Pixie-4 Express

Programmer's Manual

Version 4.46

December 15, 2017

Hardware Revision: B

Software Revision: 4.46

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

1 Overview

This manual is divided into three major sections. The first section is a description of the Pixie application program interface (API). Advanced users can build their own user interface using these API functions. The second section is a reference guide to program the Pixie modules using the Pixie-4 API. This will be interesting to those users who want to integrate the Pixie modules into their own data acquisition system. The third section describes the variables controlling the functions of the Pixie modules. Advanced and curious users can use this section to better understand the operation of the Pixie modules. Additionally, this manual includes instructions on how to integrate custom user code into the digital signal processor (DSP) and Igor.

The API and most of the variables are identical for the Pixie-4, the Pixie-4 Express, and the Pixie-500 Express. Most of the text will mention "Pixie modules" and stand for all of the 3 devices. For functions and variables specific to a particular model, the use and value are described explicitly and the text is highlighted.

1.1 Changes in Version 3.00 and 4.00

In version 3.00, WinDriver libraries and corresponding functions have been added to support the PCI Express I/O for the Pixie-500 Express modules. While many functions and features are identical, and top level API functions are almost unchanged, the following differences exist:

- Pixie-500 Express list mode runs are type 0x400, uninterupted transfer of data to host
- List mode 0x400 output data is written as one file per module
- List mode 0x400 "events" are single channel records, with a new data structure as described in the user manual. However, the same run tasks in Pixie_Acquire_Data are used to process list mode files and return data to a host program.
- The file name list has been expanded to include files for Pixie-4, Pixie-500, and Pixie-500 Express. (N_BOOT_FILES = 16)
- Pixie_Acquire_Data should always be called with a valid filename and the lower 12 bits set according to the runtype. In the past this was not always required, now it is used e.g. in Start Run to distinguish the different list mode types and write a file header in 0x400 list mode runs.
- The number of DSP parameters has been increased to 512 (was 416). Some parameters shifted position, others have been removed. Corresponding user global module and channel parameters have been removed as well, see tables 3.5 b and c.

In version 4.00, support for the Pixie-4 Express was added and support for the Pixie-500 was removed. The Pixie-4 Express uses the same drivers as the Pixie-500 Express, and overall changes are minor. In addition, data acquisition has been revised to make use of interrupts issued by the modules' DMA sequencer. The changes include

- In the file name list, Pixie-4 Express replaces Pixie-4 Rev B system files. Pixie-500 files are no longer loaded
- Increased the number of module globals to 128 x17

- Revised polling for list mode runs (poll function simply returns number of spills written to file by interrupt service routine)
- In list mode data files, the run concludes with an "End Of Run" record – equivalent to 32 word event header with special event pattern
- In both writing and parsing of list mode data, data is analyzed for transmission errors (buffer QC). Event patterns are marked with an error bit to indicate such errors found (and corrected).
- Some system variables have been moved to a new module parameter C_CONTROL (so that it can be saved to the .set file), which also specifies options for printing error and debug messages, buffer QC, and polling mode.
- List mode 0x402 collects data from all 4 channels as a group and stores it in 4-channel event records
- (4.20) Added NPPI and PPR to run statistics

2 Pixie API

The Pixie API consists of a set of C functions for building various coincidence data acquisition applications. It can be used to configure Pixie modules, make MCA or list mode runs and retrieve data from the Pixie modules. The API can be compiled as a WaveMetrics Igor XOP file which is currently used by the Pixie Viewer, or a dynamic link library (DLL). It can also be compiled for Linux and a sample ROOT interface is available from XIA. In order to better illustrate the usage of these functions, an overview of the Pixie operation is given below and the usage of these functions is described in the following.

At first the Pixie API needs to be initialized. This is a process in which the names of system configuration files are downloaded to the API. The function **Pixie_Hand_Down_Names** is used to achieve this.

The second step is to boot the Pixie modules. It involves initializing each PXI(e) slot where a Pixie module is installed, 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 (DACs). All this has been encapsulated in a single function **Pixie_Boot_System**.

Now, the instrument is ready for data acquisition. The function used for this purpose is **Pixie_Acquire_Data**. By setting different run types, it can be used to start, stop or poll a data acquisition run (list mode run, MCA run, or special task runs like acquiring ADC traces). It can also be used to retrieve list mode or histogram data from the Pixie modules.

After checking the quality of a MCA spectrum, a Pixie user may decide to change one or more settings like energy filter rise time or flat top. The function used to change Pixie settings is **Pixie_User_Par_IO**. This function converts a user parameter like energy filter rise time in μs into a number understood by the Pixie hardware or vice versa.

Pixie_Get_Par_Idx and **Pixie_Get_Par_Name** are the two functions employed to assist in using **Pixie_User_Par_IO** when the Pixie C library is linked dynamically to the user's application (.dll library). In this case the names of the user-controlled variables and their indexes in arrays are not accessible to the user's application. The user, therefore, can inquire the library regarding a particular parameter of interest either through an index or a string.

Another function, **Pixie_Buffer_IO**, is used to read data from DSP's internal memory to the host or write data from the host into the internal memory. This is useful for diagnosing Pixie modules by looking at their internal memory values. The other usage of this function is to read, save, copy or extract Pixie's configurations though its settings files.

In a multi-module Pixie-4 system, it is essential for the host to know which module or channel it is communicating to. The function **Pixie_Set_Current_ModChan** is used to set the current module and channel. The detailed description of each function is given below.

2.1 Pixie_Hand_Down_Names

Syntax

```
S32 Pixie_Hand_Down_Names (
U8 *Names[],       // An array containing the names to be downloaded
U8 *Name);         // A string indicating the type of names (file or
                   // variable names) to be downloaded
```

Description

Use this function to download the file or variable names from the host user interface to the Pixie API. The API needs these file names so that it can read the Pixie hardware configurations from the files stored in the host computer and download these configurations to the Pixie. The variable names are used by the API to obtain the indices of the DSP variables when the API converts user variable values into DSP variable values or vice versa.

Note: Since variable names are now defined in the C library itself, this function is only required for the Igor and LabView host interfaces. Other, C based interfaces usually do not require it, see the sample programs for examples on how to specify the file names in the library array "Boot_File_Name_List".

Parameter description

Names is a two dimensional string array containing either the file names or the variable names. The API will know which type of names is being downloaded by checking the other parameter *Name*:

1. **ALL_FILES**: This indicates we are downloading boot files names. In this case, *Names* is a string array which has N_BOOT_FILES elements. Currently N_BOOT_FILES is defined as 16. The elements of *Names* are the names of FPGA files, DSP executable code binary file, DSP I/O parameter values file, DSP code I/O variable names file, and DSP code memory variable names file, see table 3.2. All file names should contain the complete path name.
2. **SYSTEM**: This indicates we are downloading System_Parameter_Names. System_Parameter_Names are those global variables that are applicable to all modules in a Pixie system, e.g. number of Pixie modules in the chassis, etc. System_Parameter_Names currently can hold 64 names. If less than 64 names are needed (which is the current case), the remaining names should be defined as empty strings.
3. **MODULE**: This indicates we are downloading Module_Parameter_Names. Module_Parameter_Names are those global variables that are applicable to each individual module, e.g. module number, module CSR, coincidence pattern, and run type, etc. Module_Parameter_Names can currently hold 128 names. If less than 128 names are needed (which is the current case), the remaining names should be defined as empty strings.
4. **CHANNEL**: This indicates we are downloading Channel_Parameter_Names. Channel_Parameter_Names are those global variables that are applicable to individual channels of the Pixie modules, e.g. channel CSR, filter rise time, filter flat top, voltage gain, and DC offset, etc. Channel_Parameter_Names currently can hold 64 names. If less than 64 names are needed (which is the current case), the remaining names should be defined as empty strings.

A detailed description of System/Module/Channel_Parameter_Names is given in Table 3.5.

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid name	Check the second parameter *Name*

Usage example

```
S32 retval;

// download system parameter names; define System_Parameter_Names first
retval = Pixie_Hand_Down_Names(System_Parameter_Names, "SYSTEM");
if(retval < 0)
{
  // error handling
}

// download module parameter names; define Module_Parameter_Names first
retval = Pixie_Hand_Down_Names(Module_Parameter_Names, "MODULE");
if(retval < 0)
{
  // error handling
}

// download channel parameter names; define Channel_Parameter_Names
// first
retval = Pixie_Hand_Down_Names(Channel_Parameter_Names, "CHANNEL");
if(retval < 0)
{
  // error handling
}

// download boot file names; define All_Files first
retval = Pixie_Hand_Down_Names(All_Files, "ALL_FILES");
if(retval < 0)
{
  // error handling
}
```


2.2 Pixie_Boot_System

Syntax

```
S32 Pixie_Boot_System (
U16 Boot_Pattern);        // The Pixie-4 boot pattern
```

Description

Use this function to boot all Pixie modules in the system. Before booting the modules, it scans all PXI(e) crate slots and finds the address for each slot where a Pixie module is installed.

Parameter description

Boot_Pattern is a bit mask used to control the boot pattern of Pixie modules:

Bit 0: Boot communication FPGA

Bit 1: Boot signal processing FPGA

Bit 2: Boot DSP

Bit 3: Load DSP parameters

Bit 4: Apply DSP parameters (call Set_DACs and Program_FIPPI)

Bit 5: Release Handle

Under most of the circumstances, the first 5 tasks should be executed to initialize the Pixie modules, i.e. the *Boot_Pattern* should be 0x1F. Programs executing only a subset of functions, for example starting a list mode run after the Pixie modules have been booted by another program, may have to use *Boot_Pattern* 0x0 to initialize the communication (without rebooting the modules). *Boot_Pattern* 0x20 is used to release the handle for the device; this is useful when switching e.g. from the Igor based Pixie Viewer to the LabView interface.

Return values

Value	Description	Error Handling
0	Success	None
-1	Unable to scan crate slots	Check PXI slot map
-2	Unable to read communication FPGA configuration (Rev. B)	Check comFPGA file
-3	Unable to read communication FPGA configuration (Rev. C)	Check comFPGA file
-4	Unable to read signal processing FPGA configuration	Check SPFPGA file
-5	Unable to read DSP executable code	Check DSP code file
-6	Unable to read DSP parameter values	Check DSP parameter file
-7	Unable to initialize DSP parameter names	Check DSP .var file
-81	Downloading communication FPGA failed	See error log
-82	Downloading FiPPI configuration failed	See error log
-83	Downloading DSP code failed	See error log
-84	Failed to set DACs	See error log
-85	Failed to program FiPPI	See error log
-86	Failed to enable detector input	See error log
-87	Incorrect boot pattern	Check *Boot_Pattern*
-13	Failed to release handle	Power cycle module

Usage example

```
S32 intval;
retval = Pixie_Boot_System(0x1F);
if(ret < 0)
    // error handling
```


2.3 Pixie_User_Par_IO

Syntax

```
S32 Pixie_User_Par_IO (
double *User_Par_Values,   // A double precision array containing
                           // the user parameters to be transferred
U8 *User_Par_Name,         // A string variable indicating which user
                           // parameter is being transferred
U8 *User_Par_Type,         // A string variable indicating which type
                           // of user parameters is being transferred
U16 Direction,             // I/O direction (read or write)
U8 ModNum,                 // Number of the module to work on
U8 ChaNum);                // Channel number of the Pixie module
```

Description

Use this function to transfer user parameters between the user interface, the API and DSP's I/O memory. Some of these parameters (`User_Par_Type = "SYSTEM"`) are applicable to all Pixie modules in the system, like the total number of Pixie modules in the system. Other parameters (`User_Par_Type = "MODULE"`) are applicable to a whole Pixie module (independent of its four channels), e.g. coincidence pattern, Module CSRA, etc. The final set of parameters (`User_Par_Type = "CHANNEL"`) are applicable to each individual channel in a Pixie module, e.g. energy filter settings or voltage gain, etc. For those parameters which need to be transferred to or from DSP's internal memory (other parameters such as number of modules are only used by the API), this function will call another function **UA_PAR_IO** which first converts these parameters into numbers that are recognized by both the DSP and the API then performs the transfer.

Parameter description

User_Par_Values is a double precision array containing the parameters to be transferred. Depending on another input parameter *User_Par_Type*, different *User_Par_Values* array should be used. Totally three *User_Par_Values* arrays should be defined and all of them are one-dimensional arrays. The corresponding relationship between *User_Par_Values* and *User_Par_Type* is shown in Table 2.1.

User_Par_Type	User_Par_Values		
	Name	Size	Data Type
SYSTEM	System_Parameter_Values	64	Double precision
MODULE	Module_Parameter_Values	128 x 17	Double precision
CHANNEL	Channel_Parameter_Values	64×17×4	Double precision

Table 2.1: The Combination of User_Par_Name and User_Par_Values.

The way to fill the *Channel_Parameter_Values* array is to fill the channel first then the module. For instance, first 64 values are stored in the array for channel 0, and then repeat this for other three channels. After that, 64×4 values have been filled for module 0. Then repeat this for the remaining modules. For the *Module_Parameter_Values* array, first store 64 values for module 0, and then repeat this for the other modules.

User_Par_Name is the name of the variable which is to be transferred. It is one element of one of the arrays System/Module/Channel_Parameter_Names listed in Table 3.5. In addition, the following keywords are recognized:

Key Word	**Action**
ALL_SYSTEM_PARAMETERS	(read only) read all system parameters
ALL_MODULE_PARAMETERS	(read only) read all module parameters of a module, except RUN_TYPE
MODULE_RUN_STATISTICS	(read only) read all module run statistics parameters of a module
ALL_CHANNEL_PARAMETERS	(read only) read all channel parameters of a channel
CHANNEL_RUN_STATISTICS	(read only) read all channel run statistics parameters of the 4 channel in a module

Note: The key words "UPDATE_FILTERRANGE_PARAMS" and "FIND_TAU" are obsolete. The former action is performed automatically after changing FILTERRANGE. The latter is now a control task.

direction indicates the transfer direction of parameters:

0 - download (write) parameters from the user interface to the API;

1 - upload (read) parameters from the API to the user interface.

ModNum is the number of the Pixie module being communicated to.

ChanNum is the channel number of the Pixie module being communicated to.

Return values

Value	Description	Error Handling
0	Success	None
-1	Null pointer for User_Par_Values	Check User_Par_Values
-2	Invalid user parameter name	Check User_Par_Name
-3	Invalid user parameter type	Check User_Par_Type
-4	Invalid I/O direction	Check direction
-5	Invalid Pixie module number	Check ModNum
-6	Invalid Pixie channel number	Check ChanNum

Usage example

```
U16 direction, modnum, channum;
S32 retval;
```



```
direction = 0;   // download
modnum = 0;      // Module #0
channum = 1;     // Channel #1

// set module parameter COINCIDENCE_PATTERN to 0xFFFF
Module_Parameter_Values[Coincidence_Pattern_Index]=0xFFFF;

// download COINCIDENCE_PATTERN to the DSP
retval = Pixie_User_Par_IO(Module_Parameter_Values,
"COINCIDENCE_PATTERN", "MODULE", direction, modnum, channum);
if(retval < 0)
{
  // error handling
}

// set channel parameter ENERGY_RISETIME to 6.0 µs
Channel_Parameter_Values[ENERGY_RISETIME_Index]=6.0;

// download ENERGY_RISETIME to DSP
retval = Pixie_User_Par_IO(Channel_Parameter_Values, "ENERGY_RISETIME",
                                "CHANNEL", direction, modnum, channum);
if(retval < 0)
{
  // error handling
}
```

2.4 Pixie_Get_Par_Name

Syntax

```
S32 Pixie_Get_Par_Name (
U16 Idx,              // Parameter index in the array of parameter
                      // names
U8 *User_Par_Type,    // A string variable indicating which type
                      // of user parameters is being requested
U8 *User_Par_Name);   // The pointer to a string array to contain
                      // the variable name being requested
```

Description

Use this function to identify user parameters by type ("SYSTEM", "MODULE", "CHANNEL") and index in the type-associated list. The typical use of this function is to check whether a particular variable exists in the list, or to create a local copy of the entire list. The parameter name then can be used in Pixie_User_Par_IO to change the value of the parameter

Parameter description

Idx is an unsigned short variable indicating the sequential number of a user-controlled parameter in the list of SYSTEM, MODULE or CHANNEL type parameters.

User_Par_Type is a string describing the type of the parameter in consideration: "SYSTEM", "MODULE" or "CHANNEL".

User_Par_Name is a character array of at least size 80 to contain the name of the requested parameter.

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid system parameter index	Check if Idx exceeds 63
-2	Invalid module parameter index	Check if Idx exceeds 127
-3	Invalid channel parameter index	Check if Idx exceeds 63
-4	Invalid parameter type	Check if "SYSTEM", "MODULE" or "CHANNEL"
-5	NULL parameter name array ptr	Check if User_Par_Name is allocated

Usage example

```
// Check if parameter exists
U8 *ParName = NULL;
U16 i = 0;

ParName = malloc(80 * sizeof(char));
if (!ParName) {
    //Not enough memory error handling
}

while(Pixie_Get_Par_Name(i,"MODULE", ParName) != -1) {
    if(!strcmp(ParName, "FILTER_RANGE")) {
      printf("FILTER_RANGE exists\n");
      break;
    }
    i++;
}
if(i > 63) {
  //error handling
  }
```

2.5 Pixie_Get_Par_Idx

Syntax

```
S16 Pixie_Get_Par_Idx (
U8 *User_Par_Name,       // A string variable indicating
                         // the variable name being requested
U8 *User_Par_Type);      // A string variable indicating which type
                         // of user parameters is being requested
```

Description

Use this function to identify parameter index by type ("SYSTEM", "MODULE", "CHANNEL") in the type-associated list. The typical use of this function is to determine the index in `User_Par_Values of` **Pixie_User_Par_IO** to transfer parameter values to and from the device.

Parameter description

User_Par_Type is a string describing the type of the parameter in consideration: "SYSTEM", "MODULE" or "CHANNEL".

User_Par_Name is the name of the variable for which the index is being requested.

Return values

Value	Description	Error Handling
Idx	Success	None
-1	Invalid parameter type	Check if "SYSTEM", "MODULE" or "CHANNEL"
-2	Invalid system parameter name	Check spelling
-3	Invalid module parameter name	Check spelling
-4	Invalid channel parameter name	Check spelling

Usage example

```
// Write to module
SysParValues[Pixie_Get_Par_Idx("NUMBER_MODULES",   "SYSTEM")] = 1;
SysParValues[Pixie_Get_Par_Idx("OFFLINE_ANALYSIS", "SYSTEM")] = 1;
SysParValues[Pixie_Get_Par_Idx("AUTO_PROCESSLMDATA", "SYSTEM")] = 0;
SysParValues[Pixie_Get_Par_Idx("MAX_NUMBER_MODULES", "SYSTEM")] = 7;
SysParValues[Pixie_Get_Par_Idx("KEEP_CW",            "SYSTEM")] = 1;
Pixie_User_Par_IO(SysParValues,"NUMBER_MODULES", "SYSTEM", MOD_WRITE, 0, 0);
Pixie_User_Par_IO(SysParValues,"OFFLINE_ANALYSIS","SYSTEM", MOD_WRITE, 0, 0);
Pixie_User_Par_IO(SysParValues,"AUTO_PROCESSLMDATA","SYSTEM",MOD_WRITE, 0,0);
Pixie_User_Par_IO(SysParValues,"MAX_NUMBER_MODULES","SYSTEM",MOD_WRITE,0, 0);
Pixie_User_Par_IO(SysParValues,"KEEP_CW",        "SYSTEM", MOD_WRITE, 0, 0);
```


2.6 Pixie_Acquire_Data

Syntax

```
S32 Pixie_Acquire_Data (
U16 Run_Type,      // Data acquisition run type
U32 *User_data,    // An unsigned 32-bit integer array containing the
                   // data to be transferred
U8 *file_name,     // Name of the file used to store list mode or MCA
                   // histogram data
U8 ModNum);        // The number of the Pixie module
```

Description

Use this function to acquire ADC traces, MCA histogram, or list mode data. The string variable *file_name* should always be specified – for start/stop poll of acquisition runs it is the filename for the data, in other cases it is the filename to be processed. In rare cases, *file_name* can be specified as an empty string. The unsigned 32-bit integer array *User_data* is only used for acquiring ADC traces (control task 0x84), reading out list mode data or MCA spectra. In all other cases, *User_data* can be any unsigned integer array with arbitrary size. Make sure that *User_data* has the correct size and data type before reading out ADC traces, list mode data, or MCA spectrum.

Parameter description

file_name is a string variable which specifies the name of the output file. It needs to have the complete file path.

ModNum is the number of the module addressed, counting from 0 to (number of modules - 1). If *ModNum* == (number of modules), all modules are addressed in a for loop, however this option is not valid for all RunTypes.

Run_Type is a 16-bit word whose lower 12-bit specifies the type of either data run or control task run and upper 4-bit specifies actions (start\stop\poll) as described below. Controltasks are described in detail in section 4.5.

Lower 12-bit:

0x100,0x101,0x102,0x103	list mode runs (Pixie-4)
0x400,0x401	list mode runs (Pixie-4 Express, Pixie-500 Express)
0x301	MCA run
0x1 -> 0x7F	control task runs handled by DSP
0x80 -> 0xFF	control task runs handled by C library

Upper 4-bit:	
0x0000	start a control task run
0x0000	set offset DACs[1]
0x0001	connect inputs (Pixie-4 only)
0x0002	disconnect inputs (Pixie-4 only)
0x0005	program Fippi FPGA[1]
0x0006	measure baselines
0x0016	test writing to external list mode memory
0x001A	test writing to external histogram memory
0x0080	measure baselines and compute BLcut for all modules
0x0081	find values of τ for all modules
0x0083	adjust offsets[2]
0x0084	collect ADC traces[2]
0x0085	adjust offsets in DSP (faster, not for Pixie-4)
0x1000	start a new data run[3]
0x2000	resume a data run[3]
0x3000	stop a data run[3]
0x4000	poll run status
0x4000	check if run in progress, return 0 or 1
0x40FF	check if run in progress, return CSR value
0x4400	check if data is ready, if yes save to binary file
0x4401	check if data is ready, if yes save to ASCII file
0x5000	read histogram data and save it to a file
0x6000	read list mode buffer data and save it to a file[3]
0x7000	offline list mode data parse routines
0x7001	parse list mode data file
0x7002	locate traces
0x7003	read traces (Pixie-4 only)
0x7004	read energies
0x7005	read PSA values
0x7006	read extended PSA values
0x7007	locate events
0x7008	read event
0x7009	extract channel header data
0x7010	custom parsing with user defined functions
0x7020	perform data error check and save as new file
0x7021	process .b## file and save in (Pixie-4) .bin format
0x7030	process .b## file's waveforms for PSA into dt3 file
0x8000	manually read MCA histogram from a MCA file

[1] Should be called after most parameter changes applied directly to the DSP (using Pixie_Buffer_IO). Automatically called after parameter changes handled by the C library (using Pixie_User_Par_IO).

[2] Combined action of DSP and C library. DSP will perform the controltask 3 or 4 described in section 4.5 and C library will use the results. For backwards compatibility, RunType 0x0003 or 0x0004 are translated into 0x0083 or 0x0084, respectively in the C library.

[3] Lower 3 digits must be set to selected runtype

0x9000	external memory (EM) I/O
0x9001	read histogram memory section of EM
0x9002	write to histogram memory section of EM
0x9003	read list mode memory section of EM
0x9004	write to list mode memory section of EM
0x9005	read out 2D section EM
0x9006	write to 2D section of EM
0x9007	read out first N word of MCA section of EM (N is word 0 of User_data array

User_data has the following format for the run types listed below:

0x84: Get ADC traces

Length must be ADCTraceLen*NumberOfChannels, i.e. 8192 * 4 = 32Ki.[4]

All array elements are return values.

The Nth 8Ki of data are the ADC trace of channel N.

0x7001:Parse list mode data file

Length must be 2* PRESET_MAX_MODULES, i.e. 34

All array elements are return values.

User_data[i] = NumEvents of module i

User_data[i+PRESET_MAX_MODULES] = TotalTraces of module i

0x7002: Locate Traces of all events

Length must be (TotalTraces of ModNum)*3*NumberOfChannels

All array elements are return values.

User_data[i*3n] = Location of channel n's trace in file for event i (word number)

User_data[i*3n+1] = length of channel n's trace

User_data[i*3n+2] = energy for channel n

0x7003: Read Traces of one event

Length must be (NumberOfChannels*2+combined tracelength of channels)

First (NumberOfChannels*2) elements are input values:

User_data[2n] = Loc. of ch. n's data in file for selected event (word number)

User_data[2n+1] =length of channel n's trace

The remaining array elements are return values.

User_data[8 …] = Trace data of channel 0 followed by channels 1,2, and 3.

0x7004: Read Energies of all events

Length must be (NumEvents of ModNum * NumberOfChannels)

[4] In this manual, we use the IEC notation "Ki" for 1024 and the SI notation "k" for 1000

All array elements are return values.

User_data[i*4+n] = energy of channel n for event i

0x7005: Read PSA values of all events

Length must be (NumEvents of ModNum*2 * NumberOfChannels)

All array elements are return values.

User_data[i*2n] = XIAPSA word of channel n for event i

User_data[i*2n+1] = UserPSA word of channel n for event i

0x7006: Read extended PSA values of all events

Length must be (NumEvents of ModNum*8 * NumberOfChannels)

All array elements are return values.

User_data[i*8n] = Channel Header 0 or 4/5 (usually timestamp) of ch n for event i
16 bit for Pixie-4, 32 bit for Pixie-500e and Pixie-4 Express

User_data[i*8n+1] = Channel Header 1 or 8 (energy) of ch. n for event i

User_data[i*8n+2] = Channel Header 2 or 11 (XIAPSA) of ch. n for event i

User_data[i*8n+3] = Channel Header 3 or 10 (UserPSA) of ch. n for event i

User_data[i*8n+4] = Channel Header 4 or 12 (ExtendedPSA1) of ch. n for event i

User_data[i*8n+5] = Channel Header 5 or 13 (ExtendedPSA2) of ch. n for event i

User_data[i*8n+6] = Channel Header 6 or 14 (ExtendedPSA3) of ch. n for event i

User_data[i*8n+7] = Channel Header 7 or 15 (RealTime High or ExtendedPSA4) of channel n for event i

Notes: Headers 1-7 can be modified by DSP user routine in customized code.

In Pixie-4 standard DSP code, Header 7 contains RealTime High word

0x7007: Locate all events

Length must be (NumEvents of ModNum)*3

All array elements are return values.

User_data[i*3] = Location of event i in file (word number)

User_data[i*3+1] = Location of buffer header start for event i in file

User_data[i*3+2] = Length of event i (event header, channel header, traces)

0x7008: Read one event

Length must be (length of selected event) + 7 +36

(this is longer than actually used, but ensures enough room for channel headers in all runtypes)

First 4 elements are input values:

User_data[0] = Location of selected event in file (word number)

User_data[1] = Location of buffer header start for selected event in file

User_data[2] = Length of selected event

User_data[3] = Display coincidence window (in ticks), .b## file only.
If >64Ki, single event is returned, else up 4 closest in window.

The following array elements are return values.

User_data[1] is the ADC rate in MHz

User_data[3 …6] are the tracelengths of channel 0-3

User_data[7 …6+BHL] contain the buffer header for to the selected event

User_data[7+BHL .. 6+BHL+EHL] contain the event header

User_data[7+BHL+ELH .. 6+BHL+EHL+4*CHL] are the channel headers for channel 0-3; always 9 words per channel header, but in compressed runtypes some entries are be invalid

User_data[7+BHL+EHL+4*CHL …] contain the traces of channel 0-3,

. followed by some undefined values (use tracelength to parse traces)

Summary of *User_data* array in run task 0x7008

Location	P4 value (Header location)	P500e/P4e value (Header location)	Notes
0	Location of event	Location of event	
1	I: Location of event O: ADC rate in MHz	O: ADC rate in MHz	
2	Length of event	Length of event	
3	O: TL0	I: Coinc. Window (ticks) O: TL0	
4	TL1	TL1	
5	TL2	TL2	
6	TL3	TL3	
7	BUF_NDATA(BH0)	Unused	
8	Module number(BH1)	Module number(FH1)	
9	Run type (data format indicator): 0x100-0x403 (BH2)	Run type (data format indicator): 0x400-0x403 (FH2)	
10	BUF_TIMEHI (BH3)	TrigTimeHI (CH6)	High 16 bits of Trigger or Buffer time
11	BUF_TIMEMI (BH4)	TrigTimeMI (CH5)	Middle 16 bits of Trigger or Buffer time
12	BUF_TIMELO (BH5)	TrigTimeLO (CH4)	Lower 16 bits of Trigger or Buffer time
13	EVT_PATTERN (EH0)	EvtPattern and EvtInfo (CH1,0)	32bit for P4e, P500e 16bit for P4
14	EVT_TIMEMI (EH1)	TrigTimeMI (CH5)	Middle 16 bits of Trigger or Event time

15	EVT_TIMELO (EH2)	TrigTimeLO (CH4)	Lower 16 bits of Trigger or Event time
16	CHAN_NDATA (CH0)	Unused	Channel data length
17	CHAN_TRIGTIME (CH1)	TrigTimeMI and TrigTimeLO (CH5,4)	32bit for P4e, P500e 16bit for P4
18	CHAN_ENERGY (CH2)	CHAN_ENERGY (CH8)	energy
19	XIAPSA (CH3)	XIAPSA (CH11)	XIA PSA value (or user return)
20	USERPSA (CH4)	USERPSA (CH10)	USER PSA value (or user return)
21	Unused (CH5)	Unused (CH12)	(user return)
22	Unused (CH6)	Unused (CH13)	(user return)
23	Unused (CH7)	Unused (CH14)	(user return)
24	TIME HI (CH8)	Unused (CH15)	(user return) or High 16 bits of time

Channel header repeated 4 times. Followed by waveforms

0x7030: Process .b## file's waveforms for PSA into dt3 file

Length must be 17 or more. Definitions are as follows:

Location	P500e/P4e value (Header location)	Notes
0	reserved	
1	LoQ0	Sum 0 length
2	LoQ1	Sum 1 length
3	SoQ0	Sum 0 start
4	SoQ1	Sum 1 Start
5	RTlow	RT start level of full amplitude, in %
6	RThigh	RT end level of full amplitude, in %
7	PSAoption	0: Q1/Q0, 1: (Q1-Q0)/Q0
8	PSAdiv8	If 1: divide PSA result by 8
9	PSAletrig	If 1, use leading edge trigger
10	PSAth	Threshold for leading edge trigger
11-16	reserved	Carries output parameters internally

0x8000: Read MCA data from file (one module)

Length must be HISTOGRAM_MEMORY_LENGTH = 131072

0x9001: Read MCA data from one module
Length must be HISTOGRAM_MEMORY_LENGTH = 131072

0x9002: Write MCA data to one module (debug)
Length must be HISTOGRAM_MEMORY_LENGTH = 131072

0x9003: Read List mode data in external memory from one module
Length must be LIST_MEMORY_LENGTH = 131072

0x9004: Write List mode data to external memory one module (debug)
Length must be LIST_MEMORY_LENGTH = 131072

0x9005: Read 2D sectiondata from one module
Length must be MCA2D_MEMORY_LENGTH = 262144

0x9006: Write 2D section to one module (debug)
Length must be MCA2D_MEMORY_LENGTH = 262144

0x9007: Read first N words of MCA data from one module (debug)
First element is input value:

User_data[0] = Number of words to return (N)

Length must be MCA2D_MEMORY_LENGTH = 262144

Return values

Return values depend on the run type:

Run type = 0x0000

Value	Description	Error Handling
0	Success	None
-0x1	Invalid Pixie module number	Check ModNum
-0x2	Failure to adjust offsets	Reboot module
-0x3	Failure to acquire ADC traces	Reboot module
-0x4	Failure to start the control task run	Reboot module

Run type = 0x1000

Value	Description	Error Handling
0x10	Success	None
-0x11	Invalid Pixie module number	Check ModNum
-0x12	Failure to start the data run	Reboot module
-0x13	system contains both P4/500e and P4/500	Choose one type of module

-0x14	list mode file not found or created (0x40#)	Check file names
-0x15	list mode memory allocation failure	PC memory?
-0x16	error setting up or initializing DMA transfer	Reboot module

Run type = 0x2000

Value	Description	Error Handling
0x20	Success	None
-0x21	Invalid Pixie module number	Check ModNum
-0x22	Failure to resume the data run	Reboot module

Run type = 0x3000

Value	Description	Error Handling
0x30	Success	None
-0x31	Invalid Pixie module number	Check ModNum
-0x32	Failure to end the run	Reboot module
-0x34	DMA transfer timeout	Reboot module
-0x35	memory allocation failure	PC memory?
-0x36	error setting up or initializing DMA transfer	Reboot module
-0x37	error starting dummy run after DMA run	Reboot module
-0x38	error writing DMA data to file	

Run type = 0x4000

Value	Description	Error Handling
0	No run is in progress	N/A
1	Run is in progress	N/A
CSR value	When run type = 0x40FF	N/A
Total no. spills written to file	When run type = 0x4401 or 0x4401	
-0x41	Invalid Pixie module number	Check ModNum

Run type = 0x5000

Value	Description	Error Handling
0x50	Success	None
-0x51	Failure to save histogram data to a file	Check the file name

Run type = 0x6000

Value	Description	Error Handling
0x60	Success	None
-0x61	Failure to save list mode data to a file	Check the file name

Run type = 0x7000

Value	Description	Error Handling
0x70	Success	None
-0x71	Failure to parse the list mode data file	Check list mode data file

-0x72	Failure to locate list mode traces	Check list mode data file
-0x73	Failure to read list mode traces	Check list mode data file
-0x74	Failure to read event energies	Check list mode data file
-0x75	Failure to read PSA values	Check list mode data file
-0x76	Failure to read extended PSA values	Check list mode data file
-0x77	Failure to locate events	Check list mode data file
-0x78	Failure to read events	Check list mode data file
-0x79	Invalid list mode parse analysis request	Check run type

Run type = 0x8000

Value	Description	Error Handling
0x80	Success	None
-0x81	Failure to read out MCA spectrum from the file	Check the MCA data file

Run type = 0x9000

Value	Description	Error Handling
0x90	Success	None
-0x91	Failure to read out MCA section of external memory	Reboot the module
-0x92	Failure to write to MCA section of external memory	Reboot the module
-0x93	Failure to read out LM section of external memory	Reboot the module
-0x94	Failure to write to LM section of external memory	Reboot the module
-0x95	Invalid external memory I/O request	Check the run type
-0x96	Failure to write to 2D section of external memory	Reboot the module
-0x97	Failure to read out 2D section of external memory	Reboot the module
-0x98	Failure to read out first N words of MCA section of external memory	Reboot the module

Usage example (Pixie-4, Pixie-500)

```
S32 retval;
U16 RunType;
U32 dummy[2];
U8 ModNum;
String LMfilename =  "C:\XIA\Pixie4\PulseShape\Listdata0001.bin"

RunType = 0x1100;      // start a new list mode run
```

```
ModNum = 0;
retval = Pixie_Acquire_Data(RunType, dummy, LMfilename, ModNum);
if(retval != 0x10)
{
  // Error handling
}

// wait until the run has ended
RunType = 0x4100;
while( ! Pixie_Acquire_Data(RunType, dummy, LMfilename, ModNum) ) {;}

// Read out the list mode data from all Pixie modules and save to a file
RunType = 0x6100;
retval = Pixie_Acquire_Data(RunType, dummy, LMfilename, ModNum);
if(retval != 0x60)
{
  // Error handling
}

// Read out the histogram data from all Pixie modules and save to a file
RunType = 0x5100;
retval = Pixie_Acquire_Data(RunType, dummy,
"C:\XIA\Pixie4\MCA\Histdata0001.bin", ModNum);
if(retval != 0x50)
{
  // Error handling
}
```

Usage example (Pixie-4 Express and Pixie-500 Express)

```
(see section 3.3.2 for further examples)
S32 retval;
U16 RunType;
U32 dummy[2];
U8 ModNum;
U16 RequestedSpills, WrittenSpills;
String LMfilename =  "C:\XIA\Pixie4\PulseShape\Listdata0001.bin"

RunType = 0x1400;      // start a new list mode run
ModNum = 0;
retval = Pixie_Acquire_Data(RunType, dummy, LMfilename, ModNum);
if(retval != 0x10)
{
  // Error handling
}

// check for data, will be saved to file until the run has ended
RunType = 0x4400;
while(RequestedSpills <= WrittenSpills )
      { WrittenSpills =
          Pixie_Acquire_Data(RunType, dummy, LMfilename, ModNum) }

// end run and and save data remaining in SDRAM to file
RunType = 0x3400;
retval = Pixie_Acquire_Data(RunType, dummy, LMfilename, ModNum);
if(retval != 0x30)
```

```
{
  // Error handling
}

// Read out the histogram data from all Pixie modules and save to a file
RunType = 0x5400;
retval = Pixie_Acquire_Data(RunType, dummy,
"C:\XIA\Pixie4\MCA\Histdata0001.mca", ModNum);
if(retval != 0x50)
{
  // Error handling
}
```


2.7 Pixie_Set_Current_ModChan

Syntax

```
S32 Pixie_Set_Current_ModChan (
U8 Module,            // Module number to be set
U8 Channel);          // Channel number to be set
```

Description

Use this function to set the current module number and channel number.

Parameter description

Module specifies the current module to be set. Module should be in the range of 0 to NUMBER_MODULES in the System_Parameter_Values. (Currently the overall maximum as defined by PRESET_MAX_MODULES is 17).

Channel specifies the current channel to be set. Channel should be in the range of 0 to NUMBER_OF_CHANNELS - 1 (currently NUMBER_OF_CHANNELS is set to 4).

Return values

Value	Description	Error Handling
0	Success	None
-1	Invalid module number	Check Module
-2	Invalid channel number	Check Channel

Usage example

```
// Set current module to 1 and current channel to 3
Pixie_Set_Current_ModChan(1, 3);
```

2.8 Pixie_Buffer_IO

Syntax

```
S32 Pixie_Buffer_IO (
U16 *Values,       // An unsigned 16-bit integer array containing the
                   // data to be transferred
U8 type,           // Data transfer type
U8 direction,      // Data transfer direction
U8 *file_name,     // File name
U8 ModNum);        // Module number
```

Description

Use this function to:

- Download or upload DSP parameters between the user interface and the Pixie modules;
- Save DSP parameters into a settings file or load DSP parameters from a settings file and applies to all modules present in the system;
- Copy parameters from one module to others or extracts parameters from a settings file and applies to the selected modules.

Parameter description

Values is an unsigned 16-bit integer array used for data transfer between the user interface and Pixie modules. *type* specifies the I/O type. *direction* indicates the data flow direction. The string variable *file_name* contains the name of settings files. Different combinations of the three parameters - *Values, type, direction* – designate different I/O operations as listed in Table 2.2.

Table 2.2: Different I/O operations using function Pixie_Buffer_IO.

Type	Direction	Values	I/O Operation
0	0	DSP I/O variable values	Write DSP I/O variable values to modules
	1		Read DSP I/O variable values from modules
1	0*	Values to be written	Write to certain locations of the data memory
	1	All DSP variable values	Read all DSP variable values from modules
2	0	N/A**	Save current settings in all modules to a file
	1		Read settings from a file and apply to all modules in the system
3	0	Values[0] – source module number; Values[1] – source channel number; Values[2] – copy/extract pattern bit mask; Values[3], Values[4], … - destination channel pattern	Extract settings from a file and apply to selected modules
	1		Copy settings from a source module to destination modules
4	N/A***	Values[0] – address; Values[1] – length	Specify the location and number of words to be written into the data memory

Notes:
*Special care should be taken for this I/O operation since mistakenly writing to some locations of the data memory will cause the system to crash. The Type 4 I/O operation should be called first to specify the location and the number of words to be written before calling this one. If necessary, please contact XIA for assistance.
**Any unsigned 16-bit integer array could be used here.
***Direction can be either 0 or 1 and it has no effect on the operation.

Return values

Value	Description	Error Handling
0	Success	None
-1	Failure to set DACs after writing DSP parameters	Reboot the module
-2	Failure to program Fippi after writing DSP parameters	Reboot the module
-3	Failure to set DACs after loading DSP parameters	Reboot the module
-4	Failure to program Fippi after loading DSP parameters	Reboot the module
-5	Can't open settings file for loading	Check the file name
-6	Can't open settings file for reading	Check the file name
-7	Can't open settings file to extract settings	Check the file name
-8	Failure to set DACs after copying or extracting settings	Reboot the module
-9	Failure to program Fippi after copying or extracting settings	Reboot the module
-10	Invalid module number	Check ModNum

-11	Invalid I/O direction	Check direction
-12	Invalid I/O type	Check type

Usage example

```
S32 retval;
U8 type, direction, modnum;

modnum = 0;        // Module number

// Download DSP parameters to the current Pixie module; DSP_Values is a
// pointer pointing to the DSP parameters; no need to specify file name
// here.
direction = 0;     // Write
type = 0;          // DSP I/O values
retval = Pixie_Buffer_IO(DSP_Values, type, direction, "", modnum);
if(retval < 0)
{
  // Error handling
}

// Read DSP memory values from the current Pixie module; Memory_Values
// is a pointer pointing to the memory block; no need to specify file
// name Here.
direction = 1;     // Read
type = 1;          // DSP memory values
retval = Pixie_Buffer_IO(Memory_Values, type, direction, "", modnum);
if(retval < 0)
{
  // Error handling
}
```

2.9 Options for Compiling Pixie API

The Pixie API can be compiled as either a WaveMetrics Igor XOP file used by the Pixie Viewer or a dynamic link library (DLL) for other programs. The latter option can be used by advanced users to integrate Pixie modules into their own data acquisition systems. As part of the software distribution, we provide a number of sample C programs illustrating aspects of the operation from the command line using the DLL.[5] DLL and sample programs can also be compiled under Linux; a ROOT demo interface is available on request. Sample make files and MS Visual Studio project files (version 2005) are included in the distribution.

There are three compiler switches (options) that have to be set by the makefiles or project files:

- COMPILE_IGOR_XOP, if "defined" enables sections of the code used for compiling the Igor xop. It must not be defined for compilation of the dll or sample code.
- XIA_WINDOZE, if "defined" enables sections of the code used for compiling code for Windows. XIA_Linux, if "defined" enables sections of the code used for compiling code for Linux. The two options are exclusive.
- WINDRIVER_API, if "defined", enables those sections of the code related to the PCIe I/O for the Pixie-500 Express. If compiling purely for the Pixie-4 using the PLX PCI interface, this option can be undefined

The Pixie-4 C library is ANSI C compatible as much as possible, but a number of non-standard functions are used. These are:

- Sleep. Throughout the code, the function *Pixie_Sleep* is used to wait for a time (in ms). In utilities.c, this function is implemented using the Windows API function *Sleep*. For Linux, *usleep* is used.

To compile under Linux, files from the PLX software development kit have to be present. This SDK is available from PLX (www.plxtech.com) and not distributed by XIA. The Linux make file expects the complete SDK to be present under …\PixieClib\Linux. The Pixie software uses version 6.5 of the PLX drivers, the SDK has to match this version.

The following table summarizes the required files for these options.

[5] By default, compilation as DLL provides additional functions used by XIA's demo LabView interface. These functions can normally be ignored, they are simply wrappers of the main API functions. The LabView demo interface is available upon request.

Table 2.3: Options for compiling the Pixie C Driver.

Compilation Option	Required Files			Notes
	C source files	C header files	Library files	
Files required for all options	Boot.c, eeprom.c, globals.c, pixie_c.c, reader.c, ua_par_io.c, utilities.c,	boot.h, defs.h, globals.h, sharedfiles.h, utilities.h, reader.h, PciRegs.h, PciTypes.h, PexApi.h, Plx.h, PLX_sysdep.h, PlxApi.h, PlxTypes.h, Reg9054.h, PlxIoctl.h, PlxNetIoctl.h, PlxNetTypes.h plxdefck.h, plxstat.h pci_Jungo.h	PlxApi.lib,	
Additional Files for Pixie-500 Express functions (PCIe I/O)	pixie500e_lib.c	pixie500e_lib.h, vdma_seqcode.h	wdapi1031.lib	Only used if WINDRIVER_API is defined
Additional files for a dynamic link library (DLL)	pixie4VI_DLL.c	pixie4VI_DLL.h		These files provides wrapper functions for the LabView demo interface. They are not necessary for a basic DLL with the API functions described above, but are included by default.
Additional files for Igor XOP	pixie4_iface.c, pixie4_igor.c, PixieWinCustom.rc	pixie4_iface.h, IgorXOP.h, VCExtraIncludes.h, XOP.h, XOPResources.h, XOPStandardHeaders.h, XOPSupport.h, XOPSupportWin, XOPWinMacSupport.h, XOPWMWinMacSupport.h,	XOPSupport.lib IGOR.lib	These files provide wrapper functions for the Igor Pro Pixie Viewer. Variables COMPILE_IGOR_XOP and XIA_WINDOZE must be defined
Additional files for standalone program e.g. sample.c	sample____.c or equivalent SystemConfig.c	sample____.h or equivalent		These files provide example command line executable programs for specific tasks (e.g. booting). The executables require the DLL.

3 Control Pixie Modules via C program

3.1 Initializing

We describe here how to initialize Pixie modules in a PXI(e) chassis using the functions described in Section 2. As an example, we assume two Pixie-4 modules – residing in slot #3 and #4, respectively. Users are also encouraged to read the sample C code included with the API.

3.1.1 Initialize Global Variables

As discussed in Section 2, we assume that three global variable arrays have been defined: System_Parameter_Values, Module_Parameter_Values and Channel_Parameter_Values. For these three global variable arrays, three corresponding global name arrays are defined: System_Parameter_Names, Module_Parameter_Names and Channel_Parameter_Names, respectively. Table 3.5 lists the names contained in each of these name arrays. The API uses search functions to locate a variable value by name at run time.

Table 3.2: File Names in All_Files.

Index	Default File Name	Note
0	C:\XIA\Pixie\Firmware\P4e_16_125_vdo.bin	FPGA configuration (Pixie-4 Express 16/125 or others)
1	C:\XIA\Pixie\Firmware\syspixie_revC.bin	Communication FPGA configuration (Pixie-4 Rev C, D, E)
2	C:\XIA\Pixie\Firmware\pixie.bin	Signal processing FPGA configuration (Pixie-4)
3	C:\XIA\Pixie\DSP\PXIcode.bin	DSP executable binary code for the Pixie-4
4	C:\XIA\Pixie\Configuration\default.set	Settings file
5	C:\XIA\Pixie\DSP\P500e.var	File of DSP I/O variable names Do not use the var file from Pixie-4 DSP!
6	C:\XIA\Pixie\DSP\PXIcode.lst	File of DSP memory variable names for the Pixie-4
7	C:\XIA\Pixie\Firmware\syspixie_RevC.bin unused	Communication FPGA configuration (Pixie-500 Rev B)
8	C:\XIA\Pixie\Firmware\P4e_14_500_vdo.bin	FPGA configuration (Pixie-4 Express 14/500)
9	C:\XIA\Pixie\Firmware\p500e_zdt.bin	FPGA configuration (Pixie-500 Express)
10	C:\XIA\Pixie\DSP\P500code.bin unused	DSP executable binary code for the Pixie-500
11	C:\XIA\Pixie\DSP\P500e.ldr	DSP executable binary code for the Pixie-500 Express
12	C:\XIA\Pixie\DSP\P500e.lst	File of DSP memory variable names for the Pixie-500 Express

The file names for the DSP and firmware files have to be copied to the internal array "Boot_File_Name_List" Table 3.2 lists the names and order of files needed to initialize the

Pixie-4 modules. The API expects all files to be present, even if they are not used for the particular module type currently used. The exception are files reserved for the Pixie-500 (Index 7. and 10). Some module types, in particular the Pixie-4 Express, come in several variants. The files for two major Pixie-4 Express variants, 16/125 and 14/500, are listed in index 0 and index 8. For other variants, the default name has to be changed to the appropriate variant file name.

The non-read-only variables in the global variable array System_Parameter_Values also need to be initialized before the API functions are called to start the initialization. Table 3.3 lists those global variables and the definition of their allowed values.

Table 3.3: Initialization of Module_Global_Values.

Module_Global_Names	**Module_ Global_ Values Default Value**	**Definition**
NUMBER_MODULES	2	The total number of Pixie-4 modules present Range: 1.. PRESET_MAX_MODULES
OFFLINE_ANALYSIS	0	0 for online analysis, 1 for offline analysis (no I/O with modules)
AUTO_PROCESSLMDATA	0	0 to not writing results in a .dat file while parsing list mode output data 1 to write standard .dat file during parsing 2 to write extended dt2 file during parsing 3 to write extended dt3 file during parsing
MAX_NUMBER_MODULES	7	Select chassis type 7 = 4,5,6,8-slot chassis 13 = 14-slot XIA 6U chassis 17 = 14,18-slot NI 3U chassis 62 = 8-slot PXI/PXIe NI chassis 1062Q use 62 for any PXI Express chassis
KEEP_CW	1	0 Channel COINC_DELAY is set by user. 1 Channel COINC_DELAY is automatically adjusted to compensate for difference in energy filter length Note: not required for Pixie-500 Express and Pixie-4 Express
SLOT_WAVE[0]	110	Module 0 is s/n 110 (use physical slot number for Pixie-4)
SLOT_WAVE[1]	111	Module 1 is s/n 111
SLOT_WAVE[...]	...	Module ... is s/n ...

3.1.2 Boot Pixie Modules

The boot procedure for Pixie-4 modules includes the following steps: First, the function **Pixie_User_Par_IO** should be called to initialize the global value array System_Parameter_Values. Then, **Pixie_Boot_System** should be called to boot the modules. The following code is an example showing how to boot the Pixie-4 modules using the API functions.

An Example Code Illustrating How to Boot Pixie-4 Modules

```
S32 retval;
U8 d, m, c;

// copy file names into Boot_File_Name_List
strcpy(Boot_File_Name_List[0],"C:\\\...Pixie4\\firmware\\p4e_16_125_vdo.bin\0");
strcpy(Boot_File_Name_List[1],"C:\\\...Pixie4\\firmware\\syspixie_RevC.bin\0");
strcpy(Boot_File_Name_List[2],"C:\\\...Pixie4\\firmware\\pixie.bin\0");
strcpy(Boot_File_Name_List[3],"C:\\\...Pixie4\\dsp\\PXIcode.bin\0");
strcpy(Boot_File_Name_List[4],"C:\\\...Pixie4\\configuration\\default.set\0");
strcpy(Boot_File_Name_List[5],"C:\\\...Pixie4\\dsp\\P500e.var\0");
strcpy(Boot_File_Name_List[6],"C:\\\...Pixie4\\dsp\\PXIcode.lst\0");
// etc for 7-12

// initialize system parameter VALUES in C program
// Note: indices have to be derived from NAME arrays (table 3.5)
System_Parameter_Values[NUMBER_MODULES_Index] = 2;
System_Parameter_Values[OFFLINE_ANALYSIS_Index] = 0;
System_Parameter_Values[AUTO_PROCESSLMDATA_Index] = 0;
System_Parameter_Values[MAX_NUMBER_MODULES_Index] = 7;
System_Parameter_Values[KEEP_CW]=1;
System_Parameter_Values[SLOT_WAVE_Index] = 110;
System_Parameter_Values[SLOT_WAVE_Index+1] = 111;

// download system parameter VALUES initialized above to API
d = 0;      // direction download
m = 0;      // Module #0
c = 0;      // Channel #0

retval = Pixie_User_Par_IO(System_Parameter_Values,
 "NUMBER_MODULES", "SYSTEM", d, m, c);
if( retval < 0 )     // Error handling
retval = Pixie_User_Par_IO(System_Parameter_Values,
          "OFFLINE_ANALYSIS", "SYSTEM", d, m, c);
if( retval < 0 )     // Error handling
retval = Pixie_User_Par_IO(System_Parameter_Values,
          "AUTO_PROCESSLMDATA", "SYSTEM", d, m, c);
if( retval < 0 )     // Error handling
retval = Pixie_User_Par_IO(System_Parameter_Values,
          "MAX_NUMBER_MODULES", "SYSTEM", d, m, c);
if( retval < 0 )     // Error handling
retval = Pixie_User_Par_IO(System_Parameter_Values,
          "KEEP_CW", "SYSTEM", d, m, c);
if( retval < 0 )     // Error handling

retval = Pixie_User_Par_IO(System_Parameter_Values,
"SLOT_WAVE", "SYSTEM", d, m, c);
if( retval < 0 )     // Error handling
// Note: System_Parameter_Values contains entries for module 0 and 1
// (slot 3 and 4), the last command (argument "SLOT_WAVE") applies both

// boot Pixie-4 modules
retval = Pixie_Boot_System(0x1F);
if( retval < 0 )     // Error handling

// set current module and channel number
Pixie_Set_Current_ModChan(0, 0);
```


3.2 Setting DSP variables

The host computer communicates with the DSP by setting and reading a set of variables called DSP I/O variables. These variables, totally 512 unsigned 16-bit integers, sit in the first 512 words of the data memory. The first 256 words, which store input variables, are both readable and writeable, while the remaining 256 words, which store pointers to various data buffers and run summary data, are only readable. The exact location of any particular variable in the DSP code will vary from one code version to another. To facilitate writing robust user code, we provide a reference table of variable names and addresses with each DSP code version. Included with the software distribution is a file called P500e.var. It contains a two-column list of variable names and their respective addresses. Thus you can write your code such that it addresses the DSP variables by name, rather than by fixed location.

It should come as no surprise that many of the DSP variables have meaningful values and ranges depending on the values of other variables. A complete description of all interdependencies can be found in Section 4. All of these interdependencies have been taken care of by the Pixie API. So instead of directly setting DSP variables, users only need to set the values of those global variables defined in Table 3.5. The API will then convert these values into corresponding DSP variable values and download them into the DSP data memory. On the other hand, if users want to read out the data memory, the API will first convert these DSP values into the global variable values. Tables 3.5a-c give a complete description of all the global variables being used by the Pixie API. The code shown below is an example of setting DSP variables through the API.

An Example Code Illustrating How to Set DSP Variables through the API

```
S32 retval;
U8 direction, modnum, channum;

direction = 0;       // download
modnum = 0;          // Module #0
channum = 0;         // Channel #0
// Note xxx_Index has to be derived from NAME array (table 3.5)

// set COINCIDENCE_PATTERN to 0xFFFF
Module_Parameter_Values[COINCIDENCE_PATTERN_Index] = 0xFFFF;

// download COINCIDENCE_PATTERN to the DSP
retval = Pixie_User_Par_IO(Module_Parameter_Values,
"COINCIDENCE_PATTERN", "MODULE", direction, modnum, channum);
if( retval < 0 // Error handling

// set ENERGY_RISETIME to 6.0 µs
Channel_Parameter_Values[ENERGY_RISETIME_Index] = 6.0;

// download ENERGY_RISETIME to the DSP
retval = Pixie_User_Par_IO(Channel_Parameter_Values,
             "ENERGY_RISETIME", "CHANNEL", direction, modnum, channum);
if( retval < 0 // Error handling
```

Table 3.5a: Descriptions of System Global Variables

System_Parameter_Names	I/O Type	Unit	Corresponding DSP Variables
NUMBER_MODULES	Read/Write	N/A	N/A
OFFLINE_ANALYSIS	Read/Write	N/A	N/A
AUTO_PROCESSLMDATA	Read/Write	N/A	N/A
C_LIBRARY_RELEASE	Read only	N/A	N/A
C_LIBRARY_BUILD	Read only	N/A	N/A
KEEP_CW	Read/Write	N/A	N/A
SLOT_WAVE	Read/Write	N/A	N/A

SLOT_WAVE is followed by up to 16 slot entries. Entries are addressed as SLOT_WAVE[N]

Table 3.5b: Descriptions of Module Global Variables

Module_Parameter_Names	I/O Type	Unit	Corresponding DSP Variables
MODULE_NUMBER	Read only	N/A	MODNUM
MODULE_CSRA	Read/Write	N/A	MODCSRA
MODULE_CSRB	Read/Write	N/A	MODCSRB
C_CONTROL	Read/Write	N/A	CCONTROL
MAX_EVENTS	Read/Write	N/A	MAXEVENTS
COINCIDENCE_PATTERN	Read/Write	N/A	COINCPATTERN
ACTUAL_COINCIDENCE_WAIT	Read/Write	ns	COINCWAIT
MIN_COINCIDENCE_WAIT	Read only	ticks	COINCWAIT
SYNCH_WAIT	Read/Write	N/A	SYNCHWAIT
IN_SYNCH	Read/Write	N/A	INSYNCH
RUN_TYPE	Read/Write	N/A	RUNTASK
FILTER_RANGE	Read/Write	N/A	FILTERRANGE
MODULEPATTERN	Read/Write	N/A	MODULEPATTERN
NNSHAREPATTERN	Read/Write	N/A	NNSHAREPATTERN
DBLBUFCSR	Read/Write	N/A	DBLBUFCSR
MODULE_CSRC	Read/Write	N/A	MODCSRC
BUFFER_HEAD_LENGTH	Read only	N/A	BUFHEADLEN
EVENT_HEAD_LENGTH	Read only	N/A	EVENTHEADLEN
CHANNEL_HEAD_LENGTH	Read only	N/A	CHANHEADLEN
NUMBER_EVENTS	Read only	N/A	NUMEVENTSX, A, B
RUN_TIME	Read only	s	RUNTIMEX, A, B, C
EVENT_RATE	Read only	cps	NUMEVENTSX, A, B RUNTIMEX, A, B, C
TOTAL_TIME	Read only	s	TOTALTIMEX, A, B, C
BOARD_VERSION	Read only	N/A	<Hardware PROM>
SERIAL_NUMBER	Read only	N/A	<Hardware PROM>
DSP_RELEASE	Read only	N/A	DSPRELEASE
DSP_BUILD	Read only	N/A	DSPBUILD
FIPPI_ID	Read only	N/A	FIPPIID
SYSTEM_ID	Read only	N/A	HARDWAREID
XET_DELAY	Read/Write	N/A	XETDELAY
PDM_MASKA	Read/Write	N/A	PDMMASKA
PDM_MASKB	Read/Write	N/A	PDMMASKB
PDM_MASKC	Read/Write	N/A	PDMMASKC
USER_IN 0..15	Read/Write	N/A	USERIN 0..15
USER_OUT 0..15	Read Only	N/A	USEROUT 0..15
ADC_RATE	Read Only	MHz	<Hardware PROM>
ADC_BITS	Read Only	N/A	<Hardware PROM>
NUM_COINC_TRIG	Read Only	N/A	NCOINCTRIGX, A, B, C
COINC_SFDT	Read Only	s	CSFDTX, A, B, C
COINC_COUNT_TIME	Read Only	S	CCTX,A,B,C
COINC_INPUT_RATE	Read Only	Cps	NCOINCTRIGX, A, B, C CCTX,A,B,C
EXTRA_IN	Read/Write	N/A	EXTRAIN
EXTRA_OUT	Read Only	N/A	EXTRAOUT

Table 3.5c: Descriptions of Channel Global Variables

Channel_Parameter_Names	I/O Type	Unit	Corresponding DSP Variables
CHANNEL_CSRA	Read/Write	N/A	CHANCSRA
CHANNEL_CSRB	Read/Write	N/A	CHANCSRB
ENERGY_RISETIME	Read/Write	µs	SLOWLENGTH
ENERGY_FLATTOP	Read/Write	µs	SLOWGAP
TRIGGER_RISETIME	Read/Write	µs	FASTLENGTH
TRIGGER_FLATTOP	Read/Write	µs	FASTGAP
TRIGGER_THRESHOLD	Read/Write	N/A	FASTTHRESH
VGAIN	Read/Write	V/V	SGA, DIGGAIN
VOFFSET	Read/Write	V	TRACKDAC
TRACE_LENGTH	Read/Write	µs	TRACELENGTH
TRACE_DELAY	Read/Write	µs	USERDELAY
COINC_DELAY	Read/Write	µs	COINCDELAY, RESETDELAY
PSA_START	Read/Write	µs	PSAOFFSET
PSA_END	Read/Write	µs	PSAOFFSET , PSALENGTH
FCFD_THRESHOLD	Read/Write	N/A	FCFDTHRESHOLD
BINFACTOR	Read/Write	N/A	LOG2EBIN
TAU	Read/Write	µs	PREAMPTAUA, PREAMPTAUB
BLCUT	Read/Write	N/A	BLCUT
XDT	Read/Write	N/A	XWAIT, XAVG
BASELINE_PERCENT	Read/Write	N/A	BASELINEPERCENT
CFD_THRESHOLD	Read/Write	N/A	CFDTHR
INTEGRATOR	Read/Write	N/A	INTEGRATOR
CHANNEL_CSRC	Read/Write	N/A	CHANCSRC
GATE_WINDOW	Read/Write	µs	GATEWINDOW
GATE_DELAY	Read/Write	µs	GATEDELAY
BLAVG	Read/Write	N/A	LOG2BWEIGHT
COUNT_TIME	Read only	s	COUNTTIMEX, A, B, C
INPUT_COUNT_RATE	Read only	cps	FASTPEAKSX, A, B, C COUNTTIMEX, A, B, C, FTDTX, A, B, C,
FAST_PEAKS	Read only	N/A	FASTPEAKSX, A, B, C
OUTPUT_COUNT_RATE	Read only	cps	NOUTX,A,B, COUNTTIMEX, A, B, C
NOUT	Read only	N/A	NOUTX,A,B,
GATE_RATE	Read only	cps	GCOUNTX, A, B COUNTTIMEX, A, B, C
GATE_COUNTS	Read only	N/A	GCOUNTX,A,B
FTDT	Read only	s	FTDTX, A. B, C
SFDT	Read only	s	SFDTX, A, B, C
GDT	Read only	s	GDTX, A, B ,C
CURRENT_ICR	Read only	cps	ICR
CURRENT_OORF	Read only	%	OORF
PSM_GAIN_AVG	Unused	N/A	
PSM_GAIN_AVG_LEN	Unused	N/A	
PSM_TEMP_AVG	Unused	N/A	
PSM_TEMP_AVG_LEN	Unused	N/A	
PSM_GAIN_CORR	Unused	N/A	
QDC0_LENGTH	Read/Write	N/A	QDC0length
QDC1_LENGTH	Read/Write	N/A	QDC1length
QDC0_DELAY	Read/Write	N/A	QDC0delay
QDC1_DELAY	Read/Write	N/A	QDC1delay

NPPI	Read only	N/A	NPPIX,B,C
PASS_PILEUP_RATE	Read only	cps	NPPIX, A, B COUNTTIMEX, A, B, C
CH_EXTRA_IN	Read/Write	N/A	CHEXTRAIN

3.3 Access spectrum memory or list mode data

3.3.1 Access spectrum memory

The MCA spectrum memory is fixed to 32Ki words (32 bits per word) per channel, residing in the external memory. Therefore, the starting address of the MCA spectrum in the external memory for Channel #0, 1, 2 and 3 will be 0x00000000, 0x000080000, 0x00010000, 0x00018000, respectively. The spectrum memory is accessible even when a data acquisition run is in progress. The following code is an example of how to start a MCA run and read out the MCA spectrum after the run is finished.

An Example Code Illustrating How to Access MCA Spectrum Memory

```
S32 retval;
U8 direction, modnum, channum;
String MCAfile;
U32 User_Data[131072];   // an array for holding the MCA spectrum data of
                         // 4 channels

direction = 0;   // download
modnum = 0;      // Module #0
channum = 0;     // Channel #0
MCAfile = "C:\\XIA\\Pixie4\\MCA\Data0001.mca"

// start a MCA run
retval = Pixie_Acquire_Data(0x1301, User_Data, MCAfile, modnum);
if( retval < 0 // Error handling

// wait for 30 seconds
Sleep(30000);

// stop the MCA run
retval = Pixie_Acquire_Data(0x3301, User_Data, MCAfile, modnum);
if( retval < 0 // Error handling

// save MCA spectrum to a file
retval = Pixie_Acquire_Data(0x5301, User_Data, MCAfile, modnum);
if( retval < 0 // Error handling

// read out the MCA spectrum and put it to array User_data
retval = Pixie_Acquire_Data(0x9001, User_data, MCAfile, modnum);
if( retval < 0 // Error handling
```

Note that in clover addback mode, the spectrum length is fixed to 16Ki for each channel plus 16Ki of addback spectrum. Therefore, the starting address of the MCA spectrum in the external memory for Channel #0, 1, 2 and 3 will be 0x00000000, 0x000040000, 0x00008000, 0x00010000, respectively, for the addback spectrum it is 0x00018000.

3.3.2 Access list mode data

In the Pixie-4, there are two data buffers to choose from: the DSP's local I/O buffer (8Ki 16-bit words), and a section of the external memory (128Ki 32-bit words). Bit 1 in the variable MODCSRA selects which data buffer is filled during the run:

If the external memory is chosen to hold the output data, the local buffer is transferred to the external memory when it has been filled. Then the run resumes automatically, without interference from the host, until 32 local buffers have been transferred. The data can then be read from external memory in a fast block read starting from location 0x00020000.

If the local buffer is chosen, the run stops when the local buffer is filled. The data has to be read out from local memory.

With any data buffer, you can do any number of runs in a row. The first run would be started as a NEW run. This clears all histograms and run statistics in the memory. Once the data has been read out, you can RESUME running. Each RESUME run will acquire another either 32 or 1 8Ki buffers of data, depending on which buffer has been chosen. In a RESUME run the histogram memory is kept intact and you can accumulate spectra over many runs. The example code shown below illustrates this.

In the Pixie-4 Express and Pixie-500 Express, there is only one data buffer, the 256 MB SDRAM FIFO. Runs do not stop and resume; data is continuously added to the SDRAM and transferred to a 2MB buffer on the host PC. The host hast to check periodically if the 2MB buffer is filled and write the data to file. This is normally handled by an interrupt routine, and the top level interface only has to read the number of buffers stored. There still is an option for the top level interface to poll status and write each individual buffer, which however is much slower in the Igor based Pixie Viewer. There usually is still some data left in the buffer when a run is stopped. This data is read out as part of the run stop routine, the total number of buffers is therefore somewhat larger than the requested value.

For a description of the list mode data formats, see section 4.2 of the user manual.

In the Pixie-4, *Run_Type* = 0x4000 or 0x40FF is used to check if the run stopped, which implies data is ready for readout to file. The top level program then (i) calls *Run_Type* = 0x6### to save data, (ii) calls *Run_Type* = 0x2### to resume the run, (iii) increments the spill count and (iv) ends if the requested number of spills is reached.

In the Pixie-500 Express, *Run_Type* = 0x4400 or 0x4401 are used instead to check if data is ready and if yes the data is also saved to file. In interrupt mode, the routine really only computes the **total** number of spills written to file. The return value informs the top level program how many spills have been saved. The top level program only has to end the run once the the requested number of spills is reached.

An Example Code Illustrating How to Access List Mode Data (Pixie-4)

```
S32 retval;
U8 direction, modnum, channum;
U32 User_Data[131072]; // an array for holding the MCA spectrum data of
// 4 channels
U16 k, Nruns;
char *DataFile = {"C:\\XIA\\Pixie4\\PulseShape\\Data.bin"};

direction = 0;  // download
modnum = 0;     // Module #0
channum = 0;    // Channel #0
Nruns = 10;     // 10 repeated list mode runs
k = 0;          // initialize counter

// start a general list mode run
retval = Pixie_Acquire_Data(0x1100, User_Data, DataFile, modnum);
if( retval < 0 // Error handling

do
{
// wait until run has ended
while( ! Pixie_Acquire_Data(0x4100, User_Data, DataFile, modnum) ) {;}

// read out the list mode data and save it to a file
retval = Pixie_Acquire_Data(0x6100, User_Data, DataFile, modnum);
if( retval < 0 // Error handling

k ++;
if(k > Nruns)
{
    break;
}

// issue RESUME RUN command
retval = Pixie_Acquire_Data(0x2100, User_Data, DataFile, modnum);
if( retval < 0 // Error handling

}while(1);

// read out the MCA spectrum and put it to array User_Data
retval = Pixie_Acquire_Data(0x9001, User_Data, DataFile, modnum);
if( retval < 0 // Error handling
```


An Example Code Illustrating How to Access List Mode Data (Pixie-4 Express and Pixie-500 Express)

```
S32 retval;
U8 direction, modnum, channum;
U32 User_Data[131072]; // an array for holding the MCA spectrum data of
// 4 channels
U16 k, Nruns;
char *DataFile = {"C:\\XIA\\Pixie4\\PulseShape\\Data.bin"};

direction = 0;   // download
modnum = 0;      // Module #0
channum = 0;     // Channel #0
Nruns = 10;      // 10 spills


// start a general list mode run
retval = Pixie_Acquire_Data(0x1400, User_Data, DataFile, modnum);
if( retval < 0 // Error handling

    // check for data ready, if so it is saved to file
while(  Nruns>retval)
{retval = Pixie_Acquire_Data(0x4400,User_Data,DataFile,modnum) }

// stop list mode run, save remaining data to file
retval = Pixie_Acquire_Data(0x3400, User_Data, DataFile, modnum);
if( retval < 0 // Error handling


// read out the MCA spectrum and put it to array User_Data
retval = Pixie_Acquire_Data(0x9001, User_Data, DataFile, modnum);
if( retval < 0 // Error handling
```

To process the list mode data after it is saved to a file, the Pixie API provides several utility routines to parse the list mode data and read out the waveform, energy of each individual trace or PSA values (tasks 0x7000). The code below shows how to read waveforms from a list mode file.

Internally, the API distinguishes between Pixie-4 .bin files and Pixie4/500 Express .b## files. The latter are also checked (and marked) for data transmission errors. In task 0x7008, argument 3 of the data exchange wave specifies a window to search for coincident events near the requested one in the other 3 channels. Argument 3 is then overwritten with the trace length of channel 0. Argument 1 is overwritten with the ADC rate in MHz as extracted from the file, which can be used to set the time scale in a display of the waveform.

An Example Code Illustrating How to Parse List Mode Data

```
S32 retval;
U8 direction, modnum, channum, i, len;
U32 List_Data[2* PRESET_MAX_MODULES];  // list mode trace information for
          the maximum possible number of modules in a system.
char *DataFile = {"C:\\XIA\\Pixie4\\PulseShape\\Data.bin"};
U32 totaltraces;  // total number of traces in the list mode data file
U32 *eventposlen; // point to positions of the events in the file
U32 *EventWave;       // point to the event data from the file

direction = 0;  // download
modnum = 0;     // Module #0
channum = 0;    // Channel #0
DisplayTimeWindow = 1000; // in clock ticks


// parse list mode file
retval = Pixie_Acquire_Data(0x7001, List_Data, DataFile, modnum);
if( retval < 0 // Error handling

totaltraces = 0;
for(i=0; i<PRESET_MAX_MODULES; i++)
{
  // sum the total number of traces for all modules
  totaltraces += List_Data[i+ PRESET_MAX_MODULES];
}

// allocate memory to hold the starting address, trace length, and
// energy of each trace (therefore, 3 32-bit words are needed for each
// trace.)
eventposlen = (U32)malloc(totaltraces*3*NUMBER_OF_CHANNELS);

// populate event locations from the data file
retval = Pixie_Acquire_Data(0x7007, eventposlen, DataFile, modnum);
if( retval < 0 // Error handling

// allocate memory to hold the event data;
// headers plus traces plus location information
// (ModLoc is offset for position of module in eventposlen)
len = eventposlen[ModLoc+ChosenEvent*3+2]+7+36
EventWave = (U16)malloc(len);

EventWave[0] = eventposlen[ModLoc+ChosenEvent*3+0];// event position
EventWave[1] = eventposlen[ModLoc+ChosenEvent*3+1];// buffer header pos.
EventWave[2] = eventposlen[ModLoc+ChosenEvent*3+2];// length of event

// define coincidence window for display
// API searches this width around requested event and returns up to 4
// coincident events. If >64Ki, only requested event is returned
EventWave[3] = DisplayTimeWindow;  // valie only for .b## files

// read event into EventWave
// see Pixie_Acquire_Data above for data format.
retval = Pixie_Acquire_Data(0x7008, EventWave, DataFile, modnum);
if( retval < 0 // Error handling
```



```
// fill trace for channel 0
trace0 = (U16)malloc(EventWave[3]);
for(k=0;k<EventWave[3],k++)
  trace0[k] =  EventWave[7+36+k]

// dt between samples of trace0 is 10E-6ns/EventWave[1]
```

4 User Accessible DSP Variables

User parameters are stored in the data memory space of the on-board DSP. There are two sets of user-accessible parameters. 256 words in data memory are used to store input parameters. These must be set properly by the user application. A second set of 256 words is used for results furnished by the Pixie module. The total of 512 words are stored in the .set files – the first 256 words are copied to the module during booting and loading of settings, the second 256 words are read from the DSP during updates of run statistics etc and in the .set files created after the data acquisition.

As of this writing the two blocks are contiguous in memory space. We provide an ASCII file named P500e.var which contains in a 2-column format the offset and name of every user accessible variable. We suggest that user code use this information to create a name-address lookup table, rather than relying on the parameters retaining their address offsets with respect to the start address.

The input parameter block is partitioned into 5 subunits. The first contains 64 words that pertain to the Pixie-4 as a whole. It is followed by four blocks of 48 words, which describe the settings of the four channels.

Below we describe the module and channel parameters in turn. Also listed are the corresponding C global variable names and the corresponding control in Igor where the user can set the value of the variable. Unless it is a simple 1-1 copy, we also show the conversion process from the Igor variable or C global variable to the DSP parameter.

4.1 Module input parameters

MODNUM: Logical number of the module. This number will be written into the header of the list mode buffer to aid offline event reconstruction**.** It is normally set by the settings file during the boot process.

Igor controls: Slot Wave.

C global: SLOT_WAVE

ControlTask to apply change: none

MODCSRA: The Module Control and Status Register A

Bit 0: reserved

Bit 1: If set, DSP acquires 32 data buffers in each list mode run and stores the data in external memory. If not set, only one buffer is acquired and the data is kept in local memory. **Must be set/cleared for all modules in the system.** If set, clear bit 0 of DBLBUFCSR
Ignored for Pixie-4 Express and Pixie-500 Express

Bit 2: Bits 2 and 15 control trigger distribution over the backplane. If neither bit 2 or bit 15 are set, triggers are distributed only between channels of this module. Otherwise, triggers are distributed as follows:

Bit 2	Bit 15	Function
0	0	Triggers are distributed within module only, no connection to backplane
1	0	Module shares triggers using bussed wire-OR line. In systems with less than 8 modules and no PXI bridge boundaries, all modules sharing trigger should be set this way
0	1	Module receives triggers from master trigger lines, but uses neighboring lines to distribute triggers from right to left. In systems with more than 7 modules and/orPXI bridge boundaries, all modules except the leftmost should be set this way
1	1	Module puts own triggers and triggers received from right neighbor on the master trigger lines and responds to triggers on master trigger line. In systems with more than 7 modules and/orPXI bridge boundaries, the leftmost module should be set this way

Bit 3: If set, compute sum of channel energies for events with hits in more than one channel and put into addback spectrum
Ignored Pixie-500 Express. For Pixie-4 Express, set this bit to correctly display spectra in run task 0x402

Bit 4: If set, spectra for individual channels contain only events with a single hit. Only effective if bit 3 is set also.
Ignored for Pixie-4 Express and Pixie-500 Express

Bit 5: If set, use signal on front panel input "DSP-OUT" (between channel 1 and 2) and distribute on backplane to all modules as Veto signal (GFLT). **Note that only one module may enable this option to avoid a conflict on the backplane**.
Ignored for Pixie-4 Express and Pixie-500 Express

Bit 6: If set, channel 3's hit status contributes to the STATUS backplane line.
Ignored for Pixie-4 Express and Pixie-500 Express

Bit 7: Sensitive edge selection for front panel pulse counter and for generation of run statistics record (opposite edges)
Ignored for Pixie-4 and Pixie-500 Express

Bit 8: If set, store run statistics as a special list mode record triggered by rising/falling edge of Veto

Bit 9: If set, module writes the value of NNSHAREPATTERN to its left neighbor during ControlTask 5, using a PXI neighbor line. The left neighbor should be a PXI PDM. The value NNSHAREPATTERN specifies which coincidence test is applied in the PDM.
Ignored for Pixie-4 Express and Pixie-500 Express

Bit 10: Reserved

Bit 11: Bypass SDRAM (use smaller/faster in-FPGA buffer instead)
Ignored for Pixie-4 and Pixie-500 Express

Bit 12: If set, the module will drive low the TOKEN backplane line (used to distribute the result of the global coincidence test) if its local coincidence test fails. This way a module can inhibit all other module from acquiring data according to its local coincidence test.
Ignored for Pixie-4 Express and Pixie-500 Express

Bit 13: If set, the module will send out its hit pattern to slot 2 using the PXI STAR trigger line for each event.
This option must not be enabled in slot 2, because slot 2 can not send signals to itself. The line is instead used for chassis clock distribution and therefore should be left alone.
Ignored for Pixie-4 Express and Pixie-500 Express

Bit 14: If set, the front panel input "DSP out" is connected as an input to the "Status" line on the backplane. The Status line is set up as a wire-OR, so more than one module can enable this option.
Ignored for Pixie-4 Express and Pixie-500 Express

Bit 15: Controls sharing of triggers over backplane. See bit 2.

Igor controls: checkboxes, mostly in the CHASSIS SETUP panel.

C global: MODULE_CSRA. The C library checks for the dependencies listed above

ControlTask to apply change: 5.

MODCSRB: The Module Control and Status Register B

Bit 0: Execute user code routines programmed in user.dsp.

Bits 1-15: Reserved for user code.

Igor controls: variable in the USER CONTROL panel.

C global: MODULE_CSRB.

ControlTask to apply change: none

MODCSRC: The Module Control and Status Register C

Bits 0-15: Reserved

Igor controls: none.

C global: MODULE_CSRC.

ControlTask to apply change: none

CHANNUM: The chosen channel number. May be modified internally for tasks looping over all 4 channels, or to pass on current channel to user code. Should be set by host before starting controltask 4 and 6 to indicate which channel to operate on. (Previously HOSTIO was used in controltask 4). We recommend to always change CHANNUM when changing the channel that is addressed in the user interface.

Igor controls: none

C global none

ControlTask to apply change: none

RUNTASK: This variable tells the Pixie module what kind of run to start in response to a run start request. 8 run tasks are currently supported.

RunTask	Mode	Trace Capture	CHAN HEADLEN
0	Slow control run	N/A	N/A
256 (0x100)	Standard list mode (P4)	Yes	9
257 (0x101)	Compressed list mode (P4)	Yes	9
258 (0x102)	Compressed list mode (P4)	Yes	4
259 (0x103)	Compressed list mode (P4)	Yes	2
769 (0x301)	MCA mode	No	N/A
1024 (0x400)	Standard list mode (P500e/P4e)	Yes	32
1025 (0x401)	Text list mode (P500e/P4e)	No	32
1026 (0x406)	Group mode (P500e/P4e)	Yes	32

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 for Pixie-4 or Pixie-500. In this run type all bells and whistles are available. The scope of event processing includes computing energies to 16-bit accuracy, and performing pulse shape analyses for improved energy resolution and better time of arrival measurements. Nine words of results, including time of arrival, energy, XIA pulse shape analysis, user pulse shape analysis, etc. are written into the I/O buffer for each channel. Level-1 buffer is not used in this RunTask.

RunTask 257 (0x101) requests a compressed list mode run for Pixie-4 or Pixie-500. Both Level-1 buffer and I/O buffer are used in this RunTask, but no traces are written into the I/O buffer. Nine words of results, including time of arrival, energy, XIA pulse shape analysis, user pulse shape analysis, etc. are written into the I/O buffer for each channel.

RunTask 258 (0x102) requests a compressed list mode run for Pixie-4 or Pixie-500. The only difference between RunTask 258 and 257 is that in RunTask 258, only four words of results (time of arrival, energy, XIA pulse shape analysis, user pulse shape analysis) are written into the I/O buffer for each channel.

RunTask 259 (0x103) requests a compressed list mode run for Pixie-4 or Pixie-500. The only difference between RunTask 259 and 257 is that in RunTask 259, only two words of results (time of arrival and energy) are written into the I/O buffer for each channel.

RunTasks 512-515 (0x200-0x203) are no longer supported

RunTask 769 (0x301) requests a MCA run. The raw data stream is always sent to the level-1 buffer, independent of MODCSRA. The data-gathering interrupt routine fills that buffer with raw data, while the event processing routine removes events after processing. If the interrupt routine finds the level-1 buffer to be full, it will ignore events until there is room

again in the buffer. The run will not abort due to buffer-full condition. This run type does not write data to the I/O buffer.

RunTask 1024 (0x400) requests a standard list mode run for Pixie-500 Express. The scope of event processing includes computing energies to 16-bit accuracy, and optionally performing pulse shape analysis. 32 words of results, including time of arrival, energy, XIA pulse shape analysis, user pulse shape analysis, etc. and waveforms are written into the SDRAM FIFO for each channel. Data is saved in binary format by the C library

RunTask 1025 (0x401) requests a compressed list mode run for Pixie-500 Express. The scope of event processing includes computing energies to 16-bit accuracy, and optionally performing pulse shape analysis. 32 words of results, including time of arrival, energy, XIA pulse shape analysis, user pulse shape analysis, etc. are written into the SDRAM FIFO for each channel. No waveforms are acquired internally (value of TRACELENGTH is ignored); any pulse shape analysis is operating on the incoming data stream in the FPGA. Data is saved in ASCII format (.dt3) by the C library

RunTask 1026 (0x402) requests a group list mode run for Pixie-500 Express. Similar to 0x400, this task specifically assumes all channels share triggers and formats event data into 4-channel events.

Igor controls: *Run Type* in the *Run Control* tab

C global: RUN_TYPE. The C library checks for the RUNTASK to be of the form 0x0N0N, with 0 <=N <=4.

ControlTask to apply change: none

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 you issue a run start command with RUNTASK=0.

See section 4.5 for a list of acceptable values

Igor controls: none.

C global: none.

ControlTask to apply change: none

MAXEVENTS: The module ends its run when this number of events has been acquired. In the Pixie Viewer, the maximum value for MAXEVENTS is automatically calculated and applied when a run mode is chosen from the run type pulldown menu. The calculation is based on the RUN_TYPE, the TRACELENGTH and CHANCSRA bit 2 ("good channel" bit), using BUFHEADLEN, EVENTHEADLEN, CHANHEADLEN given by the RUN_TYPE and the length of the output buffer (8Ki words):

$$\text{Event length} = \text{EVENTHEADLEN} + \Sigma_i\,(\text{CHANHEADLEN} + \text{TRACELENGTH}_i) \times \text{good}_i$$

$$\text{MAXEVENTS}_{max} = (8\text{Ki} - \text{BUFHEADLEN})/\,\text{Event length}$$

Set MAXEVENTS = 0 if you want to switch off this feature, e.g., when logging spectra or when there is no need to enforce a fixed number of events. In particular, if 4 channels are

"good" and MAXEVENTS is computed accordingly, but the majority of events have only a single channel with a pulse, the buffer is only filled up to about 1/4 when MAXEVENTS is reached. So in this case it would be more efficient to disable the MAXEVENT limit by setting it to zero. The parameter is ignored in runtypes 0x301 and 0x40#.

Igor controls: *Events/buffer* in the *Run Control* tab

C global: MAX_EVENTS. The C library enforces an upper limit for MAXEVENTS.

ControlTask to apply change: none

COINCPATTERN: The user can request that certain coincidence/anticoincidence patterns are required for the event to be accepted. With four channels there are 16 different hit patterns, and each can be individually selected or marked for rejection by setting the appropriate bit in the COINCPATTERN mask.

Consider the 4-bit hit pattern 1010. The two 1's indicate that channel 3 (MSB) and channel 1 have reported a hit. Channels 2 and 0 did not. The 4-bit word reads as 10(decimal). If this hit pattern qualifies as an acceptable event, set bit 10 in the COINCPATTERN to 1. The 16 bit in COINCPATTERN cover all combinations. Setting COINCPATTERN to 0xFFFF causes the Pixie-4 to accept any hit pattern as valid.

Igor controls: Checkboxes in the *Coincidence* tab

C global: COINCIDENCE_PATTERN.

ControlTask to apply change: 5

COINCWAIT: Duration of the coincidence time window in clock cycles.

This parameter is used to define a time window for a coincidence test.

For the Pixie-4, the window is from the first event validation until the final hit pattern is latched. Since in the Pixie-4 the coincidence test is applied only after validation, each channel may require individual delay if validation requires different amounts of time due to differences in the energy filter settings. For example, a channel with the energy filter rise time set to 6 μs would start the coincidence window 2 μs before a channel with a filter rise time of 8 μs, and thus simultaneous events in the second channel will be lost unless the coincidence window is at least 2 μs or the channel parameter COINCDELAY is set to 2 μs for the faster channel. The following formula should be used to determine COINCDELAY:

$$\text{COINCDELAY<ch>} = \max(\text{PEAKSEP}*2^{\wedge}\text{FILTERRANGE})_{ch0\text{-}ch3} - (\text{PEAKSEP}*2^{\wedge}\text{FILTERRANGE})_{ch}$$

Only channels marked as "good" in the CHANCSRA need to be included in this computation. The maximum pileup inspection time is reported back to the user for reference in the variable MIN_COINCIDENCE_WAIT (in clock cycles).

When energy filter rise time or flat top (or CHANCSRA) are changed from Igor, the C library computes if channels need to be delayed to compensate for energy filter differences. The application of the delay depends on the System global KEEP_CW.

If KEEP_CW is 0, COINCDELAY0..3 are set to the user specified time. No compensation is applied.

If KEEP_CW is 1, COINCDELAY0..3 are set to the required compensation time. Any user input is ignored.

For the Pixie-4 Express and Pixie-500 Express the window is **around** the fast trigger at the rising edge of the pulse. This avoids the complications from trigger validation in the Pixie-4.

The Igor control and the C global are in units of μs, the DSP variable is in units of clock cycles (13.3ns for both Pixie-4 and Pixie-500, 8ns for Pixie-500 Express and Pixie-4 Express)

Constraints**:** COINCWAIT >= 1
COINCWAIT <= 65531 for Pixie-4 and Pixie-500
COINCWAIT <= min(511, minEFT)
for Pixie-500 Express and Pixie-4 Express
minEFT is the smallest energy filter time of the four channels,
(SLOWLENGTH+SLOWGAP)*2^ FILTERRANGE

Igor controls: *Window Width* in the *Coincidence* tab

C global: ACTUAL_COINCIDENCE_WAIT.

ControlTask to apply change: 5

SYNCHWAIT: Controls run start behavior. When set to 0 the module simply starts or resumes a run in response to the corresponding request. When set to 1, the module will pull down a wire-OR line in the PXI(e) backplane during run initialization, and start acquisition only when the line goes high again. This ensures that the last module ready to actually begin data taking will start the run in all modules. Similarly, the first module to end the run will stop the run in all modules.

Igor controls: Checkboxes in the *Run Control* tab

C global: SYNCH_WAIT

ControlTask to apply change: none

INSYNCH: InSynch is used to clear the Pixie on-board clock at the start of the data acquisition. When INSYNCH is 1, no particular action is taken. If this variable is 0 and SYNCHWAIT =1, then all system timers are cleared at the beginning of the next data acquisition run.

For the Pixie-4, INSYNCH is automatically set to 1 by the DSP after run start. This options should normally be only set for the first run to preserve time correlation across runs.

For the Pixie-500 Express and Pixie-4 Express, INSYNCH is unchanged by the DSP. Module timers are synchronized at every DSP reboot. This option should therefore only be set if all runs should start with a time stamp near zero.

Igor controls: Checkboxes in the *Run Control* tab

C global: IN_SYNCH

ControlTask to apply change: none

HOSTIO: A 4 word data block that is used to specify command options.

Igor controls: none

C global: none

ControlTask to apply change: none

RESUME: Set this variable to 1 to resume a data run; otherwise, set it to 0. Set to 2 before stopping a list mode run prematurely.

Ignored for Pixie-500 Express and Pixie-4 Express

Igor controls: none

C global: none

ControlTask to apply change: none

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

Igor controls: *Filter Range* in the *Energy* tab

C global: FILTER_RANGE. C library computes dependencies.

ControlTask to apply change: 5

MODULEPATTERN: To determine if an event is acceptable according to local or global coincidence tests, the DSP computes the quantity (MODULEPATTERN AND (HITPATTERN AND 0x0FFF). If nonzero, the event is accepted.

HITPATTERN bits 4..7 contain the following status information:

4: Logic level of FRONT panel input
5: Result of LOCAL coincidence test
6: Logic level of backplane STATUS line
7: Result of GLOBAL coincidence test (TOKEN backplane line)

Logic levels are captured at the time the coincidence window closes.

Consequently, to accept events based only on the local coincidence test, bit 5 of MODULEPATTERN must be 1, and all others zero. To accept events based only on the global coincidence test, bit 7 of MODULEPATTERN must be 1, and all others zero. To accept events based on both tests (either test passed => accept), set bits 5 and 7 of

MODULEPATTERN to one, others to zero.
Other values of MODULEPATTERN can in principle be used, but are not tested and/or supported at this time

Igor controls: Checkboxes in the CHASSIS SETUP panel

C global MODULEPATTERN

ControlTask to apply change: none

NNSHAREPATTERN: 16 bit user defined control word for PXI-PDM. If enabled (MODCSRA), the Pixie-4 module writes this word to its left neighbor using a PXI left neighbor line, which hopefully is a PDM. The PDM uses this word to make a coincidence accept/reject decision based on the hit pattern from all modules.

Currently not implemented for the Pixie-500 Express and Pixie-4 Express

Igor controls: Checkboxes in the CHASSIS SETUP panel

C global NNSHAREPATTERN

ControlTask to apply change: none

XETDELAY: If a front panel signal is used as an external fast trigger for the backplane, this variable specifies the delay for the auto-generated event trigger. The event trigger indicates that at least one trigger enabled channel has a valid hit (pulse present, no pileup) so the delay should typically be set to a value slightly larger than the energy filter rise time plus energy filter flat top of the channels to be triggered externally

If XETDELAY=0, the front panel signal is ignored for triggering and neither external fast triggers nor external event triggers are issued to the backplane.

Currently not implemented for the Pixie-500 Express and Pixie-4 Express

Constraints**:** XETDELAY >= 0
XETDELAY <= 65535

Igor controls: Validation Delay... in the Chassis Setup Panel

C global: XET_DELAY.

ControlTask to apply change: 5

DBLBUFCSR: A register containing several bits to control the double buffer (ping pong) mode to read out external memory. In the future, these control bits may be moved to the CSR register in the System FPGA.

Bit 0: Enable double buffer: If this bit is set, transfer list mode data to external memory in double buffer mode. **Must be set/cleared for all modules in the system.** If set, clear bit 1 of MODCSRA. Set by host, read by DSP.

Bit 1: Host read: Host sets this bit after reading a block from external memory to indicate DSP can write into it again. Set by host, read and cleared by DSP.

Bit 2: reserved

Bit 3: Read_128Ki_first: If run halted because host did not read fast enough and both blocks in external memory are filled, DSP will set this bit to indicate host to first read from block 1 (starting at address 128Ki), else (if zero) host should first read from block 2. Set by DSP, read by host. Cleared by DSP at runstart or resume

Not recommended for Pixie-4. Ignored for Pixie-500 Express and Pixie-4 Express

Igor controls: Radio buttons in the *Run Control* tab

C global DBLBUFCSR

ControlTask to apply change: 5

CCONTROL: A register containing several bits to specify options for the C library execution. This variable is part of the DSP parameters so that it will be saved with all the others in the .set file, but it is not used in the DSP.

Bit 0-3: Reserved.

Bit 4: If set, print debug messages during the boot process.

Bit 5: If set, print detailed error messages during error check of the list mode data.

Bit 6: If set, print additional debug messages during error check of the list mode data.

Bit 7: If set, print other debug messages that are normally suppressed.

Bit 8: If cleared, run list mode data acquisitions in interrupt mode. The interrupts automatically write the list mode data to file when a buffer in PC memory is filled by the DMA data transfer. Runtask 0x440# called by the top level program (e.g. Pixie Viewer) does nothing more than return the current total number of buffers written to file. Runtask 0x440# can be executed quite infrequently.
If set, run list mode data acquisition in polling mode. No interrupts are active. Runtask 0x440# called by the top level program checks for list mode data to be ready to be written to file, and if so it performs the write. Fast, frequent execution of runtask 0x440# is essential to avoid readout dead time.
Ignored for Pixie-4

Bit 9: If set, check list mode data for errors before writing to file. This bit must be set for Run type 0x401. Ignored for Pixie-4

Bit 10: Reserved.

Bit 11: If set, return "new" data in DMA buffer when polled for run progress by task 0x440x. Ignored for Pixie-4

Bit 12: If set, run 0x400 and 0x10x as a separate thread

Bit 13: If set, print debug messages for run start/stop

Bit 14: If set, debug messages are printed in Igor and to a file PIXIEmsg.txt

Bit 15: If set, the BLcut parameter is not automatically recomputed after gain or filter changes

Igor controls: Various advanced checkboxes

C global C_CONTROL

ControlTask to apply change: n/a

U00: Many unused, but reserved, data blocks have names of the structure Unn.

Those unused data blocks which reside in the block of input parameters for each channel are called UNUSEDA and UNUSEDB.

EXTRAIN: A block of 8 input variables used by customized or user-written DSP code.

HOSTIODATA: A 4 word data block that is used for data input in some control tasks.

Igor controls: none

C global: none

ControlTask to apply change: none

PDMMASKA, PDMMASKB, PDMMASKC: reserved

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

4.2 Channel input variables

All channel-0 variables end with "0", channel-1 variables end with "1", etc. In the following explanations the numerical suffix has been removed. Thus, e.g., CHANCSRA0 becomes CHANCSRA, etc.

CHANCSRA: The control and status register bits switch on/off various aspects of the Pixie-4 operation.

Bit 0: Respond to group triggers only.

Set this bit if you want to the acquisition for this channel to be controlled by the distributed group trigger instead of the local trigger for this channel. Waveforms are then always captured based on the distributed triggers, but channel time stamps are based on either local or group triggers ("local" option, bit 13 below), and energies are based on local triggers. The group trigger is a combination of triggers from all channels enabled for triggers (bit 4). Bit 0 should be set for all channels in the group, trigger enabled or not, to ensure the same routing delays for trigger distribution in all channels.

Note: To distribute group triggers between modules, bit 2 in the variable MODCSRA has to be set as well.

Bit 1: reserved

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

Bit 3: Read always
Channels marked as such will contribute to list mode data, even if they did not report a hit. This is most useful when acquiring induced signal waveforms on spectator electrodes, i.e., electrodes that did not collect any net charge, but only saw a transient induced signal.
Ignored for Pixie-500 Express and Pixie-4 Express. Channels set to respond to group trigger will always be recorded based on that trigger.

Bit 4: Enable trigger.
Set this bit for channels that are supposed to contribute to an event trigger.

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 handles negative signals by inverting them immediately after entering the FPGA.

Bit 6: GFLT (Veto) required / Reject at Veto Low
Pixie-4: Set this bit if you want to validate or veto events based on a global external signal distributed over a PXI bussed line named GFLT or Veto. When the bit is cleared, the GFLT (Veto) signal is ignored. When set, the event is accepted only if validated by GFLT (Veto). To be validated, the GFLT (veto) signal must be a logic 0 at time PEAKSEP * 2^(FILTERRANGE) after the rising edge of a pulse. Polarity can be reversed in CHANCSRC
Pixie-4 Express and Pixie-500 Express: Set bit to reject events during which Veto is low. The timing is the same as for the Pixie-4.

Bit 7: Histogram energies.
Set this bit to histogram energies from this channel in the on-board MCA memory.

Bit 8: Reserved.
Set to 0.

Bit 9: If set, allow negative number as the result of the pulse height computation. This may be useful in list mode runs to return a rough measure of an inverted pulse. Due to the binary representation of negative numbers, the pulse height will be histogrammed into bin 64Ki-abs(pulse height) of the spectrum. This option is ignored in MCA runs.

Bit 10: Compute constant fraction timing.
This pulse shape analysis computes the time of arrival for the signal from the recorded waveform. The result is stated in units of $1/256^{th}$ of a sampling period (13.3 ns). Time zero is the start of the waveform. Ignored for Pixie-500 Express and Pixie-4 Express

Bit 11: Reserved.
(Enabled multiplicity contribution in DGF-4C)

Bit 12: GATE required / Reject at Gate Low

Pixie-4: Set this bit if you want to validate or veto events based on a individual signal. GATE is distributed over a PXI left neighbor line, for example from a PDM in the slot to the left of the Pixie-4. Each channel has its own line. When the bit is set, the event is accepted only if validated by GATE. To be validated, the GATE input must have a rising edge within a time window (defined by GATEWINDOW and GATEDELAY) around the rising edge of a detector pulse. When the bit is cleared, the GATE input is not used for event validation, but its status is reported in the list mode output data. Polarities of the edge starting the Window and the status required for accepting events can be selected in CHANCSRC.

Pixie-4 Express: Set bit to reject events during which GATE is low. The timing is the same as for the Pixie-4.

Ignored for Pixie-500 Express

Bit 13: Local trigger timestamp

If set, use the local trigger to latch the time stamp even in group trigger mode, else use the distributed group trigger. Ignored for Pixie-500 Express and Pixie-4 Express

Bit 14: Estimate energy if channel not hit.

If set, the DSP reads out energy filter values and computes the pulse height for a channel that is not hit, for example when "read always" in group trigger mode. If not set, the energy will be reported as zero if the channel is not "hit"

For Pixie-500 Express and Pixie-4 Express, pulse heights computed in group trigger mode are always based on the distributed (not local) trigger.

Bit 15: Reserved.

Igor controls: Checkboxes in the PARAMETER SETUP panel

C global CHANNEL_CSRA

ControlTask to apply change: 5

CHANCSRB: Control and status register B. (for user code)

Bit 0: If set, call user written DSP code.

Bit 1: If set, all words in the channel header except Ndata, TrigTime and Energy will be overwritten with the contents of URETVAL. Depending on the run type, this allows for 6, 2 or 0 user return values in the channel header.

Bit2..15: reserved. Set to 0.

Bits 2 and 3 are used in MPI custom code.

Igor controls: Variables in the USER CONTROL panel

C global CHANNEL_CSRB

ControlTask to apply change: none

CHANCSRC: Control and status register C.

Bit 0: GFLT polarity / Reject at Veto High
Pixie-4: Controls polarity of GFLT to be considered present for accepting events, i.e. GFLT must be zero to record events instead of 1
Pixie-4 Express: Set bit to reject events during which VETO is high. Ignored for Pixie-500 Express

Bit 1: GATE acceptance polarity / Reject at Gate High
Pixie-4: Controls polarity of GATE to be considered present for accepting events, i.e. the GATE status latched at the time of the rising edge of the detector pulse must be zero to record events instead of 1.
Pixie-4 Express: Set bit to reject events during which GATE is high.
Ignored for Pixie-500 Express

Bit 2: Use GFLT for GATE.
If set, use GFLT/VETO as the input the GATE logic. Counters for GATE/VETO rate and time then only count the GATE signals, not GATE OR VETO as ususal. Ignored for Pixie-500 Express

Bit 3: Disable pileup inspection.
If set, pulses are accepted even if a pileup was detected.

Bit 4: Disable out-of-range rejection (Pixie-4 only)
If set, pulses are accepted even if the ADC input goes out of range. This can be used for detectors with occasional very large pulses. The energy filter essentially saturates for the time the signal is out of range, which means the reported energy is a measure of how long the signal is out of range and thus a coarse measure of the energy (assuming exponential decay)

Bit 5: Invert pileup inspection
If set, only accept events with pileup. May be useful to capture double pulse events.

Bit 6: Pause pileup inspection
If set, disable pileup inspection for 32 clock cycles (426 ns). May be useful for systems where the detector output shows significant ringing that causes two or more triggers on the same pulse (especially those with higher amplitude), to avoid these events to be rejected as piled up.

Bit 7: Gate edge polarity inverted / Count @ GATE falling
Pixie-4: If set, the GATE Window counter is started at the falling edge of the GATE signal instead of on the rising edge.
Pixie-4 Express: If set, count falling edge of GATE signal for GATE/VETO rate (GCOUNT) and count the time GATE is low in the GATE time (GDT). Otherwise, count rising edge and time high.
Ignored for Pixie-500 Express

Bit 8: Gate Statistics
If set, run statistics are in GATE mode where all counters except RUN TIME and TOTAL TIME are only active if a GATE or VETO signal is present. Essentially requires GATE or VETO to be present for the channel to be counted as live.(Rejection of pulses by GATE or VETO must be enabled separately.) GDT counts the time GATE or VETO are high

independent of a run being in progress.
If not set, count time and other counters are independent of GATE or VETO. GATE or VETO presence is counted separately in GDT only if the module is live. Ignored for Pixie-500 Express

Bit 9: GDT counts "allowed" time / Count @ VETO falling
Pixie-4: Invert the GDT counter for GATE or VETO presence. If not set, GDT counts the time during which pulses would be rejected (rejection must be enabled separately). If set, GDT counts the time during which pulses are allowed.
Pixie-4 Express: If set, count falling edge of VETO signal for GATE/VETO rate (GCOUNT) and count the time VETO is low in the GATE time (GDT). Otherwise, count rising edge and time high.
Ignored for Pixie-500 Express

Bit 10: No gate pulse
If set, the gate pattern bits in the event hit pattern is simply the status of the gate input at the time of local or group fast trigger.
If not set, the gate pattern bits reflect the coincidence of a (delayed and stretched) gate pulse and the local fast trigger. Ignored for Pixie-500 Express

Bit 11: 4x traces (Pixie-4 only)
If set, each waveform sample contains the average of 4 ADC samples. The time between waveform samples is 4x the ADC sampling time. If this bit is enabled for channel 0..3, bit 4..7 in the List mode data RUNFORMAT word is set to indicate the 4x time scale.

Bit 12: Out of range to Veto (Pixie-4 only)
If set, whenever the signal goes out of range on the lower limit (<=0), the module issues a signal on the VETO backplane line (for all modules).

Bit 13: Final GATE to front panel (Pixie-4 Express only)
If set, the high density front panel connector outputs the final gate applied to the channel, after all optional inversions and user defined window length are applied. If not set, the signal is the Hit status of the channel.
If set, the final gate also replaces bit 0 of the ADC samples shown in the oscilloscope panel, plotted separately in the ADC filter display. This can be used to set up delays for proper timing with the detector signal.

Bit 14: Include GCOUNT in List mode record (Pixie-4 Express only)
If set, the XIA_PSA and USER_PSA words of the list mode event record contain the value [lower 32 bit] of the GCOUNT counter, which counts rising (or falling) edges on GATE or VETO. This can be used to tag events with an externally controlled "cycle number" of the experiment.

Notes: - Only one of bits 3, 5, and 6 is meant to be set at the same time, but this is not enforced.

- It is possible to enable rejection of events for VETO high and low (or GATE high and low), but that would mean the no events are recorded.

Igor controls: Checkboxes in the PARAMETER SETUP panel

C global CHANNEL_CSRC

ControlTask to apply change: 5

TRACKDAC: This DAC determines the DC-offset voltage. The offset in volts displayed in Igor and contained in the C global VOFFSET can be calculated using the following formula:

Offset [V] = 2.5V * ((32768 - TRACKDAC) / 32768)

Constraints**:** TRACKDAC >= 0
TRACKDAC <= 65535

Igor controls: *Offset [V]* in the OSCILLOSCOPE panel

C global VOFFSET

ControlTask to apply change: 0

SGA: The index of the relay combinations of the switchable gain amplifier.

For the Pixie-4, the analog SGA gain is G = (1+Rf/Rg)/2 with

Rf = 2150 - 120*((SGA & 0x1)>0) - 270*((SGA &0x2)>0) - 560*((SGA &0x4)>0)

Rg = 1320 - 100*((SGA & 0x10)>0) - 300*((SGA & 0x20)>0)- 820*((SGA & 0x40)>0)

The C library computes the closest SGA setting for a given voltage gain VGAIN and adjusts the parameter DIGGAIN to compensate differences between VGAIN and the gain from SGA up to ±10%.

For the Pixie-500 and Pixie-500 Express, only SGA bit zero is active and it changes the gain from 1 to 3.

For the Pixie-4 Express, only SGA bit 0..2 are active and it changes the gain as follows. In the future, this may depend on the hardware version, for example high rate digitizers may have fewer gain options.

SGA = 0	gain = 1.6
SGA = 1	gain = 6.7
SGA = 2	gain = 2.4
SGA = 3	gain = 9.9
SGA = 4	gain = 3.5
SGA = 5	gain = 14.7
SGA = 6	gain = 5.4
SGA = 7	gain = 22.6

Constraints**:** SGA >= 0
SGA <= 127

Igor controls: *Gain [V/V]* in the OSCILLOSCOPE panel

C global VGAIN

ControlTask to apply change: 0

DIGGAIN: The digital gain factor for compensating the difference between the user-desired voltage gain and the SGA gain. This is computed by the C library and limited to 10% in the following way:

DG = voltage gain / SGA gain - 1.0;
DIGGAIN = 65535*DG if DG >0
= 65536 + 65535*DG if DG <0

Constraints**:** DIGGAIN >= 0 for positive DG
DIGGAIN <= 6554
. DIGGAIN >= 64Ki - 6554 for negative DG
DIGGAIN <= 64Ki

Other values between 6554 and 64Ki-6554 are possible, but may lead to binning errors or other undesirable effects

Igor controls: *Gain [V/V]* in the OSCILLOSCOPE panel

C global VGAIN

ControlTask to apply change: none

GAINDAC: Reserved and not supported.

BASELINEPERCENT: This variable sets the target DC-offset level for automatic adjustment (ControlTask 0x83) in terms of the percentage of the ADC range.

Constraints**:** BASELINEPERCENT >= 0
BASELINEPERCENT <= 100

Igor controls: *Offset (%)* in the OSCILLOSCOPE

C global: BASELINE_PERCENT.

ControlTask to apply change: none

UNUSEDA0 or UNUSEDA1: Reserved.

SLOWLENGTH: The rise time of the energy filter (also called peaking time) depends on SLOWLENGTH:

Energy Filter Rise Time = SLOWLENGTH * 2^FILTERRANGE * dt

Constraints**:** SLOWLENGTH >= 2
SLOWLENGTH + SLOWGAP <= 127

dt = 13.3ns for Pixie-4, 8ns for Pixie-4-Express and Pixie-500 Express

Igor controls: Energy Filter Rise Time in the Energy tab

C global: ENERGY_RISETIME

ControlTask to apply change: 5

SLOWGAP: The flat top of the energy filter (also called gap time) depends on SLOWGAP:

Energy Filter Flat Top = SLOWGAP * 2^ FILTERRANGE * dt.

Constraints**:** SLOWGAP >= 3
SLOWLENGTH + SLOWGAP <= 127

dt = 13.3ns for Pixie-4, 8ns for Pixie-4-Express and Pixie-500 Express

Igor controls: Energy Filter Flat Top in the Energy tab

C global: ENERGY_FLATTOP

ControlTask to apply change: 5

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

Trigger Filter Rise Time = FASTLENGTH * dt.

Constraints**:** FASTLENGTH >= 2
FASTLENGTH + FASTGAP <= 63

dt = 13.3ns for Pixie-4, 8ns for Pixie-4-Express and Pixie-500 Express

FASTLENGTH is computed by the C library from the Trigger Filter Rise Time value entered by the user in the *Trigger* tab in Igor.

Igor controls: Trigger Filter Rise Time in the Trigger tab

C global: TRIGGER_RISETIME

ControlTask to apply change: 5

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

Trigger Filter Flat Top = FASTGAP * dt.

Constraints**:** FASTGAP >= 0
FASTLENGTH + FASTGAP <= 63

dt = 13.3ns for Pixie-4, 8ns for Pixie-4-Express and Pixie-500 Express

FASTGAP is computed by the C library from the Trigger Filter Flat Top value entered by the user in the *Trigger* tab in Igor.

Igor controls: Trigger Filter Flat Top in the Trigger tab

C global: TRIGGER_FLATTOP

ControlTask to apply change: 5

INTEGRATOR: 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.

For Pixie-4: INTEGRATOR = 3,4,5: same as 1, but multiply energy by 2, 4, or 8

For Pixie-4 Express and Pixie-500 Express, the resulting pulse height value is scaled with the length of gap sum (INTEGRATOR = 1) or leading sum (INTEGRATOR =2). In addition, the result is multiplied with the decay time tau, which is here used as an arbitrary scaling factor. If tau is set to 1, the pulse height will always fall into the visible range of the spectrum.

Igor controls: *Integrator* in the *Energy* tab

C global: INTEGRATOR.

ControlTask to apply change: none

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 +5

Constraints: If PEAKSEP >128, PEAKSEP = PEAKSAMPLE +1
0 < PEAKSEP - PEAKSAMPLE < 7

Igor controls: none

C global: none

ControlTask to apply change: 5

LOG2EBIN: This variable controls the binning of the histogram. Energy values are always calculated to 16 bits precision. The energy value corresponds to 4 times the 14-bit ADC amplitude. The modules, however, do not have enough histogram memory available to record 64Ki 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 by downshifting the computed energy. The following formula describes which MCA bin a value of Energy will contribute:

MCAbin = (Energy) * 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 C global BINFACTOR contains the absolute value of LOG2EBIN

Constraints**:** LOG2EBIN >= 65520 (equiv. -16)
LOG2EBIN <= 65535 (equiv. -1)
and additionally LOG2EBIN = 0 is allowed

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

Igor controls: *Binning Factor* in the *Advanced* tab

C global: BINFACTOR

ControlTask to apply change: none

FASTTHRESH: This is the trigger threshold used by the trigger/filter FPGA. It is compared to the output of the fast filter; if the filter output is greater or equal to FASTTHRESH, a trigger is issued. For a pulse with a given height, the trigger filter output scales with the trigger filter rise time FASTLENGTH, i.e.

filter output = FASTLENGTH * pulse amplitude/4 * form factor

where "pulse amplitude" is the amplitude in ADC units (as displayed in the oscilloscope) and form factor describes the effect of the shape of the pulse during FASTLENGTH. For a square pulse, form factor = 1; for a slow rising or fast decaying pulse, form factor will be less than 1 because the amplitude is not constant during FASTLENGTH.

The threshold value TRIGGER_THRESHOLD set in the *Trigger* tab in Igor is scaled in the C library for FASTLENGTH so that a given value of TRIGGER_THRESHOLD causes triggering at the same pulse amplitude independent of FASTLENGTH:

FASTTHRESH = TRIGGER_THRESHOLD * FASTLENGTH

If FASTTHRESH = 0, the trigger logic is switched off. (The "trigger enable" bit in ChanCSRA only disables triggers being sent to the DSP for event processing and to other channels/modules for synchronous waveform capture. FASTTHRESH = 0 switches off local trigger generation, so e.g. the input count rate becomes zero.)

Constraints**:** FASTTHRESH >= 0
FASTTHRESH <= 4095
the lower 3 bits are ignored

Igor controls: *Threshold*, *Rise Time* in the *Trigger* tab

C global: TRIGGER_THRESHOLD, TRIGGER_RISETIME

ControlTask to apply change: 5

MINWIDTH: Unused.

MAXWIDTH: Unused for Pixie-4 Express and Pixie-500 Express. This value aids the pile up inspection. MAXWIDTH is the maximum duration, in sample clock ticks (13.3 ns), which the output from the fast filter may spend over threshold. Pulses longer than that will be rejected as piled up. The recommended setting is zero to disable this feature. Otherwise, a possible starting value is

MAXWIDTH = FASTLENGTH + FASTGAP+ SignalRiseTime / dt.

Constraints**:** MAXWIDTH >= 0
MAXWIDTH <= 255

dt = 13.3ns for Pixie-4

Once the other parameters have been optimized with MAXWIDTH =0, one can use the MAXWIDTH cut to improve the pile up rejection at high count rates. MAXWIDTH should be tuned by observing the main energy peak in the spectrum for fixed time intervals. Once the MAXWIDTH cut is too tight there will be a loss of efficiency in the main peak. Setting MAXWIDTH to such a value that the efficiency loss in the main peak is acceptable will give the best overall performance in terms of efficiency and pile up rejection.

Igor controls: none

C global: none

ControlTask to apply change: 5

PREAMPTAUA, PREAMPTAUB: High word and low word of the preamplifier's exponential decay time. The two variables are used to store the value with higher precision. The time τ is measured in μs. The two words are computed as follows.

PREAMPTAUA = floor(τ)
PREAMPTAUB = 65536 * (τ - PreampTauA)

To recover τ use: τ = PREAMPTAUA + PREAMPTAUB / 65536

Constraints**:** TAU >= 1/65536 μs
TAU <= 65535 μs

Note In integrator mode (INTEGRATOR >0), the decay time correction is meaningless. In the Pixie-4 Express and Pixie-500 Express, PREAMPTAU is therefore used as an arbitrary scaling factor for the computed pulse height in integrator mode. In the Pixie-4, PREAMPTAU is ingnored in integrator mode

Igor controls: *Tau* in the *Energy* tab

C global: TAU.

ControlTask to apply change: none.

RESETDELAY: This variable sets the timeout to restart processing in the trigger/filter FPGA automatically after it captured an event, but has not received event validation. In most circumstances, event capture constitutes validation and this timeout does not apply. However, if a channel is not trigger enabled or responds to group triggers only, a local event by itself is not valid, so the channel waits for RESETDELAY clock cycles to receive external validation. In other words, RESETDELAY is the window for trigger disabled channels to be included in an event *later* triggered by another channel

Constraints**:** RESETDELAY >= 0
RESETDELAY <= 65535

The default value is max(29, CD+4) and should normally not be changed by the user. CD is the value of COINCDELAY in processing clock ticks:

CD = COINCDELAY for the Pixie-4
Ignored for Pixie-500 Express and Pixie-4 Express

Igor controls: none

C global: none

ControlTask to apply change: 5

COINCDELAY: In the Pixie-4, this variable is used to delay (individually for each channel) the start of the coincidence window.

Since the coincidence test is applied only after validation, each channel may require individual delay if validation requires different amounts of time due to differences in the energy filter settings. For example, a channel with the energy filter rise time set to 6 μs would start the coincidence window 2 μs before a channel with a filter rise time of 8 μs, and thus simultaneous events in the second channel will be lost unless the coincidence window is at least 2 μs or the channel parameter COINCDELAY is set to 2 μs for the faster channel. The following formula should be used to determine COINCDELAY:

$$\text{COINCDELAY<ch>} = \max(\text{PEAKSEP}*2^{\text{FILTERRANGE}})_{\text{ch0-ch3}} - (\text{PEAKSEP}*2^{\text{FILTERRANGE}})_{\text{ch}}$$

User may choose different values based on the physics of the experiment.

In the Pixie-4 Express, this variable is used to delay (individually for each channel) the incoming ADC data by a small amount to compensate for cable delays or other factors.

The Igor control and the C global are in units of ns, the DSP variable is in units of delay cycles (13.3ns for Pixie-4, 4*2ns for the Pixie-4 Express with 500 MHz ADCs, 2*8ns for the Pixie-4 Express with 125 MHz ADCs). Currently not implemented for Pixie-500 Express

Constraints**:** COINCDELAY >= 0
COINCDELAY <= 65533 (for Pixie-4)
COINCDELAY <= 253 (for Pixie-4 Express)

Igor controls: *Delay* in the *Coincidence* tab

C global: none

ControlTask to apply change: 5

TRACELENGTH: This variable determines the length of captured waveforms in list mode runs (in clock cycles).

TRACELENGTH = Trace Length/dt

Constraints**:** TRACELENGTH >= 0
TRACELENGTH <= TLMAX

dt = 13.3ns for Pixie-4, 2ns for the Pixie-500 Express, 8ns for the Pixie-4 Express

TRACELENGTH must be a multiple of 32 for the Pixie-500 Express and Pixie-4 Express

TLMAX = 1024 for Pixie-4

TLMAX = 4096 for Pixie-500 Express and Pixie-4 Express

Igor controls: *Trace Length* in the *Waveform* tab

C global: TRACE_LENGTH

ControlTask to apply change: 5

USERDELAY: This variable specifies the number of pre-trigger samples in the captured waveform.:

USERDELAY = Trace Delay/dt

Constraints**:** USERDELAY >= 0
USERDELAY <= TRACELENGTH for Pixie-4
USERDELAY <= 2047 for Pixie-500 Express

dt = 13.3ns for Pixie-4, 2ns for the Pixie-500 Express, 8ns for the Pixie-4 Express

USERDELAY must be a multiple of 4 for the Pixie-4 Express and the Pixie-500 Express

Igor controls: *Trace Delay* in the *Waveform* tab

C global: TRACE_DELAY

ControlTask to apply change: 5

XWAIT: Extra wait states. XWAIT is used when acquiring untriggered traces in a control run with ControlTask = 0x84, e.g. the traces in the Igor oscilloscope display. The time ΔT between data points in the output buffer is

for Pixie-4

ΔT = XWAIT *13.3ns

Constraints**:** XWAIT >= 4
XWAIT <= 65533
If XWAIT >12, it is limited to XWAIT = 13+ N*5

If XWAIT > 12, a filter is implemented during the acquisition to return each data point as the average over (XWAIT-3)/5 samples.

In a test/debug mode, this parameter also controls how many extra clock cycles the DSP waits when reading extra long waveforms in real time from the ADC register rather than out of the FIFO memory. This occurs when acquiring data in list mode and asking for trace lengths longer than FIFOlength (1024), which is possible if the C library's tests are bypassed. The time between recorded samples is then ΔT = (3+XWAIT)*13.3ns.

for Pixie-500 Express and Pixie-4 Express

ΔT = XWAIT *5 *13.3ns

Constraints**:** XWAIT must be a power of 2 between 1 and 8192

If XWAIT > 1, a filter is implemented during the acquisition to return each data point as the average over XWAIT samples.

Igor controls: *dT* in the OSCILLOSCOPE

C global: XDT

ControlTask to apply change: none

XAVG: Only used in Controltask 0x84 for reading untriggered traces. XAVG stores the weight in the geometric-weight averaging scheme to remove higher frequency signal and noise components. The value is calculated as follows:

for Pixie-4

For a given dT (in μs), calculate the integer intdt = dT/0.0133
If intdt > 13, XAVG = floor(65536/((intdt-3)/5))
If intdt < 13, XAVG = 65535.

for Pixie-500 Express and Pixie-4 Express

$XAVG = \log_2 XWAIT$

Igor controls: *dT* in the OSCILLOSCOPE tab

C global: XDT

ControlTask to apply change: none

CFDTHR: Pixie-4: This sets the threshold of the software constant fraction discriminator. The threshold fraction CFD_THRESHOLD (f) is encoded as round(f*65536), with 0<f<1.

Pixie-500 Express, Pixie-4 Express: This variable is used for the PSA logic, setting a pulse detect threshold for individual samples.

Constraints**:** CFDTHR >= 0
CFDTHR <= 65535

Igor controls: *CFD Threshold* in the *Waveform* tab

C global: CFD_THRESHOLD

ControlTask to apply change: none

PSAOFFSET, PSALENGTH: 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.

PSAOFFSET = PSA Start / dt

PSALENGTH = (PSA End - PSA Start) / dt

Constraints**:** PSALENGTH >= 0
PSALENGTH <= TRACELENGTH – PSAOFFSET
PSAOFFSET >= 0
PSAOFFSET <= TRACELENGTH

dt = 13.3ns for Pixie-4

Ignored for Pixie-500 Express and Pixie-4 Express

Igor controls: *PSA Start*, *PSA End* in the *Waveform* tab

C global: PSA_START, PSA_END.

ControlTask to apply change: none

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. To reduce processing time, set BLCUT to zero to not check baselines.

ControlTask 6 can be used to measure baselines. The host computer can then histogram these baseline values and determine the appropriate value for BLCUT for each channel according to the standard deviation of the averaged baseline value. This is done automatically in the C library every time the decay time or the energy filter rise time or flat top is changed, setting BLCUT to 4 times the standard deviation.

The value of BLCUT depends on decay time, gain, filter settings, and the noise from the detector. Automatically computed values below 15 are suspicious and 15 is used instead. Values have to be measured as above and can not be derived easily from first principles.

Constraints: BLCUT >= 0
BLCUT <= 32767

Igor controls: *Baseline Cut* in the *Advanced* tab

C global: BLCUT.

ControlTask to apply change: none. Calling Controltask 128 (in the C library) automatically determines BLCUT.

LOG2BWEIGHT: The Pixie-4 module 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:

$$DC\_avg_{new} = DC\_avg_{old} + (DC - DC\_avg_{old}) * 2\text{^}LOG2BWEIGHT$$

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.

The DSP ensures that LOG2BWEIGHT will be negative. Larger (absolute) numbers mean the previous baseline measurements contribute more. 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-4 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 (in 16bit signed integer format), and this parameter does not need to be adjusted afterwards.

Constraints**:** LOG2BWEIGHT >= 65520 (equiv. -16)
LOG2BWEIGHT <= 65535 (equiv. =-1)
and additionally LOG2BWEIGHT = 0

Igor controls: *Baseline Averaging* in the *Advanced* tab

C global: BLAVG.

ControlTask to apply change: none.

GATEDELAY, GATEWINDOW: These variables set the coincidence window for the Gate signal to reject events. At the rising edge of the Gate signal, and internal Gate status bit goes high for the duration of GATEWINDOW. A GATEDELAY after a fast trigger the status bit is latched into GATEBIT. GATEBIT can be used to reject events in the FPGA, and it is reported in the hit pattern in the list mode data stream for offline processing if no online rejection is desirable.

GATEWINDOW = Gate Window /dt

GATEDELAY = Gate Delay /dt

Constraints: GATEWINDOW >= 1
GATEWINDOW <= 255
. GATEDELAY >= 1
GATEDELAY <= 254 for Pixie-4
<= 510 for Pixie-4 Express
<= 126 for Pixie-500 Express
GATEDELAY < PEAKSEP*2^FILTERRANGE
(for online rejection only)

dt = 13.3ns for Pixie-4, 8ns for Pixie-4 Express

Ignored for Pixie-500 Express

Igor controls: *Gate Delay*, *Gate Window* in the *Gate* tab

C global: GATE_WINDOW, GATE_DELAY

ControlTask to apply change: 5

QDC#DELAY, QDC#LENGTH: These two parameters specify the position and length of two "QDC" sums on the rising edge of a pulse, computed by the FPGA in real time. (#=0,1) The position is relative to the rising edge trigger point. The units are in ADC samples. Results are placed in the "User PSA" fields in the list mode data.

Constraints**:**
QDC#LENGTH >= 4
QDC# LENGTH <= 120
QDC#LENGTH must be multiple of 4
QDC# DELAY >= 0
QDC# DELAY <= 380
QDC1DELAY + QDC1LENGTH
> QDC0DELAY + QDC0LENGTH
The lowest 2 bits of QDC1DELAY and QDC0DELAY must be the same.

Ignored for Pixie-4, not yet implemented in Pixie-4 Express

Igor controls: IN PSA panel

C global: QDC#_DELAY, QDC0_LENGTH.

ControlTask to apply change: 5

FCFDTH: A threshold reserved for CFD timing implemented in the FPGA firmware. Pxiie-4e only

Igor controls: N/A

C global: FCFD_THRESHOLD

ControlTask to apply change: 5.

CHEXTRAIN: A block of 8 input variables used by customized or user-written DSP code

Igor controls: N/A

C global: CH_EXTRA_IN.

ControlTask to apply change: none.

This ends the block of channel input data. Note that there are four equivalent blocks of input channel data, one for each Pixie-4 input channel.

4.3 Module output parameters

We now show the output variables, again beginning with module variables and continuing afterwards with the channel variables. The output data block begins at the address 0x4100. Note, however, that this address could change. The output data block comprises of 256

words; 1 block of 64 is reserved for module data; 4 blocks of 48 words each hold channel data.

DECIMATION: This variable is a copy of the input parameter FILTERRANGE. It is copied as an output parameter for backwards compatibility

REALTIMEA, REALTIMEB: The real time clock. A,B are the high and middle word of a 48 bit counter; the lowest 16 bit are not visible as an output parameter. The clock is zeroed on power up, and in response to a synch request at run start (INSYNCH = 0, SYNCHWAIT = 1). The main purpose of this parameter is to see that the DSP is active, not for timing. RealTimeA is unused. Time units are 13.33ns.

RUNTIMEX, RUNTIMEA, RUNTIMEB, RUNTIMEC: The 64-bit run time. X,A,B,C words are the 16bit numbers that combine into the total 64 bit time. This time counter is active only while a data acquisition run is in progress. Comparing the run time with the total time allows judging the overhead due to data readout. Compute the run time using the following formula:

RunTime = 13.3ns * (RUNTimeX *64Ki^3 + RUNTimeA *64Ki^2 +
RUNTimeB * 64Ki + RUNTimeC)

NUMEVENTSX, NUMEVENTSA, NUMEVENTSB: Number of valid events serviced by the DSP.

Number of Events =
NUMEVENTSX * 64Ki^2 + NUMEVENTSA* 64Ki + NUMEVENTSB

NCOINCTRIGX, NCOINCTRIGA, NCOINCTRIGB: Reserved for number of coincidence rising edge trigger detected by the logic (all channels). Pixie-4 Express only

Number of Events =
NCOINCTRIGX * 64Ki^2 + NCOINCTRIGA* 64Ki + NCOINCTRIGB

BUFHEADLEN: At the beginning of each run the DSP writes a buffer header to the list mode data buffer. BUFHEADLEN is the length of that header. Currently, BUFHEADLEN is 6 in runtask 0x10# and 32 in runtask 0x400, but this value should not be hardcoded, it should be read from the DSP to ensure upgrade compatibility.

EVENTHEADLEN: For each event in the list mode buffer, or the level-1 buffer, there is an event header containing time and hit pattern information. EVENTHEADLEN is the length of that header. Currently, EVENTHEADLEN is 3 in runtask 0x10# and 0 in runtask 0x400, but this value should not be hardcoded, it should be read from the DSP to ensure upgrade compatibility.

CHANHEADLEN: For each channel that has been read, there is a channel header containing energy and auxiliary information. CHANHEADLEN is the length of this header. CHANHEADLEN varies between 2 and 32 words depending on the run type (see RUNTASK).

For Pixie-4, in runtasks 0x10#, the event and channel header lengths plus the requested trace lengths determine the maximum logically possible event size. The maximum event size is the sum of EVENTHEADLEN and the CHANHEADLENs plus the TraceLengths for all channels marked as good, i.e. which have bit 2 in the CHANCSRA set.

Example: With all four channels marked as good and required trace lengths of 1000 (i.e. 13.3μs) the maximum event size will be

```
MaxEventSize      = EVENTHEADLEN + 4*( CHANHEADLEN + 1000)
                  = 4039
```

In the last line typical values for EVENTHEADLEN (3) and CHANHEADLEN (9) were substituted. BUFHEADLEN equals 6. Thus there is room for at least 2 events in the list mode data buffer, which is 8192 words long. But there is only room for 1 event in the level-1 buffer used in compressed RUNTASKs 0x101-103, which contains only 4060 words.

CSFDTX, CSFDTA, CSFDTB, CSFDTC: Pixie-4 Express only. Combined Slow Filter Dead Time from all channels. This is the time the associated with each pulse that prohibited acquisition of a second pulse due to pileup inspection, ORed for all 4 channels. See the user manual for a description of the measured time. Convert the four words into a time using the formula:

```
CSFDT = 13.3ns * (  CSFDTX *64Ki^3 + CSFDTA *64Ki^2 +
                    CSFDTB *64Ki + CSFDTC      )
```

TOTALTIMEX, TOTALTIMEA, TOTALTIMEB, TOTALTIMEC: A timer to track the total time an acquisition was requested by the host. RUNTIME excludes the time waiting for host readout, TOTALTIME is the closest measure to the true lab time passed since the most recent "new run" command (the first spill in a series). X,A,B,C words are as for the RunTime. Compute the total time using the following formula:

```
TotalTime = 13.3ns *  ( TOTALTIMEX * 64Ki^3 +   TOTALTIMEA * 64Ki^2 +
                        TOTALTIMEB * 64Ki  +     TOTALTIMEC      )
```

CCTX, CCTA, CCTB, CCTC: Pixie-4 Express only. Combined Count Time from all channels. This is the time the associated with each pulse that prohibited acquisition of a second pulse due to pileup inspection. See the user manual for a description of the measured time. Convert the four words into a time using the formula:

```
CCT = 13.3ns * (      CCTX *64Ki^3 +     CCTA *64Ki^2 +
                      CCTB *64Ki +       CCTC          )
```

HOSTODATA: A 4 word data block that is used for data output in some control tasks.

Igor controls: none

C global: none

ControlTask to apply change: none

EMWORDS, EMWORDS2: Each of these variables are two-word arrays (high word first) counting the number of 16 bit words written to external memory. (Pixie-4 only)

DSPERROR: This variable reports error conditions:

= 0 (NOERROR), no error

= 1 (RUNTYPEERROR), unsupported RunType

= 2 (RAMPDACERROR), Baseline measurement failed

= 3 (EMERROR), writing to external memory failed

SYNCHDONE: This variable is set to 1 after the DSP comes out of its synchronization loop when a run start request was issued with SYNCHWAIT=1.

TEMPERATURE: reserved.

AOUTBUFFER: Address of the IO data buffer. 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 may change with every new compilation. Therefore your code should never assume to find any given buffer at a fixed address. The length of the IO data buffer is always 8192.

HARDWAREID: ID of the system FPGA configuration.

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

FIPPIID: ID of the FiPPI FPGA configuration

INTRFCID: Unused

DSPRELEASE: DSP software release number

DSPBUILD: DSP software build number

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

4.4 Channel output parameters

The following channel variables contain run statistics. Again the variable names carry the channel number as a suffix. For example the COUNTTIME words for channel 2 are COUNTTIMEX2, COUNTTIMEA2, COUNTTIMEB2, COUNTTIMEC2. Channel numbers run from 0 to 3.

COUNTTIMEX, COUNTTIMEA, COUNTTIMEB, COUNTTIMEC: Total counting time as measured by the trigger/filter FPGA of that channel. See the user manual for a description of the measured time. Convert the four COUNTTIME words into a CountTime using the formula:

CountTime = dt * (COUNTTIMEX *64Ki^3 + COUNTTIMEA *64Ki^2 +
COUNTTIMEB *64Ki + COUNTTIMEC)

dt = 13.3ns *16 for Pixie-4, 8ns *32 for Pixie-4 Express and Pixie-500 Express

FASTPEAKSX, FASTPEAKSA, FASTPEAKSB: The number of events detected by the fast filter is:

FAST_PEAKS = FASTPEAKSX*64Ki^2 + FASTPEAKSA*64Ki + FASTPEAKSB

FTDTX, FTDTA, FTDTB, FTDTC: Fast Trigger dead time is the time the fast trigger output was above threshold and thus not ready to detect further triggers, as measured by the trigger/filter FPGA. See the user manual for a description of the measured time. Convert the four words into a time using the formula (note missing factor 16):

FTDT = dt * (FTDTX *64Ki^3 + FTDTA *64Ki^2 +
FTDTB *64Ki + FTDTC)

dt = 13.3ns for Pixie-4, 8ns for Pixie-4 Express and Pixie-500 Express

SFDTX, SFDTA, SFDTB, SFDTC: Slow Filter Dead Time is the time the associated with each pulse that prohibited acquisition of a second pulse, for example due to pileup inspection or DSP readout. See the user manual for a description of the measured time. Convert the four words into a time using the formula:

SFDT = dt * (SFDTX *64Ki^3 + SFDTA *64Ki^2 +
SFDTB *64Ki + SFDTC)

dt = 13.3ns *16 for Pixie-4, 8ns *32 for Pixie-4 Express and Pixie-500 Express

GCOUNTX, GCOUNTA, GCOUNTB: The number of gate pulses for this channel
GCOUNT = GCOUNTX *64Ki^2 + GCOUNTA *64Ki + GCOUNTB

NOUTX, NOUTA, NOUTB: The number of output counts in this channel (high, low)
NOUT = NOUTX *64Ki^2 + NOUTA *64Ki + NOUTB

GDTX, GDTA, GDTB, GDTC: Gate Dead Time is the time during which a channel was gated. See the user manual for a description of the measured time. Convert the four words into a time using the formula:

GDT = dt * (GDTX *64Ki^3 + GDTA *64Ki^2 +
GDTB *64Ki + GDTC)

dt = 13.3ns *16 for Pixie-4, 8ns *32 for Pixie-4 Express and Pixie-500 Express

ICR: ICR is an averaged measure of the current input count rate. It is updated if a run is in progress or not. The averaging is implemented such that at every update,

$Average_{new} = (Average_{old} + Number\ fast\ triggers\ in\ update\ period)/2$

The value reported in the variable ICR is equal to $2*Average_{new}$. Updates occur every 32*64Ki clock cycles. Thus to compute the rate in counts/s, the value in ICR has to be divided by 32*64Ki * 13.3ns. The reported value is precise to about 50 counts/s, with a maximum count rate of about one million counts/s

Not computed for Pixie-500 Express

OORF: OORF is an averaged measure of the fraction of time the channel is out of range. It is updated if a run is in progress or not. The averaging is implemented such that at every update,

$Average_{new} = (Average_{old} + Time\ out\ of\ range/64)/2$

The value reported in the variable OORF is equal to $2*Average_{new}$. Updates occur every 32*64Ki clock cycles. Thus to compute the out of range fraction in percent, the value in OORF has to be multiplied by (100% / 64Ki).

Not computed for Pixie-500 Express

NPPIX, NPPIA, NPPIB: The number of pulses passing pileup inspection in this channel. Normally equal to NOUT, but in cases where output events are further limited e.g. by gating or coincidence conditions, this value can be used for dead time correction.
NPPI = NPPIX *64Ki^2 + NPPIA *64Ki + NPPIB

Not computed for Pixie-4

4.5 Control Tasks

The DSP can execute a number of control tasks, which are necessary to perform functions 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. The tasks listed here can be used by a top level program calling the API to execute specific tasks. More DSP control tasks are used within the API and can in principle be used when modifying the C library itself – recommended only for experienced programmers with full understanding of the Pixie operation.

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 (run task 0x40FF) 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 0x0: **SetDACs**
Write the GAINDAC and TRACKDAC values of all channels into the respective DACs. Reprogramming the DACs is required to make effective changes in the values of the variables GAINDAC{0…3}, TRACKDAC{0…3}.

Control Task 0x1: **Connect inputs**
Close the input relay to connect the Pixie electronics to the input connector. (Pixie-4 only)

Control Task 0x2: **Disconnect inputs**
Open the input relay to disconnect the Pixie electronics from the input connector. (Pixie-4 only)

Control Task 0x3: Reserved for internal use

ControlTask 0x4: Reserved for internal use

ControlTask 0x5: **ProgramFiPPI**
This task writes all relevant data to the FiPPI control registers.

ControlTask 0x6: Measure Baselines
Control Task 0x2A is recommended instead for Pixie-4 Express and Pixie-500 Express.
This routine is used to collect baseline values. Currently, the DSP collects six words, B0L, B0H, B1L, B1H, time stamp, and ADC value, for each baseline. 1365 baselines are collected until the 8192-word I/O buffer is almost completely filled. The host computer can then read the I/O buffer and calculate the baseline according to the formula:

$$Baseline = (B1 - B0 * \exp(-XP)) * Bnorm$$

with
B1 = (B1L+B1H*65536)
B0 = (B0L+B0H*65536)
XP = (SLOWLENGTH+SLOWGAP) * 2^{FR} / (CLK*TAU)
Bnorm = 2^{-9} / SLOWLENGTH for FR >=2 (Pixie-4)
= 2^{-8} / SLOWLENGTH for FR =1 (Pixie-4)
= 2^{-7} / SLOWLENGTH for FR =0 (Pixie-4)
= $2^{-(FR)}$/ SLOWLENGTH for FR <6 (P4e, P500e)
= $2^{-(FR-2)}$/ SLOWLENGTH for FR >=6 (P4e, P500e)

TAU = PREAMPTAUA+ PREAMPTAUB/65536
CLK = 75 for Pixie-4, 125 for Pixie-4 Express and Pixie-500 Express
FR = FILTERRANGE

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 4 times the standard deviation. Baseline values can also be plotted against time stamp or ADC value to explore the detector performance. BLCUT should be set to zero while running ControlTask 6.

Control Task 0xD (13): Find DC offset (Pixie-4 Express and Pixie-500 Express only)
This task calibrates the offset DAC. The complete function is implemented in the DSP. Normally called as a subroutine to task 0x85. On exit, the DACs are set to the value matching the request from BASELINEPERCENT.

Since the offset measurement has to take the preamplifier offset into account, this measurement must be made with the preamplifier connected to the input of the Pixie module.

The TRACKDAC values found by the DSP are reported in HOSTODATA. The API reads these values and applies them back to the module input RAM.

ControlTask 0x16 (22): Test EM write (Pixie-4 only)

This routine is used to write a test pattern from the DSP into the external memory (testing list mode data transfers). The data written is as follows:

Word (16bit)	Value	Notes
0	8002	Works as buffer length
1	MODNUM	Can be used to identify a module by writing MODNUM through CAMAC and reading the EM through USB.
2	0xAAAA	
3	0x5555	
4	0xCCCC	
5	0x3333	
6	0x1111	
7	0xEEEE	
8	0x8888	
9	0x7777	
10-8001	Repeat above words 2-9 for 999 times	
8002-8104	103, MODNUM, 25x (0x8888, 0x8888, 0x7777,0x7777), 0x8888 (testing odd sized buffer transfers)	
8105-8207	103, MODNUM, 25x (0xCCCC, 0xCCCC, 0x3333,0x3333), 0xCCCC (testing odd sized buffer transfers)	

Control Task 0x1A (26): Test histogramming

This routine is used to write a test pattern to the external memory by incrementing bin N for N times, for bins 0..4Ki. The result is a "spectrum" in channel 0 that forms a line with Ncounts = bin number for bins 0..4Ki. This procedure will take several seconds to complete.

Control Task 0x20 (32): Ramp offset DAC

This task is used for calibrating the offset DAC. For each channel the offset DAC is incremented in 2048 equal-size steps. At each DAC setting the DC-offset is determined and written into the list mode buffer. At the end of the task the list mode buffer holds the following data. Its 8192 words are divided up equally amongst the four channels. Data for channel 0 occupy the lowest 2048 words, followed by data for channel 1, etc. The first entry for each channel's data block is for a DAC value of 0, the last entry is for a DAC value of 65504. In between entries the DAC value is incremented in steps of 32. On exit, the task restores the offset DAC values to the values they had on entry.

Control Task 0x27 (39): Untriggered Traces (single channel)

This task returns 8192 ADC samples for the channel specified in CHANNUM. The results are written to the 8192 words long I/O buffer. The XWAIT variable determines the time between successive ADC samples. In the Pixie Viewer XWAIT can be adjusted through the dT variable in the Oscilloscope panel. Use this function to check if the offset adjustment was successful.

Control Task 0x28 (40): Find BLCUT (in DSP)

This task measures incoming baselines and determines their distribution around DC. The distribution width is reported in HOSTODATA0-3 for channel 0-3. This value can be used as the BLCUT parameter to remove "bad baselines" from the running baseline average. A value of 65535 is reported for failure to determine a valid distribution. Since the baseline measurement has to take the preamplifier noise into account, this measurement must be made with the preamplifier connected to the input of the Pixie module.

Control Task 0x2A (42): Collect Complete Baselines
(Pixie-4 Express and Pixie-500 Express only)

This routine is used to collect baseline values. Currently, the DSP collects 13 words for each baseline:

B0L, B0H, B1L, B1H, BGL, BGH,	(raw baseline sums)
BL_DSP_L, BL_DSP_H	(baselines computed by DSP)
DC_DSP	(DC offset computed by DSP)
BL_DSP_AVGL, BL_DSP_AVGH	(baselines averages computed by DSP)
time stamp, ADC value	

630 baselines are collected until the 8192-word I/O buffer is almost completely filled. The host computer can then read the I/O buffer and calculate the baseline according to the formula:

Baseline = (B1-B0* exp(-XP)) * Bnorm

with

B1 = (B1L+B1H*65536)
B0 = (B0L+B0H*65536)
XP = (SLOWLENGTH+SLOWGAP) * 2^{FR} / (CLK*TAU)
Bnorm = $2^{-(FR)}$/ SLOWLENGTH for FR <6 (P4e, P500e)
= $2^{-(FR-2)}$/ SLOWLENGTH for FR >=6 (P4e, P500e)
TAU = PREAMPTAUA+ PREAMPTAUB/65536
CLK = 125 for Pixie-4 Express and Pixie-500 Express
FR = FILTERRANGE

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 4 times the standard deviation. Baseline values can also be plotted against time stamp or ADC value to explore the detector performance.

Other Control Tasks <=**127:** reserved for DSP tasks

Control Task 0x80: Measure baselines and compute BLCUT for all modules

This routine is used to compute the BLCUT value for all modules. The C library will call ControlTask 6 or 40 in the DSP for each channel and module, compute baselines and set BLCUT to 4x standard deviation.

Control Task 0x81: Find values of τ for all modules

This routine is used to compute the decay time for all channels and modules. The C library will run a tau finder routine on every channel and write the new value to the DSP variables, then report success or failure. The function Pixie4_User_Par_IO should then be called with argument "ALL_CHANNEL_PARAMETERS" to update user interface parameters from the DSP variables.

Control Task 0x83: Adjust offset for all modules (C)

This routine is used to adjust DC offsets to the target value BASELINEPERCENT for all channels and modules. The C library will call a task in the DSP for each channel and module to measure ADC as a function of DC offset, compute the slope and set the offset DACs to the appropriate value.
For Pixie-4 express and Pixie-500 Express, this task is redirected to task 0x85.

Control Task 0x84: Read untriggered ADC traces

This routine is used to read untriggered ADC traces for all channels of a module. The C library will the appropriate task in the DSP, if necessary for each channel, and return the result in the User_data block.

Control Task 0x85: Adjust offset for all modules (DSP)
This routine is used to adjust DC offsets to the target value BASELINEPERCENT for all channels and modules. The C library will call a task in the DSP for each channel and module that finds the target DC offset. This is much faster for Pixie-4 Express and Pixie-500 Express than task 0x83(Pixie-4 Express and Pixie-500 Express only)

Other Control Tasks >0x80 (128): reserved for tasks performed by C library, not DSP

Example code calling a DSP Control Task:

```
Function CallControlTask(Task)
Variable Task

//Variable list
ChosenModule // selected module

runtype=Task
make/u/i/n=8192 buffer   // generate array for output
make/u/i/n=1 dummy       // generate dummy for polling array

// call control task
pixie4_Acquire_Data(Task, buffer, "",ChosenModule)

// wait for completion by polling CSR
do
  Sleep/t 5
while (pixie4_Acquire_Data(0x4000, dummy, "",ChosenModule))

  // read DSP memory for result
make/u/w/n=16384 MemoryValues    //  generate array for DSP memory
Pixie4_Buffer_IO(MemoryValues, 1, 1,"", ChosenModule)

idx = ref("AOUTBUFFER")  // find the location of AOUTBUFFER in DSP
memory
            //  this  gives  the  address  where  the  DSP's  IO
        //  buffer starts
if(ModuleType == "Pixie-4")
  Aoutbuf = MemoryValues[idx]- 0x4000
else
  Aoutbuf = MemoryValues[idx]
endif
```

```
// copy I/O buffer into local array.
buffer[p] = MemoryValues[p+Aoutbuf]  // implied loop of p from 0 to
8Ki

End
```

5 Appendix A — User supplied DSP code (Pixie-4)

5.1 Introduction

It is possible for users to enhance the capabilities of the Pixie-4 by adding their own DSP code. XIA provides an interface on the DSP level and has built support for this into the Pixie Viewer. The following sections describe the interfaces and support features.

5.2 The development environment

For the DSP code development, XIA uses and recommends version 5 or 6 of the assembler and linker distributed by Analog Devices. Both versions are in use at XIA and work fine.

It may be inconvenient, but is unavoidable to program the ADSP-2185 on board processor in assembler rather than in a higher level programming language like C. We found that code generated by the C-compiler is bloated and consequently runs very slow. As the main piece of the code could not be written in C at all, we did not burden our design by trying to be compatible with the C-compiler. Hence, using the C-compiler is currently not an option.

With the general software distribution we provide working executables and support files. To support user DSP programming we provide files containing pre-assembled forms of XIA's DSP code, together with a source code file that has templates for the user functions. The user templates have to be converted by the assembler and the whole project is brought together by the linker. XIA provides a link and a make file to assist the process.

In the Pixie Viewer we provide diagnostic tools to aid code developing and a data interface to exchange data between the host and the user code. The Pixie Viewer can, at any time, examine the complete memory content of the DSP and call any variable from any code section by name.

5.3 Interfacing user code to XIA's DSP code

When the DSP is booted it launches a general initialization routine to reach a known, and useful, state. As part of this process a routine called UserBegin is executed. It is used to communicate addresses and lengths of buffers, local to the user code, to the host. The host finds this information in the USERIN[16] buffer described in the main section of this document. The calling of **UserBegin** is not maskable. All other functions that are part of the user interface will be called only if bit 0 of MODCSRB is set at the time.

When a run starts, the DSP executes a run start initialization during which it will call **UserRunInit**. It may be used to prepare data for the event procesing routines.

When events are processed by the DSP code it may call user code in two different instances. Events are processed one channel at the time. For each channel with data, **UserChannel** is called at the end of the processing, but before the energy is histogrammed. UserChannel

has access to the energy, the acquired wave form (the trace) and is permitted one return value. This is the routine in which custom pulse shape analysis will be performed.

After the entire event, consisting of data from one to four channels, has been processed the function **UserEvent** may be called. It may be used in applications in which data have to be correlated across channels.

At the end of a run the closing routine may call **UserRunFinish**, typically for updating statistics and similar run end tasks.

The above mentioned routines are described below, including the interface variables and the permissible use of resources.

5.4 The interface

The interface consists of five routines and a number of global variables. Data exchange with the host computer is achieved via two data arrays that are part of the I/O parameter blocks visible to the host. The total amount of memory available to the user comprises 2048 instructions and 1100 data words.

5.4.1 Interface DSP routines

UserBegin:

This routine is called after rebooting the DSP. Its purpose is to establish values for variables that need to be known before the first run may start. Address pointers to data buffers established by the user are an example. The host will need know where to write essential data to before starting a run.

Since the DSP program comes up in a default state after rebooting UserBegin will always be called. This is different for the routines listed below, which will only be called if for at least one channel bit 0 of ChannelCSRB has been set.

UserRunInit:

This function is called at each run start, for new runs as well as for resumed runs. The purpose is to precompute often needed variables and pointers here and make them available to the routines that are being called on an event-by-event basis. The variables in question would be those that depend on settings that may change in between runs.

UserChannel:

This function is called for every event and every channel for which data are reported and for which bit 0 of the channel CSR_B (ChannelCSRB variable) has been set. It is called after all regular event processing for this channel has finished, but before the energy has been histogrammed.

UserEvent:

This function is called after all event processing for this particular event has finished. It may be used as an event finish routine, or for purposes where the event as a whole is to be examined.

UserRunFinish:

This routine is called after the run has ended, but before the host computer is notified of that fact. Its purpose is to update run summary information.

5.4.2 Global variables:

UserIn[16]	16 words of input data, also visible to host
UserOut[16]	16 words of output data, also visible to host
Uglobals[32]	32 words to pass global variables from the user code to the main code. The use of these variables is controlled by the main code
UretVal[6]	User output data to be incorporated into list mode data

The return value for UserChannel for list mode data is UretVal. It is an array of 6 words. If bit 1 of ChanCSRB is 0, only the first word is incorporated into the output data stream by the main code. (See Tables 4.2 to 4.6 in the user manual for the output data structure.) If the bit is 1, up to six values are incorporated, overwriting the XIA PSA value, the USER PSA value, the GSLT time, and the reserved word in the channel header. If the run type compresses the standard nine channel header words, the number of user return values is reduced accordingly (i.e only 2 words are available in RunTask 0x102, and no words in RunTask 0x103).

When entering UserChannel a number of global variables have been set by the DSP. These are listed in the file "INTRFACE.INC" as "externals:

5.4.3 Register usage:

For register usage restrictions, see the file "INTRFACE.INC".

6 Appendix B — User supplied DSP code (Pixie-4 Express and Pixie-500 Express)

6.1 Introduction

It is possible for users to enhance the capabilities of the Pixie-4 Express or Pixie-500 Express by adding their own DSP code. XIA provides an interface on the DSP level and has built support for this into the Pixie Viewer. The following sections describe the interfaces and support features.

6.2 The development environment

For the DSP code development, XIA uses and recommends VisualDSP++ version 5.0 distributed by Analog Devices. The ADSP 21369 is programmed in assembler, and all user functions are contained in file user.asm. It can be edited with any text editor, and is then compiled with precompiled XIA code into the final binary, "P500e.ldr. XIA provides a make file to assist the process.

In the Pixie Viewer we provide diagnostic tools to aid code developing and a data interface to exchange data between the host and the user code. The Pixie Viewer can, at any time, examine the complete memory content of the DSP and call any variable from any code section by name.

6.3 Interfacing user code to XIA's DSP code

When the DSP is booted it launches a general initialization routine to reach a known, and useful, state. As part of this process a routine called **UserBegin** is executed. It is used to communicate addresses and lengths of buffers, local to the user code, to the host. No parameters from settings files are available at this time, so any setup performed here must be hardcoded.

When a run starts, the DSP executes a run start initialization during which it will call **UserRunInit**. It may be used to prepare data for the event processing routines. The USERIN[16] parameters (described in the main section of this document) from the settings file are defined at this time to initialize processing parameters.

When events are processed by the DSP code it may call user code in two different instances. Events are processed one channel at the time. For each channel with data, **UserLMZChannel** or **UserMCAZChannel** is called at the end of the processing, but before the energy is histogrammed. User?ZChannel has access to the energy, time stamp (psa values), and other event information and is permitted up to 16 return values.

After the entire event has been processed, consisting of data from one to four channels, the function **UserLMZEvent** or **UserMCAZEvent** is called. It may be used in applications in which data have to be correlated across channels.

At the end of a run the closing routine may call **UserRunFinish**, typically for updating statistics and similar run end tasks.

In the Pixie-4, the user routines were only called if bit 0 of MODCSRB and/or ChanCSRB# was set at the time. In the Pixie-4 and Pixie-500 Express, these routines are called always, but MODCSRB and ChanCSRB# are avaialble to implement an immediate return if desired.

The above mentioned routines are described below, including the interface variables and the permissible use of resources.

6.4 Interfacing user variables to XIA's DSP variables

Data exchange with the host computer is achieved in the following ways:

Two data arrays that are part of the I/O settings (and saved to the .set file) are reserved for user code. While internally defined as 32 bit variables, only the lower 16 are visible to the host and saved to file. These arrays are
UserIn[16]
UserOur[16]

Most input settings are visible to the user code and declared as external variables in the user.asm file. These parameters should be treated as read only.
During event processing, the XIA code sets a number of global variables. These are not part of the I/O settings, and only visible to the host with the debug functions described below. These variables should be treated as read only, except as specifically described otherwise. Variables include

energy – computed pulse height (R/W)
timeH, timeL – time stamp

and many more, see header in the user.asm file for a list with explanations.
In addition, a number of DSP computational registers (Ri) may be set to specific values to quickly access key values.

Additional output values from the user event processing can be stored in a third user array, 32bit wide variables. Currently the first 3 are included in the list mode data (channel header).
UretVal[16]

6.5 Coding Restrictions

For pointer and register usage restrictions, see the file "user.asm".

6.6 Debugging tools

Besides debugging tools offered by the VisualDSP++ enviroment to simulate code (generally not used by XIA), the tools listed below are available. The description assumes

operation with the Pixie Viewer, but equivalent functions can be implemented with the C library's API function in a custom interface as well. The Pixie-4 Express and Pixie-500 Express do not allow VisualDSP++ real time access via the debug port of the device. Instead, it is recommended to store partial and/or complete results in some of the output parameters (UserOut or UretVal) and compare them with offline computation. Code branches can be indicated by setting debug variables to certain values. UretVal values are specific for each processed event, UserOut for an overall acquisition. In addition, all local variables defined in the user code can be read by a memory readout (lower 16 bit only).

Debug functions are

The "User Panel" has controls for key DSP input parameters (ModCSRB, UserIn, etc) and display fields for several output parameters (UserOut). More controls can be added by modifying the panel. Output values are updated after the run.

The List Mode Trace panel shows the values of XIA_PSA and User_PSA in the event data table. These two values are the lower and upper 16 bit of Uretval[0].

The "Expert Panel" (top menu XIA → Expert Panel) has a button to show "Memory" values. This reads 16Ki values from DSP memory. The first 512 are the I/O settings. Other locations contain the local and global variables defined in the code, including the 8Ki I/O buffer. For locations, see the .map file or search the table of variable names. The memory can be read while an acquisition is in progress.

7 Appendix C — User supplied Igor code

Starting in version 1.38, the Pixie Viewer contains a number of user procedures that are called at certain points in the operation. These user procedures are contained in a separate Igor procedure file "user.ipf" that is automatically loaded when opening the Pixie Viewer (Pixie.pxp). By default, the user procedures do nothing, but they can be edited to perform custom functions. It is recommended that the modified procedures be "saved as" a new procedure file user_XXX.ipf and the generic user.ipf be removed ("killed") from the main .pxp file.

7.1 Igor User Procedures

The Igor user procedures called from the current version of the main code are listed below.

Function User_Globals()

This function is called from InitGlobals. It can be used to define and create global specific for the user procedures.

By default it creates a user variable "UserVariant" which can be used to track and identify different user procedure code variants. Variant numbers 0x0 - 0x7FFF are reserved for user code written by XIA.

Function User_StartRun()

This function is called at end of Pixie_StartRun (which is executed at beginning of a data acquisition run) for runs with polling time>0. It can be used to set up customized runs, i.e. initialize parameters etc.

Function User_NewFileDuringRun(Runtype)

When Igor is set to store output data in new files every N spills or seconds, this function is called at the end of making the new files, before the run has resumed. It can be used to e.g. change acquisition parameters or save the Igor experiment during these multi-file runs. However, it will interfere with the polling routine, so the time to execute User_NewFileDuringRun should be less than the polling time.

The argument supplies the run type

Function User_UpdateMCA()

This function is called when the "update" button is clicked in MCA spectra

Function User_StopRun()

This function is called at the end of the run. By default it calls another function to duplicate the output data displayed in the standard Igor graphs and panels into a data folder called "root:results". It can be used to process output data

Function User_ChangeChannelModule()

This function is called when changing Module Number or Channel Number. By default it calls a function to update the variables in the User Control panel.

Function User_ReadEvent()

This function is called when changing event number in list mode trace display or digital filter display. By default it duplicates traces and list mode data into the "results" data folder

Function User_TraceDataFile()

This function is called when changing the file name in list mode trace display.

Function User_HomePaths()

This function is called when executing the "UseHomePaths macro. It can be used to specify custom file names.

7.2 Igor User Panels

The Igor user panels defined in the current version of the user code are listed below:

Window User_Control()

this is the main user control panel, listing DSP input and output variables and showing several action buttons. This panel can be modified to set user variables and control user procedures.

Window User_Version(ctrlName)

This panels displays the version and variants of the user code:

UserVersion	// the version of the user function calls defined by XIA
UserVariant	// the variant of the code written by the user
USEROUT[0]	// the version of the DSP code written by the user

7.3 Igor User Variables

The main Igor code defines the global variables and waves below for use in user procedures. The user code can modify these values without interfering with the main code. (An exception is the "UserVersion", which should not be modified, but used to ensure the user code is compatible with the main code.

```
NewDataFolder/o root:results      //the Igor data folder where results for user are stored

Variable/G root:results:RunTime                 // Run Time from run statistics panel
Variable/G root:results:EventRate               // Event rate from run statistics panel
Variable/G root:results:NumEvents               // Total number of events
Wave root:results:ChannelCountTime              // Channel count time 0..3
Wave root:results:ChannelInputCountRate // Channel input count rate 0..3
String/G root:results:StartTime                 // Start time from run statistics panel
String/G root:results:StopTime                  // Stop time run statistics panel

Wave root:results:MCAch0                        // Channel 0 histogram
Wave root:results:MCAch1                        // Channel 1 histogram
Wave root:results:MCAch2                        // Channel 2 histogram
Wave root:results:MCAch3                        // Channel 3 histogram
Wave root:results:MCAsum                        // Sum histogram

Wave root:results:trace0                        // channel 0 list mode trace
Wave root:results:trace1                        // channel 1 list mode trace
Wave root:results:trace2                        // channel 2 list mode trace
Wave root:results:trace3                        // channel 3 list mode trace
Wave root:results:eventposlen                   // contains trace location (in list mode file)
Wave root:results:eventwave                     // contains data for selected list mode event

NewDataFolder/o root:user          //create the folder for variables defined by user
Variable/G root:user:UserVersion   // the version of the user function calls defined by XIA
Variable/G root:user:UserVariant   // the variant of the code written by the user
```

The contents of the wave root:results:eventwave is as listed below. This is the UserData array for Runtask 0x7008 in the PixieViewer.

Position	Content
0	event location in file
1	Sampling rate in MHz for the trace data below
2	length of event in file
3..6	tracelength for channel 0..3
7..12	buffer header (see user's manual)
13..15	event header (see user's manual)
16..24	channel header for channel 0 (see user's manual)
25..33	channel header for channel 1
34..42	channel header for channel 2
43..51	channel header for channel 3
52+	trace data for channel 0,1,2,3 (use above tracelength to extract)

