Table of Contents

README

最新版本号 **Version:2018.05.12**

程序下载请访问(私有仓库，暂未开放下载) https://github.com/wuhongyi/PKUXIADAQ

网页版说明书请访问 http://wuhongyi.cn/PKUXIADAQ/

markdown版本说明书请访问 README/

离线网页版说明书请访问 docs/

pdf版本说明书请访问 README.pdf

- 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，当前基本以操作演示讲解软件的使用为主。

本说明书仅适用于 **XIA LLC Pixie-16**

- 本程序由北京大学实验核物理组开发。
- 本程序的开发参考借鉴了**TODO**程序。
- 特别感谢 谭辉*(XIA LLC)* 对我们开发的支持。

技术指导:

- 李智焕
- 谭辉(XIA LLC)

软件主要开发者:

- 吴鸿毅(wuhongyi@qq.com)
- 李晶(lijinger02@126.com)

说明书主要撰写者:

- 吴鸿毅
- 王翔
- 罗迪雯

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 添加各类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)。

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(说明书 m d版)
- README.md(github首页介绍)
- README.pdf(pdf版本说明书)
- software(pixie16驱动API，非官方标准，已经被吴鸿毅修改)
- 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:
    - FastCGI
    - JSROOT
    - 网页前端
    - ...

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
```

```
##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，时值允许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未不同的数值。
#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 表示 Run Number
```

Guide

- 这里需要介绍跳线接法
- 原始数据定义
- 算法原理
- 。。。

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。

Decode

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。

ROOT File Branch：

- sr(short):sample rate，100/250/500，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):timestamps
- ets(long int 64 bit):外部时标
- 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

TODO 这里添加一个**Branch**截图。。。

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

GUI

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: 2018.05.12
System not booted
Last run number: 1060

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* 栏中从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**。

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。

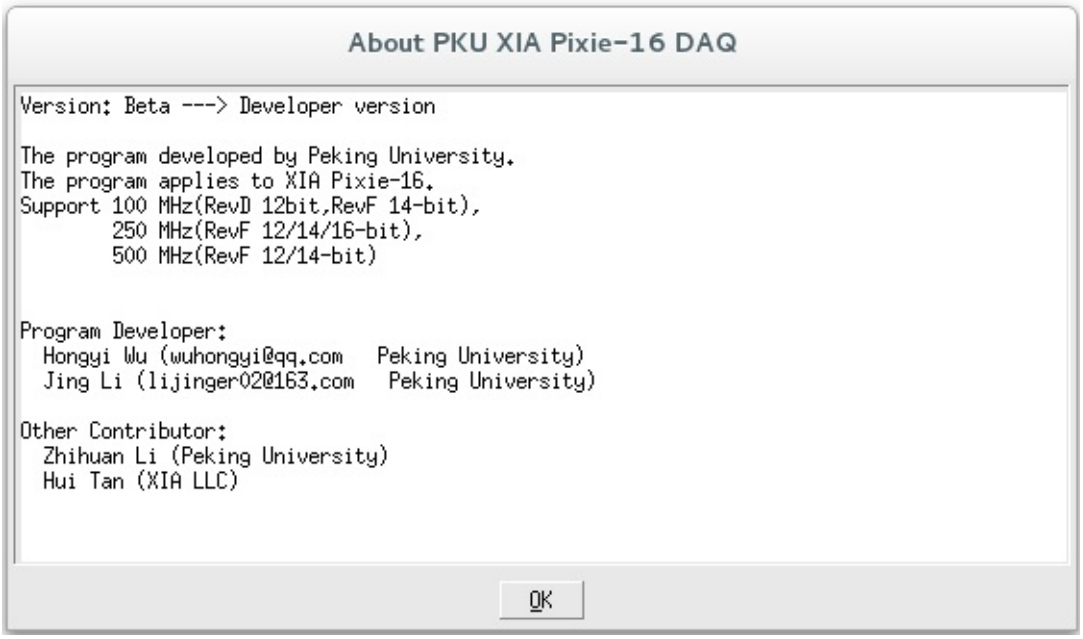
Control 栏内的 **Online Statistics** 选项开启则获取每 **3 s** 向 *OnineStatistics* 程序发送时时每路信号的输入率、输出率信息。

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。

File 下拉栏

本下拉栏内容没有实际用途。

About

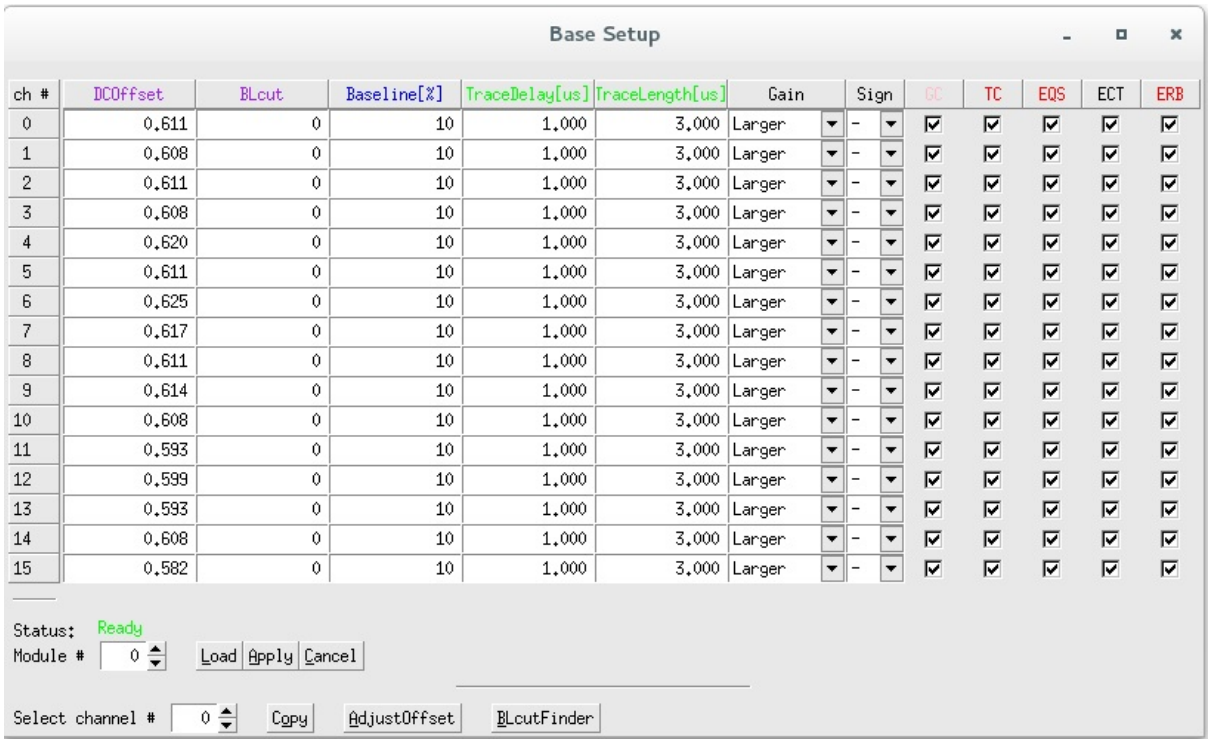


软件开发者介绍。之后将会添加主程序基本操作说明。

UV_Setup 下拉栏

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。

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。

- 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* 表示是否开启该通道，选择表示开启该通道，不选择表示不开启。
- 选项 *ECT* 选择表示开启CFD功能。

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）

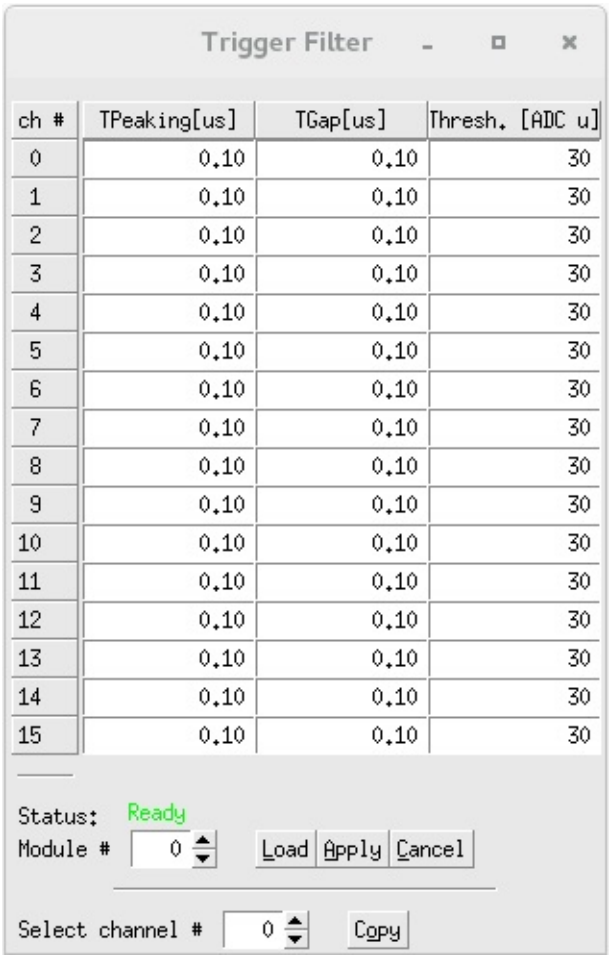
蓝色的 *Baseline* 用来调节基线位置，通过电压的补偿将基线调节到用户预期的位置：

- *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。

Trigger Filter

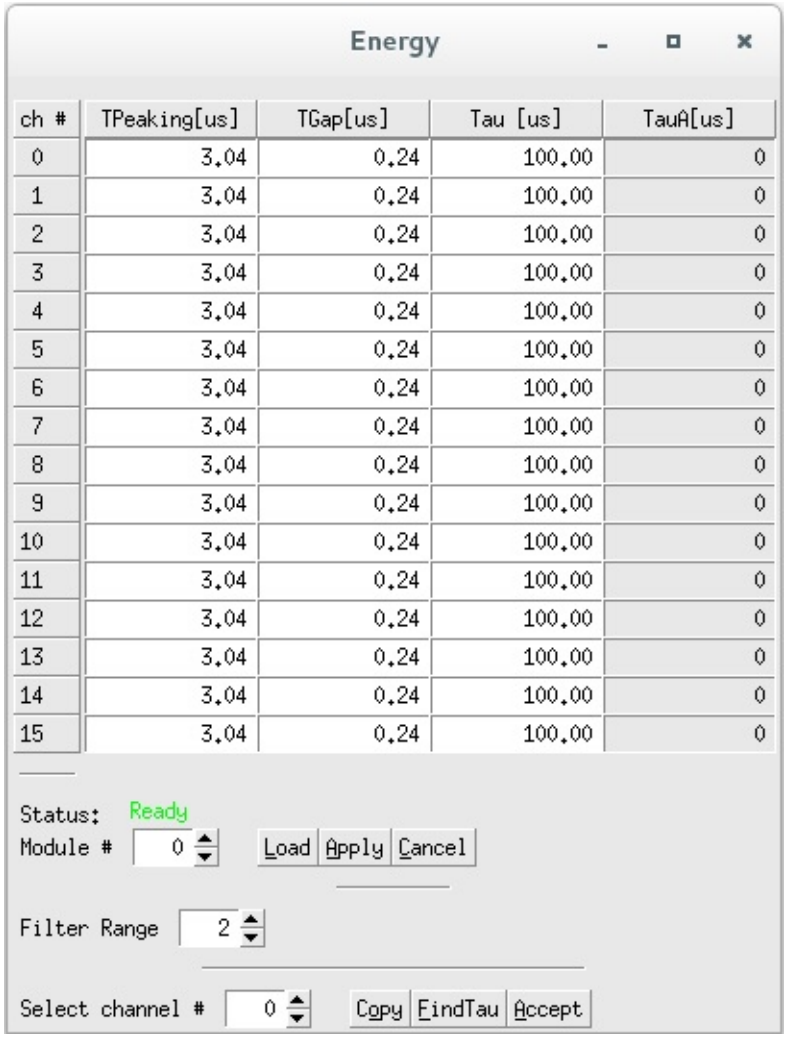


界面下方的 Status 显示为绿色的 **Ready** 时表示可操作该界面，否则需要等待。底下按钮的操作同上。

- 参数 *TPeaking* 。。。
- 参数 *TGap* 。。。
- 参数 *Thresh.* 表示阈值，该数值的设置是相对 fast filter 波形。

TODO 这里需要补充一个示意图，还有计算公式。

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。

。。。。

CFD

Cfd Par.

ch #	CFD Delay[us]	CFD Frac[0-7]	CFD Thre
0	0.10	2.00	120.00
1	0.10	2.00	120.00
2	0.10	2.00	120.00
3	0.10	2.00	120.00
4	0.10	2.00	120.00
5	0.10	2.00	120.00
6	0.10	2.00	120.00
7	0.10	2.00	120.00
8	0.10	2.00	120.00
9	0.10	2.00	120.00
10	0.10	2.00	120.00
11	0.10	2.00	120.00
12	0.10	2.00	120.00
13	0.10	2.00	120.00
14	0.10	2.00	120.00
15	0.10	2.00	120.00

Status: Ready
Module # 0 Load Apply Cancel

Select channel # 0 Copy

QDC

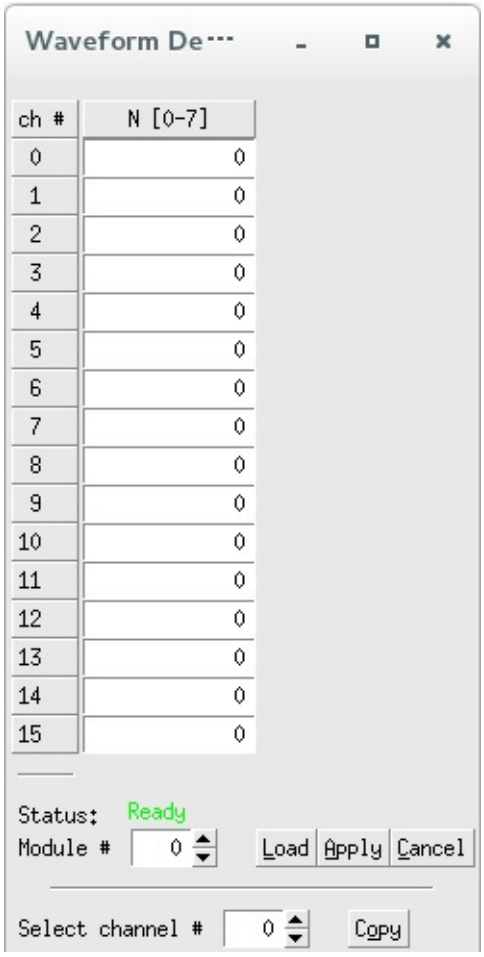
Qdc Par.

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

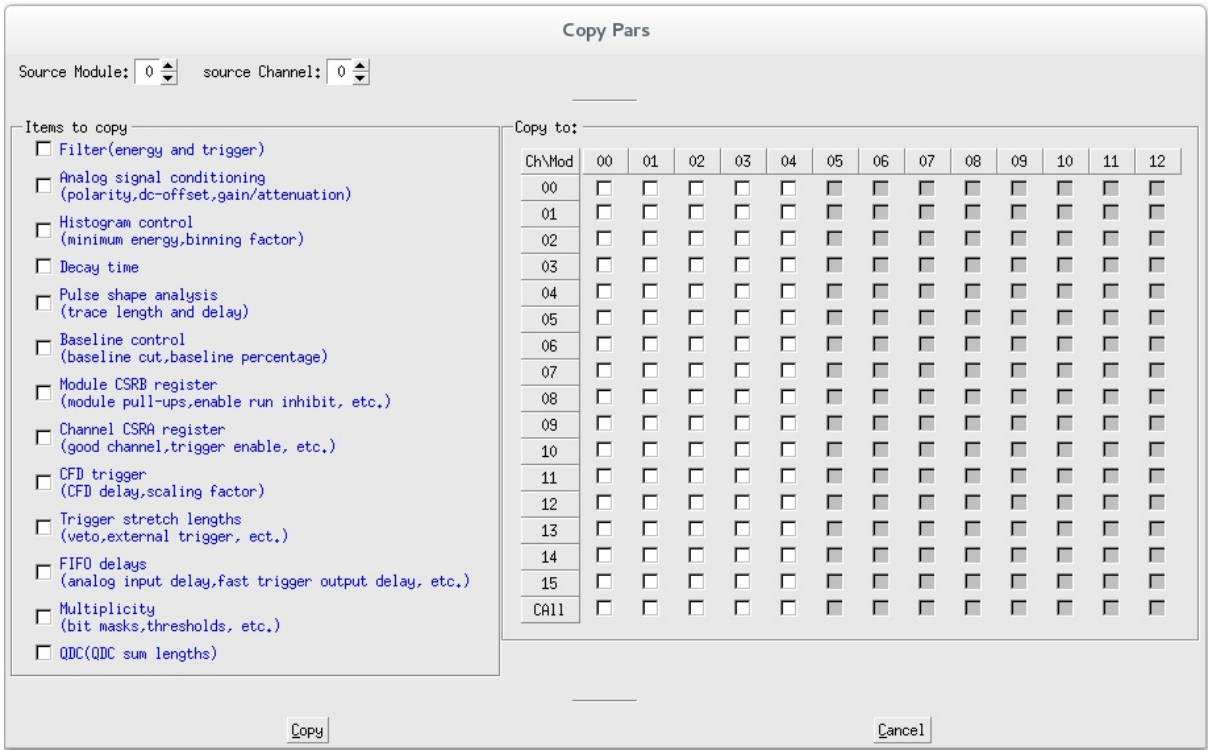
Status: Ready
Module # 0 Load Apply Cancel

Select channel # 0 Copy

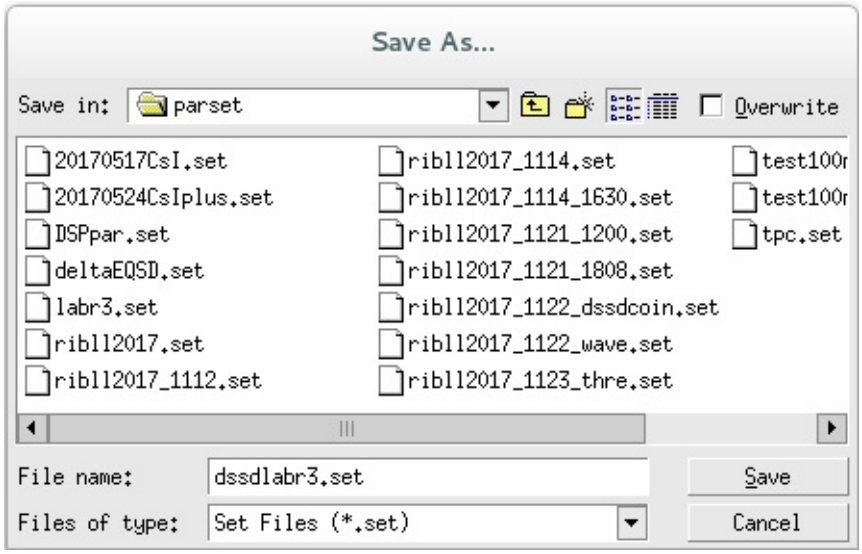
Decimation



Copy Pars



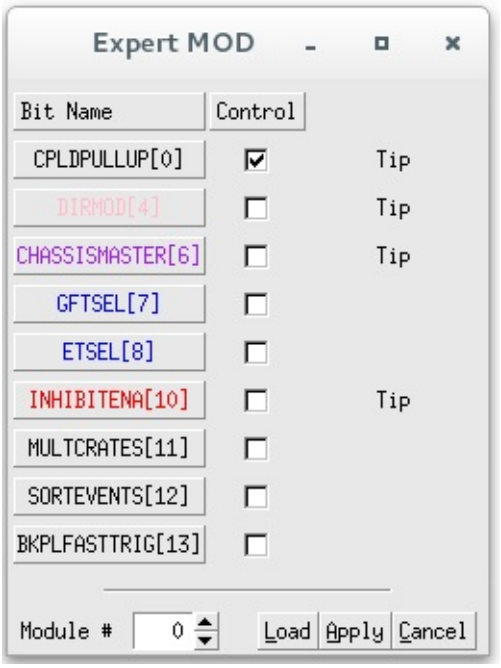
Save2File



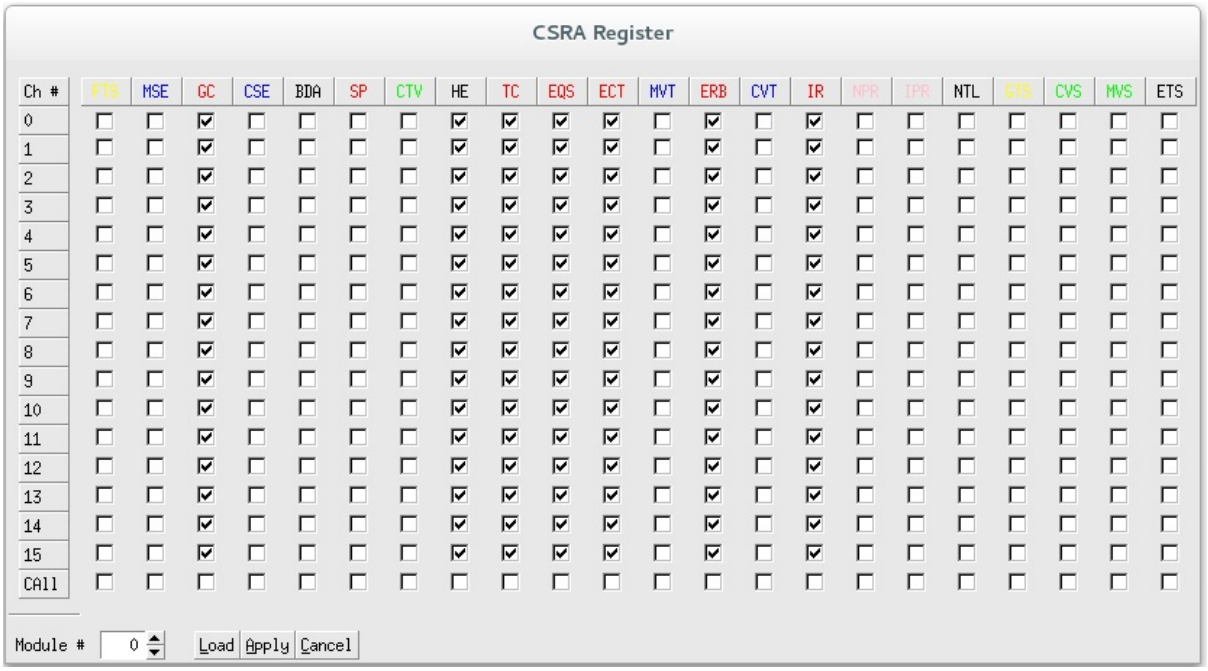
Expert 下拉栏

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。

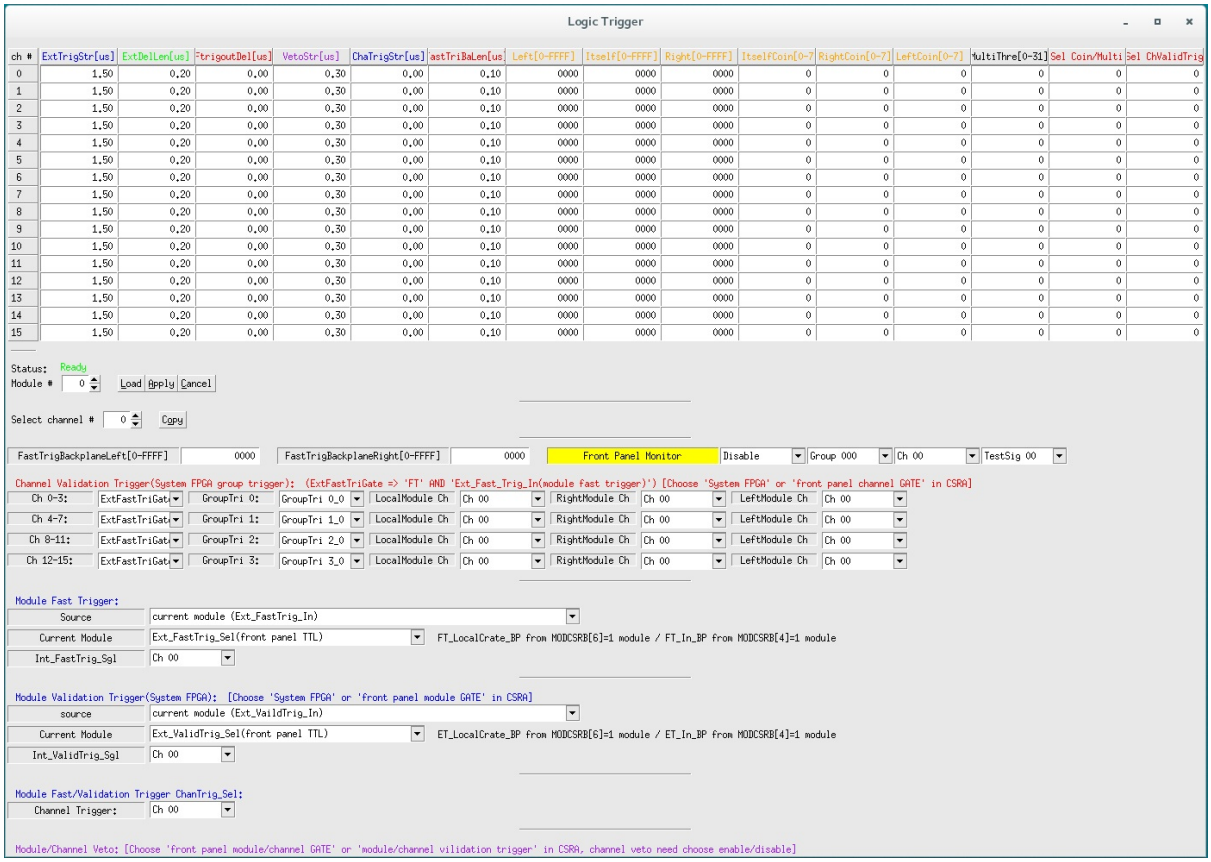
Module Variables



CSRA



Logic Set



Monitor 下拉栏

本下拉栏中调节内容为监视波形噪声水平、基线分布等。

Hist & XDT

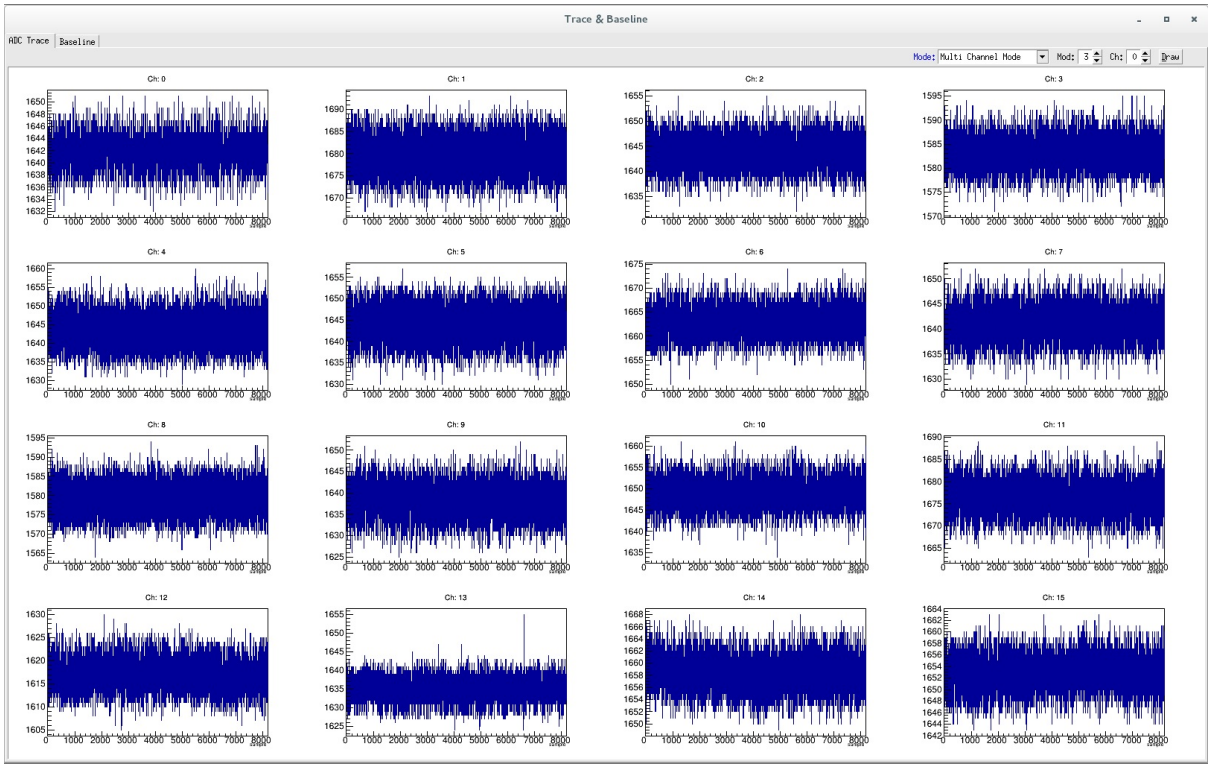
Hist & XDT

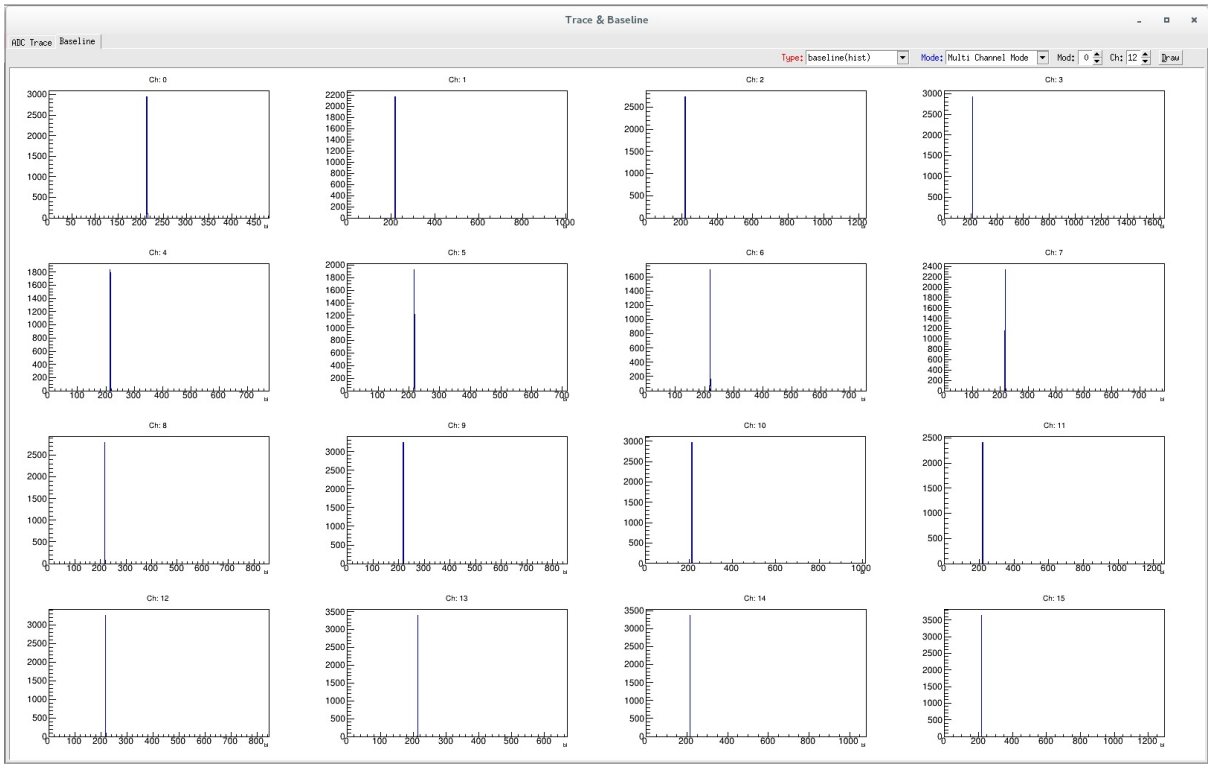
ch #	EMin [ADC u]	BinFactor	dT [us]
0	0	1	1.02
1	0	1	1.02
2	0	1	1.02
3	0	1	1.02
4	0	1	1.02
5	0	1	1.02
6	0	1	1.02
7	0	1	1.02
8	0	1	1.02
9	0	1	1.02
10	0	1	1.02
11	0	1	1.02
12	0	1	1.02
13	0	1	1.02
14	0	1	1.02
15	0	1	1.02

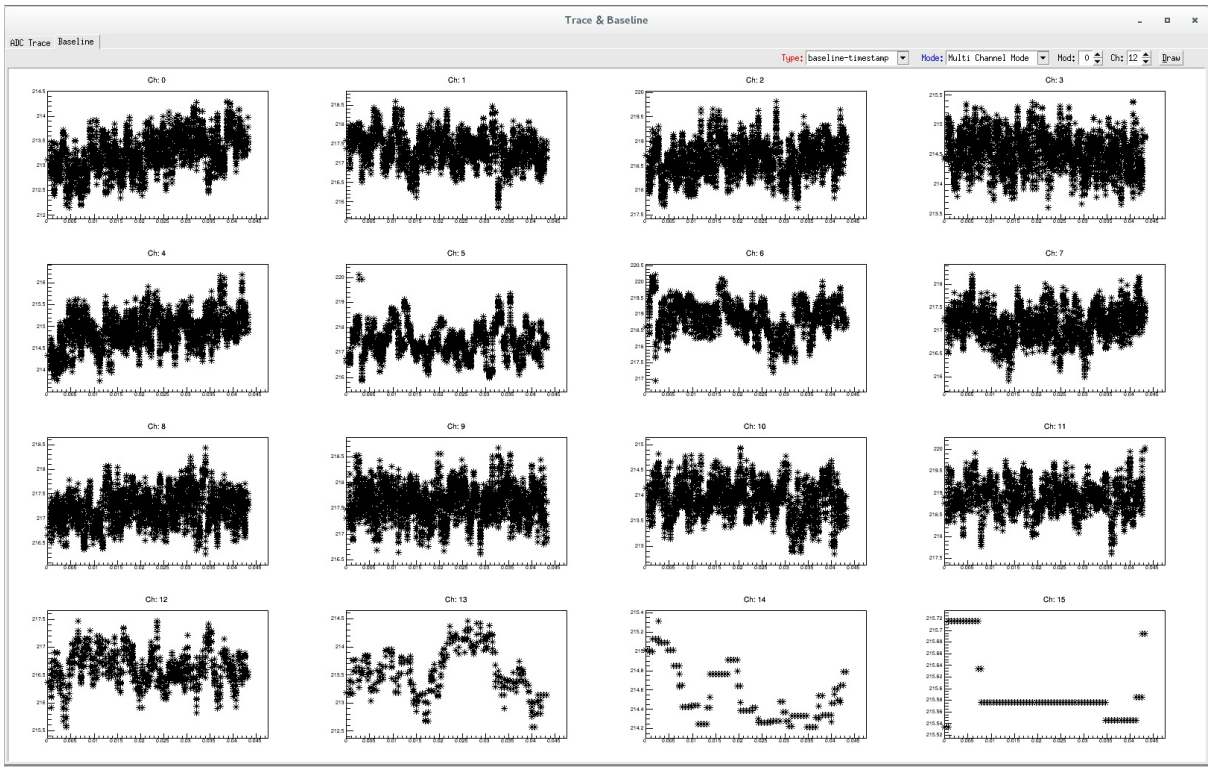
Status: Ready
Module # 0 Load Apply Cancel

Select channel # 0 Copy

Trace & Baseline



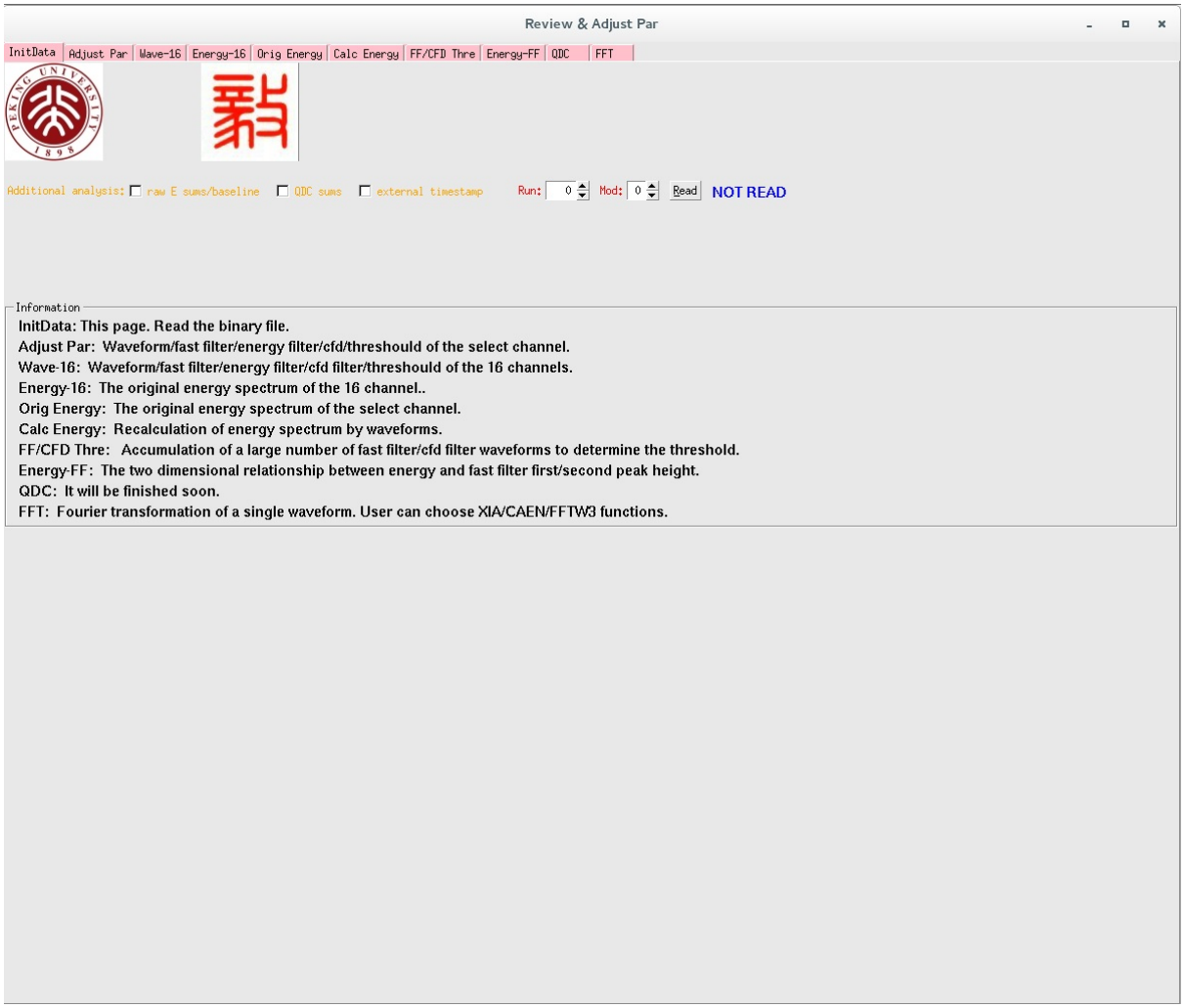




Offiline 下拉栏

本下拉栏中为离线参数优化调节。

Adjust Par



Simulation(暂未实现)

通过模型产生不同类型探测器的波形，辅助使用者学习参数优化调节的。

Online Stattics

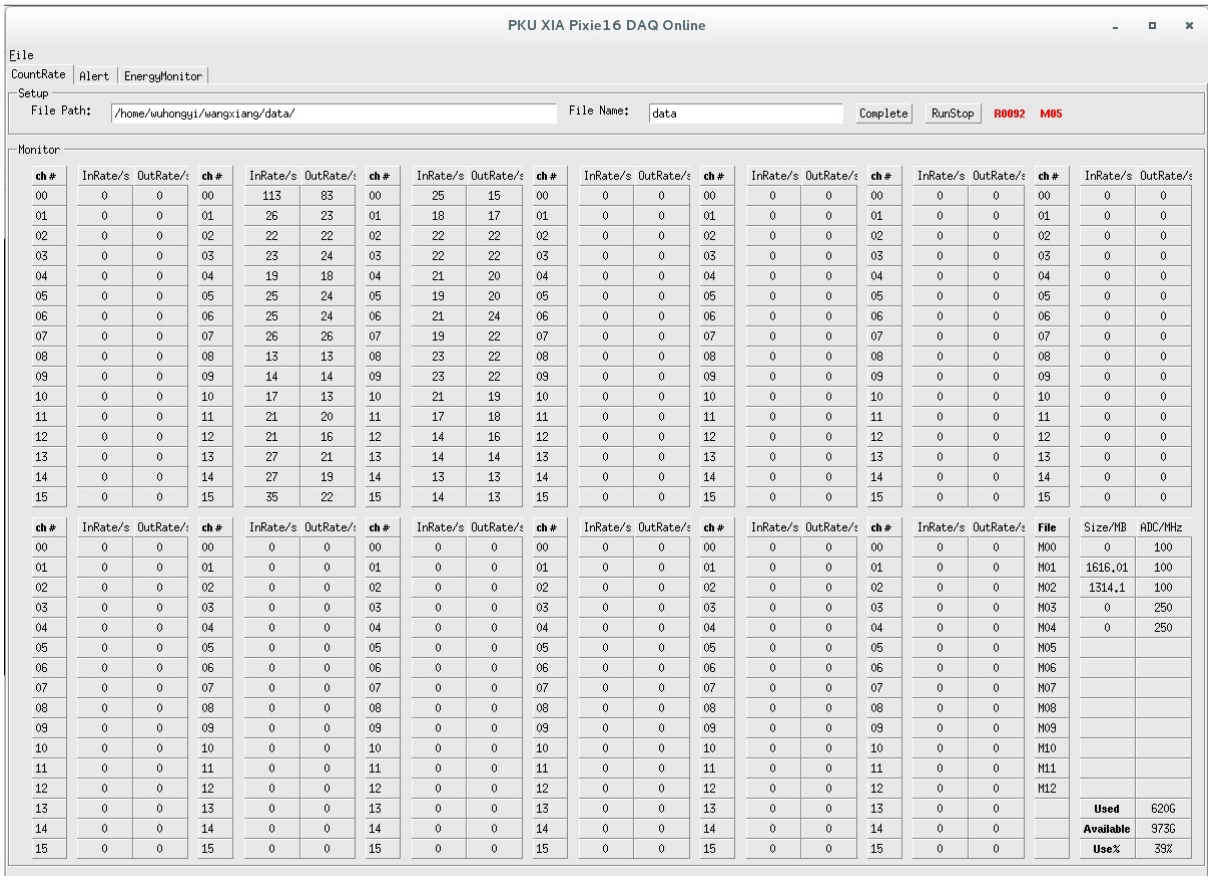
修改 **OnlineStattics** 中的文件 **PixieOnline.config**，其中第一行为原始二进制文件存放路径，第二行为文件名。通过该两行参数来监视每个文件时时大小及硬盘占用量。

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：



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。。。

Front Panel

本页内容待更新。。。

LVDS I/O port(J101)

普通网口

当前标准固件中还没定义

J155（letter"A"）

输出信号

3.3V I/O port，输出阻抗是 50 欧

A2、D1 地 A1、B1、B2、C1、C2、D2 输出信号

J151-J154

输入信号

differential LVDS signals

- 16 channel gate input
- 1 module gate input
- 1 not use current

J151-J155

输入信号

single-ended TTL external input signals

FI0、FI2、FI3、FI4、FI6、FI7

group 000

FTRIG_DELAY 采集延迟时间 只要fast filter 过阈值就会产生 FTRIG_VAL 基本同上，有效采集时候才有信号 GLBETRIG_CE stretched external global validation trigger CHANETRIG_CE stretched channel validation trigger ，开启CSRA bit13 。采集延迟必须在这个时间窗内才能采集到 看到的信号实际起始位置在 100 ns，意味着System FPGA 处理时间需要 100 ns ？ 信号宽度由 ChanTrigStretch 控制。

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。

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. PXIS
lotMap 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 physi
cal slot number that the modules reside in. The logical module number runs from 0.
For instance, in a system with 5 Pixie-16 modules, these 5 modules may occupy slots
 3 through 7. The user must fill PXISlotMap as follows: PXISlotMap = {3, 4, 5, 6, 7
 ...} since module number 0 resides in slot number 3, etc. To find out in which slo
t a module is located, any piece of subsequent code can use the expression PXISlotM
ap[ModNum], where ModNum is the logic module number.
// OfflineMode is used to indicate to the API whether the system is running in OFFL
INE mode (1) or ONLINE mode (0). OFFLINE mode is useful for situations where no Pix
ie-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 informa
tion about the Host PC’s PCI buses, including the slot enumeration scheme. XIA’s so
ftware distribution normally puts this file under the same folder as Pixie-16 softw
are installation folder. However, the user has the flexibility of putting it in oth
er folders by simply changing the definition of the string PCISysIniFile_Windows or
 PCISysIniFile_Linux in the header part of the file pixie16sys.c, depending on whic
h 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 ea
ch 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 Pi
xie-16 modules when these modules are initialized by function Pixie16InitSystem. Th
is 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 th
e module specified by ModNum will be closed. But if ModNum is equal to the total nu
mber 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 t
aking. The function downloads to the Pixie-16 modules the communication FPGA config
urations, signal processing FPGA configurations, trigger FPGA configurations (Revis
ion 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 hard
ware mounted on the modules; the DSP codes have a length which depends on the actua
l compiled code; and the set of DSP parameters always consists of 1280 32-bit words
 for each module. The host software has to make the names of those boot data files
on the hard disk available to the boot function.
// If ModNum is set to be less than the total number of modules in the system, only
 the module specified by ModNum will be booted. But if ModNum is equal to the total
 number of modules in the system, e.g. there are 5 modules in the chassis and ModNu
m = 5, then all modules in the system will be booted.
// The boot pattern is a bit mask (shown below) indicating which on-board chip will
 be booted. Under normal circumstances, all on-board chips should be booted, i.e. t
he 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 f
ile
    char *TrigFPGAConfigFile,     // name of trigger FPGA configuration file
    char *DSPCodeFile,            // name of executable code file for digital signa
l 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 modul
e 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 its ADC traces acquired. But
 if ModNum is equal to the total number of modules in the system, then all modules
in the system will have their ADC traces acquired.
// After the successful return of this function, the DSP's internal memory will be
filled with ADC trace data. A user's application software should then call another
function Pixie16ReadSglChanADCTrace to read the ADC trace data out to the host comp
uter, 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 function Pixie16AcquireADCTrace should be called to fill the
 DSP internal memory first. Also, the host code should allocate appropriate amount
of memory to store the trace data. The ADC trace data length for each channel is 81
92. Since the trace data are 16-bit unsigned integers (actually only the lower 14-b
it 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 memory. So for each channel, 4
096 32-bit words are read out first from the DSP, and then each 32-bit word is unpa
cked to form two 16-bit words.
// Specify the module using ModNum and the channel on the module using ChanNum. Not
e 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 interna
l memory of a Pixie-16 module. The DSP internal memory is split into two blocks wit
h address range 0x40000 to 0x4FFFF for the first block and address range 0x50000 to
 0x5FFFF for the second block. Within the first block, address range 0x40000 to 0x4
9FFF is reserved for program memory and shouldn’t be accessed directly by the host.
 Address range 0x4A000 to 0x4A4FF is used by the DSP I/O parameters which are store
d in the configuration files (.set files) in the host. Within this range, 0x4A000 t
o 0x4A33F can be both read and written, but 0x4A340 to 0x4A4FF can only be read but
 not written. The remaining address range (0x4A500 to 4FFFF) in the first block and
 the entire second block (0x50000 to 0x5FFFF) should only be read but not written b
y the host. Use Direction = 1 for read and Direction = 0 for 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 exter
nal memory of a Pixie-16 module. The valid memory address is from 0x0 to 0x7FFFF (3
2-bit wide). Use Direction = 1 for read and Direction = 0 for write.
// The external memory is used to store the histogram data accumulated for each of
the 16 channels of a Pixie-16 module. Each channel has a fixed histogram length of
32768 words(32-bit wide), and the placement of the histogram data in the memory is
in the same order of the channel number, i.e. channel 0 occupies memory address 0x0
 to 0x7FFF, channel 1 occupies 0x8000 to 0xFFFF, and so on.
// NOTE: another function Pixie16ReadHistogramFromModule can also be used to read o
ut the histograms except that it needs to be called channel by channel.
// In Rev-A modules, part of the external memory is also used to store the list mod
e data in ping-pong buffering mode. This function can be used to read list mode dat
```

```
a 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 collect data on an event-by-event basis, gathering energi
es, timestamps, pulse shape analysis values, and waveforms, for each event. Runs wi
ll continue until a preset number of events are reached or the user terminates the
run by calling function Pixie16EndRun. Once the run is progress, if the run is set
to terminate after a given number of events have been accumulated, another function
, Pixie16CheckRunStatus, should be called to check if the run has finished. To star
t the data acquisition this function has to be called for every Pixie-16 module in
the system. If all modules are to run synchronously, The last module addressed will
 release all others and the acquisition starts then. The first module to end the ru
n will immediately stop the run in all other modules.
// Use mode=NEW_RUN (=1) to erase histograms and statistics information before laun
ching the new run. Note that this will cause a start up delay of up to 1 millisecon
d. Use mode=RESUME_RUN (=0) to resume an earlier run. This mode has a start up dela
y 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 d
ata) and 0x103 (energy and timestamp only).
// For Rev-B, C, D modules, there are only one list mode run type supported, that i
s, 0x100. However, different output data options can be chosen by enabling or disab
ling different CHANCSRA bits.
// Histograms and statistics data are updated incrementally from run to run provide
d RESUME_RUN mode is used.
// ModNum is the module number which starts counting at 0. If ModNum is set to be l
ess than the total number of modules in the system, only the module specified by Mo
dNum will have its list mode run started. But if ModNum is set to equal to the tota
l number of modules in the system, then all modules in the system will have their r
uns 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 histog
rams, one for each channel. It launches a data acquisition run in which only energy
 information is preserved and histogrammed locally to each channel.
// Call this function for each Pixie-16 module in the system. The last module addre
ssed will allow the actual data acquisition to begin. Histogram run can be self-ter
minating when the elapsed run time exceeds the preset run time, or the user can pre
maturely 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 laun
ching 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 l
ess than the total number of modules in the system, only the module specified by Mo
dNum will have its histogram run started. But if ModNum is set to be equal to the t
otal number of modules in the system, then all modules in the system will have thei
r 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 poll
ing.
// If the return code of this function indicates the run has finished, there might
still be some data in the external memory (Rev-A modules) or external FIFO (Rev-B,
C, D modules) that need to be read out to the host. In addition, final run statisti
cs and histogram data are available for reading out too.
// In MCA histogram run mode, this function can also be called to check if the run
is still in progress even though it is normally self-terminating.
// ModNum is the module number which starts counting at 0.
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 modu
le needs to be addressed this way. It will immediately stop the run in all other mo
dule 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 t
his function, another function, Pixie16ReadStatisticsFromModule, should be called t
o read statistics data from the module first.
// *Statistics is a pointer to an array whose size is exactly 448 unsigned integer
words (32-bit). The *Statistics array is filled with data from a Pixie-16 module af
ter calling function Pixie16ReadStatisticsFromModule. ModNum is the module number w
hich 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-1
6 module. This function does not communicate with Pixie-16 modules. Before calling
this function, another function, Pixie16ReadStatisticsFromModule, should be called
to read statistics data from the module first.
// *Statistics is a pointer to an array whose size is exactly 448 unsigned integer
words (32-bit). The *Statistics array is filled with data from a Pixie-16 module af
ter calling function Pixie16ReadStatisticsFromModule. ModNum is the module number w
hich 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 modu
le has spent on data acquisition. This function does not communicate with Pixie-16
modules. Before calling this function, another function, Pixie16ReadStatisticsFromM
odule, should be called to read statistics data from the module first.
// *Statistics is a pointer to an array whose size is exactly 448 unsigned integer
words (32-bit). The *Statistics array is filled with data from a Pixie-16 module af
ter calling function Pixie16ReadStatisticsFromModule. ModNum is the module number w
hich 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 data acquisition run. This function is only used by Rev-
A modules. This function does not communicate with Pixie-16 modules. Before calling
 this function, another function, Pixie16ReadStatisticsFromModule, should be called
 to read statistics data from the module first.
// *Statistics is a pointer to an array whose size is exactly 448 unsigned integer
words (32-bit). The *Statistics array is filled with data from a Pixie-16 module af
ter calling function Pixie16ReadStatisticsFromModule. ModNum is the module number w
hich 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 function does not communicate with Pixie-16 modules. Before
 calling this function, another function, Pixie16ReadStatisticsFromModule, should b
e called to read statistics data from the module first.
// *Statistics is a pointer to an array whose size is exactly 448 unsigned integer
words (32-bit). The *Statistics array is filled with data from a Pixie-16 module af
```

```
ter calling function Pixie16ReadStatisticsFromModule. ModNum is the module number w
hich 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) o
f the Pixie-16 modules. In this 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-1
6 modules. After the host computer has written the DSP parameters to the DSP memory
```

```
, the DSP needs to write some of these parameters to the FPGAs. This function makes
 the DSP perform that action.
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 modu
le using ModNum. If ModNum is set to be less than the total number of modules in th
e system, only the module specified by ModNum will have its DC-offsets adjusted. Bu
t if ModNum is set to be equal to the total number of modules in the system, then a
ll modules in the system will have their DC-offsets adjusted.
// After the DC-offset levels have been adjusted, the baseline level of the digitiz
ed input signals will be determined by the DSP parameter BaselinePercent. For insta
nce, if BaselinePercent is set to 10(%), the baseline level of the input signals wi
ll 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 to 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 its baselines acquired. But i
f ModNum is set to be equal to the total number of modules in the system, then all
modules in the system will have their baselines acquired.
// After the successful return of this function, the DSP's internal memory will be
filled with baselines data. Users should then call another function Pixie16ReadSglC
hanBaselines to read the baselines data out to the host computer, channel by channe
l.
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 function Pixie16AcquireBaselines should be called to fill th
e 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 i
s 3640. In the DSP internal memory, each baseline data is a 32-bit IEEE floating po
int number. After being read out to the host, this function will convert each basel
ine data to a decimal number. In addition to baseline values, timestamps correspond
ing to each baseline were also returned after this function call.
// Specify the module using ModNum and the channel on the module using ChanNum. Not
e that the modules and channels 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 D
AC has 65536 steps, and the RAMP_OFFSETDACS control task ramps the DAC from 0 to 65
335 with a step size of 64, i.e., a total of 1024 steps. At each DAC step, the cont
rol task computes the baseline value as the representation of the signal baseline a
nd stores it in the DSP memory. After the control task is finished, the stored base
line values are read out to the host computer and saved to a binary file called “ra
mpdacs.bin” in the form of IEEE 32-bit floating point numbers. 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 longer needed due to the introduction of function Pixi
e16AdjustOffsets.
// If ModNum is set to less than the total number of modules in the system, only th
e module specified by ModNum will start the RAMP_OFFSETDACS control task run. But i
f ModNum is equal to the total number of modules in the system, e.g. there are 5 mo
dules in the chassis and ModNum = 5, then all modules in the system will start the
RAMP_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 th
e 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 m
odule. The baseline cut value is then downloaded to the DSP, where baselines are ca
ptured and averaged over time. The cut value would prevent a bad baseline value fro
m being used in the averaging process, i.e., if a baseline value is outside the bas
eline 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 signal that is connected to one channel of a Pixie-16 mod
ule. 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 th
e 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 t
he 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 th
e 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 histog
ram memory. Before calling this function, the host code should allocate appropriate
 amount of memory to store the histogram data. The default histogram length is 3276
8. Histogram data are 32-bit unsigned integers.
// Specify the module using ModNum and the channel on the module using ChanNum. Not
e 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 cal
```

```
ling this function, the host code should allocate appropriate amount of memory to s
tore 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 dat
a to a file. New data will be appended to the end of the file. So the same file nam
e can be used for multiple modules and the data from each 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 module will be reported.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16GetModuleEvents (
    char *FileName,               // the list mode data file name (with complete pa
th)
    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 designated module. Before calling this function to get the
individual events information, another function Pixie16GetModuleEvents should be ca
lled first to determine the number of events that have been recorded for each modul
e. If the number of events for a given module is nEvents, a memory block *EventInfo
rmation should be allocated with 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, timesta
mps, etc.) and has the following structure.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16GetEventsInfo (
    char *FileName,                   // the list mode data file name (with complet
e path)
    unsigned int   *EventInformation, // to hold event information
    unsigned short ModuleNumber);     // the module whose events are to be retrieve
d

// Read trace data from a list mode data file
// Use this function to retrieve list mode trace from a list mode data file. It use
s the trace length and file location information obtained from function Pixie16GetE
ventsInfo 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 call
ing this function, the host code should allocate appropriate amount of memory to st
ore the histogram data. The default histogram length is 32768. Histogram data are 3
2-bit unsigned integers.
// Specify the module using ModNum and the channel on the module using ChanNum. Not
e 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 pa
th)
    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 set
tings file. Each module has exactly 1280 DSP parameter values (32-bit unsigned inte
gers), 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 wh
en stored on the computer hard drive.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16SaveDSPParametersToFile (
    char  *FileName );            // the DSP parameters file name (with complete pa
th)

// Load DSP parameters to modules from a file
// Use this function to read DSP parameters from a settings file and then download
the settings to Pixie-16 modules that are installed in the system. Each module has
exactly 1280 DSP parameter values (32-bit unsigned integers), and depending on the
value of PRESET_MAX_MODULES (defined in pixie16app_defs.h), the settings file shoul
d have exactly (1280 * PRESET_MAX_MODULES * 4) bytes when stored on the computer ha
rd drive.
PIXIE16APP_EXPORT int PIXIE16APP_API Pixie16LoadDSPParametersFromFile (
    char  *FileName );            // the DSP parameters file name (with complete pa
th)

// 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 so
urce module to the destination 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 m
ask
```

```
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 DecimalNu
mber);

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


// 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 the external FIFO, this function then reads list mode dat
a (32-bit unsigned integers) from the external FIFO. So this function essentially e
ncapsulates both functions Pixie16CheckExternalFIFOStatus and Pixie16ReadDataFromEx
ternalFIFO within one function. The number of words that are read from the external
 FIFO is recorded in variable*FIFOWords.
// The function also expects setting the value of a variable called “EndOfRunRead”
to indicate whether this read is at the end of a run (1) or during the run (0). Thi
s is necessary since the external FIFO needs special treatment when the host reads
the last few words from the external FIFO due to its pipelined structure.
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 mo
dule.
// 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 acquisition run is in progress. The function returns the num
ber of words (32-bit) that the external FIFO currently has. If the number of words
is greater than a user-set threshold, function Pixie16ReadDataFromExternalFIFO can
then be used to read the data from the external FIFO. The threshold can be set by t
he user to either minimize reading overhead or to read data out of the FIFO as quic
kly 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 Pixie16CheckExternalFIFOS
tatus is called first to see how many words the external FIFO currently has, then t
his 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 comple
te path)
    unsigned short ModuleNumber,       // the module whose events are to be analyze
d
    unsigned short ChannelNumber,      // the channel whose events are to be analyz
ed
    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 comple
te path)
    unsigned short ModuleNumber,       // the module whose events are to be analyze
d
    unsigned short ChannelNumber,      // the channel whose events are to be analyz
ed
```

```
    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 comple
te path)
    unsigned short ModuleNumber,       // the module whose events are to be analyze
d
    unsigned short ChannelNumber,      // the channel whose events are to be analyz
ed
    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 );
```