Pixie-16

Programmer's Manual

Version 3.06

September 13, 2019

Hardware Revisions: B, C, D, F

XIA LLC
31057 Genstar Rd
Hayward, CA 94544 USA
Email: support@xia.com
Tel: (510) 401-5760; Fax: (510) 401-5761
http://www.xia.com/

Table of Contents

Warranty Statement

XIA LLC warrants that this product will be free from defects in materials and workmanship for a period of one (1) year from the date of shipment. If any such product proves defective during this warranty period, XIA LLC, at its option, will either repair the defective products without charge for parts and labor, or will provide a replacement in exchange for the defective product.

In order to obtain service under this warranty, Customer must notify XIA LLC of the defect before the expiration of the warranty period and make suitable arrangements for the performance of the service.

This warranty shall not apply to any defect, failure or damage caused by improper uses or inadequate care. XIA LLC shall not be obligated to furnish service under this warranty a) to repair damage resulting from attempts by personnel other than XIA LLC representatives to repair or service the product; or b) to repair damage resulting from improper use or connection to incompatible equipment.

THIS WARRANTY IS GIVEN BY XIA LLC WITH RESPECT TO THIS PRODUCT IN LIEU OF ANY OTHER WARRANTIES, EXPRESSED OR IMPLIED. XIA LLC AND ITS VENDORS DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITYOR FITNESS FOR A PARTICULAR PURPOSE. XIA'S RESPONSIBILITY TO REPAIR OR REPLACE DEFECTIVE PRODUCTS IS THE SOLE AND EXCLUSIVE REMEDY PROVIDED TO THE CUSTOMER FOR BREACH OF THIS WARRANTY. XIA LLC AND ITS VENDORS WILL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IRRESPECTIVE OF WHETHER XIA LLC OR THE VENDOR HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Contact Information:

XIA LLC

31057 Genstar Rd.

Hayward, CA 94544 USA

Telephone: +1 (510) 401-5760

Downloads: http://support.xia.com

Customer Support: support@xia.com

XIA LLC License Agreement:

SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Manual Conventions

The following conventions are used throughout this manual.

Convention	Description	Example
»	The » symbol leads you through nested menu items and dialog box options.	The sequence **Run Results»MCA Histogram** directs you to pull down the **Run Results** menu, and select the **MCA Histogram** item.
Bold	Bold text denotes items that you must select or click on in the software, such as menu items, and dialog box options.	...click on the **Startup** tab.
[Bold]	Bold text within [ ] denotes a command button.	**[Start]** indicates the command button labeled Start Run.
`monospace`	Items in this font denote text or characters that you enter from the keyboard, sections of code, file contents, and syntax examples.	`Setup.exe` refers to a file called "setup.exe" on the host computer.
"**window**"	Text in quotation refers to window titles, and quotations from other sources	"**Acquire ADC Traces**" indicates the window accessed via **Settings»Acquire ADC Traces**.
Italics	Italic text denotes a new term being introduced , or simply emphasis	*rise time* refers to the length of the slow filter. ...it is important first to set the energy filter flat top so that it is *at least one unit greater than* the preamplifier risetime...
<Key> <Shift-Alt-Delete> or <Ctrl+D>	Angle brackets denote a key on the keybord (not case sensitive). A hyphen or plus between two or more key names denotes that the keys should be pressed simultaneously (not case sensitive).	<W> indicates the W key <Ctrl+W> represents holding the control key while pressing the W key on the keyboard
Bold italic	Warnings and cautionary text.	***CAUTION:*** *Improper connections or settings can result in damage to system components.*
CAPITALS	CAPITALS denote DSP parameter names	SLOWLEN is the length of the slow energy filter
SMALL CAPS	SMALL CAPS are used for panels/windows/graphs in the GUI.	…go to the READ HISTOGRAMS panel and you see…

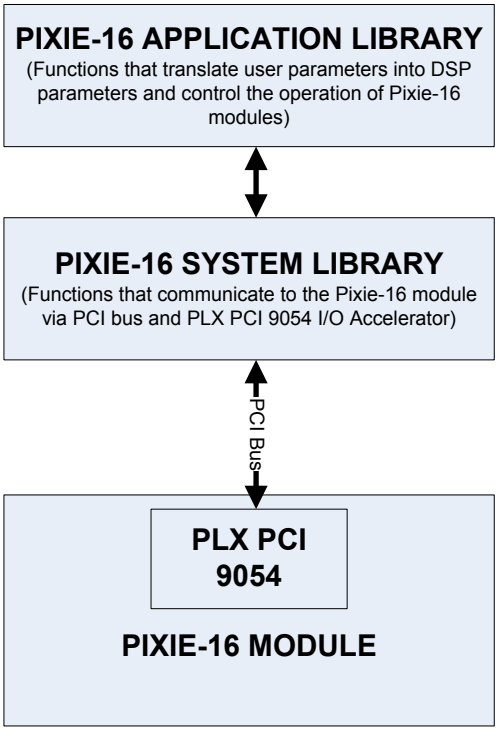
1 Introduction

1.1 About this manual

This manual provides information about the functionality of the Pixie-16 Application Programming Interface (API). The Pixie-16 API consists of a set of C functions for building various data acquisition applications that use Pixie-16 modules from XIA LLC. Users should consult this manual for details of each API function when creating their own data acquisition applications or integrating the Pixie-16 API into their existing data acquisition systems.

1.2 PIXIE-16 API Architecture

The Pixie-16 API consists of two libraries: Application Library and System Library. Each library performs different functionality, is standalone, and can be compiled separately. The following drawing illustrates the basic architecture of the Pixie-16 API.



The Application Library consists of C functions that translate user parameters into DSP parameters and control the operation of Pixie-16 modules. They call functions in the System Library to communicate to the Pixie-16 module. Since the Pixie-16 is an all-digital signal processing module whose operation is controlled by parameters in digital forms (e.g. filter length in multiples of 10 ns ADC sampling rate), there are functions in the Application Library that convert parameters from physical units (e.g. µs, volts, etc.) into digital formats that can be understood by the Pixie-16 modules. Other Application Library functions configure Pixie-16 modules, make MCA or list mode runs and retrieve data from the Pixie-16 modules.

Functions in the System Library communicate directly to the Pixie-16 hardware through the PCI bus (32-bit, 33 MHz) and the PCI 9054 I/O Accelerator from PLX Technology Inc. (PLX Technology, Inc. is now part of Broadcom Inc.) PLX PCI 9054 is a 32-bit PCI bus mastering interface chip that on one side connects to the Pixie-16 FPGAs through the local bus, and on the other side connects to the host computer through the 32-bit PCI bus. PLX provides a Software Development Kit (SDK) for users to develop their own software utilizing PLX API functions. The Pixie-16 System Library is built upon the PLX API functions to communicate to the Pixie-16 modules.

1.3 Windows and Linux Support

The Pixie-16 API supports the operation of the Pixie-16 modules in both Windows and Linux operating systems. Both the System and Application Library incorporate precompiler switches that can be used to switch the compilation of the library codes to be used under either Windows or Linux. For Windows, PLX provides an API DLL file that can be directly called upon. For Linux, PLX provides source codes for its API library that can be compiled under most standard Linux distributions, such as RedHat or Fedora. A user can download either the Windows or Linux SDK package from Broadcom's web site to gain access to PLX's API library source codes:

https://www.broadcom.com/products/pcie-switches-bridges/usb-pci/io-accelerators/pci9054#downloads

(Please search for PCI 9054 at Broadcom's web site if the above link is removed later on).

1.4 Support for Different PIXIE-16 Hardware Revisions

The Pixie-16 was first developed in 2004, and since then there have been a total of five hardware revisions: Rev-A, B, C, D, and F (E was a development revision that was never released). The main difference among these revisions is that Rev. A modules use two SRAM memories to allow alternating data readout between these two SRAM memories (ping-pong memory) while other revisions' modules have an external FIFO memory to stream data. However, Rev-A modules have become obsolete and are no longer supported in this version of the software.

Different Pixie-16 hardware revisions are managed by reading module information that are stored in the module's onboard EEPROM. Before booting a module, Pixie-16 API function *Pixie16ReadModuleInfo* should be called to retrieve the revision, serial number, number of ADC bits, and ADC sampling rate of the module so that appropriate firmware and parameter files can be chosen to successfully boot the module.

2 Getting Started

2.1 Pixie-16 API Source Code

Source code for the Pixie-16 APIs is part of the XIA software releases. Please check

http://support.xia.com/default.asp?W372

for the most recent version. The site also has links to user-developed Pixie-16 software.

2.2 Installation of PLX API Library

The Pixie-16 software is based on the PLX API. Users should first try to download the PLX SDK packages from Broadcom Inc.'s web site. However, XIA will not always update the Pixie-16 API once a new version of the PLX SDK becomes available. This is because a newer version of the PLX SDK sometimes only contains updates for a specific PLX product family, not necessarily the ones used by the Pixie-16s. Consequently, users should always contact XIA for information about the version of the PLX SDK which is being used in the Pixie-16 API.

Once users possess the PLX SDK package, please follow the instructions contained in the SDK to install the package in either Windows or Linux operating systems.

2.3 Library Compilation Tools

The Pixie-16 API libraries can be built using various tools. The following tools were used by XIA to build the libraries. There are many compatible alternative tools available for the various build environments. Customers are free to use their own preferred sets of compatible compilers.

2.3.1 Windows

Microsoft Visual C++ 6.0 was used to compile both the Application and System Library.

There are two static libraries that were needed to link to the Pixie-16 API libraries. One is PlxApi.lib, which is provided by the PLX SDK. The other is WINMM.lib, which is part of the Microsoft Visual Studio SDK package (under \Microsoft Visual Studio\VC98\Lib). If users don't have the Microsoft Visual Studio SDK package installed on their computers, they can download the Microsoft Platform SDK from Microsoft's website and install it on their computers (go to www.microsoft.com and then search "Platform SDK").

Alternatively, GCC can also be used to compile the libraries on Windows.

2.3.2 Linux

GCC was used to build both the Application and System Library in Fedora Linux distribution. Libraries can be compiled as either static or shared libraries.

3 PIXIE-16 API Reference

This section provides the details of all Pixie-16 API functions.

3.1 Overview

In order to better illustrate the usage of the functions in the Pixie-16 API, an overview of the operation of Pixie-16 is given below and the usage of the API functions is mentioned wherever appropriate.

At first the system needs to be initialized. This is a process in which the Pixie-16 modules are made known to the system and are "opened" for communication via the PCI interface. The function `Pixie16InitSystem` is used to achieve this. This has to be done once after the embedded computer, or desktop computer if using a PCI bridge, has booted.

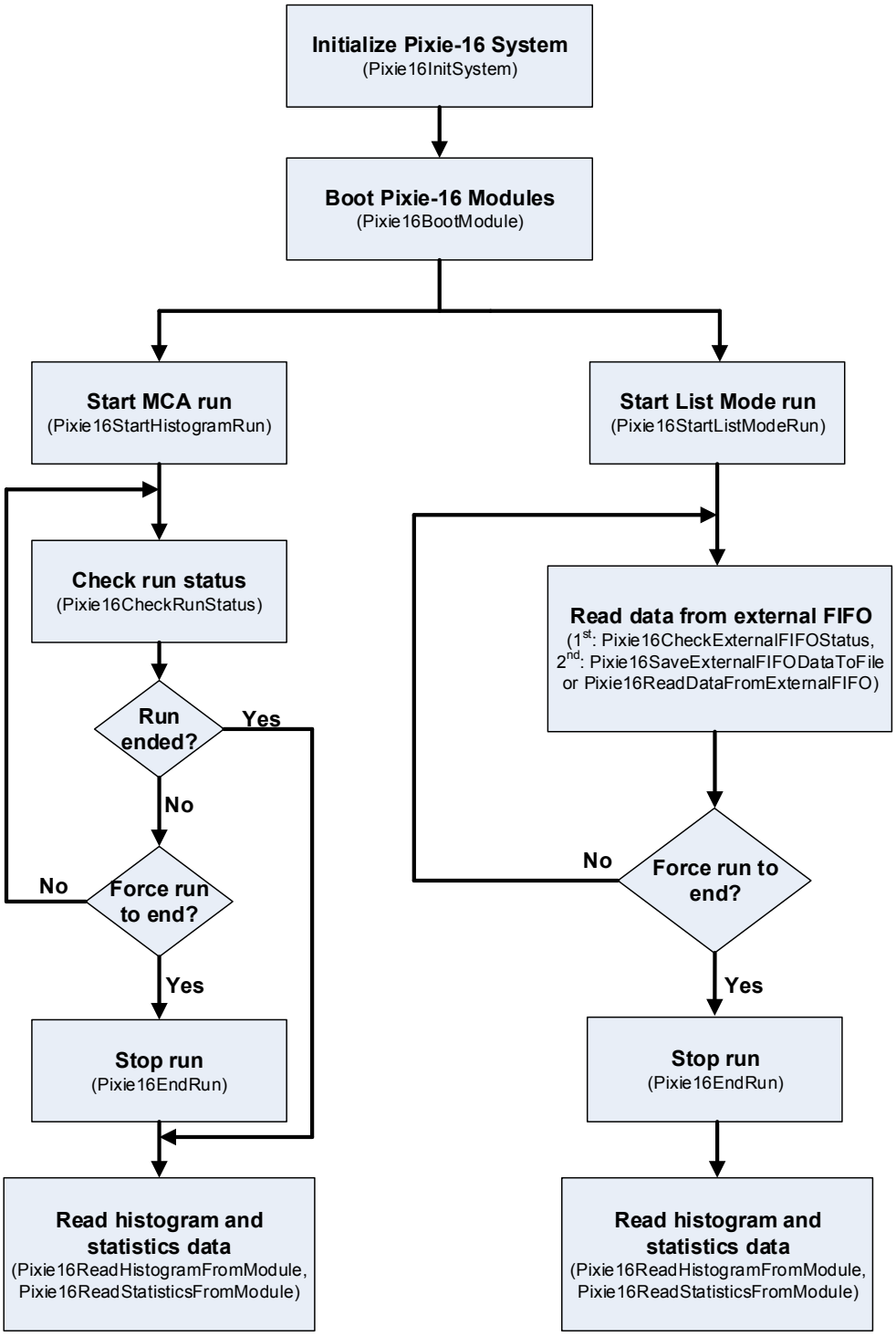
The second step is to boot the Pixie-16 modules. It involves downloading all FPGA configurations and booting the digital signal processor (DSP). It concludes with downloading all DSP parameters (the instrument settings) and commanding the DSP to program the FPGAs and the on-board digital to analog converters (DAC). All this has been encapsulated in a single function `Pixie16BootModule`. However, if the module information is unknown to the user, function `Pixie16ReadModuleInfo` should be called first to retrieve variant information of the module so that appropriate firmware, DSP and configuration parameters files can be chosen for the module prior to calling function `Pixie16BootModule`.

Now, the instrument is ready for data acquisition. An important mode of operation is MCA histogram run which can be used for diagnostic and calibration purposes. The system can be programmed to acquire independent or coincident energy histograms, or spectra, on each channel. The function to start a histogram run is `Pixie16StartHistogramRun`. MCA runs can time out by themselves when the elapsed run time reaches the preset run time, or the user can end them prematurely by using `Pixie16EndRun`, which can also be used to prematurely end list mode runs as well. Afterwards, histogram and statistics data are available for read out by the host computer (histograms can also be read out while the run is in progress) using functions `Pixie16ReadHistogramFromModule` and `Pixie16ReadStatisticsFromModule`. If the MCA run is set to time out by itself, users can call the function `Pixie16CheckRunStatus` to see whether the MCA run is still ongoing or has finished.

Normal operation of the instrument is to start the list mode data acquisition run using the NEW_RUN mode to erase any old histograms and statistics information by using function `Pixie16StartListModeRun`. The host computer will poll the status of the external FIFO by using function `Pixie16CheckExternalFIFOStatus`, and if the status indicates there is data in the FIFO, functions `Pixie16ReadExternalFIFO` or `Pixie16SaveExternalFIFODataToFile` can be used to read data from the external FIFO and save the data to a file on the local hard drives. List mode runs can be stopped by calling function `Pixie16EndRun`. After the list mode run ends, histogram and statistics data are available for read out by the host computer just as in the case of a MCA mode run.

From the statistics data, users can compute useful quantities like real time, total number of processed events, live time, input count rate, and output count rate by using the functions `Pixie16ComputeRealTime`, `Pixie16ComputeProcessedEvents`, `Pixie16ComputeLiveTime`, `Pixie16ComputeInputCountRate`, `Pixie16ComputeOutputCountRate`, respectively.

The following diagram illustrates the sequences for operating the Pixie-16 modules.



3.2 PIXIE-16 API Functions

API Function Name	Description
Pixie16AcquireADCTrace	Acquire ADC traces in single or multiple modules
Pixie16AcquireBaselines	Acquire baselines from a module
Pixie16AdjustOffsets	Adjust DC-offsets in single or multiple modules
Pixie16BLcutFinder	Compute new Baseline Cut values of a module
Pixie16BootModule	Boot modules so that they can be set up for data taking
Pixie16CheckExternalFIFOStatus	Check status of external FIFO of a module
Pixie16CheckRunStatus	Check status of a data acquisition run
Pixie16ComputeFastFiltersOffline	Compute fast filter response for offline analysis
Pixie16ComputeInputCountRate	Compute input count rate of a channel
Pixie16ComputeLiveTime	Compute live time that a channel accumulated in a run
Pixie16ComputeOutputCountRate	Compute output count rate of a channel
Pixie16ComputeProcessedEvents	Compute number of events processed by a module
Pixie16ComputeRealTime	Compute real time that a module accumulated in a run
Pixie16ComputeSlowFiltersOffline	Compute slow filter response for offline analysis
Pixie16ControlTaskRun	Execute special control tasks
Pixie16CopyDSPParameters	Copy DSP parameters from a module to others
Pixie16EMbufferIO	Transfer data between host and DSP external memory
Pixie16EndRun	Stop a data acquisition run
Pixie16ExitSystem	Release user virtual addressees assigned to modules
Pixie16GetEventsInfo	Get detailed events information from a data file
Pixie16GetModuleEvents	Parse a list mode data file to get events information
Pixie16IMbufferIO	Transfer data between host and DSP internal memory
Pixie16InitSystem	Configure modules for communication in PXI chassis
Pixie16LoadDSPParametersFromFile	Load DSP parameters to modules from a file
Pixie16ProgramFippi	Program on-board signal processing FPGAs
Pixie16RampOffsetDACs	Ramp Offset DACs of a module and record the baselines
Pixie16ReadCSR	Read Control & Status Register value from a module
Pixie16ReadDataFromExternalFIFO	Read data from external FIFO of a module
Pixie16ReadHistogramFromFile	Read histogram data from a histogram data file
Pixie16ReadHistogramFromModule	Read histogram data from a module
Pixie16ReadListModeTrace	Read trace data from a list mode data file
Pixie16ReadModuleInfo	Read information about a module stored in the EEPROM
Pixie16ReadSglChanADCTrace	Read ADC trace data from a channel in a module
Pixie16ReadSglChanBaselines	Read baselines from a channel in a module
Pixie16ReadSglChanPar	Read a CHANNEL level parameter from a module
Pixie16ReadSglModPar	Read a MODULE level parameter from a module
Pixie16ReadStatisticsFromModule	Read run statistics data from a module
Pixie16RegisterIO	Read from or write to registers on a module
Pixie16SaveDSPParametersToFile	Read DSP parameters from modules and save to a file
Pixie16SaveExternalFIFODataToFile	Read data from external FIFO and save to a file
Pixie16SaveHistogramToFile	Read histogram data from a module and save to a file
Pixie16SetDACs	Program on-board DACs
Pixie16StartHistogramRun	Start a MCA histogram mode data acquisition run
Pixie16StartListModeRun	Start a list mode data acquisition run
Pixie16TauFinder	Find the exponential decay time of a channel
Pixie16WriteCSR	Write to Control & Status Register in a module
Pixie16WriteSglChanPar	Write a CHANNEL level parameter to a module
Pixie16WriteSglModPar	Write a MODULE level parameter to a module

3.2.1 Pixie16AcquireADCTrace

Syntax

```
int Pixie16AcquireADCTrace (
  unsigned short ModNum )       // module number
```

Description

Use this function to acquire ADC traces from Pixie-16 modules. Specify the module using ModNum which starts counting at 0. If ModNum is set to be less than the total number of modules in the system, only the module specified by ModNum will have its ADC traces acquired. But if ModNum is equal to the total number of modules in the system, then all modules in the system will have their ADC traces acquired.

After the successful return of this function, the DSP's internal memory will be filled with ADC trace data. A user's application software should then call the function `Pixie16ReadSglChanADCTrace` to read the ADC trace data out to the host computer, channel by channel.

Return values

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

Usage example

```
unsigned short ModNum;
int retval;

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

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

3.2.2 Pixie16AcquireBaselines

Syntax

```
int Pixie16AcquireBaselines (
   unsigned short ModNum )        // module number
```

Description

Use this function to acquire baselines from Pixie-16 modules. Specify the module using ModNum which starts counting at 0. If ModNum is set to be less than the total number of modules in the system, only the module specified by ModNum will have its baselines acquired. But if ModNum is set to be equal to the total number of modules in the system, then all modules in the system will have their baselines acquired.

After the successful return of this function, the DSP's internal memory will be filled with baselines data. Users should then call the function `Pixie16ReadSglChanBaselines` to read the baselines data out to the host computer, channel by channel.

Return values

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

Usage example

```
unsigned short ModNum;
int retval;

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

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

3.2.3 Pixie16AdjustOffsets

Syntax

```
int Pixie16AdjustOffsets (
  unsigned short ModNum ) // module number
```

Description

Use this function to adjust the DC-offsets of Pixie-16 modules. Specify the module using ModNum which starts counting at 0. If ModNum is set to be less than the total number of modules in the system, only the module specified by ModNum will have its DC-offsets adjusted. But if ModNum is set to be equal to the total number of modules in the system, then all modules in the system will have their DC-offsets adjusted.

After the DC-offset levels have been adjusted, the baseline level of the digitized input signals will be determined by the DSP parameter *BaselinePercent*. For instance, if *BaselinePercent* is set to 10(%), the baseline level of the input signals will be ~ 409 on the 12-bit ADC scale (minimum: 0; maximum: 4095).

The main purpose of this function is to ensure the input signals fall within the voltage range of the ADCs so that all input signals can be digitized by the ADCs properly.

Return values

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

Usage example

```
int retval;
unsigned short ModNum;

// set to the first module
ModNum = 0;

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

3.2.4 Pixie16BLcutFinder

Syntax

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

Description

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

ModNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.

Return values

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

Usage example

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

// set to the first module
ModNum = 0;

// set to the first channel
ChanNum = 0;

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

3.2.5 Pixie16BootModule

Syntax

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

Description

Use this function to boot Pixie-16 modules so that they can be set up for data taking. The function downloads to the Pixie-16 modules the communication (or system) FPGA configurations, signal processing FPGA configurations, trigger FPGA configurations (Revision A modules only), executable code for the digital signal processor (DSP), and DSP parameters.

The FPGA configurations consist of a fixed number of words depending on the hardware mounted on the modules; the DSP codes have a length which depends on the actual compiled code; and the set of DSP parameters always consists of 1280 32-bit words for each module. The host software has to make the names of those boot data files on the hard disk available to the boot function.

ModNum is the module number which starts counting at 0. If ModNum is set to be less than the total number of modules in the system, only the module specified by ModNum will be booted. But if ModNum is equal to the total number of modules in the system, e.g. there are 5 modules in the chassis and ModNum = 5, then all modules in the system will be booted.

The boot pattern is a bit mask (shown below) indicating which on-board chip will be booted. Under normal circumstances, all on-board chips should be booted, i.e. the boot pattern would be 0x7F. For Rev-B, C, D, F modules, bit 1, i.e., "Boot trigger FPGA", will be ignored even if that bit is set to 1.

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

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid Pixie-16 module number	Correct ModNum
-2	Size of ComFPGAConfigFile is invalid	Correct ComFPGAConfigFile
-3	Failed to boot Communication FPGA	Check log file (Pixie16msg.txt)
-4	Failed to allocate memory to store data in ComFPGAConfigFile	Close other programs or reboot the computer
-5	Failed to open ComFPGAConfigFile	Correct ComFPGAConfigFile
-10	Size of SPFPGAConfigFile is invalid	Correct SPFPGAConfigFile
-11	Failed to boot signal processing FPGA	Check log file (Pixie16msg.txt)
-12	Failed to allocate memory to store data in SPFPGAConfigFile	Close other programs or reboot the computer
-13	Failed to open SPFPGAConfigFile	Correct SPFPGAConfigFile
-14	Failed to boot DSP	Check log file (Pixie16msg.txt)
-15	Failed to allocate memory to store DSP executable code	Close other programs or reboot the computer
-16	Failed to open DSPCodeFile	Correct DSPCodeFile
-17	Size of DSPParFile is invalid	Correct DSPParFile
-18	Failed to open DSPParFile	Correct DSPParFile
-19	Can't initialize DSP variable indices	Correct DSPVarFile
-20	Can't copy DSP variable indices	Check log file (Pixie16msg.txt)
-21	Failed to program Fippi in a module	Check log file (Pixie16msg.txt)
-22	Failed to set DACs in a module	Check log file (Pixie16msg.txt)
-23	Failed to start RESET_ADC run in a module	Check log file (Pixie16msg.txt)
-24	RESET_ADC run timed out in a module	Check log file (Pixie16msg.txt)

Usage example

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

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

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

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

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

3.2.6 Pixie16CheckExternalFIFOStatus

Syntax

```
int Pixie16CheckExternalFIFOStatus (
  unsigned int *nFIFOWords,     // returned number of words in the FIFO
  unsigned short ModNum )       // module number
```

Description

Use this function to check the status of the external FIFO of a Pixie-16 module while a list mode data acquisition run is in progress. The function returns the number of words (32-bit) that the external FIFO currently has. If the number of words is greater than a user-set threshold, function `Pixie16ReadDataFromExternalFIFO` can then be used to read the data from the external FIFO. The threshold can be set by the user to either minimize reading overhead or to read data out of the FIFO as quickly as possible. The Pixie-16 API (pixie16app_defs.h) has defined a threshold with value of 1024 for external FIFO read out (EXTFIFO_READ_THRESH).

*nFIFOWords returns the number of 32-bit words that the external FIFO currently has.

ModNum is the module number which starts counting at 0.

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid Pixie-16 module number	Correct ModNum

Usage example

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

ModNum = 0;                 // the first module

retval = Pixie16CheckExternalFIFOStatus (&nFIFOWords, ModNum);
```

3.2.7 Pixie16CheckRunStatus

Syntax

```
int Pixie16CheckRunStatus (
  unsigned short ModNum )        // module number
```

Description

Use this function to check the run status of a Pixie-16 module while a list mode data acquisition run is in progress. If the run is still in progress continue polling.

If the return code of this function indicates the run has finished, there might still be some data in the external FIFO (Rev-B, C, D, F modules) that need to be read out to the host computer. In addition, final run statistics and histogram data are available for reading out too.

In MCA histogram run mode, this function can also be called to check if the run is still in progress even though it is normally self-terminating.

ModNum is the module number which starts counting at 0.

Return values

Value	Description	Error Handling
0	No run is in progress	None
1	Run is still in progress	None
-1	Invalid Pixie-16 module number	Correct ModNum

Usage example

```
int retval;
unsigned short ModNum;

ModNum = 0;                 // the first module

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

3.2.8 Pixie16ComputeFastFiltersOffline

Syntax

```
int Pixie16ComputeFastFiltersOffline (
   char *FileName,  // the list mode data file name (with complete path)
   unsigned short ModuleNumber,  // the module to be analyzed
   unsigned short ChannelNumber, // the channel to be analyzed
   unsigned int   FileLocation,  // the location of the trace in the file
   unsigned short RcdTraceLength,// recorded trace length
   unsigned short *RcdTrace,     // recorded trace
   double         *fastfilter,   // fast filter response
   double         *cfd )         // cfd response
```

Description

Use this function to compute fast filter responses of each event in the list mode data file for offline analysis. The algorithm implemented in this offline analysis function is equivalent to the one implemented in the Pixie-16 hardware. So this function can be used to analyze how the leading edge fast trigger filter and the CFD filter implemented in the hardware respond to the shape of recorded traces. By analyzing the response of these filters, it is possible to optimize the performance of the leading edge trigger or CFD trigger by adjusting the fast filter and CFD parameters offline. Such optimized parameters can then be saved to a settings file to be used for online data acquisition.

Return values

Value	Description	Error Handling
0	Success	None
-1	Null pointer *RcdTrace	Make sure *RcdTrace has been allocated enough memory (maximum possible length is 32768, but can be shorter to match the actual trace length)
-2	Null pointer *fastfilter	Make sure *fastfilter has been allocated enough memory (maximum possible length is 32768, but can be shorter to match the actual trace length)
-3	Null pointer *cfd	Make sure *cfd has been allocated enough memory (maximum possible length is 32768, but can be shorter to match the actual trace length)
-4	Target module number is invalid	Correct ModuleNumber
-5	Trace length is too short	The length of recorded trace is too short for the offline analysis using the fast filter length (fast filter rise time and flat top). Either a different settings file with shorter fast filter length has to be used, or new traces with longer trace length have to be acquired
-6	Failed to open list mode data file	Check the list mode file name and its path are correct

Usage example

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

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

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


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

3.2.9 Pixie16ComputeInputCountRate

Syntax

```
double Pixie16ComputeInputCountRate (
  unsigned int *Statistics,                 // run statistics data
  unsigned short ModNum,                    // module number
  unsigned short ChanNum )                  // channel number
```

Description

Use this function to calculate the input count rate on one channel of a Pixie-16 module. This function does not communicate with Pixie-16 modules. Before calling this function, another function, `Pixie16ReadStatisticsFromModule`, should be called to read statistics data from the module.

*Statistics is a pointer to an array whose size is exactly 448 unsigned integer words (32-bit). The *Statistics array is filled with data from a Pixie-16 module after calling function `Pixie16ReadStatisticsFromModule`. ModNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.

Return values

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

Usage example

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

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

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

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

3.2.10 Pixie16ComputeLiveTime

Syntax

```
double Pixie16ComputeLiveTime (
  unsigned int *Statistics,                 // run statistics data
  unsigned short ModNum,                    // module number
  unsigned short ChanNum )                  // channel number
```

Description

Use this function to calculate the live time that one channel of a Pixie-16 module has spent on data acquisition. This function does not communicate with Pixie-16 modules. Before calling this function, another function, `Pixie16ReadStatisticsFromModule`, should be called to read statistics data from the module.

*Statistics is a pointer to an array whose size is exactly 448 unsigned integer words (32-bit). The *Statistics array is filled with data from a Pixie-16 module after calling function `Pixie16ReadStatisticsFromModule`. ModNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.

Return values

The return value is the live time in seconds.

Usage example

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

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

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

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

3.2.11 Pixie16ComputeOutputCountRate

Syntax

```
double Pixie16ComputeOutputCountRate (
   unsigned int *Statistics,                  // run statistics data
   unsigned short ModNum,                     // module number
   unsigned short ChanNum )                   // channel number
```

Description

Use this function to calculate the output count rate on one channel of a Pixie-16 module. This function does not communicate with Pixie-16 modules. Before calling this function, another function, `Pixie16ReadStatisticsFromModule`, should be called to read statistics data from the module.

*Statistics is a pointer to an array whose size is exactly 448 unsigned integer words (32-bit). The *Statistics array is filled with data from a Pixie-16 module after calling function `Pixie16ReadStatisticsFromModule`. ModNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.

Return values

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

Usage example

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

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

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

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

3.2.12 Pixie16ComputeProcessedEvents

Syntax

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

Description

Use this function to calculate the number of events that have been processed by a Pixie-16 module during a data acquisition run. **This function is only used by Rev-A modules.** This function does not communicate with Pixie-16 modules. Before calling this function, another function, `Pixie16ReadStatisticsFromModule`, should be called to read statistics data from the module first.

*Statistics is a pointer to an array whose size is exactly 448 unsigned integer words (32-bit). The *Statistics array is filled with data from a Pixie-16 module after calling function `Pixie16ReadStatisticsFromModule`. ModNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.

Return values

The return value is the number of processed events.

Usage example

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

ModNum = 0;           // the first module

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

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

3.2.13 Pixie16ComputeRealTime

Syntax

```
double Pixie16ComputeRealTime (
  unsigned int *Statistics,                 // run statistics data
  unsigned short ModNum )                   // module number
```

Description

Use this function to calculate the real time that a Pixie-16 module has spent on data acquisition. This function does not communicate with Pixie-16 modules. Before calling this function, another function, `Pixie16ReadStatisticsFromModule`, should be called to read statistics data from the module.

*Statistics is a pointer to an array whose size is exactly 448 unsigned integer words (32-bit). The *Statistics array is filled with data from a Pixie-16 module after calling function `Pixie16ReadStatisticsFromModule`. ModNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.

Return values

The return value is the real time in seconds.

Usage example

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

ModNum = 0;          // the first module

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

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

3.2.14 Pixie16ComputeSlowFiltersOffline

Syntax

```
int Pixie16ComputeSlowFiltersOffline (
   char *FileName,  // the list mode data file name (with complete path)
   unsigned short ModuleNumber,  // the module to be analyzed
   unsigned short ChannelNumber, // the channel to be analyzed
   unsigned int   FileLocation,  // the location of the trace in the file
   unsigned short RcdTraceLength,// recorded trace length
   unsigned short *RcdTrace,     // recorded trace
   double         *slowfilter)   // slow filter response
```

Description

Use this function to compute slow filter responses of each event in the list mode data file for offline analysis. The algorithm implemented in this offline analysis function is equivalent to the one implemented in the Pixie-16 hardware. So this function can be used to analyze how the slow filter implemented in the hardware for computing event energy responds to the shape of recorded traces. By analyzing the responses of these filters, it is possible to optimize the performance of the slow filter by adjusting its parameters offline. Such optimized parameters can then be saved to a settings file to be used for online data acquisition.

Return values

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

Usage example

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

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

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


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

3.2.15 Pixie16ControlTaskRun

Syntax

```
int Pixie16ControlTaskRun (
   unsigned short ModNum,              // module number
   unsigned short ControlTask,         // control task run type
   unsigned int Max_Poll )             // Timeout control in unit of ms
```

Description

Use this function to execute special control tasks. This may include programming the Fippi or setting the DACs after downloading DSP parameters. See section 4.4 for a list of control tasks.

ModNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.

Return values

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

Usage example

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

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

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

3.2.16 Pixie16CopyDSPParameters

Syntax

```
int Pixie16CopyDSPParameters (
   unsigned short BitMask,              // copy items bit mask
   unsigned short SourceModule,         // source module
   unsigned short SourceChannel,        // source channel
   unsigned short *DestinationMask )    // destination bit mask
```

Description

Use this function to copy DSP parameters from one module to the others that are installed in the system.

BitMask is bit pattern which designates which items should be copied from the source module to the destination module(s).

Bit	Item
0	Filter (energy and trigger)
1	Analog signal conditioning (polarity, dc-offset, gain/attenuation)
2	Histogram control (minimum energy, binning factor)
3	Decay time
4	Pulse shape analysis (trace length and trace delay)
5	Baseline control (baseline cut, baseline percentage)
7	Channel CSRA register (good channel, trigger enabled, etc.)
8	CFD trigger (CFD delay, scaling factor)
9	Trigger stretch lengths (veto, external trigger, etc.)
10	FIFO delays (analog input delay, fast trigger output delay, etc.)
11	Multiplicity (bit masks, thresholds, etc.)
12	QDC (QDC sum lengths)

SourceModule and SourceChannel is the module and channel number of the source of the DSP parameters which are to be copied to other modules and channels.

DestinationMask is an array which indicates the channel and module whose settings will be copied from the source channel and module. For instance, if there are 5 modules (total 80 channels) in the system, DestinationMask would be defined as DestinationMask[80], where DestinationMask[0] to DestinationMask[15] would select channel 0 to 15 of module 0, DestinationMask[16] to DestinationMask[31] would select channel 0 to 15 of module 1, and so on. If a given channel *i* is to be copied, then DestinationMask[*i*] should be set to **1**, otherwise, it should be set to **0**.

Return values

Value	Description	Error Handling
0	Success	None
-1	Failed to program Fippi in a module	Reboot the modules
-2	Failed to set DACs in a module	Reboot the modules

Usage example

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

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

// assume there are 6 Pixie-16 modules in the system
for(k = 0; k < (6 * 16); k ++)
{
   DestinationMask[k] = 1; // copy to all channels of all modules
}

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

3.2.17 Pixie16EMbufferIO

Syntax

```
int Pixie16EMbufferIO (
  unsigned int *Buffer,    // buffer data
  unsigned int NumWords,   // number of buffer words to read or write
  unsigned int Address,    // buffer address
  unsigned short Direction,     // I/O direction
  unsigned short ModNum ) // module number
```

Description

Use this function to directly read data from or write data to the on-board external memory of a Pixie-16 module. The valid memory address is from 0x0 to 0x7FFFF (32-bit wide). Use Direction = 1 for read and Direction = 0 for write.

The external memory is used to store the histogram data accumulated for each of the 16 channels of a Pixie-16 module. Each channel has a fixed histogram length of 32768 words (32-bit wide), and the placement of the histogram data in the memory is in the same order of the channel number, i.e. channel 0 occupies memory address 0x0 to 0x7FFF, channel 1 occupies 0x8000 to 0xFFFF, and so on.

NOTE: another function `Pixie16ReadHistogramFromModule` can also be used to read out the histograms except that it needs to be called channel by channel.

ModNum is the module number which starts counting at 0.

Return values

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

Usage example

```
int retval;
unsigned short Direction, ModNum;
unsigned int MCAData[32768], NumWords, Address;

NumWords = 32768;          // to read out the MCA data from channel 0
ModNum = 0;                // the first module in the system
Address = 0x0;             // the starting memory address
Direction = 1;             // I/O direction is read
```

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

3.2.18 Pixie16EndRun

Syntax

```
int Pixie16EndRun (
  unsigned short ModNum )              // module number
```

Description

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

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid Pixie-16 module number	Correct ModNum
-2	Failed to stop the run	Reboot the module

Usage example

```
int retval;
unsigned short ModNum;

ModNum = 0;                 // the first module

retval = Pixie16EndRun (ModNum);
if(retval < 0)
{
        // error handling
}
```

3.2.19 Pixie16ExitSystem

Syntax

```
int Pixie16ExitSystem (
  unsigned short ModNum ) // Pixie-16 module number
```

Description

Use this function to release the user virtual addressees that are assigned to Pixie-16 modules when these modules are initialized by function `Pixie16InitSystem`. This function should be called before a user's application exits.

If ModNum is set to less than the total number of modules in the system, only the module specified by ModNum will be closed. But if ModNum is equal to the total number of modules in the system, e.g. there are 5 modules in the chassis and ModNum = 5, then all modules in the system will be closed altogether. Note that the modules are counted starting at 0.

Return values

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

Usage example

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

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

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

3.2.20 Pixie16GetEventsInfo

Syntax

```
int Pixie16GetEventsInfo (
   char *FileName,   // the list mode data file name (with complete path)
   unsigned int *EventsInformation,    // to hold event information
   unsigned short ModuleNumber )       // module to get events from
```

Description

Use this function to retrieve the detailed information (except waveforms) of each event in the list mode data file for the designated module. ModNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.

Before calling this function to get the individual events information, another function `Pixie16GetModuleEvents` should be called first to determine the number of events that have been recorded for each module. If the number of events for a given module is nEvents, a memory block *EventInformation should be allocated with a length of (nEvents*68):

EventInformation = (unsigned int *)malloc(sizeof(unsigned int) * nEvents * 68);

where 68 is the length of the information records of each event (energy, timestamps, etc.) and has the following structure.

Index	Value
EventInformation [0]	Event number
EventInformation [1]	Channel number
EventInformation [2]	Slot number
EventInformation [3]	Crate number
EventInformation [4]	Header length
EventInformation [5]	Event length
EventInformation [6]	Finish code
EventInformation [7]	Event timestamp (lower 32-bit)
EventInformation [8]	Event timestamp (upper 16-bit)
EventInformation [9]	Event energy
EventInformation [10]	Trace length
EventInformation [11]	Trace location
EventInformation [67:12]	Not used

Return values

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

Usage example

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

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

ModuleNumber = 0;                // the first module
EventInformation = (unsigned int *)malloc(sizeof(unsigned int) *
ModuleEvents[ModuleNumber] * 68);

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

3.2.21 Pixie16GetModuleEvents

Syntax

```
int Pixie16GetModuleEvents (
  char *FileName,   // the list mode data file name (with complete path)
  unsigned int *ModuleEvents )  // receives number of events for modules
```

Description

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.

Return values

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

Usage example

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

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

3.2.22 Pixie16IMbufferIO

Syntax

```
int Pixie16IMbufferIO (
   unsigned int *Buffer,          // buffer data
   unsigned int NumWords,         // number of buffer words to transfer
   unsigned int Address,          // buffer address
   unsigned short Direction,      // I/O direction
   unsigned short ModNum )        // module number
```

Description

Use this function to directly transfer data between the host and the DSP internal memory of a Pixie-16 module. ModNum is the module number which starts counting at 0.

The DSP internal memory is split into two blocks with address range 0x40000 to 0x4FFFF for the first block and address range 0x50000 to 0x5FFFF for the second block. Within the first block, address range 0x40000 to 0x49FFF is reserved for program memory and shouldn't be accessed directly by the host computer. Address range 0x4A000 to 0x4A4FF is used by the DSP I/O parameters which are stored in the configuration files (.set files) in the host. Within this range, 0x4A000 to 0x4A33F can be both read and written, but 0x4A340 to 0x4A4FF can only be read but not written. The remaining address range (0x4A500 to 4FFFF) in the first block and the entire second block (0x50000 to 0x5FFFF) should only be read but not written by the host. Use Direction = 1 for read and Direction = 0 for write.

Return values

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

Usage example

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

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

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

3.2.23 Pixie16InitSystem

Syntax

```
int Pixie16InitSystem (
   unsigned short NumModules,     // total number of Pixie-16 modules
   unsigned short *PXISlotMap,    // an array containing the slot number of
                                  // each Pixie-16 module
   unsigned short OfflineMode )   // specify if using offline mode
```

Description

Use this function to configure the Pixie-16 modules in the PXI chassis.

NumModules is the total number of Pixie-16 modules installed in the system. PXISlotMap is the pointer to an array that must have at least as many entries as there are Pixie-16 modules in the chassis.

PXISlotMap serves as a simple mapping of the logical module number and the physical slot number that the modules reside in. The logical module number runs from 0. For instance, in a system with 5 Pixie-16 modules, these 5 modules may occupy slots 3 through 7. The user must fill PXISlotMap as follows: PXISlotMap = {3, 4, 5, 6, 7 ...} since module number 0 resides in slot number 3, etc. To find out in which slot a module is located, any piece of subsequent code can use the expression PXISlotMap[ModNum], where ModNum is the logic module number.

OfflineMode is used to indicate to the API whether the system is running in OFFLINE mode (1) or ONLINE mode (0). OFFLINE mode is useful for situations where no Pixie-16 modules are present but users can still test their calls to the API functions in their application software.

This function must be called as the first step in the boot process. It makes the modules known to the system and "opens" each module for communication.

The function relies on an initialization file (pxisys.ini) that contains information about the Host PC's PCI buses, including the slot enumeration scheme. XIA's software distribution normally puts this file under the same folder as Pixie-16 software installation folder. However, the user has the flexibility of putting it in other folders by simply changing the definition of the string *PCISysIniFile* in the header part of the file pixie16sys.c.

Return values

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

Usage example

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

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

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

// running in online mode
OfflineMode = 0;

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

3.2.24 Pixie16LoadDSPParametersFromFile

Syntax

```
int Pixie16LoadDSPParametersFromFile (
  char *FileName )  // DSP parameters file name (with complete path)
```

Description

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

Return values

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

Usage example

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

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

3.2.25 Pixie16ProgramFippi

Syntax

```
int Pixie16ProgramFippi (
  unsigned short ModNum)        // module number
```

Description

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

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid Pixie-16 module number	Correct ModNum
-2	Failed to start the PROGRAM_FIPPI run	Reboot the module
-3	PROGRAM_FIPPI run timed out	Reboot the module

Usage example

```
int retval;
unsigned short ModNum;

ModNum = 0;                // the first module

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

3.2.26 Pixie16RampOffsetDACs (deprecated)

Syntax

```
int Pixie16RampOffsetDACs (
   double *DCValues,              // returned DC offset values
   unsigned short NumDCVals,      // number of DC values to read
   unsigned short ModNum )        // module number
```

Description

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 and stores it in the DSP memory. After the control task is finished, the stored baseline values are read out to the host computer and saved to a binary file called "rampdacs.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 `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 RAMP_OFFSETDACS control task run. Note that the modules are counted starting at 0.

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid Pixie-16 module number	Correct ModNum
-2	Null pointer *DCValues	Correct *DCValues
-3	Requested number of DC values exceeded the limit	Reduce NumDCVals
-4	Failed to start the RAMP_OFFSETDACS run	Reboot the module
-5	RAMP_OFFSETDACS run timed out	Reboot the module
-6	Failed to read offset DAC values	Reboot the module

Usage example

```
int retval;
unsigned short ModNum;

ModNum = 0;                 // the first module

retval = Pixie16RampOffsetDACs (ModNum);
```

```
if(retval < 0)
{
        // error handling
}
```

3.2.27 Pixie16ReadCSR

Syntax

```
int Pixie16ReadCSR (
  unsigned short ModNum,        // module number
  unsigned int *CSR )           // returned CSR value
```

Description

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

See section 5.1 for CSR bit definitions

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid Pixie-16 module number	Correct ModNum

Usage example

```
unsigned short ModNum;
unsigned int CSR;

ModNum = 0;                 // the first module

Pixie16ReadCSR (ModNum, &CSR);
```

3.2.28 Pixie16ReadDataFromExternalFIFO

Syntax

```
int Pixie16ReadDataFromExternalFIFO (
  unsigned int *ExtFIFO_Data,   // to receive the external FIFO data
  unsigned int nFIFOWords,      // number of words to read from FIFO
  unsigned short ModNum )       // module number
```

Description

Use this function to read data from the external FIFO of a module.

This function reads list mode data from the external FIFO of a Pixie-16 module. The data are 32-bit unsigned integers. Normally, function `Pixie16CheckExternalFIFOStatus` is called first to see how many words the external FIFO currently has, and then this function is called to read the data from the FIFO. ModNum is the module number which starts counting at 0.

Return values

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

Usage example

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

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

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

3.2.29 Pixie16ReadHistogramFromFile

Syntax

```
int Pixie16ReadHistogramFromFile (
   char *FileName,          // histogram file name (with complete path)
   unsigned int *Histogram,      // histogram data
   unsigned int NumWords,        // number of words to be read out
   unsigned short ModNum,        // module number
   unsigned short ChanNum)       // channel number
```

Description

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.

Return values

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

Usage example

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

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

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

3.2.30 Pixie16ReadHistogramFromModule

Syntax

```
int Pixie16ReadHistogramFromModule (
   unsigned int *Histogram,      // histogram data
   unsigned int NumWords,        // number of words to be read out
   unsigned short ModNum,        // module number
   unsigned short ChanNum)       // channel number
```

Description

Use this function to read out the histogram data from a Pixie-16 module's histogram memory. Before calling this function, the host code should allocate appropriate amount of memory to store the histogram data. The default histogram length is 32768. Histogram data are 32-bit unsigned integers.

Specify the module using ModNum and the channel on the module using ChanNum. Note that both the modules and channels are counted starting at 0.

Return values

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

Usage example

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

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

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

3.2.31 Pixie16ReadListModeTrace

Syntax

```
int Pixie16ReadListModeTrace (
   char *FileName,                // list mode data file name
   unsigned short *Trace_Data,    // list mode trace data (16-bit words)
   unsigned short NumWords,       // number of 16-bit words to be read out
   unsigned int FileLocation)     // the location of the trace in the file
```

Description

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.

Return values

Value	Description	Error Handling
0	Success	None
-1	Failed to open list mode data file	Correct file name and path

Usage example

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

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

ModuleNumber = 0;                // the first module

EventInformation = (unsigned int *)malloc(sizeof(unsigned int) *
ModuleEvents[ModuleNumber]*68);

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

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


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

```
FileLocation = EventInformation[EventNumber*68 + 11];

Trace_Data= (unsigned short *)malloc(sizeof(unsigned short) * NumWords);

retval =
  Pixie16ReadListModeTrace (FileName,Trace_Data,NumWords,FileLocation);
if(retval < 0)
{
               // error handling
}
```

3.2.32 Pixie16ReadModuleInfo

Syntax

```
int Pixie16ReadModuleInfo (
   unsigned short ModNum,         // module number
   unsigned short *ModRev,        // returned module revision
   unsigned int   *ModSerNum,     // returned module serial number
   unsigned short *ModADCBits,    // returned module ADC bits
   unsigned short *ModADCMSPS )   // returned module ADC sampling rate
```

Description

Use this function to read information stored on each module, including its revision, serial number, ADC bits and sampling rate. This should be done after initializing the PCI communication. Information from the module can be used to select the appropriate firmware, DSP, and configuration parameters files before booting the module. ModNum is the module number which starts counting at 0.

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid Pixie module number	Correct file name and path
-2	Failed to read from I2C serial EEPROM	Check error message log file Pixie16msg.txt

Usage example

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

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

3.2.33 Pixie16ReadSglChanADCTrace

Syntax

```
int Pixie16ReadSglChanADCTrace (
   unsigned short *Trace_Buffer, // trace data
   unsigned int Trace_Length,    // number of trace data words to read
   unsigned short ModNum,        // module number
   unsigned short ChanNum )      // channel number
```

Description

Use this function to read ADC trace data from a Pixie-16 module. Before calling this function, another function `Pixie16AcquireADCTrace` should be called to fill the DSP internal memory first. Also, the host code should allocate appropriate amount of memory to store the trace data. The ADC trace data length for each channel is 8192. Since the trace data are 16-bit unsigned integers (for hardware variants with less than 16-bit ADCs only the lower 12-bit or 14-bit contain real data), two consecutive 16-bit words are packed into one 32-bit word in the DSP internal memory. So for each channel, 4096 32-bit words are read out first from the DSP, and then each 32-bit word is unpacked to form two 16-bit words.

Specify the module using ModNum and the channel on the module using ChanNum. Note that both the modules and channels are counted starting at 0.

Return values

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

Usage example

```
unsigned short NumWords, ModNum, ChanNum;
int retval;
unsigned short ADCTrace[8192];

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

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

// acquire the trace
```

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

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

3.2.34 Pixie16ReadSglChanBaselines

Syntax

```
int Pixie16ReadSglChanBaselines (
   double *Baselines,            // returned baseline values
   double *TimeStamps,           // timestamps for each baseline value
   unsigned short NumBases,      // number of baseline data words to read
   unsigned short ModNum,        // module number
   unsigned short ChanNum )      // channel number
```

Description

Use this function to read baseline data from a Pixie-16 module. Before calling this function, another function `Pixie16AcquireBaselines` should be called to fill the DSP internal memory first. Also, the host code should allocate appropriate amount of memory to store the baseline data. The number of baselines for each channel is 3640. In the DSP internal memory, each baseline is a 32-bit IEEE floating point number. After being read out to the host, this function will convert each baseline data to a decimal number. In addition to baseline values, timestamps corresponding to each baseline are also returned after this function call.

Specify the module using ModNum and the channel on the module using ChanNum. Note that the modules and channels are counted starting at 0.

Return values

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

Usage example

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

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

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

// acquire the baselines
retval = Pixie16AcquireBaselines (ModNum);
```

```
if(retval < 0)
{
        // error handling
}

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

3.2.35 Pixie16ReadSglChanPar

Syntax

```
int Pixie16ReadSglChanPar (
   char *ChanParName,                  // channel parameter name
   double *ChanParData,                // channel parameter value
   unsigned short ModNum,          // channel number
   unsigned short ChanNum )            // module number
```

Description

Use this function to read a channel parameter from a Pixie-16 module. ModNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0.The supported channel parameters are listed below.

Channel Parameters (Input)	Unit	Type	Corresponding DSP Variables
TRIGGER_RISETIME	µs	user set	FASTLENGTH
TRIGGER_FLATTOP	µs	user set	FASTGAP
TRIGGER_THRESHOLD	ADC units	user set	FASTTHRESH
ENERGY_RISETIME	µs	user set	SLOWLENGTH
ENERGY_FLATTOP	µs	user set	SLOWGAP
TAU	µs	user set	PREAMPTAU
TRACE_LENGTH	µs	user set	TRACELENGTH
TRACE_DELAY	µs	user set	TRIGGERDELAY, PAFLENGTH
VOFFSET	V	user set	OFFSETDAC
XDT	µs	user set	XWAIT
BASELINE_PERCENT	%	user set	BASELINEPERCENT
EMIN	None	user set	ENERGYLOW
BINFACTOR	None	user set	LOG2EBIN
BASELINE_AVERAGE	None	User set	LOG2BWEIGHT
CHANNEL_CSRA	bit pattern	user set	CHANCSRA
CHANNEL_CSRB	bit pattern	user set	CHANCSRB
BLCUT	None	user set/auto API	BLCUT
INTEGRATOR	None	user set	INTEGRATOR
FASTTRIGBACKLEN	µs	user set	FASTTRIGBACKLEN
CFDDelay	µs	user set	CFDDELAY
CFDScale	None	user set	CFDSCALE

CFDThresh	None	user set	CFDTHRESH
QDCLen0	µs	user set	QDCLEN0
QDCLen1	µs	user set	QDCLEN1
QDCLen2	µs	user set	QDCLEN2
QDCLen3	µs	user set	QDCLEN3
QDCLen4	µs	user set	QDCLEN4
QDCLen5	µs	user set	QDCLEN5
QDCLen6	µs	user set	QDCLEN6
QDCLen7	µs	user set	QDCLEN7
ExtTrigStretch	µs	user set	EXTTRIGSTRETCH
VetoStretch	µs	user set	VETOSTRETCH
MultiplicityMaskL	bit pattern	user set	MULTIPLICITYMASKL
MultiplicityMaskH	bit pattern	user set	MULTIPLICITYMASKH
ExternDelayLen	µs	user set	EXTERNDELAYLEN
FtrigoutDelay	µs	user set	FTRIGOUTDELAY
ChanTrigStretch	µs	user set	CHANTRIGSTRETCH

Return values

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

Usage example

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

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

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

3.2.36 Pixie16ReadSglModPar

Syntax

```
int Pixie16ReadSglModPar (
   char *ModParName,              // module parameter name
   unsigned int *ModParData,            // module parameter value
   unsigned short ModNum )        // module number
```

Description

Use this function to read a module parameter from a Pixie-16 module. ModNum is the module number which starts counting at 0. The supported module parameters are listed below.

Module Parameters	Unit	Type	Corresponding DSP Variables
MODULE_NUMBER	None	user set	MODNUM
MODULE_CSRA	bit pattern	user set	MODCSRA
MODULE_CSRB	bit pattern	user set	MODCSRB
MODULE_FORMAT	None	auto DSP	MODFORMAT
MAX_EVENTS	None	user set/auto DSP	MAXEVENTS
SYNCH_WAIT	logic (0, 1)	user set	SYNCHWAIT
IN_SYNCH	logic (0, 1)	User/DSP set	INSYNCH
SLOW_FILTER_RANGE	None	user set	SLOWFILTERRANGE
FAST_FILTER_RANGE	None	user set	FASTFILTERRANGE
FastTrigBackplaneEna	bit pattern	user set	FASTTRIGBACKPLANEENA
CrateID	None	user set	CRATEID
SlotID	None	user set	SLOTID
ModID	None	user set	MODID
TrigConfig0	None	user set	TRIGCONFIG[0]
TrigConfig1	None	user set	TRIGCONFIG[1]
TrigConfig2	None	user set	TRIGCONFIG[2]
TrigConfig3	None	user set	TRIGCONFIG[3]
HOST_RT_PRESET	None	user set	HOSTRUNTIMEPRESET

Return values

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

Usage example

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

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

retval = Pixie16ReadSglModPar ("SLOW_FILTER_RANGE", &ModParData, ModNum);

if(retval < 0)
{
   // Error handling
}
```

3.2.37 Pixie16ReadStatisticsFromModule

Syntax

```
int Pixie16ReadStatisticsFromModule (
  unsigned int *Statistics,                  // statistics data
  unsigned short ModNum )                    // module number
```

Description

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.

Return values

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

Usage example

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

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

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

3.2.38 Pixie16RegisterIO

Syntax

```
int Pixie16RegisterIO (
   unsigned short ModNum,        // module number
   unsigned int address,         // register address
   unsigned short direction,     // read or write
   unsigned int *value )         // holds or receives the data
```

Description

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.

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid Pixie-16 module number	Correct ModNum

Usage example

```
int retval;
unsigned short ModNum;
unsigned int address, value;

ModNum = 0;                       // the first module
address = PCI_STOPRUN_REGADDR;    // address of the register for ending run
value = 0;

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

3.2.39 Pixie16SaveDSPParametersToFile

Syntax

```
int Pixie16SaveDSPParametersToFile (
  char *FileName )  // DSP parameters file name (with complete path)
```

Description

Use this function to save DSP parameters to a settings file. It will first read the values of DSP parameters on each Pixie-16 module and then write them to the settings file. Each module has exactly 1280 DSP parameter values (32-bit unsigned integers), and depending on the value of PRESET_MAX_MODULES (defined in pixie16app_defs.h), the settings file should have exactly (1280 * PRESET_MAX_MODULES * 4) bytes when stored on the computer hard drive.

Return values

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

Usage example

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

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

3.2.40 Pixie16SaveExternalFIFODataToFile

Syntax

```
int Pixie16SaveExternalFIFODataToFile (
   char *FileName,                // list mode data file name
   unsigned int *nFIFOWords,      // number of words read from ext. FIFO
   unsigned short ModNum,         // module number
   unsigned short EndOfRunRead ) // indicator if this is end of run read
```

Description

Use this function to read data from the external FIFO of a module. ModNum is the module number which starts counting at 0.

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

Return values

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

Usage example

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

ModNum = 0;          // the first module in the system
EndOfRunRead = 0;    // this is a read during the run

retval = Pixie16SaveExternalFIFODataToFile("listmodedata_mod0.bin",
&nFIFOWords, ModNum, EndOfRunRead );
if(retval < 0)
```

```
    {
            // Error handling
    }
```

3.2.41 Pixie16SaveHistogramToFile

Syntax

```
int Pixie16SaveHistogramToFile (
  char *FileName,   // histogram data file name (with complete path)
  unsigned short ModNum)  // module number
```

Description

Use this function to read histogram data from a Pixie-16 module and save the histogram data to a file with file name specified by the user: First this function saves the histogram data to a binary file, and it then saves the histogram data to an ASCII file with run statistics data appended to the end of the ASCII file. Existing files will be overwritten. ModNum is the module number which starts counting at 0.

Return values

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

Usage example

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

ModNum = 0;                // the first module

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

3.2.42 Pixie16SetDACs

Syntax

```
int Pixie16SetDACs (
  unsigned short ModNum);       // module number
```

Description

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

Return values

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

Usage example

```
int retval;
unsigned short ModNum;

ModNum = 0;                 // the first module

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

3.2.43 Pixie16StartHistogramRun

Syntax

```
int Pixie16StartHistogramRun (
        unsigned short ModNum,        // module number
        unsigned short mode )         // run mode
```

Description

Use this function to begin a data acquisition run that accumulates energy histograms, one for each channel. It launches a data acquisition run in which only energy information is preserved and histogrammed locally to each channel.

Call this function for each Pixie-16 module in the system to initialize the run in each module. Actual data acquisition will start synchronously in all modules when the last module finished the initialization (requires the synchronization parameter to be set). Histogram runs can be self-terminating when the elapsed run time exceeds the preset run time, or the user can prematurely terminate the run by calling `Pixie16EndRun`. On completion, final histogram and statistics data will be available.

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.

Return values

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

Usage example

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

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

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

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

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

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

3.2.44 Pixie16StartListModeRun

Syntax

```
int Pixie16StartListModeRun (
        unsigned short ModNum,        // module number
        unsigned short RunType,       // run type
        unsigned short mode )         // run mode
```

Description

Use this function to start a list mode data acquisition run in Pixie-16 modules. List mode runs are used to collect data on an event-by-event basis, gathering energies, timestamps, pulse shape analysis values, and waveforms for each event. Runs will continue until the user terminates the run by calling function `Pixie16EndRun`. To start the data acquisition this function has to be called for every Pixie-16 module in the system. If all modules are to run synchronously, the last module addressed will release all others and the acquisition starts then. The first module to end the run will immediately stop the run in all other modules if run synchronization has been set up among these modules.

Use mode=NEW_RUN (=1) to erase histograms and statistics information before launching the new run. Note that this will cause a startup delay of up to 1 millisecond. Use mode=RESUME_RUN (=0) to resume an earlier run. This mode has a startup delay of only a few microseconds.

There is only one list mode run type supported, that is, 0x100. However, different output data options can be chosen by enabling or disabling different CHANCSRA bits, see section 4.2.4.

ModNum is the module number which starts counting at 0. If ModNum is set to be less than the total number of modules in the system, only the module specified by ModNum will have its list mode run started. But if ModNum is set to equal to the total number of modules in the system, then all modules in the system will have their runs started together.

Return values

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

Usage example

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

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

// Assume there are 5 modules in the system
```

```
ModNum = 5;                // start list mode run in all 5 modules

retval = Pixie16StartListModeRun (ModNum, RunType, mode);
if(retval < 0)
{
   // Error handling
}
```

3.2.45 Pixie16TauFinder

Syntax

```
void Pixie16TauFinder (
unsigned short ModNum,     // module number
double *Tau )              // Tau value
```

Description

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

Return values

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

Usage example

```
int retval;
unsigned short ModNum;
double Tau[16];

ModNum = 0;                // the first module

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

3.2.46 Pixie16WriteCSR

Syntax

```
void Pixie16WriteCSR (
   unsigned short ModNum,        // module number
   unsigned int CSR )            // CSR value to write
```

Description

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

See section 5.1 for CSR bit definitions

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid Pixie-16 module number	Correct the ModNum

Usage example

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

ModNum = 0;    // the first module

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

CSR = APP32_ClrBit(3, CSR);

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

3.2.47 Pixie16WriteSglChanPar

Syntax

```
int Pixie16WriteSglChanPar (
   char *ChanParName,                  // channel parameter name
   double ChanParData,                 // channel parameter value
   unsigned short ModNum,              // channel number
   unsigned short ChanNum )            // module number
```

Description

Use this function to write a channel parameter to a Pixie-16 module. ModNum is the module number which starts counting at 0. ChanNum is the channel number which starts counting at 0. The supported channel parameters are listed below.

Channel Parameters	Unit	Type	Corresponding DSP Variables
TRIGGER_RISETIME	µs	user set	FASTLENGTH
TRIGGER_FLATTOP	µs	user set	FASTGAP
TRIGGER_THRESHOLD	ADC units	user set	FASTTHRESH
ENERGY_RISETIME	µs	user set	SLOWLENGTH
ENERGY_FLATTOP	µs	user set	SLOWGAP
TAU	µs	user set	PREAMPTAU
TRACE_LENGTH	µs	user set	TRACELENGTH
TRACE_DELAY	µs	user set	TRIGGERDELAY, PAFLENGTH
VOFFSET	V	user set	OFFSETDAC
XDT	µs	user set	XWAIT
BASELINE_PERCENT	%	user set	BASELINEPERCENT
EMIN	None	user set	ENERGYLOW
BINFACTOR	None	user set	LOG2EBIN
BASELINE_AVERAGE	None	user set	LOG2BWEIGHT
CHANNEL_CSRA	bit pattern	user set	CHANCSRA
CHANNEL_CSRB	bit pattern	user set	CHANCSRB
BLCUT	None	user set/auto API	BLCUT
INTEGRATOR	None	user set	INTEGRATOR
FASTTRIGBACKLEN	µs	user set	FASTTRIGBACKLEN
CFDDelay	µs	user set	CFDDELAY
CFDScale	None	user set	CFDSCALE

CFDThresh	ADC units	user set	CFDTHRESH
QDCLen0	µs	user set	QDCLEN0
QDCLen1	µs	user set	QDCLEN1
QDCLen2	µs	user set	QDCLEN2
QDCLen3	µs	user set	QDCLEN3
QDCLen4	µs	user set	QDCLEN4
QDCLen5	µs	user set	QDCLEN5
QDCLen6	µs	user set	QDCLEN6
QDCLen7	µs	user set	QDCLEN7
ExtTrigStretch	µs	user set	EXTTRIGSTRETCH
VetoStretch	µs	user set	VETOSTRETCH
MultiplicityMaskL	bit pattern	user set	MULTIPLICITYMASKL
MultiplicityMaskH	bit pattern	user set	MULTIPLICITYMASKH
ExternDelayLen	µs	user set	EXTERNDELAYLEN
FtrigoutDelay	µs	user set	FTRIGOUTDELAY
ChanTrigStretch	µs	user set	CHANTRIGSTRETCH

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid Pixie-16 module number	Correct ModNum
-2	Invalid Pixie-16 channel number	Correct ChanNum
-3	Invalid channel parameter name	Correct *ChanParName
-4	Programing Fippi failed downloading channel parameter	Reboot the module
-5	Failed to find BLcut after downloading channel parameter	Reboot the module
-6	SetDACs failed downloading channel parameter	Reboot the module

Usage example

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

// Set energy filter rise time to 6.08 µs for module 0 channel 0
ModNum = 0;          // this is the first module
ChanNum = 0;         // the first channel
ChanParData = 6.08;  // energy filter rise time = 6.08 µs
```

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

3.2.48 Pixie16WriteSglModPar

Syntax

```
int Pixie16WriteSglModPar (
   char *ModParName,              // module parameter name
   unsigned int ModParData,       // module parameter value
   unsigned short ModNum )        // module number
```

Description

Use this function to write a module parameter to a Pixie-16 module. ModNum is the module number which starts counting at 0. The supported module parameters are listed below.

Module Parameters	Unit	Type	Corresponding DSP Variables
MODULE_CSRA	bit pattern	user set	MODCSRA
MODULE_CSRB	bit pattern	user set	MODCSRB
MODULE_FORMAT	None	auto DSP	MODFORMAT
MAX_EVENTS	None	user set/auto DSP	MAXEVENTS
SYNCH_WAIT	logic (0, 1)	user set	SYNCHWAIT
IN_SYNCH	logic (0, 1)	user/DSP set	INSYNCH
SLOW_FILTER_RANGE	None	user set	SLOWFILTERRANGE
FAST_FILTER_RANGE	None	user set	FASTFILTERRANGE
FastTrigBackplaneEna	bit pattern	user set	FASTTRIGBACKPLANEENA
CrateID	None	user set	CRATEID
SlotID	None	user set	SLOTID
ModID	None	user set	MODID
TrigConfig0	None	user set	TRIGCONFIG[0]
TrigConfig1	None	user set	TRIGCONFIG[1]
TrigConfig2	None	user set	TRIGCONFIG[2]
TrigConfig3	None	user set	TRIGCONFIG[3]
HOST_RT_PRESET	seconds	user set	HOSTRUNTIMEPRESET

Return values

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

Usage example

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

// Set SlowFilterRange in module 0 to 4
ModNum = 0;                // this is the first module
ModParData = 4;            // SlowFilterRange = 4

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

3.3 PIXIE-16 Utility Functions

There are a few utility functions that can also be used in users' applications.

Utility Function Name	Description
APP16_ClrBit	Clear one bit of a 16-bit unsigned integer
APP16_SetBit	Set one bit of a 16-bit unsigned integer
APP16_TstBit	Test one bit of a 16-bit unsigned integer
APP32_ClrBit	Clear one bit of a 32-bit unsigned integer
APP32_SetBit	Set one bit of a 32-bit unsigned integer
APP32_TstBit	Test one bit of a 32-bit unsigned integer
Decimal2IEEEFloating	Convert a decimal into IEEE 32-bit floating point number
IEEEFloating2Decimal	Convert an IEEE 32-bit floating point number to a decimal

Syntax

```
unsigned short APP16_ClrBit (
               unsigned short bit,
               unsigned short value )

unsigned short APP16_SetBit (
               unsigned short bit,
               unsigned short value )

unsigned short APP16_TstBit (
               unsigned short bit,
               unsigned short value )

unsigned int APP32_ClrBit (
               unsigned short bit,
               unsigned int value )

unsigned int APP32_SetBit (
               unsigned short bit,
               unsigned int value )

unsigned int APP32_TstBit (
               unsigned short bit,
               unsigned int value )

unsigned int Decimal2IEEEFloating (
               double DecimalNumber )

double IEEEFloating2Decimal (
               unsigned int IEEEFloatingNumber )
```

The return value of the setbit and clrbit functions is the input `value` with bit number `bit`-set or cleared. The tstbit function returns logic 1 if bit number `bit`, is set in the input `value`, or returns logic 0 if bit number `bit`, is not set in the input `value`. The return values of IEEE floating functions are either IEEE floating values or decimal values, respectively.

4 Control Parameters

4.1 User and DSP parameter overview

The host computer communicates with the Pixie-16 by setting and reading a set of variables called DSP parameters. These parameters are divided into input and output parameters. The exact location of any particular variable in the DSP memory could vary from one code version to another. To facilitate writing robust user code, a reference table of variable names and addresses with each DSP code version was provided. Included with the software distribution is a file called Pixie16DSP_*variant_version*.var. It contains a two-column list of variable names and their respective addresses. Further, the Pixie-16 API library initializes the DSP parameters' address at module's boot up by calling function `Pixie_Init_DSPVarAddress` in file `utilities.c`. Thus user code can be written in such a way that it uses the DSP parameters' address set by the Pixie-16 API, rather than uses fixed locations.

Many of the DSP variables depend on the values of other variables. A complete description of all interdependencies can be found in the following sections. All of these interdependencies have been taken care of by the PIXIE-16 API, which calculates the DSP parameter values from a number of user parameters with meaningful units. So instead of directly setting DSP variables and their dependencies, users only need to set the values of user parameters; for example instead of entering a filter time in DSP units and also computing the dependent pileup inspection settings etc, users enter a filter time in microseconds, which is converted into DSP units by the API.

Table 4.1: Descriptions of User and DSP parameters in PIXIE-16.

System Parameters	**Unit**	**Type**	**Corresponding DSP Variables**
NUMBER_MODULES	None	user set	None
OFFLINE_ANALYSIS	None	user set	None
PXI_SLOT_MAP	None	user set	None
Module Parameters (Input)	**Unit**	**Type**	**Corresponding DSP Variables**
MODULE_NUMBER	None	auto API	MODNUM
MODULE_CSRA	bit pattern	user set	MODCSRA
MODULE_CSRB	bit pattern	user set	MODCSRB
MODULE_FORMAT	None	auto DSP	MODFORMAT
RUN_TYPE	None	auto API	RUNTASK, CONTROLTASK
MAX_EVENTS	None	user set/auto DSP	MAXEVENTS
SYNCH_WAIT	logic (0, 1)	user set	SYNCHWAIT
IN_SYNCH	logic (0, 1)	user/DSP set	INSYNCH
SLOW_FILTER_RANGE	None	user set	SLOWFILTERRANGE

FAST_FILTER_RANGE	None	user set	FASTFILTERRANGE
FastTrigBackplaneEna	bit pattern	user set	FASTTRIGBACKPLANEENA
CrateID	None	user set	CRATEID
SlotID	None	user set	SLOTID
ModID	None	user set	MODID
TrigConfig0	bit pattern	user set	TrigConfig[0]
TrigConfig1	bit pattern	user set	TrigConfig[1]
TrigConfig2	bit pattern	user set	TrigConfig[2]
TrigConfig3	bit pattern	user set	TrigConfig[3]
HOST_RT_PRESET	seconds	user set	HOSTRUNTIMEPRESET
Module Parameters (Output)	**Unit**	**Type**	**Corresponding DSP Variables**
RUN_TIME	seconds	DAQ result	RUNTIMEA, B
NUMBER_EVENTS	counts	DAQ result	NUMEVENTSA, B
BUFFER_HEAD_LENGTH	None	data format	BUFHEADLEN
EVENT_HEAD_LENGTH	None	data format	EVENTHEADLEN
CHANNEL_HEAD_LENGTH	None	data format	CHANHEADLEN
OUTPUT_BUFFER_LENGTH	None	auto DSP	LOUTBUFFER
HARDWARE_ID	None	auto DSP	HARDWAREID
HARDWARE_VARIANT	None	auto DSP	HARDVARIANT
FIFO_LENGTH	None	auto DSP	FIFOLENGTH
FIPPI_ID	None	auto DSP	FIPPIID
FIPPI_VARIANT	None	auto DSP	FIPPIVARIANT
DSP_RELEASE	None	auto DSP	DSPRELEASE
DSP_BUILD	None	auto DSP	DSPBUILD
DSP_VARIANT	None	auto DSP	DSPVARIANT
Channel Parameters (Input)	**Unit**	**Type**	**Corresponding DSP Variables**
CHANNEL_CSRA	bit pattern	user set	CHANCSRA
CHANNEL_CSRB	bit pattern	user set	CHANCSRB
VOFFSET	V	user set	OFFSETDAC
ENERGY_RISETIME	µs	user set	SLOWLENGTH
ENERGY_FLATTOP	µs	user set	SLOWGAP
PEAK_SAMPLE	None	auto API	PEAKSAMPLE
PEAK_SEP	None	auto API	PEAKSEP

TRIGGER_RISETIME	µs	user set	FASTLENGTH
TRIGGER_FLATTOP	µs	user set	FASTGAP
TRIGGER_THRESHOLD	ADC units	user set	FASTTHRESH
TRACE_LENGTH	µs	user set	TRACELENGTH
TRACE_DELAY	µs	user set	TRIGGERDELAY, PAFLENGTH
EMIN	None	user set	ENERGYLOW
BINFACTOR	None	user set	LOG2EBIN
BLCUT	None	user set/auto API	BLCUT
BASELINE_PERCENT	%	user set	BASELINEPERCENT
TAU	µs	user set	PREAMPTAU
XDT	µs	user set	XWAIT
BASELINE_AVERAGE	None	user set	LOG2BWEIGHT
FASTTRIGBACKLEN	µs	user set	FASTTRIGBACKLEN
INTEGRATOR	None	user set	INTEGRATOR
CFDDelay	µs	user set	CFDDELAY
CFDScale	None	user set	CFDSCALE
CFDThresh	ADC units	user set	CFDTHRESH
QDCLen0	µs	user set	QDCLEN0
QDCLen1	µs	user set	QDCLEN1
QDCLen2	µs	user set	QDCLEN2
QDCLen3	µs	user set	QDCLEN3
QDCLen4	µs	user set	QDCLEN4
QDCLen5	µs	user set	QDCLEN5
QDCLen6	µs	user set	QDCLEN6
QDCLen7	µs	user set	QDCLEN7
ExtTrigStretch	µs	user set	EXTTRIGSTRETCH
VetoStretch	µs	user set	VETOSTRETCH
MultiplicityMaskL	bit pattern	user set	MULTIPLICITYMASKL
MultiplicityMaskH	bit pattern	user set	MULTIPLICITYMASKH
ExternDelayLen	µs	user set	EXTERNDELAYLEN
FtrigoutDelay	µs	user set	FTRIGOUTDELAY
ChanTrigStretch	µs	user set	CHANTRIGSTRETCH
Channel Parameters (Output)	**Unit**	**I/O Type**	**Corresponding DSP Variables**
LIVE_TIME	seconds	DAQ result	LIVETIMEA, B

FAST_PEAKS	number	DAQ result	FASTPEAKSA, B
INPUT_COUNT_RATE	cps	auto API	LIVETIMEA, B, FASTPEAKSA, B
OUTPUT_COUNT_RATE	cps	auto API	CHANEVENTSA, B, RUNTIMEA, B

4.2 User Parameters

User parameters are input/output variables for the Pixie-16 API. In the Pixie-16 VB demo interface, these parameters are displayed in variable fields and checkboxes. Custom interfaces can set and read these variables in whatever form is most suitable. For all parameters, also see the corresponding entry in DSP parameter section for details.

In most cases, the Pixie-16 API will adjust the input parameter to the closest value that corresponds to a valid DSP parameter value, for example entering a filter time of 22ns will be adjusted to 20ns.

4.2.1 System Parameters

NUMBER_MODULES: User specified number of modules in the system.
Valid range: 1-24.

OFFLINE_ANALYSIS: Set to 1 to run the Pixie-16 API on analysis-only mode, i.e., without a Pixie-16 module present.
Valid range: 0 and 1.

C_LIBRARY_RELEASE:
C_LIBRARY_BUILD: Currently not implemented. Output parameters indicating version of the Pixie-16 API.

PXI_SLOT_MAP: User specified array listing the occupied slots in the PXI chassis.
Valid range for slot numbers: 2-8 (for 8-slot chassis) or 2-14 (for 14-slot chassis).

4.2.2 Module Parameters (Input)

MODULE_NUMBER: Number assigned to the module by the API during booting. The number will be written in the module's output data header for identification. Also used to address individual modules in the API.
Valid range: 0 to (number of modules intended for booting – 1)

MODULE_CSRA: Bit pattern controlling data acquisition in this module.
Valid range: 0 – 2^{32}-1 (theoretical).

MODULE_CSRB: Bit pattern controlling data acquisition in this module.
Valid range: 0 – 2^{32}-1 (theoretical).

MODULE_FORMAT: Currently unused.

RUN_TYPE: Number specifying the data acquisition type. See entry for RUNTASK in DSP parameter section for details.
Valid range: 0 (control run), 0x100 (list mode run), 0x301 (MCA run).

MAX_EVENTS: Currently not implemented. Number specifying the maximum numbers of events the module will acquire in one list mode spill run.
Valid range: depends on Trace_Lengths selected.

SYNCH_WAIT: Number specifying if modules should start data acquisition runs simultaneously.
Valid range: 0 (not simultaneously), 1 (simultaneously).

IN_SYNCH: Number specifying if modules should reset their clock counters to zero at the start of a data acquisition run. Set this number to 0 to reset clock counters to zero at the start of the run. After synchronization, IN_SYNCH is set to one by the DSP to indicate module clocks are synchronized simultaneously.
Valid range: 0 (simultaneously), 1 (not simultaneously).

SLOW_FILTER_RANGE: Number specifying the how many ADC samples will be averaged for one step in the energy filters. Since the number of steps in the filter is limited, this is used to extend the range of filter times.
Valid range: 1 to 6.

FAST_FILTER_RANGE: Number specifying the how many ADC samples will be averaged for one step in the trigger filters. Since the number of steps in the filter is limited, this is used to extend the range of filter times.
Valid range: 0 (i.e., currently no fast filter averaging is supported).

FastTrigBackplaneEna: Bit mask pattern to specify whether a channel should send its fast trigger to its nearest neighboring module: bits [15:0] of FastTrigBackplaneEna indicating whether sending (1) or not sending (0) each of the 16 channels' fast trigger to nearest neighboring module on its left; bits [31:16] of FastTrigBackplaneEna indicating whether sending (1) or not sending (0) each of the 16 channels' fast trigger to nearest neighboring module on its right.

CrateID: The ID of the crate.
Valid range: 0 to (Number of Crates – 1).

SlotID: The ID of the slot in the crate.
Valid range: 2 to 8 (8-slot crate) or 2 to 14 (14-slot crate).

ModID: The ID of the Pixie-16 module.
Valid range: 0 to (Number of Modules – 1).

TrigConfig0: The first trigger configuration registers. Please refer to the user's manual for its detailed usage.

TrigConfig1: The second trigger configuration registers. Please refer to the user's manual for its detailed usage.

TrigConfig2: The third trigger configuration registers. Please refer to the user's manual for its detailed usage.

TrigConfig3: The fourth trigger configuration registers. Please refer to the user's manual for its detailed usage.

HOST_RT_PRESET: The preset run time requested by the host. It is only used by MCA histogram mode run. Its corresponding DSP parameter is HOSTRUNTIMEPRESET, which is a 32-bit IEEE standard floating point number. This is the only Module level parameter that needs to use this special data format. The reason for this is that preset run time can be non-integer, e.g. run for 30.5 seconds. All other Module Level parameters are integers. To set its value correctly, Pixie-16 c-library function `Decimal2IEEEFloating` has to be called to convert a decimal value into an IEEE 32-bit floating point number in the form of unsigned integer. Then that converted unsigned integer can be used in the function `Pixie16WriteSglModPar` to set the preset MCA run

time with ModParName 'HOST_RT_PRESET'.
Valid range: 0 – infinite (theoretical)

4.2.3 Module Parameters (Output)

RUN_TIME: Duration of the data acquisition in seconds.

NUMBER_EVENTS: Currently not implemented.

BUFFER_HEAD_LENGTH: Currently not implemented.

EVENT_HEAD_LENGTH: Currently not implemented.

CHANNEL_HEAD_LENGTH: Currently not implemented.

OUTPUT_BUFFER_LENGTH: Currently not implemented.

HARDWARE_ID: Currently not implemented.

HARDWARE_VARIANT: Currently not implemented.

FIFO_LENGTH: Length of the trace buffer in the signal processing FPGAs.

FIPPI_ID: Currently not implemented.

FIPPI_VARIANT: Currently not implemented.

DSP_RELEASE: DSP code release number.

DSP_BUILD: DSP code build number.

DSP_VARIANT: Currently not implemented.

4.2.4 Channel Parameters (Input)

CHANNEL_CSRA: Bit pattern controlling data acquisition in this channel
Valid range: 0 – 2^{32}-1 (theoretical).

CHANNEL_CSRB: Bit pattern controlling data acquisition in this channel
Valid range: 0 – 2^{32}-1 (theoretical).

VOFFSET: Voltage offset in V.
Valid range: -1.5V - +1.5V.

ENERGY_RISETIME: Energy filter (trapezoidal) rise time in µs.
Valid range: $2*2^{SLOWFILTERRANGE}*10ns$ – $124*2^{SLOWFILTERRANGE}*10ns$ (100 MHz or 500 MHz modules) or $2*2^{SLOWFILTERRANGE}*8ns$ – $124*2^{SLOWFILTERRANGE}*8ns$ (250 MHz modules).

ENERGY_FLATTOP: Energy filter (trapezoidal) flat top time in µs.
Valid range: $3*2^{SLOWFILTERRANGE}*10ns$ – $125*2^{SLOWFILTERRANGE}*10ns$ (100 MHz or 500 MHz modules) or $3*2^{SLOWFILTERRANGE}*8ns$ – $125*2^{SLOWFILTERRANGE}*8ns$ (250 MHz modules).

TRIGGER_RISETIME: Trigger filter (trapezoidal) rise time in µs.
Valid range: 2*10ns – 127*10ns (100 MHz or 500 MHz modules) or 2*8ns – 127*8ns (250 MHz modules).

TRIGGER_FLATTOP: Trigger filter (trapezoidal) flat top time in µs.
Valid range: 0ns – 125*10ns (100 MHz or 500 MHz modules) or 0ns – 125*8ns (250 MHz modules).

TRIGGER_THRESHOLD: Threshold of the trigger filter in ADC steps. When the trigger filter output is greater than this number, a trigger is issued.
Valid range: 1 – 65535.

TRACE_LENGTH: Length of waveform acquired in microseconds.
Valid range: 0 – 81.92 µs, 10ns steps (Rev B/C/D modules); 0 – 163.8 µs, 10ns steps (Rev F, 100 MHz modules); 0 – 131 µs, 8ns steps (Rev F, 250 MHz modules); 0 – 40.94 µs, 2ns steps (Rev F, 500 MHz modules).

TRACE_DELAY: Length of pre-trigger time of waveform acquired in microseconds.
Valid range: 0 – 10.24 µs, 10ns steps (Rev B/C/D, or Rev F-100 MHz, or Rev F-500 MHz modules); 0 – 8.192 µs, 8ns steps (Rev F-250 MHz modules).

EMIN: Energy cutoff value. Option for not recording trace if computed energy is above EMIN (or EnergyLow in the DSP code).
Valid Range: 0 – 65535.

BINFACTOR: This variable controls the binning of the histogram. Energy values are calculated to 16 bits precision. The LSB corresponds to $1/16^{th}$ of a 12-bit ADC. The Pixie-16s, however, do not have enough histogram memory available to record 64K spectra, nor would this always be desirable. The user is therefore free to choose the binning control. Observe the following formula to find to which MCA bin a value of Energy will contribute:

$$MCAbin = Energy * 2^{(-BINFACTOR)}$$

Valid Range: 1 – 16.

BLCUT: This variable sets the cutoff value for baselines in baseline measurements. If BLCUT is not set to zero, the DSP checks continuously each baseline value to see if it is outside of the limit set by BLCUT. If the baseline value is within the limit, it will be used to calculate the average baseline value. Otherwise, it will be discarded. Set BLCUT to zero to not check baselines, therefore reduce processing time, but with the risk of worse energy resolution.
ControlTask 6 can be used to measure baselines. Host computer can then histogram these baseline values and determine the appropriate value for BLCUT for each channel according to the standard deviation SIGMA for the averaged baseline value. BLCUT could be set to be three times SIGMA.

BASELINE_PERCENT: This variable sets the DC-offset level in terms of the percentage of the ADC range during automatic offset adjustments.

TAU: Preamplifier exponential decay time. This variable is used to store the preamplifier decay time. The time τ is measured in µs. The DSP uses this variable to compute coefficients for the energy calculation.

XDT: Time interval between ADC samples acquired for the oscilloscope view.

BASELINE_AVERAGE: This variable sets the value of DSP parameter LOG2BWEIGHT for averaging baseline values during baseline computations in the DSP.

FASTTRIGBACKLEN: This variable sets the fast trigger backplane length.

INTEGRATOR: Currently not implemented

CFDDELAY: This variable sets the CFD delay value.

CFDSCALE: This variable sets the CFD scaling factor.

CFDTHRESH: Threshold for arming the search for CFD zero crossing point.

QDCLEN0: This variable sets the QDC length #0.

QDCLEN1: This variable sets the QDC length #1.

QDCLEN2: This variable sets the QDC length #2.

QDCLEN3: This variable sets the QDC length #3.

QDCLEN4: This variable sets the QDC length #4.

QDCLEN5: This variable sets the QDC length #5.

QDCLEN6: This variable sets the QDC length #6.

QDCLEN7: This variable sets the QDC length #7.

EXTTRIGSTRETCH: This variable sets the stretched length of the external trigger.

VETOSTRETCH: This variable sets the stretched length of the veto signal.

MULTIPLICITYMASKL: This variable sets the lower 32-bit of the Multiplicity mask.

MULTIPLICITYMASKH: This variable sets the upper 32-bit of the Multiplicity mask.

EXTERNDELAYLEN: This variable sets the external delay length.

FTRIGOUTDELAY: This variable sets the delay value for outputting fast trigger to the System FPGA.

CHANTRIGSTRETCH: This variable sets the stretched length of the channel validation trigger.

4.2.5 Channel Parameters (Output)

LIVE_TIME: Time in seconds the channel was active.

FAST_PEAKS: Number of fast triggers seen in this channel.

INPUT_COUNT_RATE: Input count rate for this channel.

OUTPUT_COUNT_RATE: Output count rate for this channel.

4.3 DSP Parameters

Below we describe the module and channel parameters in turn. Where appropriate, we show how a variable can be viewed using the Pixie-16 VB demo interface. Note that even though there are functions available to simply read and write these parameters to and from DSP memory, in some cases the variables have to be applied to the FPGAs by starting a CONTROLTASK in the DSP. Other variables are only stored in the DSP parameter space and used by higher level functions.

4.3.1 Module input parameters

MODNUM: Logical number of the module. This number can be used to be part of the list mode binary data file name to aid offline event reconstruction. It is assigned by the C driver API at time of booting.
Valid range: 0 to (number of modules intended for booting – 1).

MODCSRA: Currently not implemented.

Bits 0-31: Reserved.

MODCSRB: The Module Control and Status Register B:

Bit 0: If set, wired-OR trigger lines on the backplane connect to a pullup resistor. This bit should be set for only one module in the backplane segment.

Bit 1-3: Reserved.

Bit 4: Set this module as the Director module (1) or non-Director module (0).

Bit 5: Reserved.

Bit 6: Control chassis master module: 1: chassis master module; 0: chassis non-master module

Bit 7: Select global fast trigger source

Bit 8: Select external trigger source

Bit 9: Reserved.

Bit 10: Control external INHIBIT signal: use INHIBIT (1) or don't use INHIBIT (0)

Bit 11: Distribute clock and triggers in multiple crates: multiple crates (1) or only single crate (0)

Bit 12: Sort (1) or don't sort events based on their timestamps

Bit 13: Enable connection of fast triggers to backplane

Bits 14-31: Reserved.

MODFORMAT: List mode data format descriptor. Currently it is not in use.

RUNTASK: This variable tells the Pixie-16 what kind of run to start in response to a run start request. Three run tasks are currently supported.

RunTask	Mode	Trace Capture
0	**Slow control run**	**N/A**
256 (0x100)	**Standard list mode**	**Yes**
769 (0x301)	**MCA mode**	**No**

RunTask 0 is used to request slow control tasks. These include programming the trigger/filter FPGAs, setting the DACs in the system, transfers to/from the external memory, and calibration tasks.

RunTask 256 (0x100) requests a standard list mode run. In this run type triggered waveforms together with time of arrival (trigger time), event energy, and other event information are written into the External FIFO for each channel and module. The raw data stream is always sent to the intermediate buffer in the signal processing FPGAs. The data-gathering routine in the DSP reads the raw data from the FPGAs, computes the event energy, and then writes those data to the external FIFO. If the intermediate buffer in the FPGA is full, newly arrived events will be ignored until there is room again in the buffer.

RunTask 769 (0x301) requests a MCA run. Similar to RunTask 256, the event raw data stream is always sent to the intermediate buffer in the signal processing FPGAs. The data-gathering routine in the DSP reads the raw data from the FPGAs, computes the event energy, and then writes the energy value to the External Histogram Memory. If the intermediate buffer in the FPGA is full, newly arrived events will be ignored until there is room again in the buffer. This run type does not write data to the External FIFO.

The RunTask can be chosen as the run type in the Run tab of the VB interface.

CONTROLTASK: Use this variable to select a control task. Consult the control tasks section of this manual for detailed information. The control task will be launched when a run start command is issued with RUNTASK=0.

MAXEVENTS: currently not implemented

COINCPATTERN: currently not implemented.

COINCWAIT: currently not implemented

SYNCHWAIT: Controls run start behavior. When set to 0 the module simply starts or resumes a run in response to the corresponding host request. When set to 1,

modules will run synchronously through the backplane. This will ensure that the last module ready to actually begin data taking will start the run in all modules and the first module to end the run will stop the run in all modules. This way it never happens that a multi-Pixie system is only partially active.

INSYNCH: InSynch is an input/output variable. It is used in multi-Pixie systems in which the modules are driven by a common clock. When InSynch is 1, the module assumes it is in synch with the other modules and no particular action is taken at run start. If this variable is 0, then all system timers are cleared at the beginning of the next data acquisition run (RunTask>0). The timers are reset when the entire system actually starts the run. After run start, InSynch is automatically set to 1 by the DSP.

RESUME: Set this variable to 1 to resume a data run; otherwise, set it to 0.

SLOWFILTERRANGE: The energy filter range downloaded from the host to the DSP and FPGA. It sets the number of ADC samples ($2^{SLOWFILTERRANGE}$) to be averaged before entering the energy filtering logic. The currently supported filer range in the signal processing FPGA includes 1, 2, 3, 4, 5, and 6.

FASTFILTERRANGE: The trigger filter range downloaded from the host to the DSP and FPGA. It sets the number of ADC samples ($2^{FASTFILTERRANGE}$) to be averaged before entering the trigger filtering logic. The currently supported filer range in the signal processing FPGA is only 0.

CHANNUM: The chosen channel number of a Pixie module. Mainly used by the host to set the designated channel.

HOSTIO: A 16 word data block that is used to specify command options.

USERIN: A block of 16 input variables used by user-written DSP code.

FASTTRIGBACKPLANEENA: Enables sending fast trigger to backplane

CRATEID: ID number for chassis. Reported in list mode data for purposes of event building. Limited to 0..15.

SLOTID: ID number for physical slot in chassis. Reported in list mode data for purposes of event building. Limited to 0..15.

MODID: ID number for module. Unused

TRIGCONFIG[3:0]: Four bit pattern words used to configure various trigger options. See user manual for details

U00: unused, reserved

HOSTRUNTIMEPRESET: Used to set exact run time in MCA mode.
A 32-bit IEEE standard floating point number in seconds.

POWERUPINITDONE: Indicates whether routine to initialize DSP on RESET is done (1) or not (0).

4.3.2 Channel input parameters

All channel-0 variables end with "[00]", channel-1 variables end with "[01]", etc. In the following explanations the numerical suffix has been removed. Thus, e.g., CHANCSRA[00] becomes CHANCSRA, etc.

Once a new variable has been written to DSP memory, it has to be activated by starting a run with RunTask 0 (Set DACs) and ControlTask 5 (Program FiPPI). Strictly speaking, not all variables require this activation, but it is easiest to apply this to them all.

CHANCSRA: The control and status register bits switch on/off various aspects of the PIXIE-16 operation.

Bit 0: Fast trigger selection - 1: select external fast trigger; 0: select local fast trigger

Bit 1: Module validation signal selection - 1: select module gate signal; 0: select global validation signal

Bit 2: Good channel.
Only channels marked as good will contribute to spectra and list mode data.

Bit 3: Channel validation signal selection - 1: select channel gate signal; 0: select channel validation signal

Bit 4: Block data acquisition if trace or header DPMs are full - 1: enable; 0: disable.

Bit 5: Trigger positive.
Set this bit to trigger on a positive slope; clear it for triggering on a negative slope. The trigger/filter FPGA can only handle positive signals. The PIXIE-16 handles negative signals by inverting them immediately after entering the FPGA.

Bit 6: Veto channel trigger - 1: enable; 0: disable

Bit 7: Histogram energies.
Set this bit to histogram energies from this channel in the on-board MCA memory.
NOTE: in the current DSP code implementation, the DSP always histograms event energies in the on-board MCA memory. So the value of this bit has no effect.

Bit 8: Enable trace capture.
Set to 1 to enable trace capture for this channel. Set to 0 to disable trace capture.

Bit 9: Enable QDC sums capture.
Set to 1 to enable QDC sums capture for this channel. Set to 0 to disable QDC sums capture.

Bit 10: Enable CFD trigger mode.
Set to 1 to enable CFD trigger mode for this channel. Set to 0 to disable CFD trigger.

Bit 11: Enable the requirement for module validation trigger.
Set to 1 to require module validation trigger for events validation for this channel. Set to 0 to disable the requirement for module validation trigger.

Bit 12: Enable capture raw energy sums and baselines.
Set to 1 to store raw energy sums and baselines for events captured in this channel. Set to 0 to not capture raw energy sums and baselines.

Bit 13: Enable the requirement for channel validation trigger.
Set to 1 to require channel validation trigger for events validation for this channel. Set to 0 to disable the requirement for channel validation trigger.

Bit 14: Enable input relay.
This bit controls the ON or OFF position switching of the input relay of each channel of the Pixie-16, resulting in two discrete fixed gains for the input signal: one high and one low. The actual gain value depends on the input design of each particular Pixie-16 hardware variant.

Bit 15,16: Pileup rejection control. Set bits [16:15] to
00: record all events (trace, timestamps, etc., but no energy for piled-up events)
01: only record single events (trace, energy, timestamps, etc.) (i.e., reject piled-up events)
10: record trace, timestamps, etc. for piled-up events but do not record trace for single events
11: only record trace, timestamps, etc., for piled-up events (i.e., reject single events)

Bit 17: Enable "no trace for large pulses" feature - 1: enable; 0: disable

Bit 18: Group trigger selection - 1: external group trigger; 0: local fast trigger.

Bit 19: Channel veto selection - 1: channel validation trigger; 0: front panel channel veto.

Bit 20: Module veto selection - 1: module validation trigger; 0: front panel module veto.

Bit 21: External timestamps in event header - 1: enable; 0: disable.

CHANCSRB: Control and status register B.
Bit0..31: are reserved. Set to 0.

GAINDAC: Reserved and not supported.

OFFSETDAC: This DAC determines the DC-offset voltage. The offset can be calculated using the following formula:

Offset [V] = 1.5 * ((32768 - OFFSETDAC) / 32768)

DIGGAIN: Unused. The digital gain factor for compensating the difference between the user-desired voltage gain and the SGA gain.

SLOWLENGTH: The rise time of the energy filter depends on SlowLength:

RiseTime = SlowLength * $2^{\text{SlowFilterRange}}$ * 10 ns *(100 MHz or 500 MHz modules)*
or
RiseTime = SlowLength * $2^{\text{SlowFilterRange}}$ * 8 ns *(250 MHz modules)*

Note the constraint: SlowLength > 2

SLOWGAP: The flat top of the energy filter depends on SlowGap:

FlatTop = SlowGap * $2^{\text{SlowFilterRange}}$ * 10 ns *(100 MHz or 500 MHz modules)*
or
FlatTop = SlowGap * $2^{\text{SlowFilterRange}}$ * 8 ns *(250 MHz modules)*

Note the constraint: SlowGap > 2

There is a constraint concerning the sum value of the two parameters:
SlowLength + SlowGap < 127

FASTLENGTH: The rise time of the trigger filter depends on FastLength:

RiseTime = FastLength * $2^{\text{FastFilterRange}}$ * 10 ns *(100 MHz or 500 MHz modules)*
or
RiseTime = FastLength * $2^{\text{FastFilterRange}}$ * 8 ns *(250 MHz)*

Note the constraint: FastLength > 2

FASTGAP: The flat top of the trigger filter depends on FastGap:

FlatTop = FastGap * $2^{\text{FastFilterRange}}$ * 10 ns *(100 MHz or 500 MHz modules)*
or
FlatTop = FastGap * $2^{\text{FastFilterRange}}$ * 8 ns *(250 MHz modules)*

There is a constraint concerning the sum value of the two parameters: FastLength + FastGap < 127

PEAKSAMPLE: This variable determines at what time the value from the energy filter will be sampled.
PeakSample = SlowLength + Slow Gap – 1

If the sampling point is chosen poorly, the resulting spectrum will show energy resolutions of 10% and wider rather than the expected fraction of a percent. For some parameter combinations PeakSample needs to be varied by one or two units in either direction, due to the pipelined architecture of the trigger/filter FPGA.

Do not set manually, it is computed by the DSP and/or C driver library from the filter.

PEAKSEP: This value governs the minimum time separation between two pulses. Two pulses that arrive within a time span shorter than determined by PeakSep will be rejected as piled up.

The recommended value is:
PeakSep = PeakSample+2 if SlowFilterRange <= 2;
PeakSep = PeakSample+5 if SlowFilterRange > 2.

Do not set manually, it is computed by the DSP and/or C driver library from the filter.

CFDTHRESH: This sets the threshold of the constant fraction discriminator (CFD) trigger that is implemented in the trigger/filter FPGA.

FASTTHRESH: This is the trigger threshold used by the trigger/filter FPGA. The value relates to a trigger threshold through the formula:

FASTTHRESH = TriggerThreshold * FASTLENGTH

Note the constraint FASTTHRESH < 65535

THRESHWIDTH: Unused.
Reserved for width for trigger above threshold

PAFLENGTH: PAFLENGTH and the next parameter TRIGGERDELAY are legacy parameters used by obsolete designs. They are kept only for setting the value of TraceDelay, which is the length of the trace prior to the trigger. The acquired waveform will start rising from the baseline at a time delay (TraceDelay) after the beginning of the trace. This delay is a quantity that the user will want to set. In the PIXIE-16 interface the TraceDelay (measured in microseconds) is available through the Settings tab.

The PafLength computed as follows:

PafLength = TriggerDelay / $2^{FASTFILTERRANGE}$ + TraceDelay/10ns *(100 MHz or 500 MHz modules)* or
PafLength = TriggerDelay / $2^{FASTFILTERRANGE}$ + TraceDelay/8ns *(250 MHz modules)*

Note the constraint: PafLength < FifoLength.

Do not set manually, it is computed by the DSP and/or C driver library from the filter and trace settings.

TRIGGERDELAY: This is a partner variable to PafLength. For *all* filter ranges,

TriggerDelay = (PeakSep -1)*$2^{SlowFilterRange}$

Do not set manually, it is computed by the DSP and/or C driver library from the filter and trace settings.

RESETDELAY: Unused**.** This variable controls the restarting of the FIFO after it was halted to read the waveform. When triggers are distributed across channels and modules, a halted FIFO is automatically restarted if the trigger/filter FPGA does not receive the distributed event trigger within RESETDELAY 10ns clock ticks after the internal event trigger. The default value written by the PIXIE module should not be changed by the user.

CHANTRIGSTRETCH: The "channel validation trigger" from the system FPGA is extended by this value (in clock cycles). See section 3.3.10 in the user manual.

TRACELENGTH: This tells the DSP how many words of trace data to read for each event. The action taken depends on FIFOlength, whose value depends on hardware variants and specific firmware implementations. If TraceLength < FIFOlength, the DSP will read from the FIFO. In that case individual samples are either 10 ns (100 MHz or 500 MHz modules) or 8 ns (250 MHz) apart. If FIFOlength <= TraceLength, the PIXIE-16 code will force the TraceLength to be equal to FIFOlength.

XWAIT: This parameter controls the number of clock cycles between untriggered ADC traces in control run with ControlTask = 4. The time between recorded samples is ΔT = XWAIT*10ns.

TRIGOUTLEN: unused

ENERGYLOW: Start energy histogram at ENERGYLOW. Only applies to list mode runs.

LOG2EBIN: This variable controls the binning of the histogram. Energy values are calculated to 16 bits precision. The LSB corresponds to $1/16^{th}$ of a 12-bit ADC. The PIXIEs, however, do not have enough histogram memory available to record 64K spectra, nor would this always be desirable. The user is therefore free to choose a lower cutoff for the spectrum (EnergyLow) and control the binning. Observe the following formula to find to which MCA bin a value of Energy will contribute:

$$MCAbin = (Energy-EnergyLow) * 2^{Log2Ebin}$$

As can be seen, Log2Ebin should be a negative number to achieve the correct behaviour. At run start the DSP program ensures that Log2Ebin is indeed negative by replacing the stored value by -abs(Log2Ebin).

The histogramming routine of the DSP takes care of spectrum overflows and underflows.

MULTIPLICITYMASKL:
MULTIPLICITYMASKH: bit patterns controlling the coincidence logic. See user manual for details.

PSAOFFSET:
PSALENGTH: currently not implemented

When recording traces and requiring any pulse shape analysis by the DSP, these two parameters govern the range over which the analysis will be applied. The analysis begins at a point PSAOFFSET sampling clock ticks into the trace, and is applied over a piece of the trace with a total length of PSALENGTH clock ticks.

INTEGRATOR: currently not implemented

This variable controls the energy reconstruction in the DSP.

INTEGRATOR == 0: normal trapezoidal filtering
INTEGRATOR == 1: use gap sum only; good for scintillator signals
INTEGRATOR == 2: ignore gap sum; pulse height=leading sum – trailing sum; good for step-like pulses.

BLCUT: This variable sets the cutoff value for baselines in baseline measurements. If BLCUT is not set to zero, the DSP checks continuously each baseline value to see if it is outside of the limit set by BLCUT. If the baseline value is within the limit, it will be used to calculate the average baseline value. Otherwise, it will be discarded. Set BLCUT to zero to not check baselines, therefore reduce processing time.

ControlTask 6 can be used to measure baselines. Host computer can then histogram these baseline values and determine the appropriate value for BLCUT for each channel according to the standard deviation SIGMA for the averaged baseline value. BLCUT could be set to be three times SIGMA.

BASELINEPERCENT: This variable sets the DC-offset level in terms of the percentage of the ADC range. The DSP uses this variable to set the DC-offset level when it is executing the ADJUST OFFSET control task.

FTRIGOUTDELAY: Delay for sending trigger to system FPGA for coincidence logic. See user manual section 3.3.10 for details.

LOG2BWEIGHT: The PIXIE measures baselines continuously and effectively extracts DC-offsets from these measurements. The DC-offset value is needed to apply a correction to the computed energies. To reduce the noise contribution from this correction baseline samples are averaged in a geometric weight scheme. The averaging depends on LOG2BWEIGHT:

$$DC\_avg = DC\_avg + (DC - DC\_avg) * 2^{LOG2BWEIGHT}$$

DC is the latest measurement and DC_avg is the average that is continuously being updated. At the beginning, and at the resuming, of a run, DC_avg is seeded with the first available DC measurement.

As before, the DSP ensures that LOG2BWEIGHT will be negative. The noise contribution from the DC-offset correction falls with increased averaging. The standard deviation of DC_avg falls in proportion to sqrt(2^LOG2BWEIGHT).

When using a BLCUT value from a noise measurement (cf control task 6) the PIXIE will internally adjust the effective LOG2BWEIGHT for best energy resolution, up to the maximum value given by LOG2BWEIGHT. Hence, the Log2Bweight setting should be chosen at low count rates (dead time < 10%). Best energy resolutions are typically obtained at values of -3 to -4, and this parameter does not need to be adjusted afterwards.

PREAMPTAU: Preamplifier exponential decay time.
This variable is used to store the preamplifier decay time. It is stored in the DSP as a 32-bit floating point number (IEEE standard floating point format). The DSP uses this variable to compute coefficients for the event energy calculations.

XAVG: unused

FASTTRIGBACKLEN: Length of trigger for coincidence logic in system FPGA. See user manual section 3.3.10 for details.

CFDDELAY: The CFD algorithm builds the difference of original and delayed/scaled ADC signal. This variable defines the delay. See user manual for details.

CFDSCALE: The CFD algorithm builds the difference of original and delayed/scaled ADC signal. This variable defines the scale. See user manual for details.

EXTTRIGSTRETCH: This parameter is used to stretch the module validation trigger pulse. Only relevant when module validation is required by setting bit 11 CCSRA_GLOBTRIG

VETOSTRETCH: This parameter is used to stretch the veto pulse for this channel.

EXTERNDELAYLEN: This parameter 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.

QDCLEN0-7: Length of the QDC sums.

4.3.3 Module output parameters

REALTIMEA:
REALTIMEB: The 64-bit real time clock. A, B are the high and low word, respectively. The clock is zeroed on power up, and also in response to a synchronous data acquisition start when InSynch was set to 0 prior to the run start.

RealTime = (RealTimeA * 2^{32} + RealTimeB) * 10ns

RUNTIMEA:
RUNTIMEB: The 64-bit run time clock. A, B words are as for the RealTime clock. This time counter is active only while a data acquisition run is in progress and thus counts the elapsed run time. Compute the run time using the following formula:

RunTime = (RunTimeA * 2^{32} + RunTimeB) * 10ns

SYNCHDONE: This variable is used to indicate whether the synchronization interrupt has successfully occurred in the DSP when a run start request was issued with SYNCHWAIT=1. If the DSP is stuck in an infinite loop caused by a malfunctioning Busy-Synch loop, indicated by the value of SynchDone being always 0, the module should be rebooted and then the reason which caused the malfunctioning Busy-Synch loop should be investigated.

Below follow the addresses and lengths of a number of data buffers used by the DSP program. The addresses are generated by the assembler/linker when creating the executable. On power up the DSP code makes these values accessible to the user. Note that the addresses will typically change with every new compilation. Therefore your code should never assume to find any given buffer at a fixed address.

USEROUT: 16 words of user output data, which may be used by the user written DSP code.

HARDWAREID: ID of the hardware version.
HARDVARIANT: Variant of the hardware.

FIFOLENGTH: Length of the onboard FIFOs, measured in storage locations.

DSPRELEASE: DSP software release number.
DSPBUILD: DSP software build number.

4.3.4 Channel output parameters

The following channel variables contain run statistics. Again the variable names carry the channel number as a suffix. For example the LIVETIME words for channel 2 are LIVETIMEA[2], LIVETIMEB[2]. Channel number runs from 0 to 15.

LIVETIMEA:
LIVETIMEB: Total live time as measured by the trigger/filter FPGA of that channel. It excludes times during which the input signal is out of the ADC's voltage range, or when the run was stopped. Convert the two LiveTime words into a live time using the formula:

LiveTime = (LiveTimeA * 2^{32} + LiveTimeB) *10ns *(100 MHz or 500 MHz modules)*
or
LiveTime = (LiveTimeA * 2^{32} + LiveTimeB) *8ns *(250 MHz modules)*

FASTPEAKSA: The number of events detected by the fast filter is:
FASTPEAKSB: NumEvents = FASTPEAKSA* 2^{32} + FASTPEAKSB

CHANEVENTSA:
CHANEVENTSB: Total number of events that have been processed by the DSP for a given channel:
ChanEvents = (ChanEventsA * 2^{32} + ChanEventsB)

4.4 DSP Control Tasks

The DSP can execute a number of control tasks, which are necessary to control hardware blocks that are not directly accessible from the host computer. The most prominent tasks are those to set the DACs and program the trigger/filter FPGAs. The following is a list of control tasks that will be of interest to the programmer.

To start a control task, set RUNTASK=0 and choose a CONTROLTASK value from the list below. Then start a run by setting bit 0 in the control and status register (CSR).

Control tasks respond within a few hundred nanoseconds by setting the RUNACTIVE bit (#13) in the CSR. The host can poll the CSR and watch for the RUNACTIVE bit to be deasserted. All control tasks indicate task completion by clearing this bit.

Execution times vary considerably from task to task, ranging from under a microsecond to 10 seconds. Hence, polling the CSR is the most effective way to check for completion of a control task.

Control Task 0: **SetDACs**
Write the OFFSETDAC values of all channels into the respective DACs. Reprogramming the DACs is required to make effective changes in the values of the variables OFFSETDAC{0…15}.

Control Task 1: **Enable Input**
This control task was declared in pixie16app_defs.h, but it was not implemented in the DSP code nor it is being used.

Control Task 3: **Ramp offset DAC**
Ramp offset DACs of a module from 0 to 65535 at a step size of 64. Baseline values are captured at each offset DAC level, and a plot of baseline values versus offset DAC levels can then be used to find the appropriate offset DAC value for a given baseline level. The number of baseline values for each channel that are available after calling this control task is 1024. So for 16 channels, there will be a total of 16384 baseline values to read out from the DSP internal memory.

Control Task 4: **Untriggered ADC Traces**
This task provides ADC values measured on all 16 channels and gives the user an idea of what the noise and the DC-levels in the system are. This function samples 8192 ADC words (16-bit) for each of the 16 channels. The XWAIT variable determines the time between successive ADC samples (samples are XWAIT * 10ns apart). In the Pixie-16 VB demo interface XWAIT can be adjusted through the dT variable in the ADC Trace Display panel. The results are written to the 65536 words (32-bit) long I/O buffer. Use this function to check if the offset adjustment was successful.

Control Task 5: **ProgramFiPPI**
This task writes all relevant data to the FiPPI registers in the FPGA, for example, the fast filter rise time and flat top, etc. These data have already been converted to FPGA recognizable values by the APP DLL/Clib functions before they are downloaded to the DSP, e.g. slow filter rise time has been converted from units of μs to units of FPGA clock cycles.

Control Task 6: **Measure Baselines**

This routine is used to collect baseline values. Currently, DSP collects three words, time stamp (low 32-bit word), time stamp (high 16-bit word), and baseline value, for each baseline. 3640 baselines are collected until the 65536-word (32-bit) I/O buffer is almost completely filled. The host computer can then read the I/O buffer and retrieve the baseline values. Please note, the baseline values retrieved from the DSP are 32-bit IEEE floating point numbers, and they have to be converted to decimal values first before they can be used for analysis.

Baseline values can then be statistically analyzed to determine the standard deviation associated with the averaged baseline value and to set the BLCUT. BLCUT should be about 3 times the standard deviation. Baseline values can also be plotted against time stamp to explore the detector performance. BLCUT should be set to zero while running Control Task 6.

Control Task 7: **Automatically adjust DC offsets**

This task automatically adjusts the DC offsets of each of the 16 channels of a Pixie-16 module through setting Offset DACs to appropriate values. The signal baseline levels after the adjustment will be determined by DSP parameter BaselinePercent. For instance, if the BaselinePercent is set to 10% on a Pixie-16 module installed with 14-bit ADCs, the DC offset will be about 1640 after successful DC offset adjustment.

Control Task 8: **Find decay time Tau**

This task tries to find the single exponential decay time constant of the input analog signals of each channel. It uses the untrigged ADC traces to find and fit the input pulses, so reasonably high count rate for the input signals is required to find the decay time successfully.

Control Task 23: **Reset ADCs (Rev. F modules only)**

This task resets the ADCs of a Pixie-16 module, and is applicable to Rev. F modules only. This task is only used while booting the Rev. F modules.

5 Control Registers

A Pixie-16 module has several control registers that are used to activate runs, download DSP code or reset DSP, control pullup resistor for the trigger lines on the backplane, and/or indicate module status to the host computer. Software on the host computer can set and/or read bits in these registers to control the operation of the Pixie-16 module or monitor its internal status.

5.1 PCI Host Control Register

Bit name	Bit #	Direction	Description
RUNENABLE	0	Read/Write	Enable run: =1: start a run; =0: stop a run
Download DSP code	1	Read/Write	Enable DSP code download: =1: Download DSP code; =0: DSP code is running
PCI Active read or write	2	Read/Write	Indicate PCI I/O is active: =1: PCI is reading or writing memory; =0: no PCI read or write
Pull-up resistor control	3	Read/Write	Control pull-up for the SYNC lines: =1: wired-OR trigger lines on the backplane connect to a pullup resistor; =0: not connected
Reset DSP	4	Read/Write	Generate a pulse to reset DSP: calling function Pixie_ReadCSR to read the value of this control register, and then set this bit to 1, and finally write the control register back to the System FPGA by calling function Pixie_WrtCSR
Reserved	5		
EXTFIFO_WML	6	Read only	External FIFO watermark level indicator: =1: number of data words in the external FIFO exceeds the watermark level, so the external FIFO is ready to be read out; =0: number of data words in the external FIFO is still below the watermark level, so the external FIFO is not yet ready to be read out
Reserved	[12:7]		
RUNACTIVE	13	Read only	Run active indicator: =1: run is active; =0: run has ended
Reserved	14		
CLREXTMEM_ACTIVE	15	Read only	Clearing external memory active indicator: =1: clearing external memory is still ongoing; =0: clearing external memory has completed