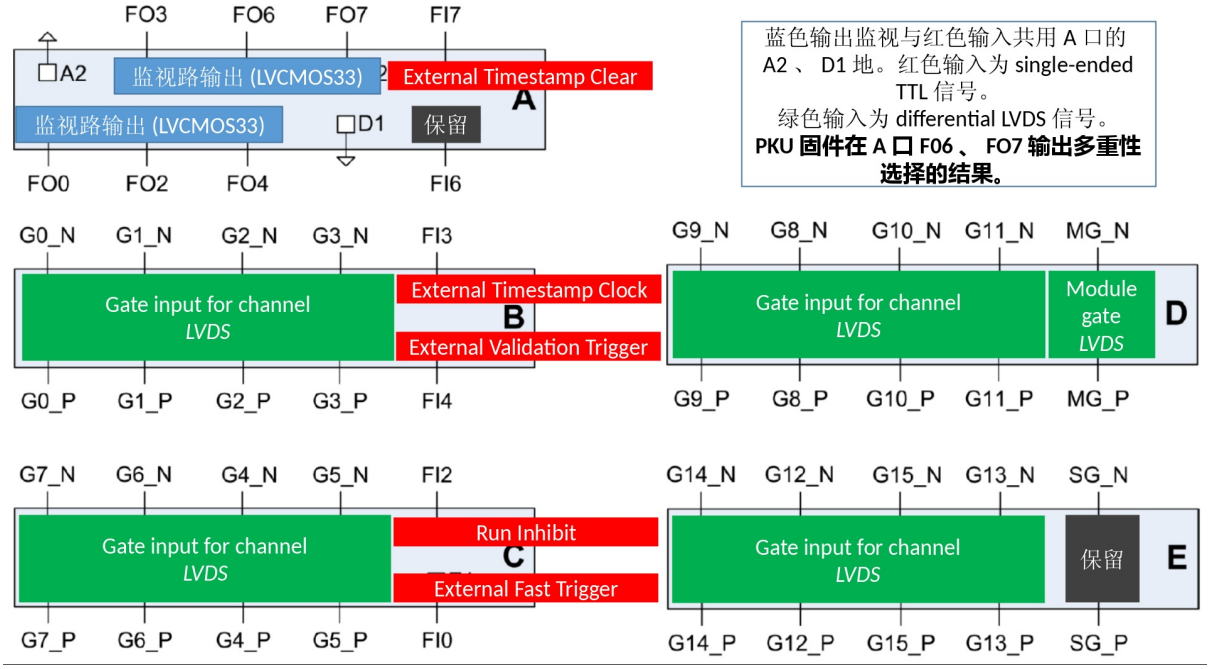


Figure: frontpanel

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)**

Figure: Front Panel LEDs for the Pixie-16 Modules

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.

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)

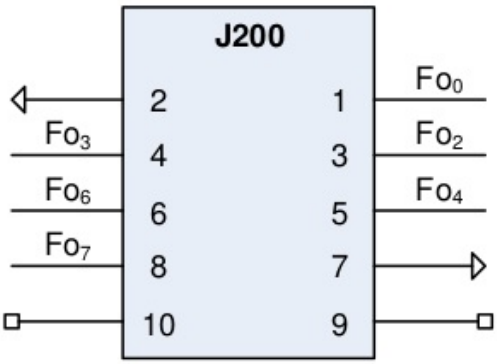


Figure: Rev. D Module's 3.3V I/O Connector

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.

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)

Figure: Rev. D Module's GATE Inputs

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.

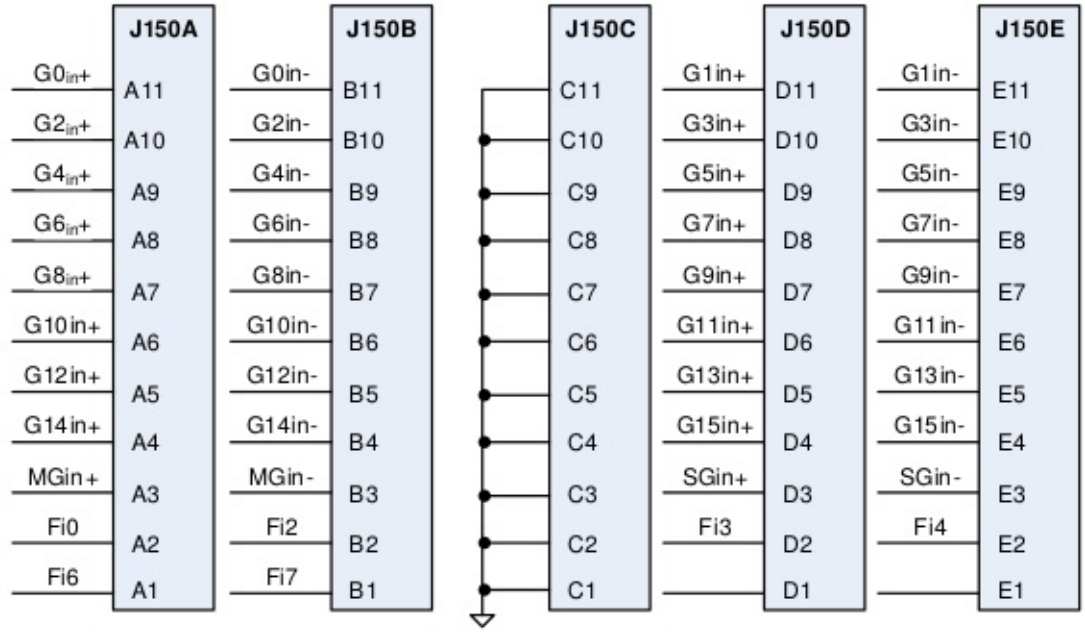


Figure: Rev. D Module's GATE INPUTS Connector

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*.

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)

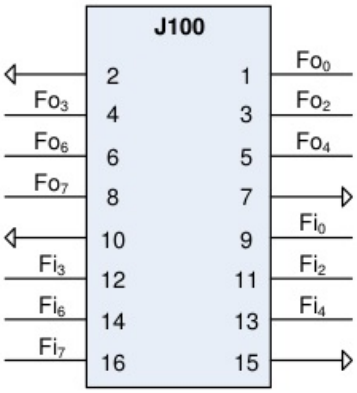


Figure: Rev. B and C Module's 3.3V I/O Connector

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.

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

Figure: TTL Digital Input Signals

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)

Figure: Connector J101 LVDS I/O Port Signals

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

Figure: TTL Digital Output Signals

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.

用于逻辑转换的可编程逻辑插件


Lupo
IN1
IN2
IN3
IN4
LVDS
OUT1
OUT2
OUT3
OUT4
FPGA ISP
CAEN
Mod. V2495
DTACK
GENERAL
PURPOSE
LOGIC UNIT

Figure: Lupo x495

 updated: 2018-11-03 15:08:17

Logic

- Module Fast Trigger (for trigger)
- Module Validation Trigger (for control logic)
- Channel Validation Trigger(for trigger/control logic)
- Veto
- System FPGA （coincidence/multiplicity）

对于某路信号的每个事件是否有效被记录取决于：

- Fast trigger select (一级trigger)
- Control logic (二级trigger)

Fast trigger select：

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

Control logic：

- Module validation trigger
- Channel validation trigger
- Veto
- Pileup
- ...

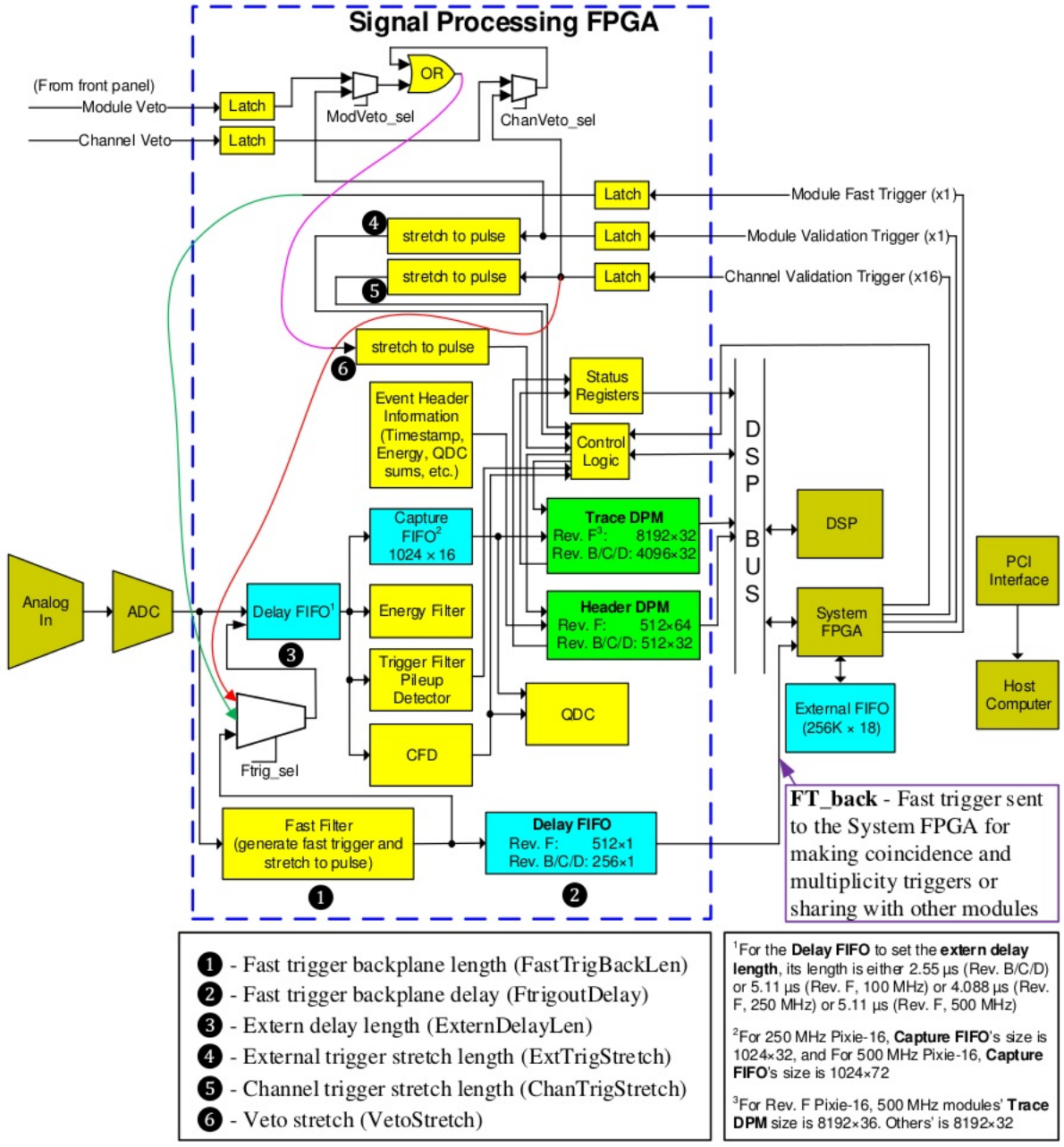


Figure: illustrates the signal processing in the Pixie-16 modules

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:

- 1) energy filter which samples energy running sums at the PeakSample time;
- 2) trigger filter which detects pulse and performs pileup inspection;
- 3) 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;
- 4) 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

Figure: Range for Trigger Stretch Lengths and FIFO Delays in Pixie-16 Modules

Module Fast Trigger (for trigger)

Module fast trigger 有以下四种来源可供选择：

- Ext_FastTrig_In(来源于本插件)
  - Ext_FastTrig_Sel(前面板 TTL 输入)
  - Int_FastTrig_Sgl(内部某路 FT)
  - FTIN_Or(内部 FT 的 OR)
  - LVDS_ValidTrig_FP(前面板网口输入)
  - ChanTrig_Sel(内部某路的 valid trigger)(与 module validation trigger 共用一个设置)
- FT_LocalCrate_BP(本机箱中指定插件送出的 trigger)
- FT_In_BP(多机箱中指定机箱上指定插件发送的 trigger)
- FT_WiredOr(本机箱中所有插件发送出 trigger 的 OR)

Module Validation Trigger (for control logic)

Module validation trigger 有以下来源可供选择：

- Ext_ValidTrig_In(来源于本插件)
  - Ext_ValidTrig_Sel(前面板 TTL 输入)
  - Int_ValidTrig_Sgl(内部某路 FT)
  - FTIN_Or(内部 FT 的 OR)
  - LVDS_ValidTrig_FP(前面板网口输入)
  - ChanTrig_Sel(内部某路的 valid trigger))(与 module fast trigger 共用一个设置)
- ET_LocalCrate_BP(本机箱中指定插件送出的 trigger)
- ET_In_BP(多机箱中指定机箱上指定插件发送的 trigger)
- ET_WiredOr(本机箱中所有插件发送出 trigger 的 OR)
- 前面板 module GATE 输入 LVDS 信号

Channel Validation Trigger(for trigger/control logic)

Channel validation trigger来源有以下选择：

- 每路独立设置，来源于多重性选择
- 每路独立设置，来源于符合
- 每4路共用一个设置，来源于左、中、右插件某路的 FT
- 每4路共用一个设置，来源于自身 FT 与 Ext_FastTrig_In 的符合
- 每路独立设置，前面板 channel GATE 输入 LVDS 信号(与前面板 Veto 共用一个输入口)

Veto

来源于 ModuleVeto 与 ChannelVeto 的 OR

- ModuleVeto 来源有两个选择：
  - Module Validation Trigger
  - 前面板 Module Gate
- ChannelVote 来源有两个选择：
  - Channel Validtion Trigger
  - 前面板 Gate input for channel (与前面板 Channel validation trigger 共用一个输入口)

System FPGA （coincidence/multiplicity）

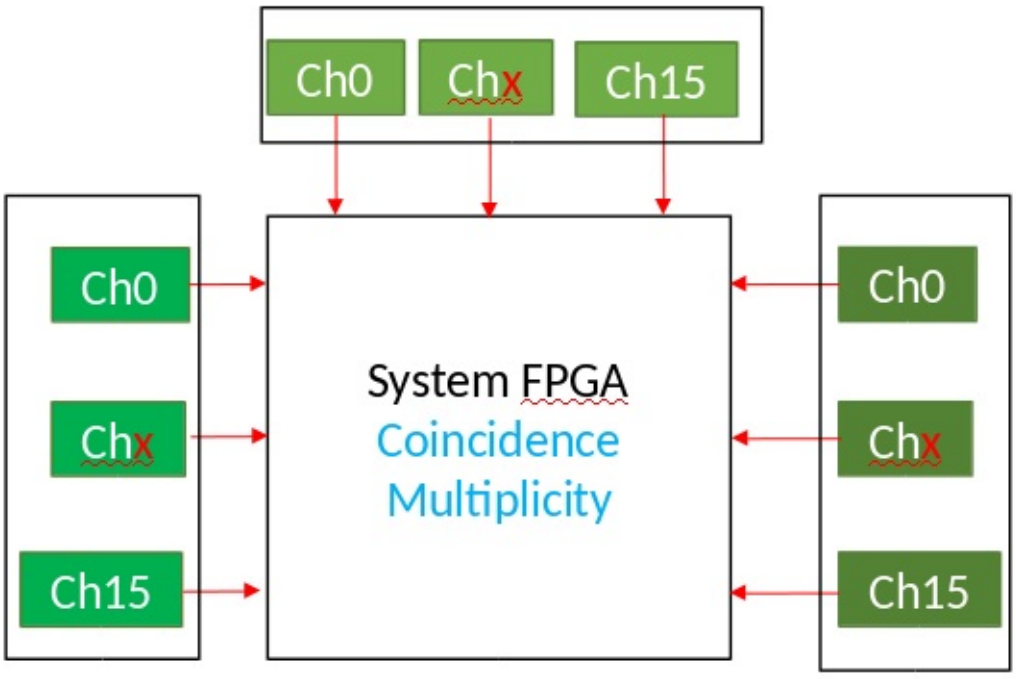


Figure: System FPGA

Multiplicity：对设置的该channel来说，左邻插件、自身插件、右邻插件共48路，可以选择参与多重性选择的路数

Coincidence：对设置的该channel来说，左邻插件、自身插件、右邻插件，每个插件均满足设置的符合条件，才能给出符合

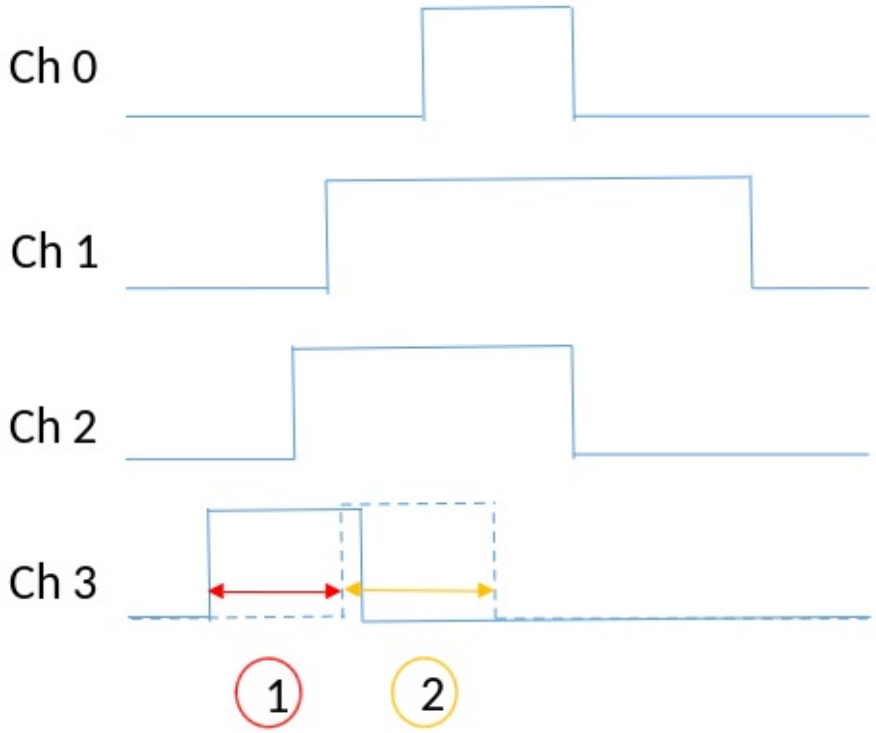


Figure: fast trigger stretch/delay

其它插件的fast filter通过机箱背板传到该插件需要时间，大约100ns左右。因此通过调节门宽、延迟来保证符合、多重性选择。

- Fast trigger stretch length 设置 fast filter 门宽，
- fast trigger delay length 设置 fast filter 延迟。

Control logic (module/channel validation trigger)

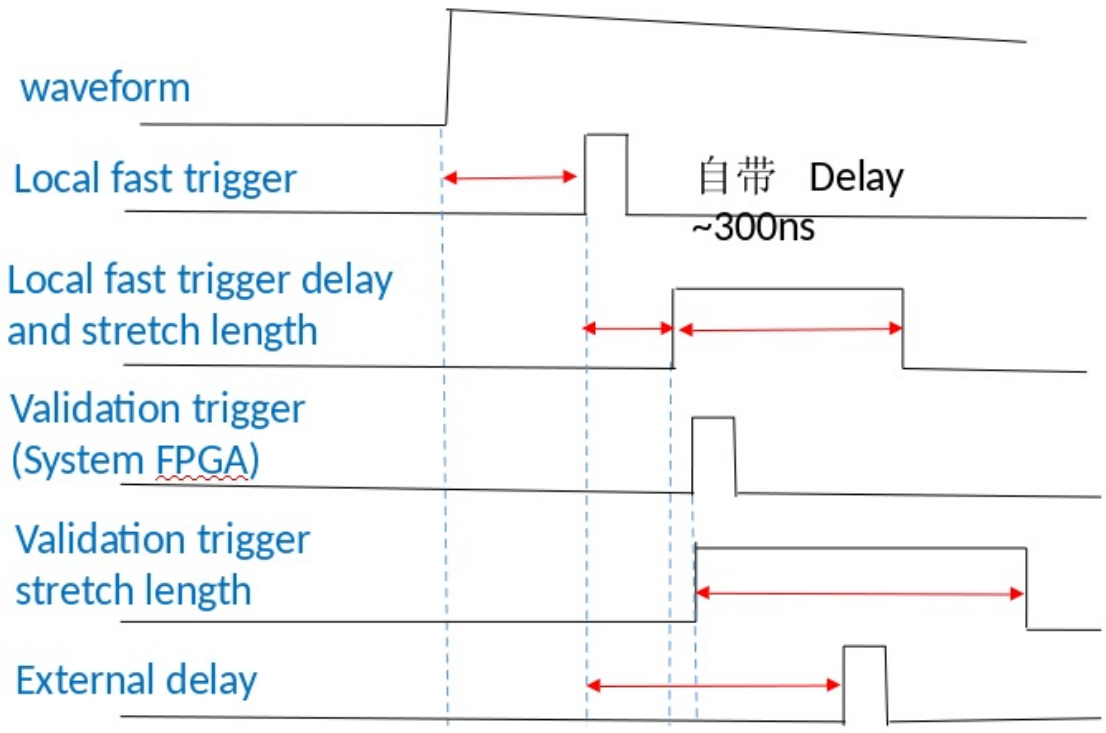


Figure: validation trigger

特别需要注意信号经过背板传输大约需要时间 100 ns。

 updated: 2018-11-03 15:08:17

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.

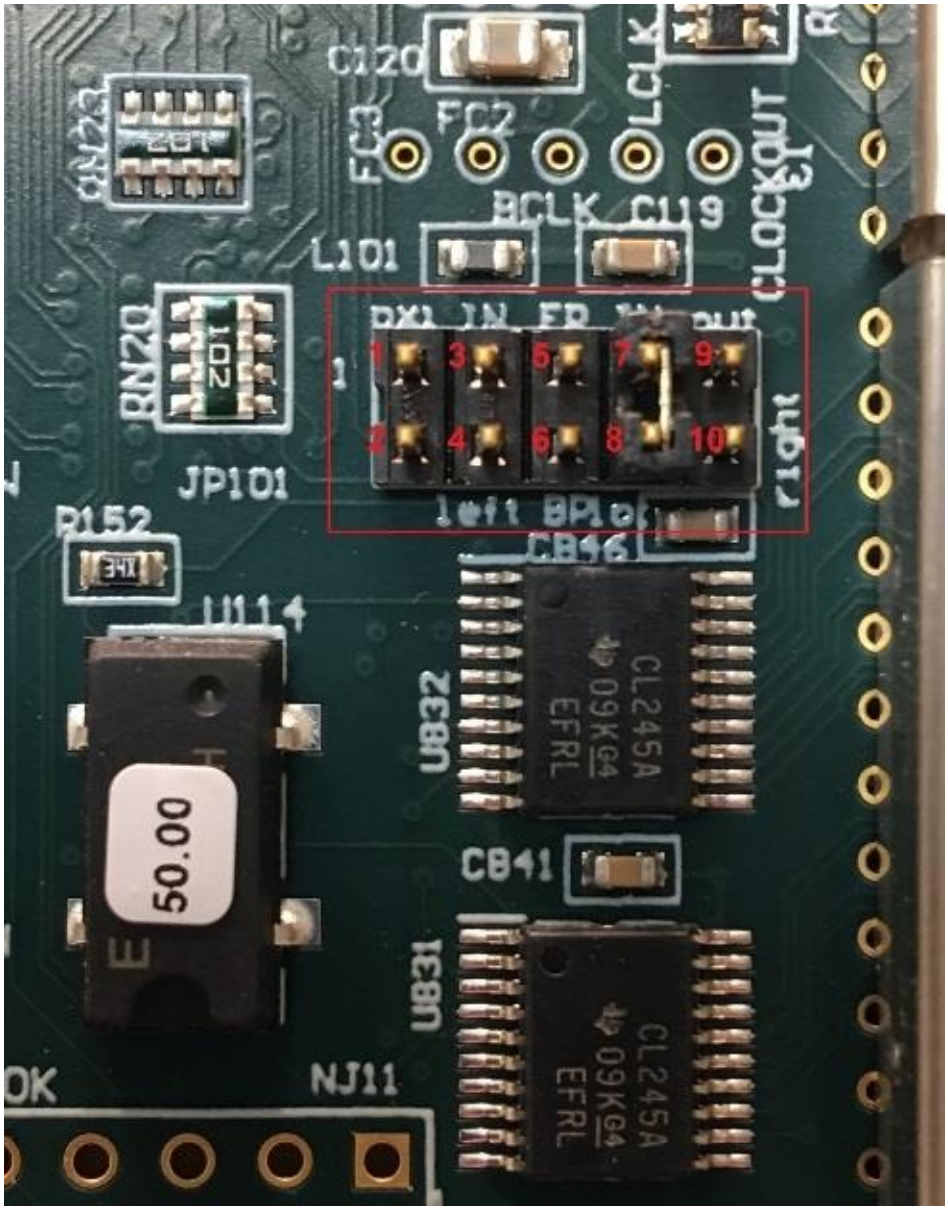


Figure: JP101

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.

> **[warning] Please Note**
>
> 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.

Individual Clock Mode

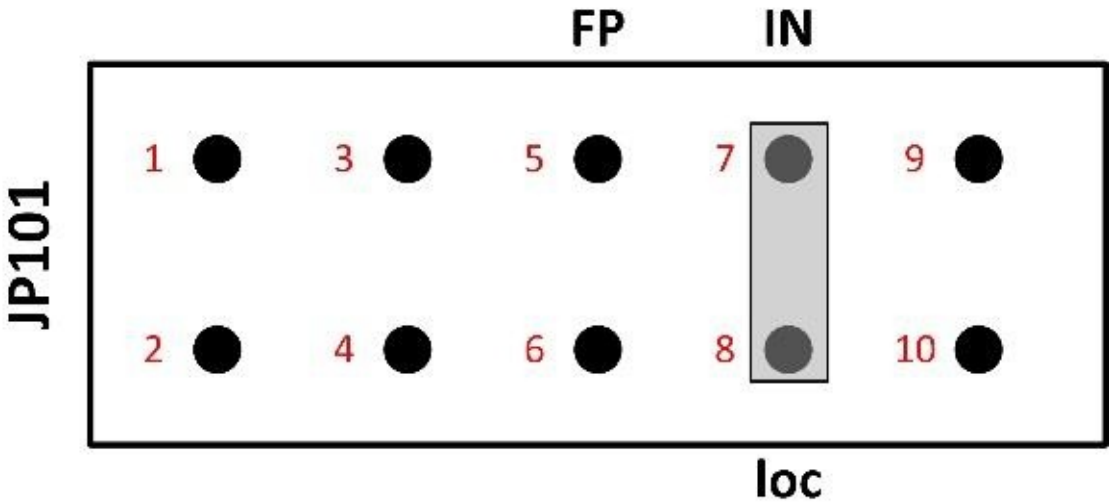


Figure: individual clock mode

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.

PXI Clock Mode

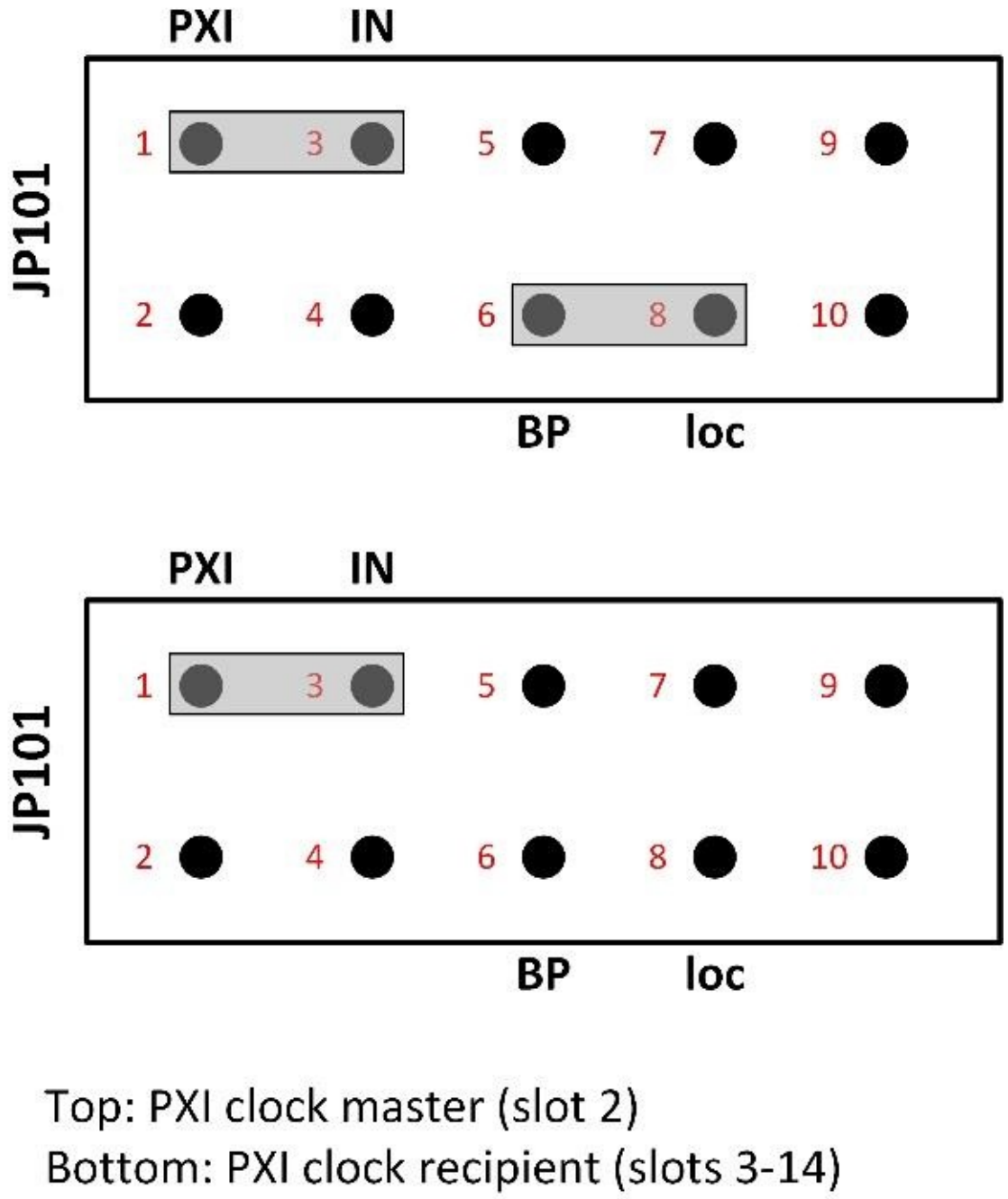


Figure: PXI clock mode

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.

Daisy-chained Clock Mode

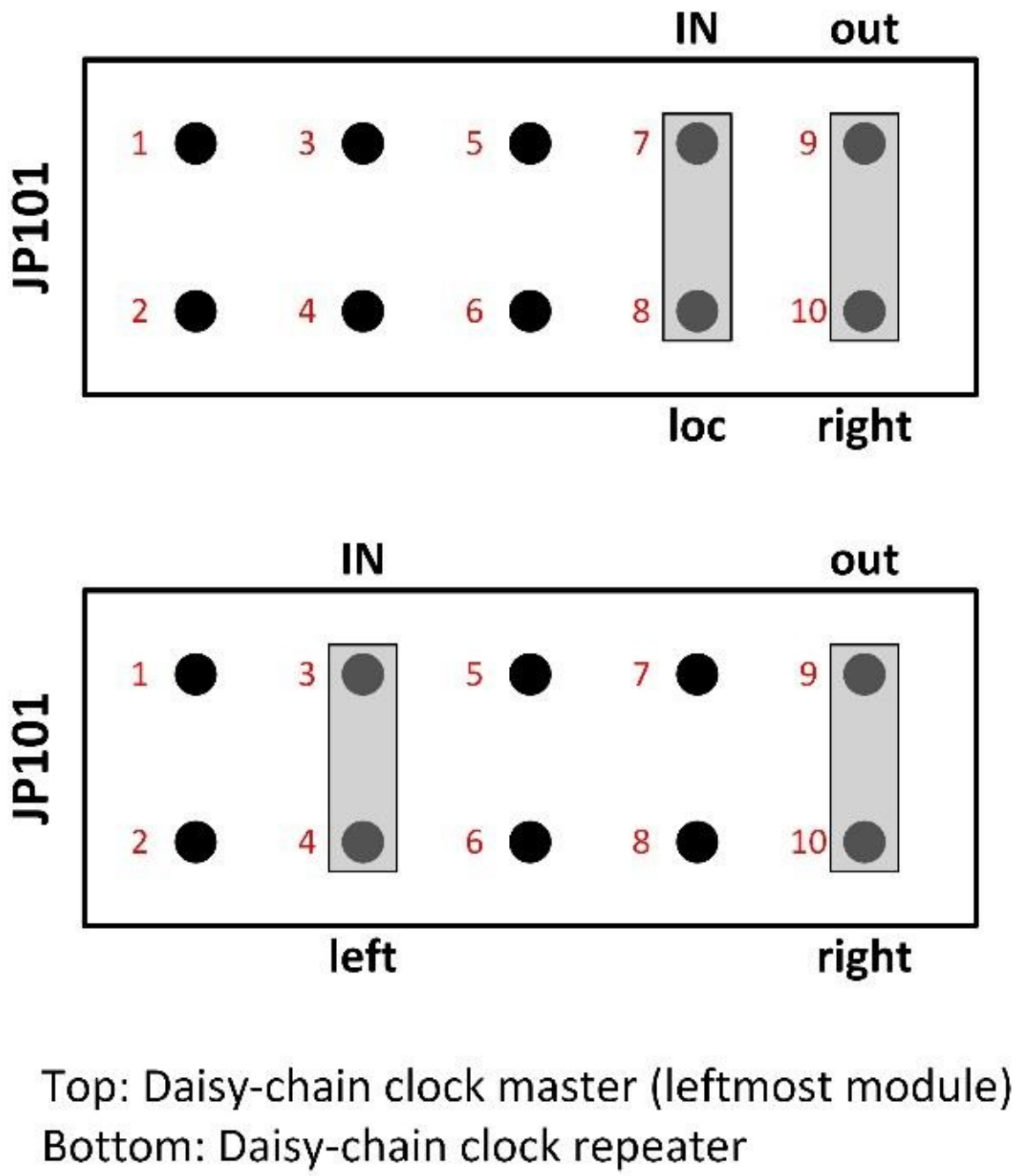
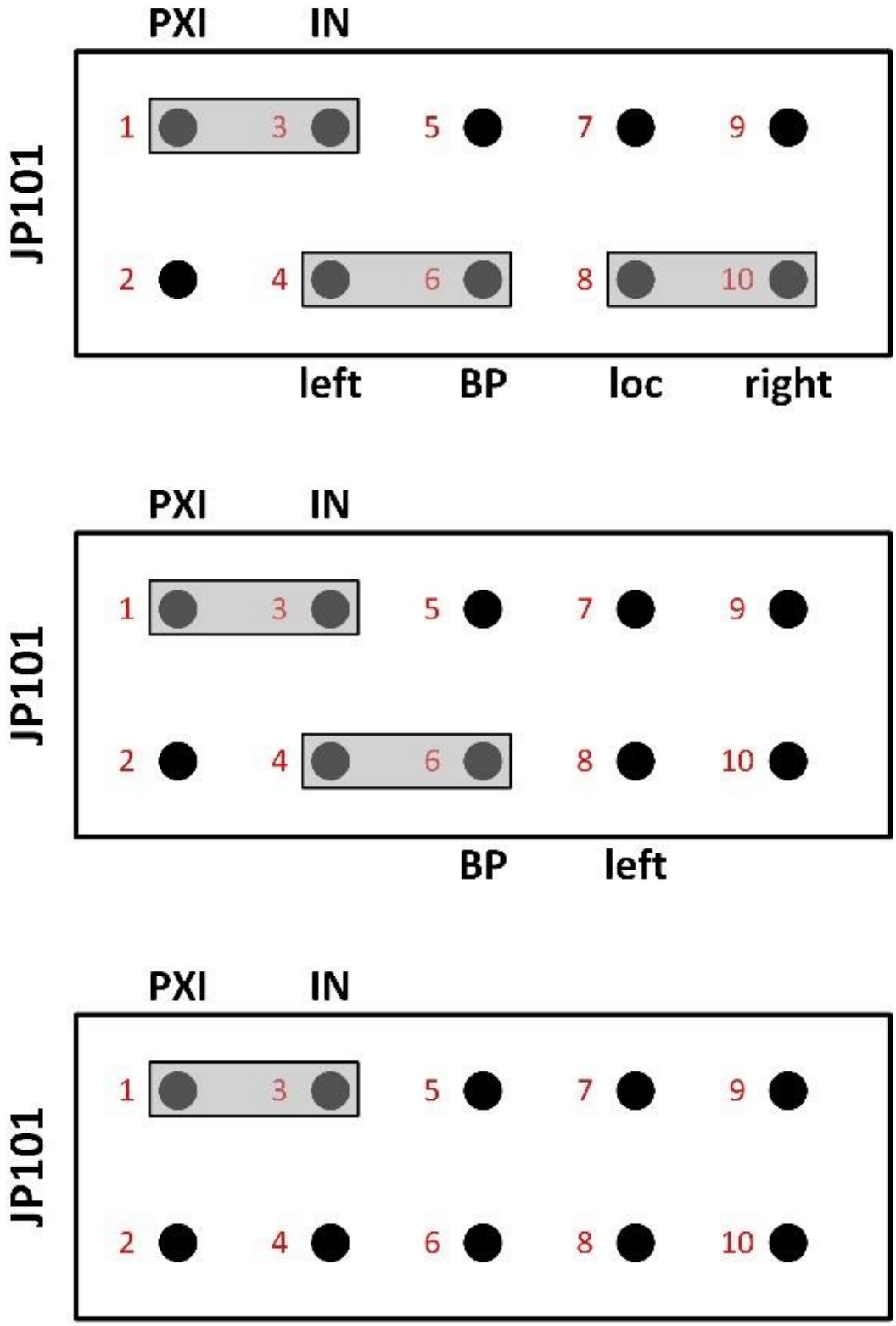


Figure: daisy-chained clock mode

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.

Multi-Crate Clock Mode



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

Figure: multi-crate clock mode

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.

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

Figure: Module Definitions in a 2-crate System

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

or 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.

Figure: Clock Jumper JP101 Settings in a 2-crate System

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: rear I/O trigger modules

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.

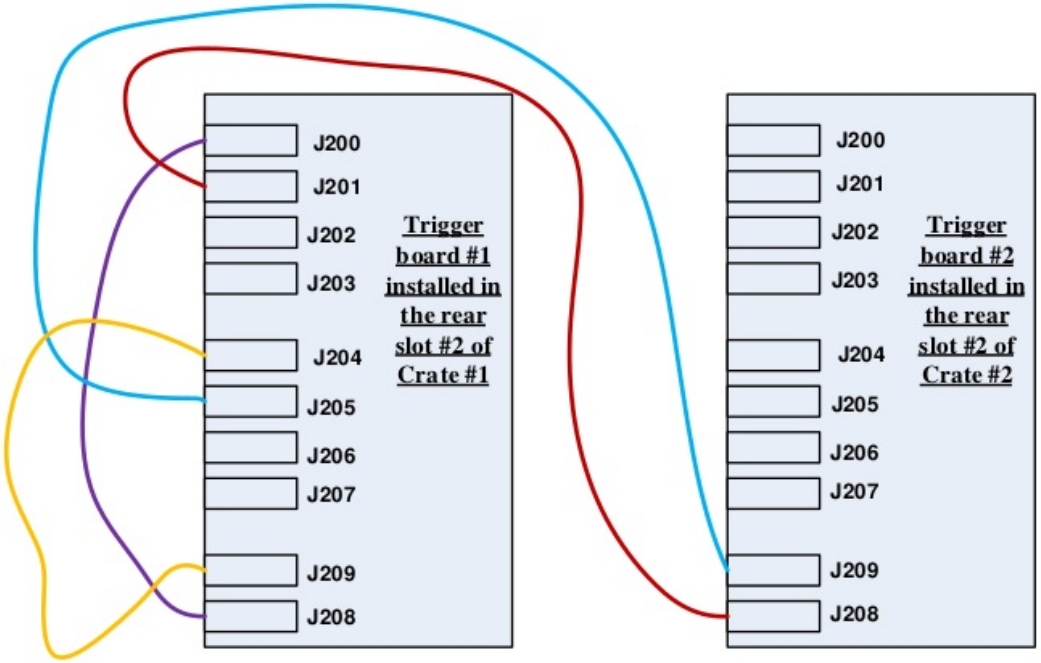


Figure: Cable connections between two Pixie-16 rear I/O trigger modules

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.

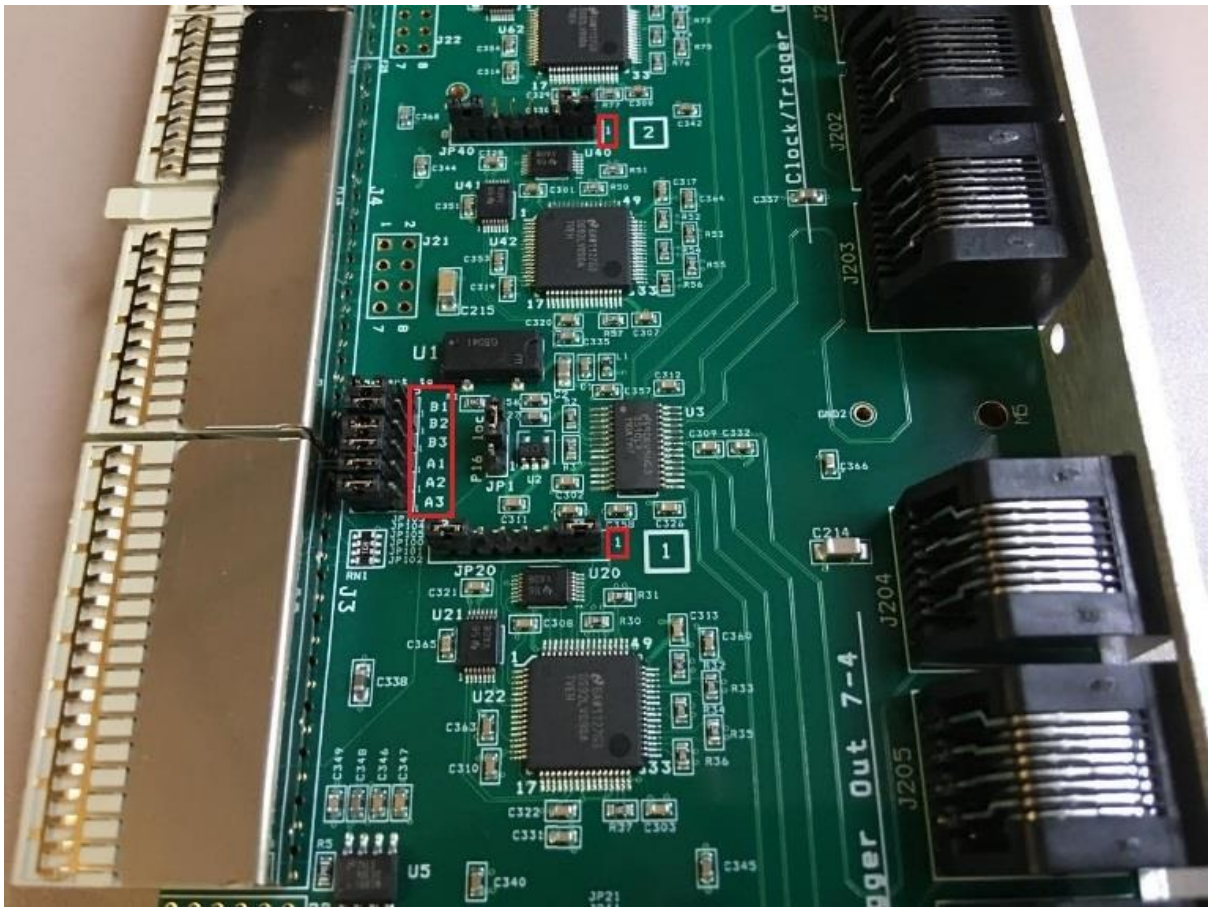


Figure: Pin numbering for the jumpers on the Pixie-16 rear I/O trigger module

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)

Figure: Rear I/O Trigger Module #1's Jumper Settings

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)

Figure: Rear I/O Trigger Module #2's Jumper Settings

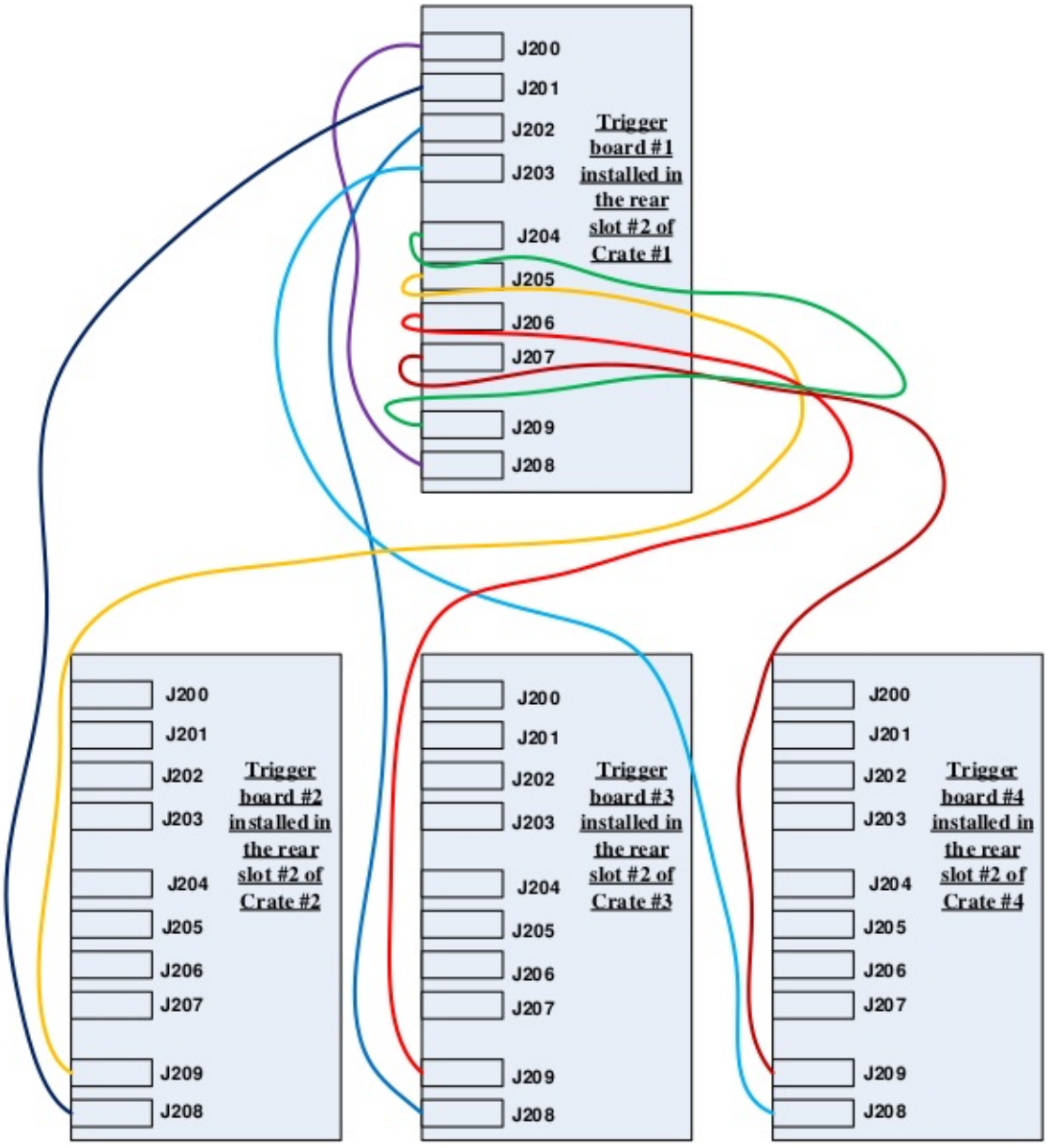


Figure: Cable connections among four Pixie-16 rear I/O trigger modules

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 1-17. 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.

 updated: 2018-11-03 15:08:17

应用案例

本章节介绍一些实验应用案例、测试结果等。

 updated: 2018-11-27 20:14:06

开发者指南

本章节介绍 Pixie16 开发中使用的Pixie-16 API 函数及获取程序的基本原理。

为用户提供基于我们获取程序开发的可能。

 updated: 2018-11-03 15:08:17

XIA API

It from **Programmer's Manual Digital Gamma Finder (DGF) PIXIE-16** *Version 1.40, October 2009*

```
// 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 mod
ules 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 pi
ece 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.
// 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 sa
me folder as Pixie-16 software installation folder. However, the user has the flexibility of putting it in othe
r folders by simply changing the definition of the string PCISysIniFile_Windows or PCISysIniFile_Linux in the h
eader part of the file pixie16sys.c, depending on which operating system is being used.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16InitSystem (
    unsigned short NumModules,     // total number of Pixie16 modules in the system
    unsigned short *PXISlotMap,    // an array containing the PXI slot number for each pixie16 module
    unsigned short OfflineMode ); // specify if the system is in offline mode
```

```
// 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 mo
dules 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.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ExitSystem (
    unsigned short ModNum );      // module number
```

```
PIXIE16APP_EXPORT int PIXIE16APP_API 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
```

```
// 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 download
s to the Pixie-16 modules the communication 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 co
nsists of 1280 32-bit words for each module. The host software has to make the names of those boot data files o
n the hard disk available to the boot function.
// If ModNum is set to be less than the total number of modules in the system, only the module specified by Mod
Num will be booted. But if ModNum is equal to the total number of modules in the system, e.g. there are 5 modul
es 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 cir
cumstances, all on-board chips should be booted, i.e. the boot pattern would be 0x7F. For Rev-B, C, D modules,
bit 1, i.e., “Boot trigger FPGA”, will be ignored even if that bit is set to 1.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16BootModule (
    char *ComFPGAConfigFile,      // name of communications FPGA configuration file
    char *SPFPGAConfigFile,       // name of signal processing FPGA configuration file
    char *TrigFPGAConfigFile,     // name of trigger FPGA configuration file
    char *DSPCodeFile,            // name of executable code file for digital signal processor (DSP)
    char *DSPParFile,             // name of DSP parameter file
    char *DSPVarFile,             // name of DSP variable names file
    unsigned short ModNum,        // pixie module number
    unsigned short BootPattern ); // boot pattern bit mask
```

```
// Acquire ADC traces in single or multiple modules
// Use this function to acquire ADC traces from Pixie-16 modules. Specify the module using ModNum. If ModNum is
 set to be less than the total number of modules in the system, only the module specified by ModNum will have i
ts ADC traces acquired. But if ModNum is equal to the total number of modules in the system, then all modules i
n 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 another function Pixie16ReadSglChanADCTrace to read the ADC trac
e data out to the host computer, channel by channel.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16AcquireADCTrace (
    unsigned short ModNum );      // module number
```

```
// 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 funct
ion Pixie16AcquireADCTrace should be called to fill the DSP internal memory first. Also, the host code should a
llocate appropriate amount of memory to store the trace data. The ADC trace data length for each channel is 819
2. Since the trace data are 16-bit unsigned integers (actually only the lower 14-bit contains real data due to
the on-board 14-bit ADC), two consecutive 16-bit words are packed into one 32-bit word in the DSP internal memo
ry. So for each channel, 4096 32-bit words are read out first from the DSP, and then each 32-bit word is unpack
ed 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.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ReadSglChanADCTrace (
    unsigned short *Trace_Buffer, // trace data
    unsigned int Trace_Length,    // trace length
    unsigned short ModNum,        // module number
    unsigned short ChanNum );     // channel number
```

```
// 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 modul
e. The DSP internal memory is split into two blocks with address range 0x40000 to 0x4FFFF for the first block a
nd address range 0x50000 to 0x5FFFF for the second block. Within the first block, address range 0x40000 to 0x49
FFF is reserved for program memory and shouldn’t be accessed directly by the host. Address range 0x4A000 to 0x4
A4FF is used by the DSP I/O parameters which are stored in the configuration files (.set files) in the host. Wi
thin this range, 0x4A000 to 0x4A33F can be both read and written, but 0x4A340 to 0x4A4FF can only be read but n
ot written. The remaining address range (0x4A500 to 4FFFF) in the first block and the entire second block (0x50
000 to 0x5FFFF) should only be read but not written by the host. Use Direction = 1 for read and Direction = 0 f
or write.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16IMbufferIO (
    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
```

```
// 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 mod
ule. 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-1
6 module. Each channel has a fixed histogram length of 32768 words(32-bit wide), and the placement of the histo
```

```
gram 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 tha
t it needs to be called channel by channel.
// In Rev-A modules, part of the external memory is also used to store the list mode data in ping-pong bufferin
g mode. This function can be used to read list mode data from the buffers.
PIXIE16APP_EXPORT int PIXIE16APP_API 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
```

```
// Start a list mode data acquisition run
// Use this function to start a list mode data acquisition run in Pixie-16 modules. List mode run is used to co
llect data on an event-by-event basis, gathering energies, timestamps, pulse shape analysis values, and wavefor
ms, for each event. Runs will continue until a preset number of events are reached or the user terminates the r
un by calling function Pixie16EndRun. Once the run is progress, if the run is set to terminate after a given nu
mber of events have been accumulated, another function, Pixie16CheckRunStatus, should be called to check if the
 run has finished. To start the data acquisition this function has to be called for every Pixie-16 module in th
e system. If all modules are to run synchronously, The last module addressed will release all others and the ac
quisition starts then. The first module to end the run will immediately stop the run in all other modules.
// Use mode=NEW_RUN (=1) to erase histograms and statistics information before launching the new run. Note that
 this will cause a start up delay of up to 1 millisecond. Use mode=RESUME_RUN (=0) to resume an earlier run. Th
is mode has a start up delay of only a few microseconds.
// For Rev-A modules, currently there are 4 list mode run types supported. They are 0x100 (general purpose run)
, 0x101 (without waveforms), 0x102 (without auxiliary data) and 0x103 (energy and timestamp only).
// For Rev-B, C, D modules, there are 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.
// Histograms and statistics data are updated incrementally from run to run provided RESUME_RUN mode is used.
// 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 ru
ns started together.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16StartListModeRun (
    unsigned short ModNum,        // module number
    unsigned short RunType,       // run type
    unsigned short mode );        // run mode
```

```
// 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 ea
ch channel.
// Call this function for each Pixie-16 module in the system. The last module addressed will allow the actual d
ata acquisition to begin. Histogram run can be self-terminating when the elapsed run time exceeds the preset ru
n time, or the user can prematurely terminate the run by calling Pixie16EndRun. On completion, final histogram
and statistics data will be available.
// 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.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16StartHistogramRun (
    unsigned short ModNum,        // module number
    unsigned short mode );        // run mode
```

```
// 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 ex
ternal memory (Rev-A modules) or external FIFO (Rev-B, C, D modules) that need to be read out to the host. In a
ddition, 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 th
```

```
ough it is normally self-terminating.
// ModNum is the module number which starts counting at 0.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16CheckRunStatus (
    unsigned short ModNum );      // Pixie module number
```

```
// 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 s
top the run in all other module in the system.
// ModNum is the module number which starts counting at 0.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16EndRun (
    unsigned short ModNum );      // Pixie module number
```

```
// Compute input count rate
// 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, Pixie16ReadStatisticsFro
mModule, 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 *Statist
ics array is filled with data from a Pixie-16 module after calling function Pixie16ReadStatisticsFromModule. Mo
dNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.
PIXIE16APP_EXPORT double PIXIE16APP_API Pixie16ComputeInputCountRate (
    unsigned int   *Statistics,
    unsigned short ModNum,
    unsigned short ChanNum );
```

```
// 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, Pixie16ReadStatisticsFr
omModule, 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 *Statist
ics array is filled with data from a Pixie-16 module after calling function Pixie16ReadStatisticsFromModule. Mo
dNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.
PIXIE16APP_EXPORT double PIXIE16APP_API Pixie16ComputeOutputCountRate (
    unsigned int   *Statistics,
    unsigned short ModNum,
    unsigned short ChanNum );
```

```
// 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 acquisi
tion. 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 *Statist
ics array is filled with data from a Pixie-16 module after calling function Pixie16ReadStatisticsFromModule. Mo
dNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.
PIXIE16APP_EXPORT double PIXIE16APP_API Pixie16ComputeLiveTime (
    unsigned int   *Statistics,
    unsigned short ModNum,
    unsigned short ChanNum );
```

```
// Compute number of events processed by a channel
// Use this function to calculate the number of events that have been processed by a Pixie-16 module during a d
ata 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 *Statist
ics array is filled with data from a Pixie-16 module after calling function Pixie16ReadStatisticsFromModule. Mo
dNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.
PIXIE16APP_EXPORT double PIXIE16APP_API Pixie16ComputeProcessedEvents (
    unsigned int   *Statistics,
    unsigned short ModNum );
```

```
// Compute real time that a channel accumulated in a run
// Use this function to calculate the real time that a Pixie-16 module has spent on data acquisition. This func
tion does not communicate with Pixie-16 modules. Before calling this function, another function, Pixie16ReadSta
tisticsFromModule, 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 *Statist
ics array is filled with data from a Pixie-16 module after calling function Pixie16ReadStatisticsFromModule. Mo
dNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.
PIXIE16APP_EXPORT double PIXIE16APP_API Pixie16ComputeRealTime (
    unsigned int   *Statistics,
    unsigned short ModNum );
```

```
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16complexFFT (
    double *data,
    unsigned int length );

// Test one bit of a 16-bit unsigned integer
PIXIE16APP_EXPORT unsigned short PIXIE16APP_API APP16_TstBit (
    unsigned short bit,
    unsigned short value );

// Set one bit of a 16-bit unsigned integer
PIXIE16APP_EXPORT unsigned short PIXIE16APP_API APP16_SetBit (
    unsigned short bit,
    unsigned short value );

// Clear one bit of a 16-bit unsigned integer
PIXIE16APP_EXPORT unsigned short PIXIE16APP_API APP16_ClrBit (
    unsigned short bit,
    unsigned short value );

// Set one bit of a 32-bit unsigned integer
PIXIE16APP_EXPORT unsigned int PIXIE16APP_API APP32_SetBit (
    unsigned short bit,
    unsigned int   value );

// Clear one bit of a 32-bit unsigned integer
PIXIE16APP_EXPORT unsigned int PIXIE16APP_API APP32_ClrBit (
    unsigned short bit,
    unsigned int   value );

// Test one bit of a 32-bit unsigned integer
PIXIE16APP_EXPORT unsigned int PIXIE16APP_API APP32_TstBit (
    unsigned short bit,
    unsigned int   value );
```

```
// Program on-board DACs
// Use this function to reprogram the on-board digital to analog converters (DAC) of the Pixie-16 modules. In t
his operation the DSP uses data from the DSP parameters that were previously downloaded.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16SetDACs (
    unsigned short ModNum );
```

```
// 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 co
mputer 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.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ProgramFippi (
    unsigned short ModNum );
```

```
// 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. If ModNum i
s 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 al
```

```
l 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 det
ermined by the DSP parameter BaselinePercent. For instance, if BaselinePercent is set to 10(%), the baseline le
vel of the input signals will be ~ 1638 on the 14-bit ADC scale (minimum: 0; maximum: 16383).
// The main purpose of this function is to ensure the input signals fall within the voltage range of the ADCs t
o ensure all input signals can be digitized by the ADCs properly.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16AdjustOffsets (
    unsigned short ModNum );
```

```
// Acquire baselines from a module
// Use this function to acquire baselines from Pixie-16 modules. Specify the module using ModNum. If ModNum is
set to be less than the total number of modules in the system, only the module specified by ModNum will have it
s baselines acquired. But if ModNum is set to be equal to the total number of modules in the system, then all m
odules 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 another function Pixie16ReadSglChanBaselines to read the baselines data out to the host
computer, channel by channel.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16AcquireBaselines (
    unsigned short ModNum );      // module number
```

```
// Read baselines from a channel in a module
// Use this function to read baselines data from a Pixie-16 module. Before calling this function, another funct
ion 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 baselines data length for each channel is
 3640. In the DSP internal memory, each baseline data 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 val
ues, timestamps corresponding to each baseline were also returned after this function call.
// Specify the module using ModNum and the channel on the module using ChanNum. Note that the modules and chann
els are counted starting at 0.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ReadSglChanBaselines (
    double *Baselines,            // returned baselines values
    double *TimeStamps,           // time stamp for each baseline value
    unsigned short NumBases,      // number of baseline values to read
    unsigned short ModNum,        // module number
    unsigned short ChanNum );     // channel number
```

```
// Ramp Offset DACs of a module and record the baselines
// Use this function to execute the RAMP_OFFSETDACS control task run. Each Offset DAC has 65536 steps, and the
RAMP_OFFSETDACS control task ramps the DAC from 0 to 65335 with a step size of 64, i.e., a total of 1024 steps.
 At each DAC step, the control task computes the baseline value as the representation of the signal baseline an
d stores it in the DSP memory. After the control task is finished, the stored baseline values are read out to t
he host computer and saved to a binary file called "rampdacs.bin" in the form of IEEE 32-bit floating point num
bers. Users can then plot the baseline values vs. DAC steps to determine the appropriate DAC value to be set in
 the DSP in order to bring the input signals into the voltage range of the ADCs. However, this function is no l
onger needed due to the introduction of function Pixie16AdjustOffsets.
// If ModNum is set to less than the total number of modules in the system, only the module specified by ModNum
 will start the RAMP_OFFSETDACS control task run. 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 start the R
AMP_OFFSETDACS control task run. Note that the modules are counted starting at 0.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16RampOffsetDACs (
    double *DCValues,             // returned DC offset values
    unsigned short NumDCVals,     // number of DC values to read
    unsigned short ModNum );
```

```
// Execute special control tasks
// Use this function to call special control tasks. This may include programming the Fippi or setting the DACs
after downloading DSP parameters.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ControlTaskRun (
    unsigned short ModNum,        // Pixie module number
    unsigned short ControlTask,   // Control task number
    unsigned int   Max_Poll );    // Timeout control in unit of ms for control task run
```

```
// Find the 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 valu
e is then downloaded to the DSP, where baselines are captured and averaged over time. The cut value would preve
nt a bad baseline value from being used in the averaging process, i.e., if a baseline value is outside the base
line cut range, it will not be used for computing the baseline average. Averaging baselines over time improves
energy resolution measurement.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16BLcutFinder (
    unsigned short ModNum,        // Pixie module number
    unsigned short ChanNum,       // Pixie channel number
    unsigned int   *BLcut );      // BLcut return value
```

```
// Find the exponential decay time of a channel
// Use this function to find the exponential decay time constant (Tau value) of the detector or preamplifier si
gnal that is connected to one channel of a Pixie-16 module. The found Tau value is returned via pointer *Tau.

PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16TauFinder (
    unsigned short ModNum,        // Pixie module number
    double         *Tau );        // 16 returned Tau values, in 社
```

```
// Write a MODULE level parameter to a module
// Use this function to write a module parameter to a Pixie-16 module.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16WriteSglModPar (
    char *ModParName,             // the name of the module parameter
    unsigned int   ModParData,    // the module parameter value to be written to the module
    unsigned short ModNum );      // module number
```

```
// Read a MODULE level parameter from a module
// Use this function to read a module parameter from a Pixie-16 module.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ReadSglModPar (
    char *ModParName,             // the name of the module parameter
    unsigned int   *ModParData,   // the module parameter value to be read from the module
    unsigned short ModNum );      // module number
```

```
// Write a CHANNEL level parameter to a module
// Use this function to write a channel parameter to a Pixie-16 module.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16WriteSglChanPar (
    char *ChanParName,            // the name of the channel parameter
    double ChanParData,           // the channel parameter value to be written to the module
    unsigned short ModNum,        // module number
    unsigned short ChanNum );     // channel number
```

```
// Read a CHANNEL level parameter from a module
// Use this function to read a channel parameter from a Pixie-16 module.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ReadSglChanPar (
    char *ChanParName,            // the name of the channel parameter
    double *ChanParData,          // the channel parameter value to be read from the module
    unsigned short ModNum,        // module number
    unsigned short ChanNum );     // channel number
```

```
// 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 t
his function, the host code should allocate appropriate amount of memory to store the histogram data. The defau
lt 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.
PIXIE16APP_EXPORT int PIXIE16APP_API 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
```

```
// 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.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ReadStatisticsFromModule (
    unsigned int   *Statistics,   // run statistics data
    unsigned short ModNum );      // module number
```

```
// 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 data to a file. New data will b
e appended to the end of the file. So the same file name can be used for multiple modules and the data from eac
h module will be stored in the order that this function is called.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16SaveHistogramToFile (
    char *FileName,               // histogram data file name
    unsigned short ModNum);       // module number
```

```
// 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 mo
dule will be reported.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16GetModuleEvents (
    char *FileName,               // the list mode data file name (with complete path)
    unsigned int *ModuleEvents ); // receives number of events for each module
```

```
// Get detailed events information from a data file
// Use this function to retrieve the detailed information of each event in the list mode data file for the desi
gnated module. Before calling this function to get the individual events information, another function Pixie16G
etModuleEvents 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 w
ith a length of (nEvents*68):
// EventInformation = (unsigned long *)malloc(sizeof(unsigned long) * nEvents * 68);
// where 68 is the length of the information records of each event (energy, timestamps, etc.) and has the follo
wing structure.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16GetEventsInfo (
    char *FileName,                 // the list mode data file name (with complete path)
    unsigned int   *EventInformation, // to hold event information
    unsigned short ModuleNumber);     // the module whose events are to be retrieved
```

```
// 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.
PIXIE16APP_EXPORT int PIXIE16APP_API 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
```

```
// 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.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ReadHistogramFromFile (
    char  *FileName,              // the histogram data 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
```

```
// 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 v
alues (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 har
d drive.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16SaveDSPParametersToFile (
    char  *FileName );            // the DSP parameters file name (with complete path)
```

```
// 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 mod
ules that are installed in the system. Each module has exactly 1280 DSP parameter values (32-bit unsigned integ
ers), 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.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16LoadDSPParametersFromFile (
    char  *FileName );            // the DSP parameters file name (with complete path)
```

```
// 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 destinati
on module(s).
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16CopyDSPParameters (
    unsigned short BitMask,             // copy items bit mask
    unsigned short SourceModule,        // source module
    unsigned short SourceChannel,       // source channel
    unsigned short *DestinationMask );  // the destination module and channel bit mask
```

```
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ReadMSGFile (
    char *ReturnMsgStr );
```

```
// Convert a decimal into IEEE 32-bit floating point number
PIXIE16APP_EXPORT unsigned int PIXIE16APP_API Decimal2IEEEFloating(double DecimalNumber);

// Convert an IEEE 32-bit floating point number to a decimal
PIXIE16APP_EXPORT double PIXIE16APP_API IEEEFloating2Decimal(unsigned int IEEEFloatingNumber);
```

```
// Read data from external FIFO and save to a file
// Use this function to read data from the external FIFO of a module. This function can only be used for Pixie-
16 Revision-B, C, and D modules.
// This function first checks the status of the external FIFO of a Pixie-16 module, and if there are data in th
e 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 Pixie16ReadDataFromExt
ernalFIFO 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 rea
d is at the end of a run (1) or during the run (0). This is necessary since the external FIFO needs special tre
atment when the host reads the last few words from the external FIFO due to its pipelined structure.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16SaveExternalFIFODataToFile (
    char *FileName,               // list mode data file name
    unsigned int   *nFIFOWords,   // number of words read from external FIFO
    unsigned short ModNum,        // module number
    unsigned short EndOfRunRead); // indicator whether this is the end of run read
```

```
// 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.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16RegisterIO (
    unsigned short ModNum,        // the Pixie module to communicate to
    unsigned int   address,       // register address
```

```
    unsigned short direction,     // either MOD_READ or MOD_WRITE
    unsigned int   *value );      // holds or receives the data
```

```
// Read Control & Status Register value from a module
// Use this function to read the host Control & Status Register (CSR) value.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ReadCSR (
    unsigned short ModNum,
    unsigned int   *CSR );
```

```
// Write to Control & Status Register in a module
// Use this function to write a value to the host Control & Status Register (CSR).
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16WriteCSR (
    unsigned short ModNum,
    unsigned int   CSR );
```

```
// 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 acqui
sition run is in progress. The function returns the number of words (32-bit) that the external FIFO currently h
as. If the number of words is greater than a user-set threshold, function Pixie16ReadDataFromExternalFIFO can t
hen be used to read the data from the external FIFO. The threshold can be set by the user to either minimize re
ading overhead or to read data out of the FIFO as quickly as possible.
// *nFIFOWords returns the number of 32-bit words that the external FIFO currently has.
// ModNum is the module number which starts counting at 0.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16CheckExternalFIFOStatus (
    unsigned int   *nFIFOWords,
    unsigned short ModNum );
```

```
// Read data from external FIFO of a module
// Use this function to read data from the external FIFO of a module. This function can only be used for Pixie-
16 Revision-B, C, and D modules.
// 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, then this function is called to read the data from the FIFO.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ReadDataFromExternalFIFO (
    unsigned int   *ExtFIFO_Data, // To receive the external FIFO data
    unsigned int   nFIFOWords,    // number of words to read from external FIFO
    unsigned short ModNum );      // module number
```

```
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ComputeFastFiltersOffline (
    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
    double         *cfd );             // cfd response
```

```
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16ComputeSlowFiltersOffline (
    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
```

```
// Add by Hongyi Wu
```

```
PIXIE16APP_EXPORT int PIXIE16APP_API 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
    unsigned int   bl,
    double         sl,
    double         sg,
    double         tau,
    int            sfr,
    int            pointtobl );
```

 updated: 2018-11-03 15:08:17

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！！！

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

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

MakeEvent

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

OnlineStattics

- Linkdef.hh
- main.cc
  - 主程序
- Makefile
- Online.cc
- Online.hh
  - 在线监视界面
- PixieOnline.config

 updated: 2018-11-03 15:08:17