



# Pixie-Net XL

## User Manual

**Version 3.3E**

January 19, 2024

Hardware Revisions: B, C

Software Revision: 3.3E

### Supported Variants and Firmware Revision:

Part Number	ID	Description	FW Rev.
Pixie-Net-8-14-250	8	8 channel, 14 bit, 250 MSPS, 10G	0x3A42, 0x3E41
Pixie-Net-8-14-500	7	8 channel, 14 bit, 500 MSPS, 10G	0x3871
Pixie-Net-16-14-250W	4W	16 channel, 14 bit, 250 MSPS, 1G, White Rabbit	0x3140
Pixie-Net-16-14-250T	4T	16 channel, 14 bit, 250 MSPS, 10G, TTCL	0x3E41
Pixie-Net-16-14-250	4	16 channel, 14 bit, 250 MSPS, 10G	0x3E41
Pixie-Net-8-14-125	1	8 channel, 14 bit, 125 MSPS, 10G	0x3111

**XIA LLC**  
2744 East 11th St  
Oakland, CA 94601 USA  
Email: support@xia.com  
<http://www.xia.com/>

Information furnished by XIA LLC is believed to be accurate and reliable. However, no responsibility is assumed by XIA for its use, or for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of XIA. XIA reserves the right to change hardware or software specifications at any time without notice.

## Contents

Safety .....	6
Specific Precautions .....	6
Power Source .....	6
User Adjustments/Disassembly .....	6
Detector and Preamplifier Damage.....	6
Voltage Ratings.....	6
Servicing and Cleaning .....	6
Linux Passwords .....	6
Linux Backup.....	6
Warranty Statement .....	7
Contact Information:.....	7
Manual Conventions .....	8
1   Introduction.....	9
1.1   Pixie-Net XL Features .....	9
1.2   Specifications .....	10
1.2.1   Analog Inputs and Digitization .....	10
1.2.2   Processing Platform .....	14
1.3   System Requirements.....	16
1.3.1   Drivers and Software .....	16
1.3.2   Detector Signals .....	16
1.3.3   Power Requirements .....	17
1.3.4   Connectors and Cabling.....	17
1.4   Software and Firmware Overview .....	17
1.5   Support.....	18
2   Setup .....	19
2.1   Power .....	19
2.2   Serial Port (USB-UART).....	19
2.3   Ethernet Connections .....	20
2.3.1   Controller I/O.....	20
2.3.2   Data Output and White Rabbit.....	20
2.4   SSH login.....	21
2.5   Web Interface.....	21
2.6   SMB (Samba).....	22
2.7   Required Initial Linux Commands.....	22
2.8   Useful Linux Commands .....	22
2.9   Direct Network Connection between Pixie-Net XL and a Windows PC.....	23
3   Pixie-Net XL Operation.....	25
3.1   Adjust Settings .....	25
3.2   Basic Data Acquisition.....	26
3.3   User Interface Options .....	26
3.3.1   Terminal .....	27
3.3.2   Terminal and Webpages.....	27
3.3.3   Terminal, SMB and Windows programs .....	27
3.3.4   Graphical User Interface on Host PC.....	28
3.3.5   Webpage-Only Operation .....	29

3.3.6	Web API.....	30
3.4	Optimizing Parameters.....	32
3.4.1	Energy Filter Parameters.....	32
3.4.2	Threshold and Trigger Filter Parameters .....	32
3.4.3	Decay Time .....	32
3.4.4	Baselines and ADC calibration .....	33
4	API functions .....	34
4.1	progfippi.....	34
4.2	gettraces, cgitraces, cgiprinttraces .....	34
4.3	findsettings.....	35
4.4	runstats, cgistats .....	35
4.5	startdaq.....	36
4.6	mcadaq .....	37
4.7	pollcsr.....	37
4.8	bootfpga .....	38
4.9	cgireadsettings.cgi.....	38
4.10	cgiwritesettings.cgi .....	39
4.11	adcinit.....	40
4.12	udpena[.cgi] .....	41
4.13	udppdis[.cgi] .....	41
5	Web Pages.....	42
5.1	index.html .....	42
5.2	adcpage.html, cgitraces.cgi .....	44
5.3	mcapage.html .....	45
5.4	rspage.html, cgistats.cgi.....	46
5.5	Web Operations (webops/webopsindex.html) .....	48
5.6	Web Operations Setup (webops/adcsetuppage.html).....	49
5.7	Web Operations DAQ Control (webops/daqpage.html) .....	51
6	Parameters in the Settings Files .....	52
6.1	System Parameters .....	53
6.2	Module Parameters .....	58
6.3	Channel Parameters .....	61
7	Data Formats.....	70
7.1	List Mode Data Files.....	70
7.1.1	Pixie-16 style binary list mode files (Run Type 0x100, 0x105) .....	71
7.1.2	Modified Pixie-16 style binary list mode files (Run Type 0x110, 0x111).....	76
7.1.3	Pixie-4e style binary list mode files (Run Type 0x400).....	77
7.1.4	PSA text data without waveforms (Run Type 0x401) .....	79
7.1.5	Modified Pixie-4e style binary list mode files (Run Type 0x410).....	80
7.1.6	Gammasphere compatible binary list mode files (Run Type 0x411).....	82
7.1.7	Legacy Run Types .....	83
7.2	Run Statistics Files.....	89
7.2.1	RS.csv .....	89
7.2.2	RATES.csv.....	90
7.3	MCA Files.....	90
8	Controller (MZ) Registers Visible to Linux .....	91

8.1	I/O Registers .....	91
9	Pulse Processing (K7) Register Map .....	94
9.1	Input/Output Registers .....	95
9.2	Output Only Registers.....	100
10	Linux Configuration .....	103
10.1	Licensing Information.....	103
10.2	Software Information.....	103
10.2.1	Ubuntu 18.....	104
10.2.2	Systemd startup routine to configure FPGAs.....	104
10.2.3	Other configurations .....	105
10.3	Other Settings.....	105
10.3.1	Static IP address.....	105
11	Theory of Operation.....	106
11.1	Digital Filters for $\gamma$ -ray Detectors .....	106
11.2	Trapezoidal Filtering in a Pixie Module .....	108
11.3	Baselines and Preamplifier Decay Times .....	109
11.4	Thresholds and Pile-up Inspection .....	110
11.5	Filter Range.....	112
11.6	Data Flow.....	112
11.6.1	Data Capture Process .....	112
11.6.2	Data Flow.....	113
11.7	Dead Time and Run Statistics .....	114
11.7.1	Definitions.....	114
11.7.2	Count time and dead time counters.....	117
11.7.3	Count Rates.....	119
11.7.4	Dead time correction in the Pixie-Net XL .....	120
11.8	Trigger Distribution .....	121
11.9	Gating.....	122
11.9.1	Gating the Fast Trigger .....	122
11.9.2	Gating the Event Capture .....	123
11.9.3	Gating Notes .....	123
12	Hardware and Firmware Special Functions .....	125
12.1	White Rabbit Time Synchronization.....	125
12.1.1	Setup .....	125
12.1.2	Data Acquisition .....	126
12.1.3	Output Data.....	126
12.1.4	Ethernet Data Output .....	127
12.2	Pulse Shape Analysis .....	128
12.2.1	Overview.....	128
12.2.2	PSA Input Parameters .....	129
12.2.3	PSA Return Values .....	129
12.2.4	Reduced General Purpose Functionality.....	129
12.3	Constant Fraction Timing (CFD) .....	130
12.4	Software Triggering (discontinued) .....	133
12.4.1	Pixie-Net XL Firmware .....	133
12.4.2	Decision Maker Software.....	134
12.5	TTCL Compatible Clock and Trigger Distribution .....	134

12.6	QDC .....	134
13	Hardware Information.....	136
13.1	Access to the PCB hardware.....	136
13.2	Jumpers .....	136
13.2.1	Revision A Modules.....	136
13.2.2	Revision B and C Modules .....	137
13.3	EEPROMs.....	139
13.4	LEDs .....	139
13.5	Dimensions .....	140
13.6	Clocks .....	140
13.6.1	Clock Inputs and Outputs.....	140
13.6.2	Pulse Processing Clock Options .....	141
13.7	Switching Between Board Configurations.....	142
14	Igor Pro GUI .....	143
14.1	Introduction and Setup.....	143
14.2	Operation via web API (Recommended).....	144
14.3	Operation via Serial Port.....	147
14.4	Results.....	147
14.4.1	Oscilloscope.....	148
14.4.2	MCA Spectrum .....	148
14.4.3	Run Statistics Table .....	149
14.4.4	List Mode Traces .....	150
14.4.5	List Mode Data Analysis (under development) .....	150
14.5	Web API coding example .....	151
15	UDP Receiver Program .....	152
15.1	UDP Setup .....	152
15.2	UDP performance .....	153

# Safety

Please take a moment to review these safety precautions. They are provided both for your protection and to prevent damage to the Pixie module and connected equipment. This safety information applies to all operators and service personnel.

---

## Specific Precautions

### **Power Source**

The Pixie-Net XL module is powered through an AC/DC wall adapter. The default adapter has a variety of AC plug attachments for different localities. Please remember to shut down the Linux OS before removing the power plug

Note that “plug-in” modules require additional power, notably any SFP and PMOD modules. **SFP modules above 1W are not supported.**

### **User Adjustments/Disassembly**

To avoid personal injury, and/or damage, always disconnect power before accessing the Pixie module’s interior. Jumper and switches for user configuration should only be changed when power is off. .

### **Detector and Preamplifier Damage**

Please review all instructions and safety precautions provided with these components before powering a connected system.

The Pixie module may be able to provide analog preamp power in a custom configuration. Please contact XIA for details.

### **Voltage Ratings**

Signals on the analog inputs (gold SMB connectors) must not exceed  $\pm 3.5V$ . Exceptions apply for certain attenuation and termination settings, see Appendix.

Signals on the digital inputs (gold MMCX connector and HDMI GPIO connector) must not exceed 3.3V. Please review the pinout in the appendix before making any connections.

### **Servicing and Cleaning**

To avoid personal injury, and/or damage to the Pixie module or connected equipment, do not attempt to repair or clean the inside of these units.

### **Linux Passwords**

The Pixie-Net XL Linux OS comes with default user IDs and passwords for 1) SSH login, 2) SMB file sharing, and 3) Web Operations as described below. Users should immediately change these passwords, especially when the Pixie-Net XL is connected to external networks. Don’t let hackers take over your Pixie-Net XL!

### **Linux Backup**

The Pixie-Net XL Linux OS is stored on a removable SD card. SD cards’ file systems can become corrupted, which would crash the Linux system and make the Pixie-Net XL unable to operate. Therefore periodic backup of the SD card is recommended, for example using Win32DiskImager. (Byte for byte copy is required).

Note that all Linux passwords are stored on the SD card

# Warranty Statement

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

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

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

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

---

## Contact Information:

XIA LLC  
2744 East 11th St,  
Oakland, CA 94601 USA

Downloads: <https://github.com/xia-whennig/PNXL-Releases>  
Support: [support@xia.com](mailto:support@xia.com)

# Manual Conventions

The following conventions are used throughout this manual

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

# 1 Introduction

The Pixie-Net XL consists of a Zynq System on Module (SoM), a main processing board with 2 FPGAs (PXdesk), and two ADC daughter boards.

The Zynq is a combination of an FPGA (Programmable Logic, PL) with an ARM processor (Processing System, PS). The PS runs a basic Linux OS with gcc, webserver, etc; it has USB and Ethernet peripherals and 1GB of memory. The PL is used by the PS to communicate with the PXdesk FPGAs which process the ADC data.

Each FPGA connects to one ADC daughterboard, a clocking board for Ethernet output, and a SFP port to output list mode data at 1 Gsps (1G) or 10 Gsps (10G)

---

## 1.1 Pixie-Net XL Features

- Designed for
  - high precision  $\gamma$ -ray spectroscopy with HPGe detectors,
  - timing with fast scintillators (NaI, LaBr<sub>3</sub>, etc),
  - pulse shape analysis to extract time, position, and/or particle type in segmented or strip detectors, phoswich detectors, or neutron detectors
  - coincidence acquisition
- Flexible digitization with two replaceable daughtercards, each with either of
  - 14 bit 125 MSPS ADC, 4 channels
  - 14 bit 250 MSPS ADC, 8 channels (single ended or differential input)
  - 14 bit 250 MSPS ADC, 4 channels
  - 16 bit 250 MSPS ADC, 4 channels (under redevelopment)
  - 14 bit 500 MSPS ADC, 4 channels (under redevelopment)
  - Contact XIA for more options
- Programmable gain and input offset, depending on ADC daughtercard.
- Programmable pulse processing parameters include trigger filter length, energy filter length, decay time, threshold, and rejection criteria.
- Triggered synchronous waveform acquisition across channels.
- Simultaneous amplitude measurement, waveform capture, and pulse shape analysis.
- On-board MCA memory
- Configurable digital inputs and outputs
- Embedded Linux environment, acting as a built-in data acquisition PC with SD card drive, USB host, 10/100/1000M Ethernet, webserver, GPIO
- List mode data output via 2 dedicated 10G SFP Ethernet ports (or White Rabbit compatible 1G Ethernet ports)
- Open source DAQ software

## 1.2 Specifications

### 1.2.1 Analog Inputs and Digitization

#### 1.2.1.1 Variant DB01, 8-14-75 (non-standard)

<b>Analog Inputs</b>	
<b>Signal Input (8x)</b>	2x 4 analog SMB inputs. Switch selectable input impedance: 50Ω and 5kΩ. Switch selectable attenuation 1/1 and 1/8
<b>Analog Gain</b>	8 gain settings: 1.6 to 22.6
<b>Analog Offset</b>	Input range: After termination and attenuation, up to ±3.0V <sup>1</sup> DC can be added to compensate signal DC offsets. After DC offset compensation, signals in the range from 0V to (2V/analog gain) are accepted by the ADC. (See above for analog gain values)
<b>Digitization</b>	
<b>ADC</b>	2x 4 14bit, 75 MSPS ADC
<b>DB ID</b>	0

#### 1.2.1.2 Variant DB01, 8-14-125

<b>Analog Inputs</b>	
<b>Signal Input (8x)</b>	2x 4 analog SMB inputs. Switch selectable input impedance: 50Ω and 5kΩ. Switch selectable attenuation 1/1 and 1/8
<b>Analog Gain</b>	8 gain settings: 1.6 to 22.6
<b>Analog Offset</b>	Input range: After termination and attenuation, up to ±3.0V DC can be added to compensate signal DC offsets. After DC offset compensation, signals in the range from 0V to (2V/analog gain) are accepted by the ADC. (See above for analog gain values)
<b>Digitization</b>	
<b>ADC</b>	2x 4 14bit, 125 MSPS ADC
<b>DB ID</b>	1

<sup>1</sup> The software parameter for offset DAC universally ranges from -1.25V to +1.25V and is mapped to the full range for each variant internally. For example, on DB01, parameter value 1.25V is actually 3.0V on the input

1.2.1.3 Variant DB02, 16-12-250 (non-standard)

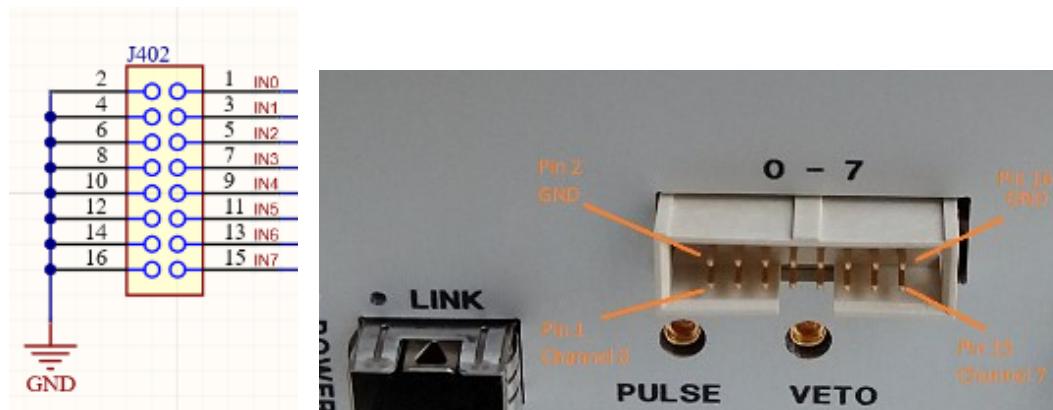
<b>Analog Inputs</b>	
<b>Signal Input (16x)</b>	2x 8 differential analog inputs, 4 per microHDMI connector Input impedance: 1KOhm to GND each side.
<b>Analog Gain</b>	Fixed gain, factory adjustable
<b>Analog Offset</b>	Fixed offset
<b>Digitization</b>	
<b>ADC</b>	2x 8 12bit, 250 MSPS ADC
<b>DB ID</b>	2
<b>Power and Digital I/O</b>	
<b>Control</b>	I2C SDA, SCL, 3 GPIO via HDMI connector 3.3V I/O, 4.7K pullup on ADC daughtercard DB02
<b>Power</b>	+5V filtered analog output via HDMI connector

1.2.1.4 Variant DB02, 16-14-250

<b>Analog Inputs</b>	
<b>Signal Input (16x)</b>	2x 8 differential analog inputs, 0.1" header Input impedance: 1KOhm to GND each side.
<b>Analog Gain</b>	Fixed gain, factory adjustable
<b>Analog Offset</b>	Fixed offset
<b>Digitization</b>	
<b>ADC</b>	2x 8 14bit, 250 MSPS ADC
<b>DB ID</b>	2

## 1.2.1.5 Variant DB04, 16-14-250

<b>Analog Inputs</b>	
<b>Signal Input (16x)</b>	2x 8 analog inputs, 0.1" header Input impedance: 50 Ohm to GND.
<b>Analog Gain</b>	Fixed gain, factory adjustable
<b>Analog Offset</b>	Input range: After termination and attenuation, up to $\pm 2.5V$ DC can be added to compensate signal DC offsets. After DC offset compensation, signals in the range from 0V to (2V/analog gain) are accepted by the ADC. (See above for analog gain values)
<b>Digitization</b>	
<b>ADC</b>	2x 8 14bit, 250 MSPS ADC
<b>DB ID</b>	4



## 1.2.1.6 Variant DB06, 8-16-250 (under redevelopment)

<b>Analog Inputs</b>	
<b>Signal Input (8x)</b>	2x 4 analog SMB inputs. Switch selectable input impedance: $50\Omega$ and $5k\Omega$ . Switch selectable attenuation 1/1 and 1/8
<b>Analog Gain</b>	2 gain settings: 2 and 5

<b>Analog Offset</b>	Input range: After termination and attenuation, up to $\pm 3.0\text{V}$ DC can be added to compensate signal DC offsets. After DC offset compensation, signals in the range from 0V to (2V/analog gain) are accepted by the ADC. (See above for analog gain values)
<b>Digitization</b>	
<b>ADC</b>	2x 4 16bit, 250 MSPS ADC
<b>DB ID</b>	6

1.2.1.7 Variant DB06, 8-14-500 (under redevelopment)

<b>Analog Inputs</b>	
<b>Signal Input (8x)</b>	2x 4 analog SMB inputs. Switch selectable input impedance: $50\Omega$ and $5\text{k}\Omega$ . Switch selectable attenuation 1/1 and 1/8
<b>Analog Gain</b>	2 gain settings: 2 and 5
<b>Analog Offset</b>	Input range: After termination and attenuation, up to $\pm 2.5\text{V}$ DC can be added to compensate signal DC offsets. After DC offset compensation, signals in the range from 0V to (2V/analog gain) are accepted by the ADC. (See above for analog gain values)
<b>Digitization</b>	
<b>ADC</b>	2x 4 14bit, 500 MSPS ADC
<b>DB ID</b>	7

1.2.1.8 Variant DB08, 8-14-250

<b>Analog Inputs</b>	
<b>Signal Input (8x)</b>	2x 4 analog inputs, SMB Switch selectable input impedance: $50\Omega$ and $5\text{k}\Omega$ . Switch selectable attenuation 1/1 and 1/8
<b>Analog Gain</b>	2 gain settings: 2 and 5

<b>Analog Offset</b>	Input range: After termination and attenuation, up to $\pm 2.5$ V DC can be added to compensate signal DC offsets. After DC offset compensation, signals in the range from 0V to $(2V/\text{analog gain})$ are accepted by the ADC. (See above for analog gain values)
<b>Digitization</b>	
<b>ADC</b>	2x 4 14bit, 250 MSPS ADC
<b>DB ID</b>	8

## 1.2.2 Processing Platform

<b>System Options</b>	
<b>Controller Options</b>	<ul style="list-style-type: none"> <li>• MicroZed (“MZ” option)</li> <li>• PicoZed (“PZ” option)</li> <li>• Z-turn (“ZT” option)</li> </ul> Only one controller option is physically installed; with feature differences as noted below.
<b>Ethernet List Mode</b> <b>Data Output Options</b>	<ul style="list-style-type: none"> <li>• 10 G Ethernet for each processing FPGA (“10G” option)</li> <li>• 1 G Ethernet with WR for each processing FPGA (“1G” option)</li> <li>• 10 G Ethernet for each processing FPGA plus 1G WR on PZ controller (“10G+WR” option)</li> </ul> Only one Ethernet option is physically installed; with feature differences as noted below.
<b>Embedded Processing (Zynq Controller)</b>	
<b>Processor</b>	Xilinx Zynq
<b>Data Interfaces</b>	10/100/1000 Ethernet USB 2.0 (host) USB-UART SFP cage with Gbit serial I/O (“PZ” option) 10/100M Ethernet with RJ-45 connector, PTP compatible, (“PZ” option)
<b>Memory</b>	1 GB of DDR3 SDRAM 128 Mb of QSPI Flash 16 GB removable micro SD card (“ZT” and “MZ” options) 8 GB eMMC memory (“PZ” option)
<b>Operating System</b>	Linux (Xillinux – based on Ubuntu 18 LTS) Operates from Linux partition on SD card or eMMC memory
<b>Real Time Clock</b>	inquire
<b>Pulse Processing Features</b>	

<b>Architecture</b>	Two Xilinx Kintex 7 FPGAs with dedicated I/O, each with - SFP cage with Gbit serial I/O - High density PCB connector for ADC daughtercard - micro HDMI connector with 13 GPIO signals (shared with MZ or PZ controller) - 512 MB SDRAM - White Rabbit voltage controlled oscillators and PROMs (“1G” option) - Oscillators for 10G Ethernet (“10G” or “10G+WR” option)
<b>Digital Gain</b>	Arbitrary gain factor for channel matching
<b>Offset</b>	DC offset adjustment from –1.25V to +1.25V, in 65535 steps.
<b>Shaping</b>	Trapezoidal filter with peaking times 0.048 to 63.4 µs Adjustable flat top to eliminate ballistic deficit effects
<b>Trigger</b>	Digital trapezoidal trigger filter with adjustable threshold. Rise time and flat top set independently.
<b>Coincidence</b>	Programmable coincidence window: 40 to 1016 ns Reject unwanted hit patterns of the channels
<b>Data Collection Options</b>	MCA number of bins 1Ki to 32Ki Waveform lengths and pre-trigger delay List mode data format (text, binary, content)

### Data Outputs

<b>Spectrum</b>	1024-32768 bins per channel, 32 bit deep (4.2 billion counts/bin). Additional memory for sum spectrum for clover detectors.
<b>Statistics</b>	Real time, counting time, filter dead time, input and throughput counts.
<b>List Mode Event Data</b>	Pulse height (energy), time stamps, pulse shape analysis results, waveform data (up to 4Ki samples), constant fraction timing, and ancillary data like hit patterns.
<b>List Mode Output Path</b>	- Store on local SD card - Store on remote network drive - UDP output stream

### Front Panel I/O

<b>Pulser Output</b>	1 MMCX coaxial connector “PULSE” Periodic, exponentially decaying reference pulser. Programmable on/off
<b>Veto Input</b>	1 MMCX coaxial connector “VETO” input only default use: Veto signal to suppress event triggering 2.5V logic, 5V compatible
<b>SFP (3x)</b>	2x SFP cage for 1/10G Ethernet, 1G and White Rabbit compatible (“1G” option) 10G (“10G” option) 1x SFP cage for Zynq processor (“PZ” option) 1G and White Rabbit compatible (“1G+WR” option)
<b>Indicators</b>	3 LEDs controlled by processor for system status information 3 LEDS for SFP Ethernet link status

<b>Analog Inputs</b>	See analog specifications
<b>Rear Panel I/O</b>	
<b>Logic Input/Output</b>	General Purpose I/O connected to programmable logic: 2 micro HDMI connectors with 13 GPIO signals ( <u>not video</u> ) single ended or differential can be used as clock input factory set to 5V (default) or 2.5V logic.
<b>Power</b>	12V DC in. AC adapter requirements: 12V, 60W Barrel Plug 2.1mm I.D. x 5.5mm O.D., center positive Usage: typical ~40W
<b>PMOD</b>	2 12-pin 0.1" connectors for 8 I/O signals, 3.3V power, GND Compatible with PMOD I/O modules (I2C, GPIO, serial, wifi, GPS, ...) 1 dedicated for each controller option
<b>Clocks and Trigger</b>	1 MMCX coaxial connector "CLK OUT", programmable clock out (3.3V ) 1 MMCX coaxial connector "CLK IN", external clock input (3.3V ) 1 MMCX coaxial connector "TRIG OUT" programmable trigger (e.g. PPS), 3.3V, output only
<b>Controller I/O</b>	USB, RJ-45 Ethernet, UART, eMMC, SD card depending on controller option

Table 1-1. Specifications for the Pixie-Net XL

## 1.3 System Requirements

The digital spectroscopy system considered here consists of a Pixie-Net XL and a gamma ray detector with appropriate power supplies. A PC, smartphone or tablet is required to communicate with the Pixie-Net XL, but data acquisition can be fully contained in the Pixie-Net XL itself.

### 1.3.1 Drivers and Software

The Pixie-Net XL operates with an embedded Linux system that includes all software and drivers to communicate with external devices via Ethernet or USB. It can

- Make DAQ results available via webserver
- Read and write USB drives for data exchanges
- Share files over a Windows network

For higher convenience of communication and data analysis, we can provide the Pixie Viewer, based on Wavemetrics' Igor Pro. (Igor Pro version 8 or higher is required).

### 1.3.2 Detector Signals

The Pixie-Net XL is designed for fast rising, exponentially decaying signals. Step pulses and short non-exponential pulses can be accommodated with specific parameter settings. Staircase type signals from reset preamplifiers generally need to be AC coupled.

Detector signals must not exceed  $\pm 3.5V$ .

### 1.3.3 Power Requirements

The Pixie-Net XL consumes roughly 40 W, requiring the following currents from the AC adapter:

12V      up to 5 A

### 1.3.4 Connectors and Cabling

The Pixie-Net XL uses SMB connectors for the analog inputs from the detectors. SMB to BNC adapter cables are provided with the module. Some variants may use HDMI or other high density cabling that requires matching detector output connectors or adapters.

MMCX connectors are used for power, analog outputs, and digital input/outputs. MMCX to BNC adapter cables are provided with the module.

MicroHDMI connectors are used for additional digital inputs and outputs or for analog differential inputs, but not video. HDMI cables are widely available and not provided with the module.

SFP cages are intended for optical Ethernet connections, but can be used for copper Ethernet or other I/O with appropriate SFP modules and/or firmware changes. SFP modules and cables are widely available and not provided with the module by default.

RJ-45 connectors are used for standard 10/100/1000M or 10/100M copper Ethernet. Matching CAT-5 cables are widely available and not provided with the module.

---

## 1.4 Software and Firmware Overview

The data acquisition (DAQ) software of the Pixie-Net XL consists of a small set of C programs (API functions) executed on the ARM section of the Zynq controller. The controller configures, starts and monitors the data acquisition that is mainly executed in the FPGA fabric of the two Kintex FPGAs. The controller can read results and store them on the SD card (or network drives), but the main data path is from the ADC through FPGA pulse processing to SFP Ethernet output directly from each FPGA.

The API functions are called from the Linux command line or as CGI scripts from a web page or other program. DAQ results can be viewed or downloaded via the web page, or copied over the network. Acquisition parameters are stored in an .ini file that can be edited by the user to adjust parameter settings.

DAQ results (list mode data) can also be streamed out as UDP packages via a dedicated Ethernet connection. The receiving device must run a program to capture UDP packages and save them to file. XIA provides a basic receiver program for Linux and Windows as a coding example.

Firmware code for the on-board pulse processing functions is loaded to the fabrics as part of the powerup boot sequence.

Users may modify the API functions (source code and gcc compiler is included in the Linux environment on the SD card). Applications can be executed from the SD card or a mounted USB drive.

More sophisticated graphical user interfaces can be developed, providing control buttons and data analysis functions. One such interface is based on Igor Pro (see below); contact XIA for details and demo code for other implementations.

---

## 1.5 Support

A unique benefit of dealing with a small company like XIA is that the technical support for our sophisticated instruments is often provided by the same people who designed them. Our customers are thus able to get in-depth technical advice on how to fully utilize our products within the context of their particular applications.

Please read through the following sections before contacting us. Contact information is listed in the first few pages of this manual.

## 2 Setup

When powered up, the Pixie-Net XL automatically boots the Zynq controller's FPGA configuration and starts the Linux OS. The Linux OS automatically configures the pulse processing FPGAs and defines the processing parameters from the settings file<sup>2</sup>. This brings the system up to the last operational state when settings were saved.

There are several ways for a user to connect to the Pixie-Net XL Linux OS on the Zynq controller:

1. Serial port via USB-UART
2. Network SSH terminal
3. Web server
4. SMB (Samba) file sharing

The typical procedure for first time setup would be to first power up the Pixie-Net XL (2.1), then install and configure drivers for the serial port on a USB master PC (2.2). Log on via serial port terminal and find the Pixie-Net XL's IP address. After that, the Pixie-Net XL can be operated through terminal, web interface and/or SMB.

### 2.1 Power

To power up the Pixie-Net XL, simply connect the 12V DC power plug from the AC adapter. The center pin must be positive and the adapter must be rated for 60W or more.

To power down the Pixie-Net XL, first shut down the Linux OS (type `halt`), then remove the 12V DC power plug. Make sure to also remove any UART cable, which can also provide power (via USB), but only for the controller!

### 2.2 Serial Port (USB-UART)

The serial port connection requires a driver to map the UART to a serial port and a terminal running on the USB master PC. Windows installation consists of the following steps:

1. Download and extract/install the Silicon Labs CP210x USB-to-UART driver, [www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers](http://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers), “CP210x VCP Windows”
2. Download and install Tera Term (or other suitable terminal program). See <http://ttssh2.osdn.jp/>
3. Power up the Pixie-Net XL (see 2.1), then connect USB cable between Pixie-Net XL and PC. (PC: any USB port, Pixie-Net XL: port labeled UART 1<sup>3</sup>)
4. From the Silicon Labs installation, run CP210xVCPInstaller\_x64.exe on the PC to create a COM port
  - Find the new COM port's number in the Windows device manager
  - The COM port assignment is device specific, so this has to be repeated every time switching to a different Pixie-Net XL unit. For the same Pixie-Net XL, this is a one-time operation.

<sup>2</sup> These steps can be executed manually by executing setup routines on the Zynq controller, see below.

<sup>3</sup> UART 0 for PZ option

5. Open Tera Term.
  - Connect via the serial port showing the COM number above
  - Select Setup > Serial port. Defaults are ok, except baud rate must be 115200
  - Adjust the font and size if desired
  - Login credentials for serial port login are required in Ubuntu 18. Factory defaults are **root/xia17pxn** (please change).

If the USB master PC is a Linux system, it is possible to use the Minicom utility as for serial port I/O. Usually no installation or drivers are required. It can be configured via

```
sudo minicom -s
```

and by specifying *ttyUSB0* as the port, with *no* HW flow control. See also <https://help.ubuntu.com/community/Minicom>.

## 2.3 Ethernet Connections

### 2.3.1 Controller I/O

The Pixie-Net XL has multiple Ethernet connections. For the initial setup described in the following sections, use the RJ-45 connector “LAN 1”<sup>4</sup> for standard CAT-5 cables near UART 1 on the rear panel. This is the 10/100/1000M Ethernet connection for the embedded Linux controller.

For this connection, MAC address is stored on a local PROM. The IP address is usually assigned automatically by the network (DHCP).

### 2.3.2 Data Output and White Rabbit

The other Ethernet connections are, depending on the Ethernet Data Output Options:

- **SFP 0, 1:**

For “1G” option:	White Rabbit compatible 1G optical Ethernet. Copper SFP cables can be used, but will limit precision
For “10G” option: “10G+WR” or “10G+TTCL”	10G Ethernet connection (copper or optic) without White Rabbit synchronization

Both ports are only used to stream out list mode data as UDP packages (and additionally for WR synchronization in 1G). The receiving device must run software to capture the UDP data and store it to disk. XIA provides a simple UDP receiver code example (see section 15).

For these connection, MAC and IP addresses are stored on a local PROM accessible via the WR UART (1G) or in the settings file (10G, 10G+WR, 10G+TTCL).

<sup>4</sup> LAN 0 for PZ option

- **SFP Z:**

For “10G+WR” option:      White Rabbit compatible 1G optical Ethernet.  
Copper SFP modules can be used, but will lose precision

This port is currently only used for White Rabbit synchronization with the PicoZed Zynq controller, no DAQ data I/O.

For this connection, MAC and IP addresses are stored on a local PROM accessible via the WR UART

## 2.4 SSH login

Tera Term or another suitable program can be used to log in via the network once the Pixie-Net XL is powered, connected to a network, and its IP address is known. The IP address can be found by

- connecting via serial port terminal as described above, and typing `ifconfig`
- executing `arp -a` in a Windows Command prompt or Linux terminal, and look for the MAC address from the label on the Pixie-Net XL (starts with 70:B3 ...)
- executing `ping pixienet` in a Windows Command prompt
- logging on to the network router, see list of connected devices
- just trying the one it had before, IP addresses seem fairly persistent through power cycles

To connect, open a terminal and make connection to the Pixie-Net XL’s IP address. Default ID/PW is **root/xia17pxn** (*please change*).

Two possible free SSH terminal program for Android are “SSH Client” and “JuiceSSH”, allowing login from a tablet or smartphone.

## 2.5 Web Interface

Any web browser can be used to log in via the network once the Pixie-Net XL is powered, connected to a network, and its IP address is known. The IP address can be found by

- connecting via serial port terminal as described above, and typing `ifconfig`
- executing `arp -a` in a Windows Command prompt or Linux terminal, and look for the MAC address from the label on the Pixie-Net XL(starts with 70:B3 ...)
- executing `ping pixienet` in a Windows Command prompt
- logging on to the network router, see list of connected devices
- just try the one it had before, they seem fairly persistent through power cycles

To connect, enter Pixie-Net XL’s IP address as the web address in your browser. Any browser should work; but XIA uses Firefox and does not test for compatibility with other browsers. No password is required for the Pixie-Net XL home page, however, login is required for web operations that duplicate terminal access to change settings and execute tasks. The default ID/PW is **webops/xia17pxn** (*please change*).

For proper operation of some links in the home page, prior execution of a function in the Linux terminal is required. See below for details.

## 2.6 SMB (Samba)

The Pixie-Net XL Linux OS is running Samba, a “Windows interoperability suite of programs for Linux and Unix”. It allows the files in the Linux partition of the Pixie-Net XL SD card to be shared with Windows networking. By default, only one folder is shared (/var/www) which is the location of the API functions.

To connect, type \\192.168.1.xxx\PNvarwww in the Windows Explorer location bar, where 192.168.1.xxx is the IP Address of the Pixie-Net XL found as above and PNvarwww is the name given to /var/www for file sharing purposes in Samba. Windows will prompt for login; the default ID/PW is **root/xia17pxn** (*please change*).

The settings file `settings.ini`, output data files, and all (source and executable) files of the API functions can now easily be copied or edited with Windows tools and programs. However, to *execute* the API functions, serial port or SSH login is required (or in some cases, API functions are executed from the web interface)

## 2.7 Required Initial Linux Commands

Once logged on via the Linux terminal or the web operations page, the following steps **should** be performed:

1. Change directory: `cd /var/www`  
This is the directory visible via the web server. Data created by execution of XIA API functions can be read via the web browser from this directory. For convenience, it contains a “release” of all XIA SW functions and is used as the default working directory. (Not required for web operations page)
2. Optional: Configure pulse processing FPGAs (see also section 4 for more options)  
`./bootfpga`
3. Optional: Apply settings to FPGA:  
`./progfippi`
4. Automatically set a few basic parameters such as DC offsets (and resolve random channel swapping): `./findsettings`  
This should be performed with the detector connected, with the correct polarity and termination setting.

Note: As of SW version 3.21, Steps 2 and 3 are already automatically executed at the startup of the Linux OS. Step 3 still has to be repeated after every change to `settings.ini`

## 2.8 Useful Linux Commands

The following commands may be helpful

1. Type `ifconfig` to find the IP address  
The network does not come up by itself occasionally. This appears to coincide with the webpage being open in a browser during (re)boot. For a change of restart, type `sudo /etc/init.d/networking restart` else power cycle
2. The Zynq temperature is reported as a raw number in `/sys/devices/soc0/amba/f8007100.adc/iio:device0/in_temp0_raw`; the actual temperature can be computed by subtracting 2219 and multiplying

with 0.12304. ADC and Zynq temperatures are also reported in the run statistics and by ./progfippi

3. To change a parameter value in the settings file without opening/editing/closing the file, a sed command as follows can be used:  

```
sed -i '/RUN_TYPE/c RUN_TYPE      0x410' settings.ini
```

 This replaces (-i = in the file) the line containing the string “RUN\_TYPE” with the text “RUN\_TYPE 0x410”.
4. To mount a USB drive (e.g. to copy data or SW updates), type  

```
mount /dev/sda1 /mnt/usb
```

 /var is not a suitable directory to mount the USB stick, as it confuses the web server
5. Type date to verify Linux time automagically updated to UTC
6. Use the mount command to mount external network drives. For example, to mount NASdrive/data from a PC with IP address 192.168.1.123, type  

```
mount //192.168.1.123/data /mnt/data -o "username=[user],password=[passwd]"
```

 with the appropriate values filled in for [user] and [passwd]
5. To mount the SD card boot partition to a folder /mnt/sd, execute  

```
mount /dev/mmcblk0p1 /mnt/sd
```

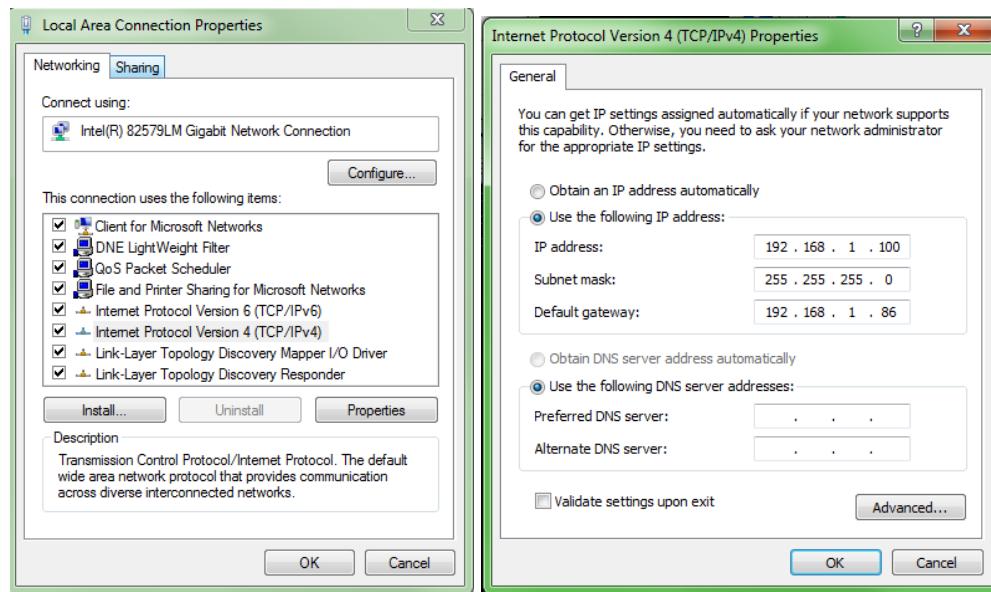
 this is useful to update the boot files without removing the SD card. The Pixie-Net XL has to be rebooted before the new boot files become effective.
6. To clear the command line history, type  

```
history -c
history -w
```

## 2.9 Direct Network Connection between Pixie-Net XL and a Windows PC

The above description of setup and operation of the Pixie-Net XL assumes both Pixie-Net XL and the user device (PC, tablet, smartphone) are connected to a local network with DHCP server and gateway. Instead, it is possible to directly connect the Pixie-Net XL to a laptop or desktop with a standard Ethernet cable (no crossover cable required). However, a number of configurations have to be set manually:

1. On the Pixie-Net XL terminal, type ifconfig eth1 up 192.168.1.86 to bring up the Ethernet connection and assign the IP number. In some cases, eth0 or eth2 has to be used instead of eth1.
2. On the Windows PC (here Windows 7), go to Start > Control Panel > Network and Sharing Center > Change Adapter Settings > Local Area Connection



In the Local Area Connection “Properties” dialog, select “Internet Protocol Version 4 (TCP/IPv4)” and click “Properties”.

In the properties dialog, set IP address, subnet mask, and gateway as shown above. (The gateway is equal to Pixie-Net XL’s IP address.)

Note: The network will not be able to connect to the internet. The date/time of the Pixie-Net XL will likely be initialized to 1970.

## 3 Pixie-Net XL Operation

Basic operation of the Pixie-Net XL uses a combination of terminal commands and web interface. The former controls all data taking and requires login to the system. The latter displays data and is accessible to anyone. This provides a measure of security in instrument control while making it convenient to view data.

We recommend reading through the description of the *basic* methods for adjusting settings and acquiring data (sections 3.1 and 3.2) to get an understanding of the internal operation. The most *convenient* methods of operation are the webpage-only operation (section 3.3.5) and the Igor Pro GUI (section 14).

### 3.1 Adjust Settings

The pulse processing parameters must be adjusted (once) to match the detector characteristics. This includes analog settings such as gain and offset, and pulse processing parameters such as decay time and trigger threshold.

All settings are stored in an .ini file. The default settings file is configured for the Pixie-Net XL internal pulser. This file, *defaults.ini*, contains all the parameter settings as described in section 6. Settings are grouped into system (e.g. requested run time), module (e.g. coincidence pattern for all channels in one FPGA), and channel (e.g. offset). A condensed settings file, *settings.ini*, contains the most important parameters and **will override the defaults**.

The most basic method to change settings is to edit *var/www/settings.ini* and then execute `./progfippi` to apply them to the FPGA. If necessary, lines with additional parameters can be copied from *defaults.ini* into *settings.ini*; also lines can be deleted from *settings.ini* if there is no need to vary specific parameters. The file *defaults.ini* should be considered a read-only file. Editing can be accomplished with a built-in Linux editor through the terminal (for example VI) or by opening the file in a Windows editor through the SMB file sharing.

The most important parameters for initial setup are

REQ_RUNTIME	Requested runtime of the data acquisition
RUN_TYPE	0x301 for MCA only, 0x100, 0x105, 0x400 for list mode runs to SD card (or 1G) 0x110, 0x111, 0x410, 0x411 for list mode runs via 10G
CCSRA_INVERT_05	If 1, invert incoming signal
ANALOG_GAIN	14/75 and 14/125 ADCs: 1.6, 2.4, 3.5, 5.4, 6.7, 9.9, 14.7, 22.6 16/250, 14/500, 14/250 ADCs: 2.4 and 5.4 all 16-channel models: fixed (ignored)
DIG_GAIN	Arbitrary gain factor for energies in MCA spectra and list mode events
VOFFSET	DC offset (ignored for 12/250 ADCs)
TAU	Exponential decay time of the input signal

(See also section 3.4 for optimizing trigger and energy filter settings)

When set up correctly, the signal baseline should be at approximately 10% of the full ADC range. The polarity and gain should be set such that pulses start with a fast rise and decay

back down to baseline, and do not go out of the ADC range (max 4096 or 16K steps for a 12 or 14 bit ADC). The decay time must be measured and specified as the parameter TAU. This is best verified by opening/refreshing the Pixie-Net XL ADC page in the web browser, or executing `./gettraces` and viewing the resulting `ADC.csv` file. You can also open/refresh the Run Statistics page in the web browser or execute `./runstats` and read the output parameters in the resulting `RS.csv` file. The current input count rate, out of range fraction, temperatures, and FPGA system time will update even when no run is in progress. The function `./findsettings` can assist in finding parameters such as DC offset. It also resolves random channel swapping between odd and even channels for the 250 MHz ADC variant (DB02, DB04, and DB08).

---

## 3.2 Basic Data Acquisition

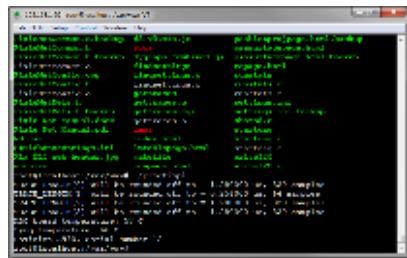
1. In the terminal, type `./startdaq` or `./mcadaq` to start a DAQ run with current settings.  
The screen will be updated with print statements of the runtime
2. In the browser, navigate to the MCA page (“view spectra”) or the Run Statistics page (“view run statistics”) under DAQ Monitoring.  
Refresh these pages with browser button to see updates during DAQ
3. < wait > *Currently the only way to stop is ctrl-c*
4. When the DAQ finishes in the terminal, the final MCA, the run statistics, and the list mode data files have been created. *The filenames are fixed to MCA.csv, RS.csv and LMdata#.bin/b00 for ./startdaq (# = module number, default 0)*
5. In the browser, the data files can be viewed or downloaded (under “DAQ Results”)
6. In the terminal, the data files can be copied to local USB drive or network drive
7. In a Windows Explorer window pointing to <\\<Pixel Net IP>\PNvarwww>, the data files can be copied or opened by Windows tools and programs.
8. For some run types, list mode data is streamed out via the SFP Ethernet connections as UDP packages. These have to be received by a suitable program on the destination device.

---

## 3.3 User Interface Options

There are a variety of interface options to operate the Pixie-Net XL. They range from basic terminal and file I/O to plug-ins for data acquisition or data analysis software. Fundamentally, the Pixie-Net XL communicates through generic interfaces such as serial port and Ethernet, so that any program emulating a serial port and reading files from a network drive can be used to operate the system. A few examples are listed in the following sections

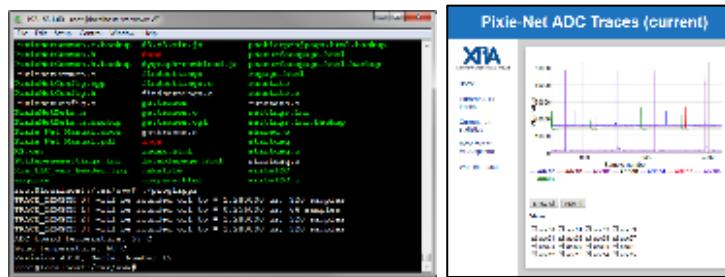
### 3.3.1 Terminal



At the most basic level, users can log in to the Pixie-Net XL with a terminal program and execute the C programs to set up and perform data acquisition (e.g. `./progfippi` and `./startdaq`). Settings are modified by editing the settings file `settings.ini` with a basic text editor like VI. Results are files on the Pixie-Net XL's SD card, which can be viewed with a text editor or copied to mounted network drives.

This environment may appeal most to users familiar with Linux, and also allows modification and recompilation of the C programs with the built-in gcc compiler.

### 3.3.2 Terminal and Webpages



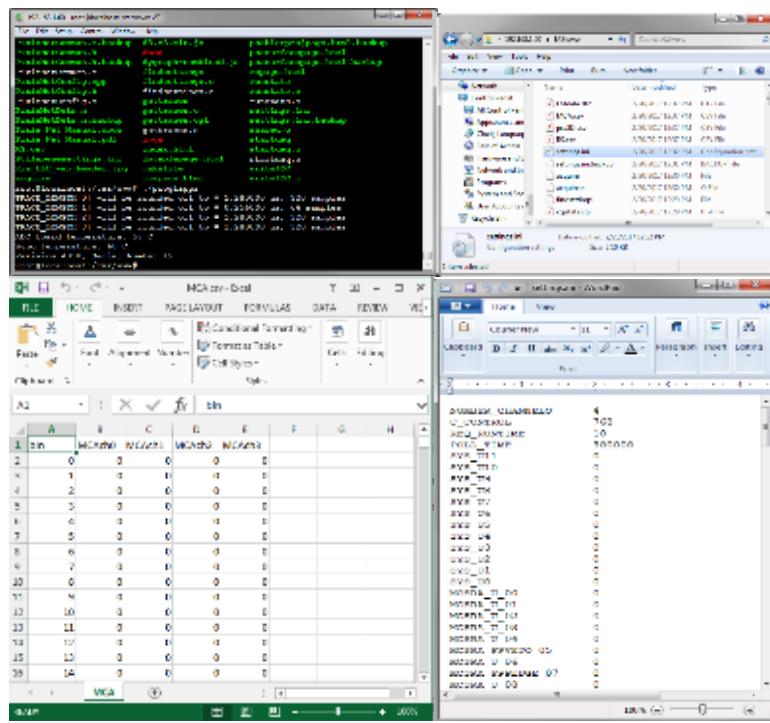
For direct graphical feedback, users can log in via terminal as in the preceding section, but view results as a webpage. The Pixie-Net XL is running a web server, and the output files are plotted on webpages with a variety of java scripts.

While still requiring familiarity with Linux, the graphical feedback makes for an easier setup and a better presentation of the results.

### 3.3.3 Terminal, SMB and Windows programs

With SMB file sharing, Windows programs can be used to directly view, edit, and analyze settings and results on the Pixie-Net XL's SD card. For example, the workflow could be

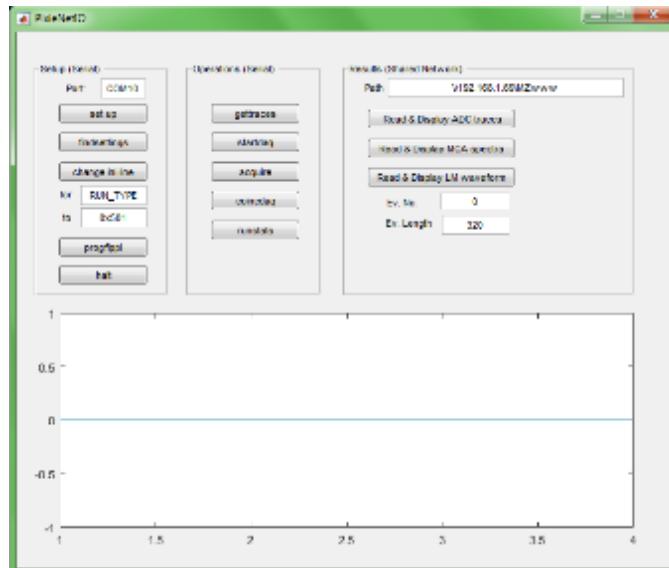
- 1) Using WordPad, open, modify, and save `settings.ini`
- 2) Execute `./progfippi` and `./startdaq` in the terminal
- 3) Open `MCA.csv` in Excel or other program to view, analyze, and plot results (or the webpages)



### 3.3.4 Graphical User Interface on Host PC

Many data acquisition and analysis program can communicate with serial ports, external files and web servers. Examples include Igor Pro, MATLAB, LabVIEW, and many more. It is therefore possible to develop graphical user interfaces with buttons and plots in such a program. As described in section 14, Igor Pro's “very dumb terminal” function (VDT2) can be used to send the commands normally typed into the terminal with the click of a button. After sending the command, Igor reads the Pixie-Net XL webpage and extracts and displays the data. Igor Pro version 8 and higher also can use the function URLrequest to communicate via the web API functions.

A demo MATLAB implementation is shown in the screenshot below. It uses serial port communication similar to the Igor Pro example described above, then reads and displays the data from files. (While in principle the Pixie-Net XL's FPGA firmware can be developed with MATLAB tools as well, this is currently not supported by XIA.)



Our current policy is to provide demo code for free, but to charge for software and support for more sophisticated user interfaces, in particular if customized for specific applications. Please contact XIA for more information.

### 3.3.5 Webpage-Only Operation

As an alternative to the terminal entry and execution of the C programs to set up and perform data acquisition (e.g. `./progfippi` and `./startdaq`), there is a [moderately] secure web page that allows execution of the C programs as a cgi script through a browser.

The Web Operation page can be reached via a link in the Pixie-Net XL Home page. In its primitive form, it essentially lists the C functions that would normally be called by typing in the terminal; clicking on the function name executes the function. Data is created in a separate subdirectory with Linux permissions for data write access for the webserver (`/var/www/webops`). The Web Operation main page shows the equivalent links from the Pixie-Net XL home page to download and display the data.

As this webpage allows generation and execution of files from any remote user, the webserver has been set up to require authentication to access this webpage. Details of the setup are described in section 10. The default user ID/password is `webops/xia17pxn`, currently specified as plain text in the file `/var/www/webopspasswords`. Obviously, this is only moderately secure and should be a) changed by the user as soon as possible and b) upgraded to encrypted password storage for sensitive environments.

Two additional webpages implement a basic graphical user interface. In the setup page (section 5.6), ADC traces can be viewed and changes in key settings such as gain, offset and polarity can be applied to the settings file. In the DAQ control page (section 5.7), runtype and runtime can be set in the settings file and DAQ runs can started.

The screenshot shows the Pixie-Net Web Operations interface. On the left is a sidebar with links: XIA Logo, Home, ADC setup, DAQ control, Most recent MCA spectra, Current run statistics, and Web interface. Below these are Help links: This set of pages allows DAQ and parameter control (after login). Proceed to ADC setup to test and configure the input parameters. For more help, try the tooltips or see the Pixie-Net XL manual and XIA Pixie-Net website.

**Current Status (cgl)**

When no other task is in progress, view/refresh

- ADC setup: Displays ADC waveforms together with parameter settings
- DAQ control: Set run type and time, start DAQ
- ADC traces: Displays ADC waveforms read just read from Pixie-Net XL
- Run statistics: Displays Run statistics read just read from Pixie-Net XL(full)

Run executables as if typed in terminal:

- bootfpga
- progfippi
- startdaq
- adcinit
- findsettings
- gettraces
- runstats

**DAQ Monitoring**

During or after the data acquisition, view most the recent

- MCA spectra
- Run statistics (short)
- PSA histogram, with projections or surface plot

**DAQ Results**

After the data acquisition, open/download

- 0x100 Binary list mode data (from run type 0x100 )
- 0x400/402 Binary list mode data (from run types 0x400, 0x402)
- 0x401 Text list mode data (from run type 0x401)
- MCA spectra as csv file
- Run statistics as csv file (full)
- PSA histogram as csv file

To view most recent ADC traces in a plot, click [here](#).

### 3.3.6 Web API

The webpage-only operation described in the previous section makes use of a number of cgi functions that are called by browser button clicks. Essentially, the browser issues an HTTP request to the Pixie-Net XL (e.g. <http://192.168.1.67/progfippi.cgi>), and the Pixie-Net XL’s web server responds by executing the cgi function (here: programming parameters into the FPGA). While some of the cgi functions simply “print” a webpage (for example, cgitraces.cgi prints the ADC page with current ADC waveforms), the more sophisticated functions respond to http requests with more complex actions and return values. For example, cgewritersettings.cgi parses the http request to update parameter values in settings.ini. These functions therefore essentially constitute a web API – a “programmatic interface consisting of one or more publicly exposed endpoints to a defined request-response message system”<sup>5</sup>. While limited in scope, this web API provides basic control of the Pixie-Net XL setup and data acquisition. It is primarily used by the webpage-only operation described above, but also can be used by any program that can issue http

<sup>5</sup> Wikipedia definition of server side web API, 2022

get commands. One such program is Igor Pro, (using the “URLRequest” function in version 8 and higher) and the demo GUI described in section 14 can serve as a coding example. A possible C implementation might make use of LibCurl.

The cgi functions are described in detail in section 4. This development is in progress and only the most common functions and parameters are supported at this time. Please contact XIA for more information.

### 3.3.6.1 Application Example

The web pages adcsetuppage (5.6) and daqpage (5.7) implement a browser based program on a remote PC utilizing the web API functions. (The program code, i.e. the javascript in the webpage, is provided by the Pixie-Net XL in the page itself but executed by the remote PC in the browser). For another implementation example of the web API we use Igor Pro (version 8 or higher). It supports a function “URLRequest” with the url given as a string argument [urlstring] and the server response returned in a string [ServerResponse]. User and password are handled automatically. Equivalent functions are assumed to exist in other tools.

Basic operation of the Pixie-Net XL would include the steps below. We use ip/wo as a shortcut to the webops folder on the Pixie-Net XL, e.g. “192.168.1.99/webops”. “PC” stands for actions on the remote PC, “PN” for actions triggered on the Pixie-Net XL. This example assumes RUN\_TYPE 0x110 or 0x410 and DATA\_FLOW=4.

#### 1. Read settings from file on Pixie-Net XL’s SD card to remote PC

PC: URLRequest with urlstring="ip/wo/cgireadsettings.cgi"

PN: executes cgireadsettings.cgi, “prints” csv table of settings

PC: receives csv table of settings in ServerResponse, sorts into local variables

#### 2. Change settings on remote PC

PC: change local variables corresponding to settings

#### 3. Write settings from remote PC to file on Pixie-Net XL’s SD card

PC: URLRequest with urlstring="ip/wo/cgiwritesettings.cgi"

[plus a string defining what is written with what value – see description of API function]

PN: executes cgiwritesettings.cgi, updating settings.ini

#### 4. Apply settings to pulse processing FPGAs for Pixie-Net XL

PC: URLRequest with urlstring="ip/wo/progfippi.cgi"

PN: executes progfippi, writing parameters in settings.ini to FPGAs

#### 5. Start UDP receiver on remote PC

PC: Execute “udp\_receive” or equivalent in command line window.

(Igor Pro can use xop to integrate into GUI)

#### 6. Start data acquisition for Pixie-Net XL

PC: URLRequest with urlstring="ip/wo/startdaq.cgi"

PN: executes startdaq, which enables the UDP output and MCA histogramming  
PC: wait for URLRequest to complete, will take entire REQ\_RUNTIME.

Alternatively, PC can use udpena.cgi which will only start the UDP output and return right away. [see section 4 for optional arguments for WR time start/stop]

#### 7. Stop data acquisition for Pixie-Net XL

PC: when using startdaq, there is no method to stop the DAQ.

- When starting a DAQ with `udpena.cgi`, it can be stopped with `upddis.cgi`
8. **Collect output data**  
Copy LM data from `udp_receive` output file, MCAs and run statistics from SD card files or from webpages to final destination.
- 

## 3.4 Optimizing Parameters

Optimization of the Pixie-Net XL's run parameters for best resolution depends on the individual systems and usually requires some degree of experimentation. Rough guidelines for setting parameters are described below.

### 3.4.1 Energy Filter Parameters

The main parameter to optimize energy resolution is the energy filter rise time (ENERGY\_RISETIME). Generally, longer rise times result in better resolution, but reduce the throughput. Optimization should begin with scanning the rise time through the available range. Try 2 $\mu$ s, 4 $\mu$ s, 8 $\mu$ s, 11.2 $\mu$ s, take a run of 60s or so for each and note changes in energy resolution. Then fine tune the rise time.

The flat top (ENERGY\_FLATTOP) usually needs only small adjustments. For a typical coaxial Ge-detector we suggest to use a flat top of 1.2 $\mu$ s. For a small detector (20% efficiency) a flat top of 0.8 $\mu$ s is a good choice. For larger detectors flat tops of 1.2 $\mu$ s and 1.6 $\mu$ s will be more appropriate. In general the flat top needs to be wide enough to accommodate the longest typical *signal rise time* from the detector. It then needs to be wider by one filter clock cycle than that minimum, but at least 3 filter clock cycles. Note that a filter clock cycle ranges from 0.026 to 0.853 $\mu$ s, depending on the filter range (FILTER\_RANGE), so that it is not possible to have a very short flat top together with a very long filter rise time.

### 3.4.2 Threshold and Trigger Filter Parameters

In general, the trigger threshold (TRIGGER\_THRESHOLD) should be set as low as possible for best resolution. If too low, the input count rate will go up dramatically and “noise peaks” will appear at the low energy end of the spectrum. If the threshold is too high, especially at high count rates, low energy events below the threshold can pass the pile-up inspector and pile up with larger events. This increases the measured energy and thus leads to exponential tails on the (ideally Gaussian) peaks in the spectrum. Ideally, the threshold should be set such that the noise peaks just disappear.

The settings of the trigger filter have only minor effect on the resolution. However, changing the trigger conditions might have some effect on certain undesirable peak shapes. A longer trigger filter rise time (TRIGGER\_RISETIME) allows the threshold to be lowered more, since the noise is averaged over longer periods. This can help to remove tails on the peaks. A long trigger filter flat top (TRIGGER\_FLATTOP) will help to trigger better on slow rising pulses and thus result in a sharper cut off at the threshold in the spectrum.

### 3.4.3 Decay Time

The preamplifier decay time  $\tau$  (TAU) is used to correct the energy of a pulse sitting on the falling slope of a previous pulse. The calculations assume a simple exponential decay with one decay constant. A precise value of  $\tau$  is especially important at high count rates where

pulses overlap more frequently. If  $\tau$  is off the optimum, peaks in the spectrum will broaden, and if  $\tau$  is very wrong, the spectrum will be significantly blurred.

A first rough estimate of  $\tau$  can be obtained from the ADC traces. Fine tuning of  $\tau$  can be achieved by exploring small variations around the estimated value ( $\pm 1\text{-}2 \mu\text{s}$ ). This is best done at high count rates, as the effect on the resolution is more pronounced. The value of  $\tau$  found through this way is also valid for low count rates. Manually enter  $\tau$ , take a short run, and note the value of  $\tau$  that gives the best resolution.

#### 3.4.4 Baselines and ADC calibration

Between detector pulses, the Pixie module continuously measures baselines, which is ultimately used to correct for the DC offset. Multiple baseline measurements can be averaged to reduce noise (BLAVG), and a threshold (BLCUT) can be set to exclude the occasional bad measurement from the average.

## 4 API functions

---

### 4.1 progfippi

#### Functions performed

- Parse .ini file, extract FPGA parameters
- Convert parameters into “FPGA units”, e.g. number of samples instead of microseconds
- Apply limits and dependencies, if not ok, give feedback and abort
- Read data from PROM and verify the current version of the Pixie-Net XL hardware is supported by the software
- Write to FPGA registers to apply settings
- Toggle I2C lines (0x002) to set gain etc. [May be moved to FPGA at some point]
- Issue DSP\_CLR and RTC\_CLR to PL resets

**Arguments:** fixed to read parameters from “settings.ini”

**Output:** Errors for bad settings, prints hardware info from PROM and temperatures

**Restrictions:** Do not execute during a data acquisition. Automatically executed at powerup

**Web API:**

- URL get request: <ip>/webops/progfippi.cgi
- Server action: See “functions performed” above
- Server response: Text as in “output” above
- Implementation example (Igor Pro):  
URLRequest/AUTH={usr,pw} url=”<ip>/webops/progfippi.cgi”

---

### 4.2 gettraces, cgitraces, cgiprinttraces

Read untriggered ADC data: 2K samples, all present channels. gettraces writes to a local file ADC.csv, cgitraces prints a webpage with the ADC data to std out, which is piped into the webserver. cgiprinttraces prints data only to std out.

**Functions performed**

- Read ADC register in PL, write to file or print

**Arguments:** none

**Output:** ADC.csv

**Restrictions:** Do not execute during a data acquisition

**Notes:** The cgitraces program requires presence of the web page template adcpage.html in the same folder.

**Web API:**

- URL get request: <ip>/webops/ cgiprinttraces.cgi
- Server action: See “functions performed” above

---

## 4.3 findsettings

Perform a series of tasks assisting in the optimization of gain, offset, tau, etc

### Functions performed

- Initialize ADCs
- Resolve random channel swapping between odd and even channels (some ADCs)
- Ramp DACs and determine DAC setting for a baseline at BASELINEPERCENT fraction of the full ADC range  
(Note: Pulse polarity has to be set correctly for this function to work properly)
- Determine pulse decay time Tau.  
(Note: the current value of TAU is used as a minimum fit range, and XDT determines the time span of ADC traces to be analyzed).

**Arguments:** none

**Output:** prints found values for offset, tau, etc

**Restrictions:** Do not execute during a data acquisition

**Note:** No changes are made to the settings file

### Web API:

- URL get request: <ip>/webops/findsettings.cgi
- Server action: See “functions performed” above
- Server response: error message or list of offset values in the form  
“Channel %u: DAC value %u, offset %fV, ADC %u\n”  
repeated for each channel
- Implementation example (Igor Pro):  
URLRequest/AUTH={usr,pw} url=”<ip>/webops/findsettings.cgi”

---

## 4.4 runstats, cgistats

These functions are used to read and output the run statistics registers. runstats writes them to a local file RS.csv. cgistats prints them as a webpage to std out, which is piped into the webserver; this is displayed as the “Run statistics” webpage. These functions can be used for development, setup, and diagnostics, for example the run statistics include module revision and serial number, temperature, current input count rate (ICR) and out of range fraction of the ADC signal (OOR).

Run statistics registers are also read and written to the file RS.csv by the data acquisition routines (acquire, startdaq, etc). During a data acquisition, do not execute runstats and cgistats (i.e. do not load webpage “Run statistics” = cgistats.cgi) to avoid conflicting access to the run statistics registers and possible delays and dead times. Instead monitor the file RS.csv (i.e. load webpage “Run status”= rpage.html)

**Functions performed**

- Read Runstats registers in PL
- Read I2C info from PROM
- write to file

**Arguments:** none**Output:** RS.csv**Restrictions:** Do not execute during a data acquisition**Notes:** The cgi program requires presence of the web page template rspage.html in the same folder.

---

## 4.5 startdaq

Start list mode run with or without waveforms (0x100, 0x105, 0x110, 0x111, 0x400, 0x401, 0x404, 0x500, 0x501, 0x502, 0x410, 0x411). Pulse shape analysis data is acquired in run types 0x105, 0x111, 0x400, 0x401, 0x410, 0x411, and 0x502.

This functions is relatively simple and limited in throughput. It serves as an introductory example to users that may want to customize data acquisition.

The parameter DATA\_FLOW controls how the data is output to the local SD card vs the Ethernet interface, which in turn affects the throughput.

**Functions performed**

- Parse .ini file, extract parameters
- Compute coefficients for E reconstruction etc
- Set bits in reg 0x000 to start run
  - o RunEnable -> 1 (an overall enable of the run)
  - o nLive -> 0 (can be used to temporarily pause acquisition (rarely used))
- Monitor DAQ
  - o Poll EVSTATS, if data ready:  
read event data from PL,  
compute E,  
read waveforms from PL,  
write to LM file (if LM run),  
increment MCA
  - o Periodically save MCA to file (4Ki bins)
  - o Periodically read runstats from PL and write to file
    - o Periodically save PSA 2D histogram data to file
    - o Periodically read BL data from PL, compute BLavg
- Stop when time is up
- Save final run statistics, PSA, and MCA to file (32Ki bins)

**Arguments:** none (file names are fixed)

**Output:** MCA.csv, RS.csv, RATES.csv, PSA.csv, LM file (text or binary)

#### Web API:

- URL get request: <ip>/webops/startdaq.cgi
- Optional argument: For WR\_RUNTIME\_CTRL > 3, add "?WR\_tm\_tai\_start=##" with ## a White Rabbit time in the future to the URL get request to start DAQ at this time.
- Server action: See "functions performed" above
- Server response: error message in case of error
- Implementation example (Igor Pro):
 

```
URLRequest/AUTH={usr,pw} url="/webops/startdaq.cgi"
```

## 4.6 mcadaq

Start MCA run (0x301) Simplified version of startdaq for MCA runs only.

#### Functions performed

- Parse .ini file, extract parameters
- Set bits in reg 0x000 to start run
  - o RunEnable -> 1 (an overall enable of the run)
  - o nLive -> 0 (can be used to temporarily pause acquisition (rarely used))
- Monitor DAQ
  - o Poll EVSTATS, if data ready: increment MCA
  - o Periodically save MCA to file (4Ki bins)
  - o Periodically read runstats from PL and write to file
- Stop when time is up
- Save final run statistics, PSA, and MCA to file (32Ki bins)

**Arguments:** none (file names are fixed)

**Output:** MCA.csv, RS.csv, RATES.csv

## 4.7 pollcsr

Reports the 16bit Control and Status Register value, which can be used to check if a run is in progress, or the 64bit White Rabbit time in seconds. (Note the use of arguments compared to Pixie-Net)

#### Functions performed

- Read CSR or WR time register, print and return value

**Arguments:** If QUERY\_STRING (from http get) is mode=

0: CSR from MZ/PZ controller

1: CSR from Kintex #0

2: CSR from Kintex #1  
 3: WR time from PZ controller  
 4: WR time from Kintex #0  
 5: WR time from Kintex #1  
 without argument, defaults to 0

**Output:** print and return CSR or WR time value

**Restrictions:** None

**Web API:** URL get request: <ip>/webops/polcsr.cgi?MODE=5

- Server action: See “functions performed” above
- Server response:  
 For argument 0-2, a string of a 4-digit hexadecimal number, i.e. “0x####” with # = 0-F  
 For argument 3-5, a string of a long unsigned integer number, e.g. “12345”
- Implementation example (Igor Pro):  
 URLRequest/AUTH={usr,pw} url=”<ip>/webops/polcsr.cgi?mode=5”

## 4.8 bootfpga

Downloads the FPGA configuration data to the FPGAs.

**Functions performed**

- Read hardware information
- Select file to download
- Initialize FPGA and download
- Apply ADC initialization steps, if required

**Arguments:** WR units only: any argument loads 10G firmware instead of WR

Use –R for custom firmware supporting run type 0x404

**Output:** Status messages

**Restrictions:** Automatically executed after powerup. Must be followed by ./progfippi.

## 4.9 cgireadsettings.cgi

Parses the settings file (settings.ini) and returns key values as a comma separated list to be used by webpages

**Functions performed**

- Read settings file
- Print key values

**Arguments:** none

**Output:** prints headers and <values> as follows

```
channel,polarity,offset,analog gain,digital gain,tau,  

00,<CCSRA_INVERT_05(00)>,<VOFFSET (00)>,<ANALOG_GAIN(00)>,<DIG_GAIN(00)>,<TAU(00)>,
```

```

...
31,<CCSRA_INVERT_05(31)>,<VOFFSET(31)>,<ANALOG_GAIN(31)>,<DIG_GAIN(31)>,<TAU(31)>,
RUN_TYPE,<RUN_TYPE,
REQ_RUNTIME,<REQ_RUNTIME>,
DATA_FLOW,<DATA_FLOW>,
WR_RUNTIME_CTRL,<WR_RUNTIME_CTRL>,

```

**Restrictions:** none

**Notes:** As an alternative, it read the whole settings file from the shard SD card folder

**Web API:**

- URL get request: <ip>/webops/cgireadsettings.cgi
- Server action: See “functions performed” above
- Server response: csv data containing parameters in the following format:  
header line  
`channel,polarity,offset,analog gain,digital gain,tau`  
values in format  
`%d,%d,%04f,%02f,%04f,%04f,`  
repeated for each channel, and  
followed by module parameters  
`RUN_TYPE,0x%x,`  
`REQ_RUNTIME,%d,`  
`DATA_FLOW,%d,`  
`WR_RUNTIME_CTRL,%d,`  
note trailing commas
- Implementation example (Igor Pro):  

```

URLRequest/AUTH={usr,pw} url=<ip>/webops/cgireadsettings.cgi"
for(ch=0;ch<MaxNchannels;ch+=1)
    polarity[ch] = str2num(StringFromList( 7+6*ch, ServerResponse, ","))
    voffset[ch] = str2num(StringFromList( 8+6*ch, ServerResponse, ","))
    analog_gain[ch] = str2num(StringFromList( 9+6*ch, ServerResponse, ","))
    digital_gain[ch] = str2num(StringFromList(10+6*ch, ServerResponse, ","))
    tau[ch] = str2num(StringFromList(11+6*ch, ServerResponse, ","))
endfor
Run_Type = str2num(StringFromList(6*(MaxNchannels+1)+2, ServerResponse, ","))
Req_Runtime = str2num(StringFromList(6*(MaxNchannels+1)+4, ServerResponse, ","))
Data_Flow = str2num(StringFromList( 6+6*(MaxNchannels+1), ServerResponse, ","))
WR_RT_CTRL = str2num(StringFromList( 8+6*(MaxNchannels+1), ServerResponse, ","))

```

(with MaxNchannels=32)

## 4.10 cgiwritesettings.cgi

Receives a parameter=value string from a webpage and writes it to the settings file (settings.ini).

**Functions performed**

- Parse parameter=value string
- Replace matching line in settings.ini

**Arguments:** none

**Output:** Status messages, regenerated settings file

**Restrictions:** Do not execute during data acquisition

**Web API:**

- URL get request:  
`<ip>/webops/cgiwritesettings.cgi?<PARAMETER>=<TYPE>&v<ch>=<value>`  
 where  
`<PARAMETER>` is the parameter name in the settings file  
`<TYPE>` is “MODULE” or “CHANNEL”  
`<ch>` is the channel number  
`<value>` is the new parameter value
- Server action: See “functions performed” above
- Server response: error message or success message
- Implementation example (Igor Pro):
 

```
cmd = "192.168.1.100/webops/cgiwritesettings.cgi?TAU=CHANNEL
for(ch=0;ch<MaxNchannels;ch+=1)
    sprintf chval,"&v%d=%4f",ch, tau[ch]
    cmd = cmd + chval
endfor
URLRequest/AUTH={usr,pw} url=cmd
```

## 4.11 adcinit

Calibration function to initialize some ADC variants

**Functions performed**

- DB01: check/initialize ADC data capture. Already done as part of ./bootfpga
- DB02, DB04: do nothing
- DB06: 14bit, 500 MSPS variant only: Attempt to calibrate ADC cores.  
 The ADC in the DB06 14/500 variant operates as 2 interleaved ADC cores digitizing at 250 MHz, that have to be matched in gain and offset. If not properly matched, even and odd samples appear offset by tens to hundreds of ADC steps.

**Arguments:** DB06 only:

- No argument: usage message
- 1 argument: 0 measure mismatch only
- 1 autoset calibration values
- 2 use default calibration values
- 3 run calibration procedure
- 4 arguments: arguments specify FPGA, channel in FPGA, address (hex), data (hex) to write; routine writes data to that register and exits

**Output:** Status messages,

**Restrictions:** Do not execute during a data acquisition

**Notes:** This function is under development. Mode 3 works reasonably well.

---

## 4.12 udpna[.cgi]

Executes the first portion of startdaq only, up to the point where the pulse processing and UDP output is enabled in the FPGAs. This will start the list mode data output stream (DATA\_FLOW must be 4), but will not update MCA and run statistics files on the SD card (seen in mcapage.html). The function completes while the DAQ is still in progress. Run statistics can be monitored by the runstats function. DAQ should be ended with udpdis (or progfippi).

### Functions performed

- Initialize DAQ, including synchronization
- WR synchronization can use WR\_RTCtrl=3 (or 4) with WR time given as QUERY\_STRING (from http get).

**Arguments:** optional QUERY\_STRING (from http get)

**Output:** Status messages. No files are created on SD card

**Restrictions:** none

---

## 4.13 udpdis[.cgi]

This is the opposite function to udpna. It ends (disables) the pulse processing and UDP output in the FPGAs. Saves final runs statistics to SD card.

### Functions performed

- Ends (disables) the pulse processing
- Saves final runs statistics to SD card

**Arguments:** none

**Output:** Status messages, RS.csv, RATES.csv

**Restrictions:** none

# 5 Web Pages

The Pixie-Net XL web pages are located in /var/www; this is the default directory for the installed lighttpd webserver. Browsing to the Pixie-Net XL’s IP address will bring up index.html, from which all other pages can be accessed.

## 5.1 index.html

**Pixie-Net XL Home Page**

**XIA**  
Instruments That Advance The Art

- [Home](#)
- [Current ADC traces](#)
- [Current run statistics](#)
- [Most recent MCA spectra](#)
- [Web interface](#)

**Help**

If you're seeing this page, it means you have successfully set up your Pixie-Net XL. For more help, see

- [Pixie-Net XL manual](#)
- [XIA Pixie-Net website](#)

or try the tooltips

**Current Status (cgi)**

When no other task is in progress, [view/refresh](#)

- [ADC traces](#)
- [Run statistics \(full\)](#)

For a web interface, [click here](#) (login required)

**DAQ Monitoring**

During or after the data acquisition, view most the recent

- [MCA spectra](#)
- [Run statistics \(short\)](#)
- [PSA histogram, with projections or surface plot](#)

**DAQ Results**

After the data acquisition, open/download

- [0x100 Binary list mode data \(from run type 0x100 \)](#)
- [0x400/402 Binary list mode data \(from run types 0x400, 0x402\)](#)
- [0x401 Text list mode data \(from run type 0x401\)](#)
- [MCA spectra as csv file](#)
- [Run statistics as csv file \(full\)](#)
- [PSA histogram as csv file](#)

To view most recent ADC traces in a plot, [click here](#).

This is the home page for the Pixie-Net XL.

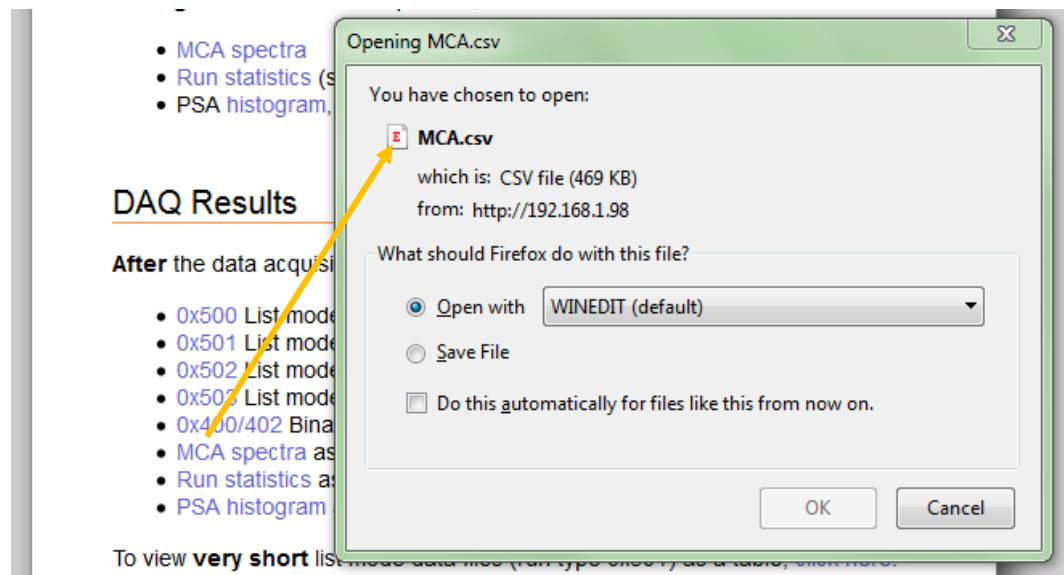
At the left sidebar of the page, there are a few shortcuts to common tasks or pages. Do not use the “current” links during a data acquisition, instead use the Run Status link in the next section which work on files generated by the data acquisition.

Under **DAQ Monitoring**, there are links to view (as plots or tables in the browser) the csv files from the most recent data acquisition for MCA spectra and run status.

Under **DAQ Results**, the final data files from the most recent tasks can be downloaded or viewed in the browser. This requires that a task in the Linux terminal, such startdaq, runstats, acquire or gettraces has been executed and completed. The screenshot

below shows opening/downloading the MCA file. The file can then be opened, and its data displayed and analyzed, in any suitable program.

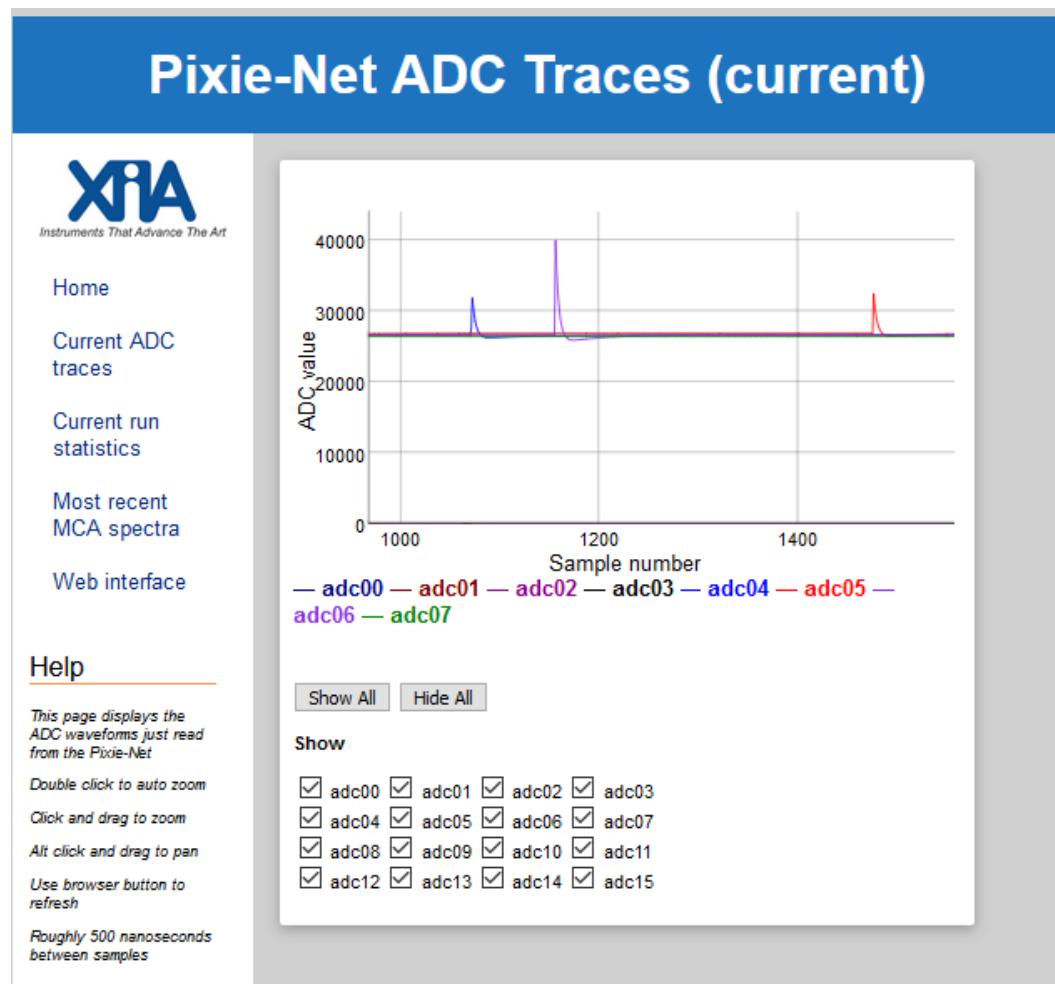
A further link runs a CGI script to extract and display a list mode waveform from the most recent 0x500 data acquisition (see below).



Under **Current Status**, procedures are executed directly on the Pixie-Net XL, their output generating webpages for ADC traces and other results instead of files. This combines the two steps of i) executing a function on the Pixie-Net XL terminal to create data and ii) refreshing the appropriate webpage to view the results. Do not execute these links during a data acquisition.

A further link opens the web operations page (see below) which allows execution of the terminal functions from the web browser. Login is required to prevent unauthorized access.

## 5.2 adcpage.html, cgitraces.cgi



The adcpage contains a chart showing untriggered waveforms read from the Pixie-Net XL's ADCs. Internally, this is a dygraph javascript that links to the file ADC.csv. The csv file has to be generated or refreshed by executing gettraces on the Pixie-Net XL terminal.

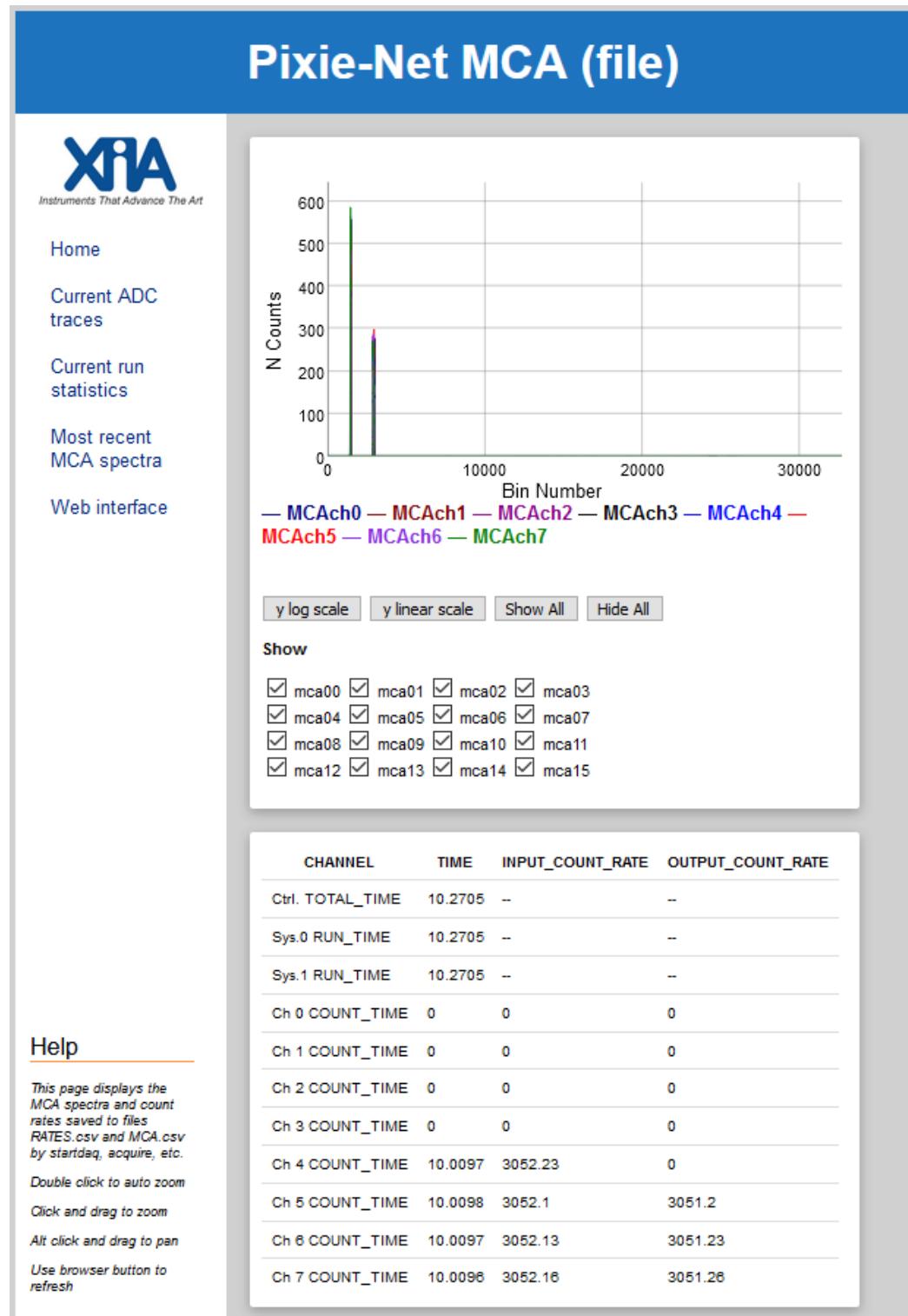
Buttons and checkboxes below the graph allow selection of the channels to view. (Checkboxes do not update from button actions.)

Mouse click and drag operations allow to zoom in and pan. See the help section for details. Use the browser **Refresh** button to update for new data.

An equivalent page reading and displaying the data can be generated by a cgi script from the web browser when no data acquisition is in progress.

Time between samples is controlled by parameter XDT in the settings file

## 5.3 mcapage.html



The mcapage contains a chart showing MCA spectra from the current or most recent data acquisition. Internally, this is a dygraph javascript that links to the files MCA.csv and

RATES.csv. The files are generated and periodically refreshed by DAQ functions like `startdaq` or `acquire`.

Buttons and checkboxes below the graph allow selection of the channels to view. (Checkboxes do not update from button actions.)

A table below the MCA plot shows the most important run statistics.

Mouse click and drag operations can be used to zoom in and pan.

## 5.4 rspage.html, cgistats.cgi

Pixie-Net Run Statistics (current)														
Parameter	Val.	Parameter	Sys. 0	Sys. 1	Parameter	Ch. 00	Ch. 01	Ch. 02	Ch. 03	Ch. 04	Ch. 05	Ch. 06	Ch. 07	Ch. 08
TOTAL_TIME	3.8e-7	RUN_TIME	4.24e-7	4.16e-7	COUNT_TIME	0	0	0	0	1.82e-7	1.82e-7	1.82e-7	1.82e-7	
PR_CODE_VERSION	0x0001	-	0	0	INPUT_COUNT_RATE	0	0	0	0	0	0	0	0	
ACTIVE	0	-	0	0	OUTPUT_COUNT_RATE	0	0	0	0	0	0	0	0	
CBROUT	0x0000	CBROUT	0x020	0x020	COUNTTIME	0	0	0	0	24	24	24	24	
reserved	0x0000	sysstatus	0x0	0x0	COUNTTIME	0	0	0	0	0	0	0	0	
reserved	0x0	MEM_O_CNT	0x0	0x0	COUNTTIME	0	0	0	0	0	0	0	0	
PFW_ROOTED	0x1	MEM_O_CNT	0x0	0x0	reserved	0	0	0	0	0	0	0	0	
reserved	0x0	MEM_O_CNT	0x0	0x0	NTRIG	0	0	0	0	0	0	0	0	
reserved	0x0	MEM_O_CNT	0x0	0x0	NTRIG	0	0	0	0	0	0	0	0	
SysTime	0xA008	ADCframe	0x0	0x0	NTRIG	0	0	0	0	0	0	0	0	
SysTime	0xF034	reserved	0xABCD	0xABCD	reserved	0	0	0	0	0	0	0	0	
PW_VERSION	0x000	RunTime	0x86	0x84	NDOUT	0	0	0	0	0	0	0	0	
HW_VERSION	0xA100	RunTime	0x0	0x0	NDOUT	0	0	0	0	0	0	0	0	
TotalTime	0x150	RunTime	0x0	0x0	NDOUT	0	0	0	0	0	0	0	0	
TotalTime	0x0	reserved	0x0	0x0	reserved	0	0	0	0	0	0	0	0	
TotalTime	0x0	PW_VERSION	0x480	0x480	reserved	0	0	0	0	0	0	0	0	
TotalTime	0x0	WH_TM_TAI	0x0	0x10	reserved	0	0	0	0	0	0	0	0	
reserved	0x0	WR_TM_TAI	0x0	0x0	reserved	0	0	0	0	0	0	0	0	
reserved	0x0	WT_TM_CV0	0x0	0x100	reserved	0	0	0	0	0	0	0	0	
PDB_VERSION	0xA101	WT_TM_CV0	0x0	0x657	reserved	0	0	0	0	0	0	0	0	
PDB_SNMMI	8	dmstatus	0	0	reserved	0	0	0	0	0	0	0	0	
SNMMI	8	dmstatus	0	0	reserved	0	0	0	0	0	0	0	0	
T_BOARD	44	T_ADC	511	512	reserved	0	0	0	0	0	0	0	0	
T_ZYNG	61	T_WR	0	0	reserved	0	0	0	0	0	0	0	0	

The rspage contains a table showing the Run Status information: the run statistics from the most recent data acquisition (which can be the one in progress). Internally, this is a d3 javascript that links to the file RS.csv. The file is generated and periodically refreshed by DAQ functions like `./startdaq` or `./acquire`. It can also be generated outside a data acquisition by executing `./runstats` on the Pixie-Net XL terminal (actual run statistics will not change, but a real time counter increments and some hardware and status information may be of interest).

A similar page reading and displaying the data can be generated by the script cgistats.cgi from the web browser. This should only be used when no data acquisition is in progress. This page is named Run Statistics to differentiate for the Run Status page described in the previous paragraph.

Units in the pages (and the underlying csv file) are in seconds or counts/s for the first 3 lines, then raw internal units (counts and ns).



## 5.5 Web Operations ([webops/webopsindex.html](#))

The web operations page allows execution of the terminal functions from the web browser. Login is required to prevent unauthorized access. Default ID/password are **webops/xia17pxn** (*please change*). Most of the links and functions from the home page are duplicated, with the following additions and differences:

**Pixie-Net Web Operations**

- XIA Logo
- Home
- ADC setup
- DAQ control
- Most recent MCA spectra
- Current run statistics
- Web interface

**Help**

*This set of pages allows DAQ and parameter control (after login). Proceed to ADC setup to boot and configure the input parameters. For more help, try the [tooltips](#) or see the [FAQ](#).*

- [Pixie-Net XL manual](#)
- [XIA Pixie-Net website](#)

**Current Status (cgi)**

**When no other task is in progress, view/refresh**

- [ADC setup](#): Displays ADC waveforms together with parameter settings
- [DAQ control](#): Set run type and time, start DAQ
- [ADC traces](#): Displays ADC waveforms read just read from Pixie-Net XL
- [Run statistics](#): Displays Run statistics read just read from Pixie-Net XL(full)

**Run executables as if typed in terminal:**

- [bootfpga](#)
- [progfippi](#)
- [startdaq](#)
- [adcinit](#)
- [findsettings](#)
- [gettraces](#)
- [runstats](#)

**DAQ Monitoring**

**During or after the data acquisition, view most the recent**

- [MCA spectra](#)
- [Run statistics \(short\)](#)
- [PSA histogram, with projections or surface plot](#)

**DAQ Results**

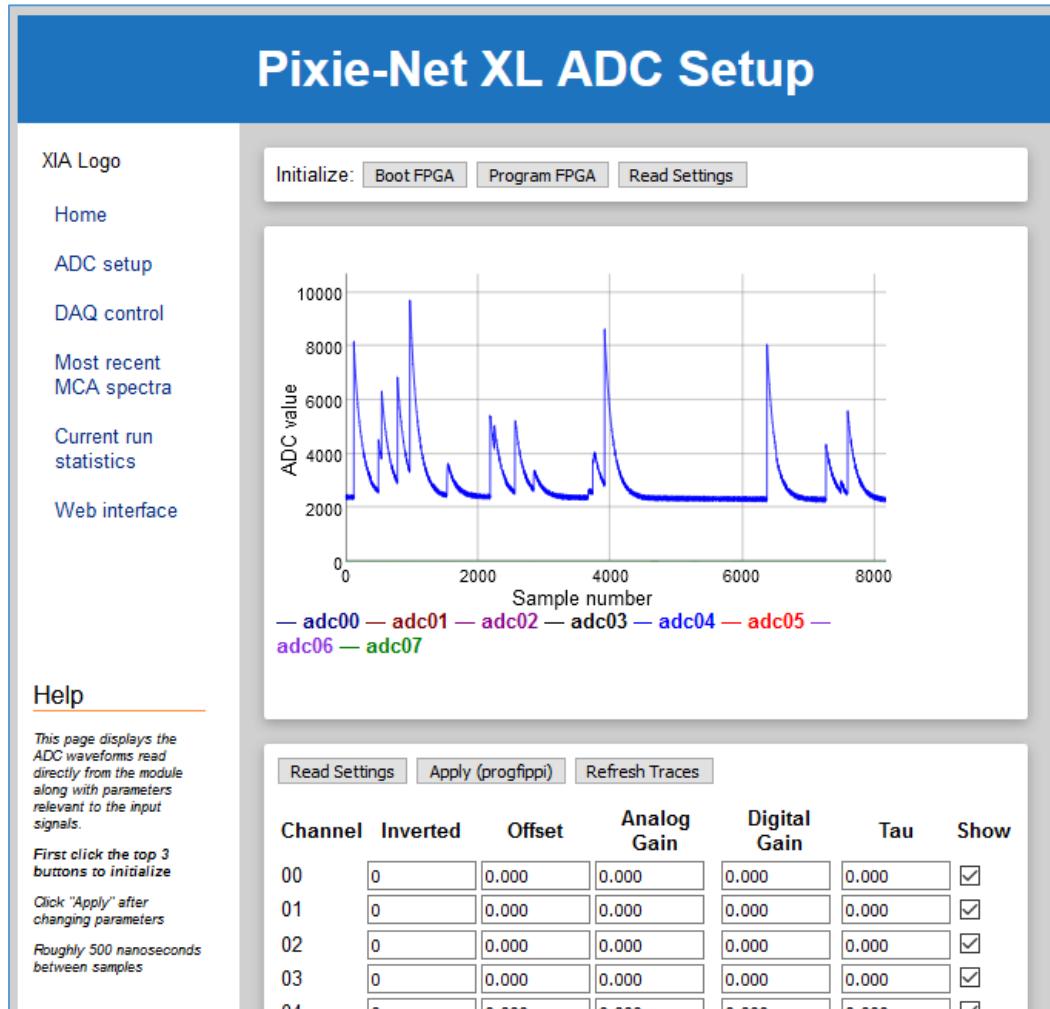
**After the data acquisition, open/download**

- [0x100 Binary list mode data](#) (from run type 0x100 )
- [0x400/402 Binary list mode data](#) (from run types 0x400, 0x402)
- [0x401 Text list mode data](#) (from run type 0x401)
- [MCA spectra as csv file](#)
- [Run statistics as csv file \(full\)](#)
- [PSA histogram as csv file](#)

To view most recent ADC traces in a plot, [click here](#).

- New list of API functions to be executed as CGI scripts. Clicking on one of these links executes the API function as if typed into the terminal.
- **Settings and data files are created in a new subfolder: /var/www/webops**
- API function output (e.g. error messages) is reported as a “plain text” web page; which is empty if no message. Use the browser back button to return to the webops page.
- Links to webpages for setup and DAQ control (see below)

## 5.6 Web Operations Setup (webops/adcsetuppage.html)



The web operations setup page can be used to initialize the Pixie-Net XL after powerup. The buttons “Boot FPGA” and “Program FPGA” are optional; these functions will automatically execute at powerup. Click “Read Settings” read the contents of the settings file into the webpage control fields below. The buttons will turn green after execution. The FPGA booting will take several seconds, users should wait for the confirmation message before clicking the next button.

After initialization, new ADC waveforms can be read from the Pixie-Net XL by clicking the “Refresh Traces” button. This runs the equivalent of “gettraces” on the Pixie-Net XL and loads the resulting file ADC data into the page. Using the fields below, gain, offset, polarity, and decay time tau can be written to the settings file. The field will turn yellow while it is selected, on de-select the webpage will communicate with the Pixie-Net XL to update the settings file. To apply the new settings to the FPGA, click the button “Apply (progfippi)”. This button turns yellow when a field (and therefore the settings file) has been modified, and back to gray after the settings have been applied. The analog settings can thus be adjusted until the signal is properly in range (baseline about 10%, not clipped, pulses start with rising edges).

Notes:

- channels that are defined as “not Good” in the settings file will return constant 1 for the ADC waveforms.
- The modified settings file is in /var/www/webops, the standard one in /var/www/ is unchanged.

The checkboxes can be used to show or hide channels from the ADC plot.

This page is under development and will be updated.

## 5.7 Web Operations DAQ Control ([webops/daqpage.html](#))

The web operations DAQ control page can be used to start and stop runs. This assumes the Pixie-Net XL has been initialized and configured, for example using the web operations setup page. After opening the page, first click “Read Settings” to read the parameters from the settings file into the control fields of the webpage. The control fields will turn yellow while selected; on de-select the webpage will communicate with the Pixie-Net XL to update the settings file. To apply the new settings to the FPGA, click the button “Apply (progfippi)”. This button turns yellow when a field (and therefore the settings file) has been modified, and back to gray after the settings have been applied.

Clicking the “Start DAQ” button will start the acquisition on the Pixie-Net XL (startdaq). MCA spectra and run statistics can be monitored with the MCA and Runstats pages (in the webops section, since the files are created in /var/www/webops). A progress bar on the DAQ page indicates the time elapsed, however, this progress is simply estimated by the time elapsed for the host PC displaying the webpage, not the actual run time elapsed on the Pixie-Net XL. Links under DAQ Results can be used to download the generated files.

Notes:

- The modified settings file is in /var/www/webops, the standard one in /var/www/ is unchanged.

This page is under development and will be updated.

## 6 Parameters in the Settings Files

The ini files contain the parameter settings for the data acquisition. There are two files, *defaults.ini* and *settings.ini*. The file *defaults.ini* should be considered read-only; it contains defaults for all parameters. The file *settings.ini* contains a subset of most relevant parameters, which will overwrite the defaults. Parameter lines can be added or removed from *settings.ini* to focus better on parameters relevant for a certain application. The .ini files are ASCII text files, one line per parameter. Parameter name spelling must be maintained, but the order is not essential. (Users customizing code can add parameters at the end of the file, and lines not matching any parameter name can be added as comments.) The parameter values are in physical units, and will be translated by the `progfippi` routine into values and bit patterns and then written into PL input registers.

Several control bit patterns are broken out as one bit per line. For other, less commonly used bit patterns, their value can be written as decimals or hexadecimals, for example 65535 or 0xFFFF.

The following sections describe the parameters. Unused parameters are shown in gray. Key parameters for *settings.ini* are shown in bold. NYI = not yet implemented.

---

## 6.1 System Parameters

Parameter Name	Units, Limits	Description
C_CONTROL	Bits, 0x0-FFFF	<p>Unused, except Bit 12: if 1, use local files to control run start/stop</p> <p>C_CONTROL is intended to modify the behavior of the DAQ routines running on Linux (written in C), rather than the pulse processing in the FPGA. Only one control bit (12) is currently implemented. If set, the startdaq routine will create a file “startok” on the SD card to indicate a DAQ has been started. It also periodically checks for the presence of a file “donesignal” created by another process; if that file is found the DAQ will be stopped even if the REQ_RUNTIME is not yet met.</p>
REQ_RUNTIME	Seconds, 5..2^32	<b>Requested run time in (full) seconds</b>
POLL_TIME	Tbd, 100..2^32	<p>Number of internal polling loops between updates of csv output files. In the order of microseconds.</p> <p>The startdaq and other DAQ routines running on the Linux side continuously poll the FPGA for new events to read out. Once in a while, output files for MCA spectra and run statistics are updated. This parameter controls how many polling loops are executed between these updates. Since the time to read out an event from the FPGA is longer than just checking if an event is waiting for readout and doing nothing if not, the actual time between file updates will vary with count rate. So if operating at high count rates and the updates are too infrequent, this number should be decreased.</p>
CRATE_ID	Number, 0-15	Pixie-16 compatible numbers to identify different chassis/slots/modules in a system.
SLOT_ID	Number, 0-15	MODULE_ID also sets the number <i>M</i> in the list mode file name “LMdata <i>M</i> .bin
MODULE_ID	Number, 0-15	List mode data files in format compatible with the Pixie-16 include these user defined numbers to identify events from different chassis, slots, or modules. There are no chassis or slots for a Pixie-Net XL, so usually these parameters simply would be 0. They could be used to insert an arbitrary user specified number in the data stream, but only in the Pixie-16 compatible data formats.
AUX_CTRL	Bits, 0x00-FF	Control bits for pulser, I2C addressing. See section 8

Parameter Name	Units, Limits	Description
		The lower 8 bits of this parameter are written into the AUX_CTRL register in the Zynq controller. These control bits are normally toggled automatically by the DAQ routines as needed. For example, bits 4-6 select which device group is addressed by the internal I2C communication. The only bit of interest to the user is usually bit 0, which turns the test pulser output on or off. Bit 1 can be used to turn the red LED on via the settings file (in addition to any internal error flags).
CLK_CTRL	Bits, 0x00-FF	<p>Control bits for clock generation. See also section 8, PLLSTART</p> <p>0x00 local clock      0x03 clock from daughterboard WRclkDB      0x0C clock from FPGA on TTCL board      0x3C clock from jitter cleaner on TTCL board</p> <p>This parameter is one of the few that is used only by the bootfpga routine, not by progfippi. It defines clock sources and clock distribution for the pulse processing and digitization. In most cases it is simply 0x00, using the local clock on the Pixie-Net XL. The other options are described in documentation specific to clock synchronization with White Rabbit or TTCL.</p>
WR_RUNTIME_CTRL	Number 0, 1, 2,3,4	<p>if 0, run starts immediately      if 1, run start/stop is synchronized to WR time in Kintex (next 10s rollover)      if 2, run start/stop is synchronized to WR time in PicoZed (next 10s rollover)      if 3, run start/stop is synchronized to WR time in Kintex (time specified by web query)      if 4, run start/stop is synchronized to WR time in PicoZed (time specified by web query)</p> <p>This parameter is only used in data acquisitions that use White Rabbit to start at a specific date/time. Rather than starting immediately when the startdaq program is executed, the actual run start in the FPGA is delayed until the White Rabbit time counters match the user specified start time.</p>
UDP_PAUSE	64ns, 0-65535	<p><b>Minimum delay between UDP packages</b></p> <p>In DAQs with DATA_FLOW=3 or 4, list mode data is buffered in SDRAM memory and transmitted as UDP packages whenever the memory is not empty. In case of high count rates, packages might therefore be sent out faster than the receiver can handle, and the Pixie-Net XL has no information on any loss of those packages. It can therefore be better to force a minimum delay</p>

Parameter Name	Units, Limits	Description
		<p>between packages, essentially setting a maximum package rate. If the memory fills up faster than the packages can be sent out, the DAQ will be paused and run statistics counters halted, which allows accounting for such data readout deadtime.</p> <p>This is especially important when funneling one or more 10G output streams into a 1G port, and for long traces (<math>&gt;672</math> samples) that are broken up into 8 UDP packages.</p>
<b>DEST_MAC0</b>	<b>Number <math>0..2^{48}</math></b>	<p><b>MAC address of the device receiving LM data through the Ethernet link from FPGA 0</b></p> <p>For the UDP packages of list mode data to arrive at the intended destination, users must specify here the MAC, IP, and/or PORT of the receiver.</p>
<b>DEST_MAC1</b>	<b>Number <math>0..2^{48}</math></b>	<p><b>MAC address of the device receiving LM data through the Ethernet link from FPGA 1</b></p> <p>For the UDP packages of list mode data to arrive at the intended destination, users must specify here the MAC, IP, and/or PORT of the receiver.</p>
<b>SRC_MAC0</b>	<b>Number <math>0..2^{48}</math></b>	<p><b>MAC address of the FPGA 0 sending LM data through the Ethernet link.</b></p> <p><b>In the 1G/WR variant, the WR PROM defines SRC_MAC instead</b></p> <p>UDP packages of list mode data require the MAC, IP, and PORT of the sender (source). In practice the actual value is often irrelevant for the package to arrive at the destination, but it can be used to identify or filter packages.</p>
<b>SRC_MAC1</b>	<b>Number <math>0..2^{48}</math></b>	<p><b>MAC address of the FPGA 1 sending LM data through the Ethernet link.</b></p> <p><b>In the 1G/WR variant, the WR PROM defines SRC_MAC instead</b></p> <p>UDP packages of list mode data require the MAC, IP, and PORT of the sender (source). In practice the actual value is often irrelevant for the package to arrive at the destination, but it can be used to identify or filter packages.</p>
<b>DEST_IP0</b>	<b>Number <math>0..2^{32}</math></b>	<p><b>IP address of the device receiving LM data through the Ethernet link from FPGA 0</b></p> <p>For the UDP packages of list mode data to arrive at the intended destination, users must specify here the MAC, IP, and/or PORT of the receiver.</p>
<b>DEST_IP1</b>	<b>Number <math>0..2^{32}</math></b>	<p><b>IP address of the device receiving LM data through the Ethernet link from FPGA 1</b></p>

Parameter Name	Units, Limits	Description
		For the UDP packages of list mode data to arrive at the intended destination, users must specify here the MAC, IP, and/or PORT of the receiver.
SOURCE_IP0	Number 0..2^32	<p><b>IP address of FPGA 0 for Ethernet data output</b></p> <p>UDP packages of list mode data require the MAC, IP, and PORT of the sender (source). In practice the actual value is often irrelevant for the package to arrive at the destination, but it can be used to identify or filter packages.</p>
SOURCE_IP1	Number 0..2^32	<p><b>IP address of FPGA 1 for Ethernet data output</b></p> <p>UDP packages of list mode data require the MAC, IP, and PORT of the sender (source). In practice the actual value is often irrelevant for the package to arrive at the destination, but it can be used to identify or filter packages.</p>
DEST_PORT0	Number 0..65535	<p><b>IP port of the device receiving LM data through the Ethernet link from FPGA 0</b></p> <p><b>Note:</b> Receiver Software by default uses port 61002</p> <p>For the UDP packages of list mode data to arrive at the intended destination, users must specify here the MAC, IP, and/or PORT of the receiver.</p>
DEST_PORT1	Number 0..65535	<p><b>IP port of the device receiving LM data through the Ethernet link from FPGA 1</b></p> <p>For the UDP packages of list mode data to arrive at the intended destination, users must specify here the MAC, IP, and/or PORT of the receiver.</p>
SOURCE_PORT0	Number 0..65535	<p><b>IP port of FPGA 0 for Ethernet data output</b></p> <p>UDP packages of list mode data require the MAC, IP, and PORT of the sender (source). In practice the actual value is often irrelevant for the package to arrive at the destination, but it can be used to identify or filter packages.</p>
SOURCE_PORT1	Number 0..65535	<p><b>IP port of FPGA 1 for Ethernet data output</b></p> <p>UDP packages of list mode data require the MAC, IP, and PORT of the sender (source). In practice the actual value is often irrelevant for the package to arrive at the destination, but it can be used to identify or filter packages.</p>
DATA_FLOW	Number, 0..5	<p><b>See section 11.6.2 for options</b></p> <p>The DATA_FLOW parameters specifies how much of the processing is handled by the FPGA (faster) and how much is handled by the Linux/C readout routine (more control). For example,</p>

Parameter Name	Units, Limits	Description
		DATA_FLOW=0 handles everything in startdaq which allows writing list mode data to SD card in pretty much any format. DATA_FLOW=4 automatically sends out list mode data as UDP packages, with no involvement of startdaq except for incrementing MCA spectra.
SLEEP_TIMEOUT	Number, 1000- 65535	FPGA Clocks for pulse processing will be disabled after 64K x 40 ns x this number to save power. (2.6s for value 1000) Set to 65535 to prevent any disabling  Slated to be discontinued. It does not save much.
USER_PACKET_DATA	Number 0..65535	A user defined number that is reported in the event header in run type 0x411.

## 6.2 Module Parameters

Parameter Name	Units Limits	Description
MCSR <sub>X</sub> _NAME_##		MCSR or Module Control and Status Register bits control pulse processing for the “module” as a whole, as opposed to control bits for each channel CCSR <sub>X</sub> _NAME_##. For historical reasons there are several such registers (X = A, B, C) and each has 16 bits (##). The NAME gives a clue as to its purpose.
MCSRA_CWGROU <sub>P</sub> _00	0 or 1	In coinc mode runs (0x503 or 0x402), save only one record per coincidence window, NYI
MCSRA_P4ERUNSTATS_01	0 or 1	Count live time etc per Pixie-4e convention  The Pixie-Net XL merges firmware from the older Pixie-4e and Pixie-16 models. One significant difference is the accounting for run statistics. Specifically, Pixie-4e considers readout dead time (i.e. FPGA memory full and pausing DAQ for readout) a reason to stop the “count time” counters while Pixie-16 does not. This affects the reported output count rates. See section 11.7 for details.
MCSRA_U_02	0 or 1	Module control bit A.2, unused
MCSRA_FPTTCL_03	0 or 1	Module control bit A.3, If 1, the front panel input is used to start the TTCL acceptance window (to validate events at time of energy filter). Use MCSRA_FP_PEDGE_07 for optional inversion of front panel input signal.
MCSRA_FP_COUNT_04	0 or 1	Module control bit A.4, If 1, enables counting edges on MMCX input as “external timestamp”. If 0, use local clock (or White Rabbit clock if available) as “external timestamp” (Note: use FP_EXTCLR to clear local counter externally)
MCSRA_FP_VETO_05	0 or 1	Module control bit A.5, If 1, enables MMCX input as global Veto
MCSRA_FP_EXTCLR_06	0 or 1	Module control bit A.6, If 1, enables MMCX input as CLR for “external clock” counter / time stamp
MCSRA_FP_PEDGE_07	0 or 1	Module control bit A.7, If 1, inverts active edge for front panel pulse counter and counter clear
MCSRA_U_08	0 or 1	Module control bit A.8, unused
MCSRA_U_09	0 or 1	Module control bit A.9, unused
MCSRA_U_10	0 or 1	Module control bit A.10, unused
MCSRA_U_11	0 or 1	Module control bit A.11, unused
MCSRA_U_12	0 or 1	Module control bit A.12, unused
MCSRA_U_13	0 or 1	Module control bit A.13, unused
MCSRA_U_14	0 or 1	Module control bit A.14, unused
MCSRA_U_15	0 or 1	Module control bit A.15, unused
MCSR <sub>B</sub> _U_00	0 or 1	Module control bit B.0, unused
MCSR <sub>B</sub> _U_01	0 or 1	Module control bit B.0, unused
MCSR <sub>B</sub> _U_02	0 or 1	Module control bit B.0, unused

Parameter Name	Units Limits	Description
MCSRBU_03	0 or 1	Module control bit B.3, unused
MCSRBU_04	0 or 1	Module control bit B.0, unused
MCSRBU_05	0 or 1	Module control bit B.0, unused
MCSRBU_06	0 or 1	Module control bit B.0, unused
MCSRBU_07	0 or 1	Module control bit B.0, unused
MCSRBU_08	0 or 1	Module control bit B.8, unused
MCSRBU_09	0 or 1	Module control bit B.9, unused
MCSRBU_10	0 or 1	Module control bit B.10, unused
MCSRBU_11	0 or 1	Module control bit B.11, unused
MCSRBU_12	0 or 1	Module control bit B.12, unused
MCSRBU_13	0 or 1	Module control bit B.13, unused
MCSRBU_14	0 or 1	Module control bit B.14, unused
MCSRBU_15	0 or 1	Module control bit B.15, unused
RUN_TYPE	<b>0x100,</b> <b>0x104,</b> <b>0x105,</b> <b>0x110,</b> <b>0x111,</b>  <b>0x301,</b>  <b>0x400,</b> <b>0x401,</b> <b>0x402,</b> <b>0x404,</b> <b>0x410,</b> <b>0x411,</b>  0x500, 0x501, 0x502, 0x503	<p><b>List mode run with waveforms, Pixie-16 style</b>            List mode run with waveforms for 10G  <b>List mode run with waveforms and PSA</b>  <b>List mode run with waveforms for 10G</b>  <b>List mode run with waveforms and PSA for 10G</b></p> <p><b>MCA only run</b></p> <p><b>List mode run with waveforms.</b>            List mode run with PSA, no waveforms (text dt2)            Coincidence list mode run            List mode run with custom headers for 10G  <b>List mode run with waveforms for 10G</b>  <b>List mode run with custom headers for 10G</b></p> <p>List mode run with waveforms (text .txt)            List mode run without waveforms (text .dat)            List mode run with PSA, no waveforms (text dt2)            Coincidence list mode run (text .dt3)</p> <p>The RUN_TYPE specifies what data is collected and how it is formatted for output in the list mode file. See section 7 for details.</p>
SYNC_AT_START	<b>0 , 1 or 2</b>	<p><b>If 0, do not reset timers at runstart</b>  <b>If 1, reset timers at runstart by write to RTC_CLR</b>  <b>If 2, reset using the Live* signal (at WR start time, if enabled)</b></p> <p>The FPGA counters providing the value for event time stamps in list mode are reset to 0 at FPGA boot time. It is often beneficial to reset these time stamp counters for every DAQ run, so that event time stamps are reported relative to run start. If SYNC_AT_START is 1, a clear command is issued by the startdaq routine before the DAQ is enabled. The Linux system time is recorded at the same time, and reported in some list mode data formats. For higher precision of the reset, SYNC_AT_START can be set to 2 which ties the clearing of the counters</p>

Parameter Name	Units Limits	Description
		to the White Rabbit time or possibly an external signal.
<b>SLOW_FILTER_RANGE</b>	<b>1..6</b>	<b>Decimation/averaging for energy filter</b> <b>See note 1</b>
<b>FAST FILTER RANGE</b>	--	unused
GROUPMODE_FIP	0..6	<p>Trigger distribution options for the TRIG_OUT output:</p> <p>mode 0: OR of local triggers</p> <p>mode 1: OR of HITs (stretched local triggers)</p> <p>mode 2: sum of HITs (reserved)</p> <p>mode 3: coinc pairs of HITs (reserved)</p> <p>mode 4: coinc pair of ( OR of 4 HITs)</p> <p>mode 5: OR of energy in range test</p> <p>mode 6: beta gated coin pairs (reserved)</p> <p>The GROUPMODE parameters configure the trigger distribution between channels and FPGAs. See section 11.8 for details. It applies to both pulse processing FPGAs</p>
VETOMODE	0..5	<p>mode 0: Front panel (see MCSRA bits for inversion and other options)</p> <p>mode 1: Veto from controller</p> <p>mode 2: stretched trigger from ch. 6 AND 7</p> <p>mode 3: stretched trigger from ch. 6 AND 7, inverted</p> <p>mode 4: stretched trigger from ch. 6 OR 7</p> <p>mode 5: stretched trigger from ch. 6 OR 7, inverted</p> <p>The VETOMODE parameter configures the gating options. See section 11.911.8 for details.</p>
GATE_TIME	8ns clock ticks 1..255	<p>The time channels 6/7 can gate (or allow) other channels</p> <p>The GATE_TIME parameter configures advanced gating options. See section 11.911.8 for details.</p>
DYNODE_WINDOW	8ns clock ticks 1..255	reserved
TTCL_APPR_WINDOW	0..65535	<p>Length of acceptance window started by TTCL accept pulse. 0 = infinite (always accept)</p> <p>In addition to gating pulses at the time they are detected, it is possible to require validation a few microseconds later before the event is recorded into FPGA memory. This allows external processes to make an accept/reject decision within the energy filter time. This parameter sets the length of the acceptance window after an external pulse, usually via the MMCS front panel connector. It has to be timed to extend to the end of the energy filter. See section 11 for details.</p>
MOD_U1		Unused
MOD_U0		Unused

## 6.3 Channel Parameters

Parameter Name	Units Limits	Description
<b>CCSRA_FTRIGSEL_00</b>	<b>0 or 1</b>	<b>Channel control bit A.0, If 0, use local trigger If 1, use distributed trigger</b>  Each channel can use its local trigger (from detecting a rising edge) or a distributed trigger from other channels (or external sources). See section 11.8 for details
CCSRA_EXTTRIGSEL_01	0 or 1	unused
<b>CCSRA_GOOD_02</b>	<b>0 or 1</b>	<b>Channel control bit A.2, if 0, channel will not be processed and not shown in ADC traces</b>  This control bit can be used to disable “bad” channels that are not connected to detectors, or otherwise should be ignored.
CCSRA_CHANTRIGSEL_03	0 or 1	unused
CCSRA_SYNCDATAACQ_04	0 or 1	Channel control bit A.4, if 0, channel count time is paused by local conditions only (memory full) If 1, channel count time is stopped if other channels are paused  If all channels should be taking data synchronously, it is important that they are actually ready to do so at the same time. This control bit can be used to enforce that if one channel is paused (e.g. due to higher count rate or a recent burst in pulses), all other channels are also paused. This is usually turned on for channels using distributed triggers.
<b>CCSRA_INVERT_05</b>	<b>0 or 1</b>	<b>Channel control bit A.5, if 1, ADC data is inverted before processing (for falling edge pulses)</b>  The pulse processing logic expects signals with a fast rising edge and slower exponential decay. If the detector output is the opposite polarity, this bit can be used to invert the digitized signal before processing in the FPGA.
CCSRA_VETOENA_06	0 or 1	Channel control bit A.6, if 0, ignore external veto if 1, reject events based on Veto  The VETO signal is used to block events. Typically a front panel input is used, but see section 11.9 for other options. VETO

Parameter Name	Units Limits	Description
		is considered at the time of trigger (rising edge of the pulse)
CCSRA_HISTOE_07	0 or 1	unused
CCSRA_TRACEENA_08	<b>0 or 1</b>	<b>Channel control bit A.8</b> <b>If 1, record waveforms in list mode runs</b>  This bit can be used to disable waveform capture in list mode acquisitions. It is equivalent to setting TRACE_LENGTH to zero. This will significantly increase maximum throughput (less data to read out) in those applications that can make use of the results computed online and reported in the list mode headers.
CCSRA_QDCENA_09	0 or 1	Channel control bit A.9 If 1, record QDC sums in list mode runs If 0, record PSA sums in list mode runs  The Pixie-Net XL firmware can process incoming pulses to capture the sum over small areas of the pulse. Two algorithms are implemented: 1) capturing 8 “QDC” sums of up to 32766 samples; 2) capturing 2 “PSA” sums of up to 60 samples with better precision at the rising edge. This control bit lets the user choose between the two options. See section 12.2 and 12.6 for details.
CCSRA_CFDMODE_10	0 or 1	Channel control bit A.10 If 1, record CFD results in list mode runs  This control bit turns on constant fraction timing for this channel. See section 12.3 for details.
CCSRA_GLOBTRIG_11	0 or 1	unused
CCSRA_ESUMSENA_12	0 or 1	unused
CCSRA_CHANTRIG_13	0 or 1	unused
CCSRA_ENARELAY_14	0 or 1	unused
CCSRA_PILEUPCTRL_15	<b>0 or 1</b>	<b>Channel control bit A.15</b> <b>If 1, reject pulses that are flagged as piled up, else record everything</b>  Pulses that follow too closely for the energy filter to accumulate the required number of samples are considered piled up. The energy computation will still be executed, but the result will not accurately represent the height of the first pulse. In most cases, such event should be rejected (not histogrammed or recorded) and the bit set to 1. In some cases, it may

Parameter Name	Units Limits	Description
		be of interest to see these pulses also and the bit can be set to 0.
CCSRB_U_00	0 or 1	Channel control bit B.0,unused
CCSRB_U_01	0 or 1	Channel control bit B.1,unused
CCSRB_U_02	0 or 1	Channel control bit B.2,unused
CCSRB_U_03	0 or 1	Channel control bit B.3,unused
CCSRB_U_04	0 or 1	Channel control bit B.4,unused
CCSRB_U_05	0 or 1	Channel control bit B.5,unused
CCSRB_U_06	0 or 1	Channel control bit B.6,unused
CCSRB_U_07	0 or 1	Channel control bit B.7,unused
CCSRB_U_08	0 or 1	Channel control bit B.8,unused
CCSRB_U_09	0 or 1	Channel control bit B.9,unused
CCSRB_U_10	0 or 1	Channel control bit B.10,unused
CCSRB_U_11	0 or 1	Channel control bit B.11,unused
CCSRB_U_12	0 or 1	Channel control bit B.12,unused
CCSRB_U_13	0 or 1	Channel control bit B.13,unused
CCSRB_U_14	0 or 1	Channel control bit B.14,unused
CCSRB_U_15	0 or 1	Channel control bit B.15,unused
CCSRC_INVERSEPILEUP_00	0 or 1	<p>Channel control bit C.0, if 1, capture piled up events</p> <ul style="list-style-type: none"> <li>- if PILEUPCTRL=0: header only for non-piled up events; header + waveforms for piled up events, NYI</li> <li>- if PILEUPCTRL=1: do not capture non-piled up events; capture header + waveforms for piled up events, NYI</li> <li>- always: do not capture non-piled up events; capture header + waveforms for piled up events</li> </ul> <p>Current implementation: If this bit is set, CCSRA_PILEUPCTRL_15 is ignored and <u>only</u> piled up events are recorded</p>
CCSRC_ENAENERGYCUT_01	0 or 1	Channel control bit C.1, reserved for advanced P16 functions
CCSRC_GROUPTRIGSEL_02	0 or 1	Channel control bit C.2, reserved for advanced P16 functions
CCSRC_CHANVETOSEL_03	0 or 1	unused
CCSRC_MODVETOSEL_04	0 or 1	unused
CCSRC_EXTTSENA_05	0 or 1	unused
CCSRC_RBADDIS_06	0 or 1	<p>Channel control bit C.6. Ignore out-of-range test</p> <p>If the input signal goes out of range while the energy filter accumulates the required number of samples, the event is flagged as out-of-range and not recorded. If this bit is set, the event is still recorded and the out-of-range flag in the list mode header indicates the status.</p>
CCSRC_PAUSEPILEUP_07	0 or 1	Channel control bit C.7. Pause pileup inspection for a short time after trigger

Parameter Name	Units Limits	Description
		Pulses with overshoots or non-monotonic rising edges can potentially trigger more than once per rising edge. If this bit is set, triggers are disabled for a short time after a first trigger to prevent such double triggering and the resulting pileup rejection.
CCSRC_CHECK_ELIMITS_08	0 or 1	<p>Channel control bit C.8. When 1 and E in user defined range, issue logic pulse on MMCX output of length VETO_STRETCH</p> <p>A custom function in the firmware checks if the computed pulse height (E) is in a user defined range. If this bit is set, a logic signal of user defined length is issued on a physical output. This allows the use of the Pixie-Net XL as a pulse height discriminator for other electronics.</p>
CCSRC_LOCAL_ENERGY_09	0 or 1	<p>Channel control bit C.9. Use local trigger for E filter capture even in group trigger mode</p> <p>When using a distributed trigger for event capture, it is often desirable to base the pulse height computation on a locally detected rising edge, to properly capture the pulse in the energy filter sums. If this bit is not set, the distributed trigger sets the starting point of the energy filter sums, which can be some time before or after the actual pulse in this channel.</p>
CCSRC_U_10	0 or 1	Channel control bit C.10, Reserved for test purposes
CCSRC_GATEADC_11	0 or 1	<p>Channel control bit C.11. If 1, lower 5 bits of ADC traces show triggers and gating information. See below</p> <p>To assist with setting up delays and windows for gating and coincidence tests, the lower 5 bits of untriggered ADC traces can be replaced with gate and trigger signals seen by the FPGA. Captured list mode traces are not affected.</p>
CCSRC_U_12	0 or 1	Channel control bit C.12, unused
CCSRC_U_13	0 or 1	Channel control bit C.13, unused
CCSRC_U_14	0 or 1	Channel control bit C.14, unused
CCSRC_U_15	0 or 1	Channel control bit C.15, unused
ANALOG_GAIN	1.0-22.4	<b>Gain with switches/relays/VGAs</b> <b>Values vary for different HW variants.</b>

Parameter Name	Units Limits	Description
		The analog front end of the Pixie-Net is implemented in removable ADC daughterboards. Some of these daughterboards have fixed gain, others have multiple gain options. This parameter sets the gain value (amplification of the signal) and so sets the dynamic range of pulses that the AD can process. See section 1.2.1 for valid values.
DIG_GAIN	Positive float	<p><b>Digital gain adjustment factor.</b> The filtered pulse amplitude is multiplied with this factor before histogramming and reporting in list mode file.</p> <p>As a pure computational factor, this parameter can be used to fine tune the overall gain, for example to align peaks from different channels in the MCA histogram. It does not change the pulse amplitude in the wave forms. Binning artefacts can appear with very small or very large numbers. Pulses larger than the ADC input range will not be brought in range by a digital gain &lt;1.0.</p>
VOFFSET	V -1.25..1.25	<p><b>Analog offset</b></p> <p>Used to compensate detector baseline offsets. Mapped to true offset range depending on HW variant (e.g. +/- 3V for DB01)</p>
ENERGY_RISETIME	μs, 0.032 .. 62.976	<p><b>Energy filter rise time</b></p> <p>See Note 1) and section 3.4.1</p>
ENERGY_FLATTOP	μs, 0.048 .. 62.976	<p><b>Energy filter flat top</b></p> <p>See Note 1) and section 3.4.1</p>
TRIGGER_RISETIME	μs, 0.016 .. 1.016	<p><b>Trigger filter rise time</b></p> <p>See note 2) and section 3.4.2</p>
TRIGGER_FLATTOP	μs, 0.000 .. 1.000	<p><b>Trigger filter flat top</b></p> <p>See note 2) and section 3.4.2</p>
TRIGGER_THRESHOLD	(ADC steps) 0.0..4096.0	<p><b>Trigger threshold</b></p> <p>See note 3) and section 3.4.2</p>
THRESH_WIDTH		unused
TRACE_LENGTH	μs, 0..16	<p><b>Captured waveform length</b></p> <p><b>Must be multiple of 32 samples, max 4096 samples.</b></p> <p>The length of waveforms in list mode data acquisitions is independent of the energy filter length. However, if offline</p>

Parameter Name	Units Limits	Description
		<p>reconstruction of the energy filter is desired, the trace length should be 2x (ENERGY_RISETIME+ENERGY_FLATTOP) with a TRACE_DELAY of 2x ENERGY_RISETIME+ENERGY_FLATTOP to record sufficient samples for the energy filter sums.</p> <p>In UDP data acquisitions, the maximum IPV4 packet size of ~1500 bytes limits the TRACE_LENGTH to 672 samples (~2.6us for 250 MHz ADC variants). Run type 0x410 can accommodate longer traces by breaking the event up into 8 packages. TRACE_LENGTH must then be a multiple of 8x32 samples. The UDP receiver utilities provided by XIA identifies such multi-packet events and assembles them without repeating the duplicate header words.</p>
TRACE_DELAY	μs, 0..16	<p><b>Pre-trigger delay</b></p> <p>This parameter specifies the length of the waveform before the trigger point.</p>
BINFACTOR	1-16	<p>MCA binning factor: divide measured pulse height by <math>2^N</math> before binning</p> <p>In applications with limited resolution or low count rates, the full 32K bins of a the MCA spectrum may slice the energy too fine to be practical. This parameter allows combining counts into fewer bins.</p> <p>Since the energy is computed as a 16 bit value (0—64K) and the MCA memory has only 32K bins per channel, the minimum binfactor is 1.</p>
INTEGRATOR	0..2	<p>Filter mode:            0-trapezoidal,            1-gap sum integral, NYI            2-ignore gap sum, NYI</p>
BLCUT	Positive integer	<p>Threshold for bad baseline measurements            See section 3.4.4</p>
BASELINE_PERCENT	0-99	<p>Target offset for baseline, nominally in percent</p> <p>The findsettings routine attempts to place the DC offset of the signal at this fraction of the full ADC range. For example, for a value of 10 and a 14 bit ADC (0-16K), the baseline would be adjusted to ~1600 ADC steps.</p>

Parameter Name	Units Limits	Description
TAU	µs positive float	Preamplifier decay time See section 3.4.3
XDT	µs	Sampling interval in untriggered traces.  Untriggered “ADC traces” shown in the ADC page or reported in the file ADC.csv use this parameter as the time between ADC samples.
BLAVG	65535.. 65528 and 0	Baseline averaging See section 3.4.4
MULTIPLICITY MASKL		unused
MULTIPLICITY MASKM		unused
MULTIPLICITY MASKH		unused
MULTIPLICITY MASKX		unused
FASTTRIG_BACKLEN		Effectively ½ of the coincidence window. See below
CFD_THRESHOLD	1..65535	CFD parameters. See section 12.3
CFD_DELAY	8ns cycles 1..63	CFD parameters. See section 12.3
CFD_SCALE	0,2..7	CFD parameters. See section 12.3 0 – scale = 1.000 1 – unsupported scale value 2 – scale = 0.750 3 – scale = 0.625 4 – scale = 0.500 5 – scale = 0.375 6 – scale = 0.250 7 – scale = 0.125
EXTTRIG_STRETCH		unused
VETO_STRETCH	8ns cycles	Length of logic output for E within ELO and EHI  See also CCSRC_CHECK_ELIMITS_08
CHANTRIG_STRETCH		unused
EXTERN_DELAYLEN	8ns cycles 0..511	Input delay for ADC data stream (equivalent to a cable delay)  The incoming, digitized ADC data stream is delayed by this many clock cycles. This can be used to compensate different cable lengths between detectors and the Pixie-Net XL inputs or to delay the signal to allow external processes to complete.
FTRIGOUT_DELAY		unused
QDCLen0-7	8ns cycles 2..32766  OR	QDC sum length for Pixie-16 compatible QDC analysis (8 consecutive sums starting from beginning of captured waveform, or equivalent if not captured).

Parameter Name	Units Limits	Description
	samples 2..60	If CCSRA_QDCENA_09 = 0, some of these parameters are used for Pixie-4e compatible pulse shape analysis (see section 12.2): QDCLen# = Length of PSA sum# [2..60] (PSA sums 2,3,... only acquired in runtype 0x404)
QDCDel0-7	samples -2 .. 500	Only used if CCSRA_QDCENA_09 = 0, for Pixie-4e compatible pulse shape analysis (see section 12.2): QDCDel# = Delay of PSA sum # relative to trigger point [-2..500] (PSA sums 2,3,... only acquired in runtype 0x404, with delays -60 .. 440)
QDC_DIV	4 or 32	Only used if CCSRA_QDCENA_09 = 0, for Pixie-4e compatible pulse shape analysis (see section 12.2): Divide all PSA sums by this factor
PSA_THRESHOLD	ADC steps 1.. 65530	PSA trigger threshold  If a new sample from the ADC is PSA_THRESHOLD above the running baseline average (taken from 4 samples a few clock cycles before), the PSA sum accumulation is started. Practical limit is maximum ADC value minus signal baseline offset
Emin	0.. 65535	Suppress records in list mode for energies smaller than this number Only applies to DATA_FLOW <3
ELO	0.. 65535	When E between ELO and EHI and enabled, issue logic pulse on MMCX output of length VETO_STRETCH  See also CCSRC_CHECK_ELIMITS_08
EHI	0.. 65535	When E between ELO and EHI and enabled, issue logic pulse on MMCX output of length VETO_STRETCH  See also CCSRC_CHECK_ELIMITS_08
GROUPMODE_CH	0..6	group trigger options: 0 OR of local triggers 1 trigger from system (res.) 2 external trigger from front panel 3 coinc. of 2 channel pairs 4 reserved 5 reserved 6 beta gated OR of dynode triggers  This parameter selects the source of the distributed trigger if CCSRA_FTRIGSEL_00 is set to 1. See section 11.8 for details.

Parameter Name	Units Limits	Description
CHAN_U0		Unused

## Notes

1. Energy filter rise time and flat top depend on FILTER\_RANGE (FR). Higher filter ranges have allow longer filter times but with increased coarseness. Maximum combined length is  $126 \times 0.008\mu s * 2^F R$

FILTER_RANGE	Filter granularity	max. T <sub>rise</sub> +T <sub>flat</sub>	min. T <sub>rise</sub>	min. T <sub>flat</sub>
1	0.016μs	2.032μs	0.032μs	0.048μs
2	0.032μs	4.064μs	0.064μs	0.096μs
3	0.064μs	8.128μs	0.128μs	0.192μs
4	0.128μs	16.256μs	0.256μs	0.384μs
5	0.256μs	32.512μs	0.512μs	0.768μs
6	0.512μs	65.024μs	1.024μs	1.536μs

Table 6-1: Filter clock decimations and filter time granularity

2. Trigger filter rise time and flat top maximum combined length is  $127 \times 0.008\mu s$
3. The internal triggering compares the output of the trigger filter (technically an area) with  $(TRIGGER\_THRESHOLD * TRIGGER\_RISETIME [in ns] / 8)$ , which must be <65536. The maximum for TRIGGER\_THRESHOLD thus depends on TRIGGER\_RISETIME. Fractional values for TRIGGER\_THRESHOLD are acceptable. TRIGGER\_THRESHOLD=0 turns off triggering for this channel.

# 7 Data Formats

## 7.1 List Mode Data Files

There are several types of list mode data acquisition, with waveforms and without, binary or text, and for special purposes. The following table gives an overview of their differences. The DATA\_FLOW parameter controls if the data is recorded on local SD card or sent as UDP packages via the FPGA Ethernet interface, and also the involvement of the Linux DAQ routine in that process.

Fully supported run types are listed in the table below. All run types can collect waveforms (See below for legacy output options that can be reactivated upon request). The most common are shown in bold.

Run Type	Function	File extension	DATA_FLOW Options	Notes
<b>0x100</b>	startdaq	.bin	0,1,2 to SD card 3,4 to UDP (1G only)	<b>General purpose, compatible with Pixie-16</b>
0x105	startdaq	.bin	0,1,2 to SD card 3,4 to UDP (1G only)	Provides raw PSA information not available in 0x100 via UDP
<b>0x110</b>	startdaq, udpena	.bin	<b>3,4 to UDP (10G only)</b>	<b>General purpose, 3 extra header words compared to 0x100 due to 10G.</b>
0x111	startdaq, udpena	.bin	3,4 to UDP (10G only)	Provides raw PSA information not available in 0x110 via UDP
<b>0x400</b>	startdaq	.b00	0,1,2 to SD card	<b>General purpose, compatible with Pixie-4e</b>
0x401	Startdaq	.dt3	0,1,2 to SD card	General purpose, compatible with Pixie-4e (not compatible with Pixie-Net 0x503)
<b>0x410</b>	startdaq, udpena	.b00	<b>3,4 to UDP (10G only)</b>	<b>General purpose, similar to Pixie-4e</b>
0x411	startdaq, udpena	.b00	3,4 to UDP (10G only)	General purpose, ANL compatible

Notes:

- As a general rule, times and time stamps are in units of 1 ns unless explicitly stated otherwise. Note that time counters are incremented at a rate of 125 MHz, i.e. by 8 every 8 ns.
- Traces longer than 704 samples (e.g. 2.8μs for 250 MHz variants) can currently only be recorded to SD card (DATA\_FLOW = 0,1,2) or in run type 0x410 via UDP (DATA\_FLOW = 4 only)

- It is quite straightforward for users to modify the code in e.g. startdaq to generate application specific output files on the local SD card. For custom formats in UDP data acquisitions, the UDP receiver program can be modified in the same manner.

### 7.1.1 Pixie-16 style binary list mode files (Run Type 0x100, 0x105)

*DATA\_FLOW 0,1,2 to SD card or 3,4 to UDP receiver (1G only)*

The Pixie-Net XL supports generation of Pixie-16 style list mode files. Run type 0x100 is identical to the one generated by the Pixie-16. However, event header words 0-7, 16-17 are always reported and -- if QDC is enabled -- additionally words 8-15. Options to suppress energy sums or external time stamps are ignored.

In run type 0x105, the energy sums are replaced with PSA and CFD results. Run types 0x100 and 0x105 are only supported for recoding to SD card (or via 1G Ethernet), not 10G Ethernet.

#### 7.1.1.1 Event Header

The list mode event header always starts with a 4-word (32-bit) block, which includes the event identification information, timestamp, energy and trace length, etc. Following this 4-word block are the optional data blocks whose recording is selectable by the user. The placement of these data blocks in the event header always follows the order as described below. If the recording of a specific data block is not enabled by the user, all subsequent data blocks would simply shift upward by the length of the disabled data block.

In word 0, Chan# is the channel number. SlotID is the PXI slot number. CrateID is the PXI crate number. Header Length specifies the length of this event header. Event length specifies the length of this event, which could include both event header and trace. If Event Length equals to Header Length, Trace Length should be zero and no trace is recorded for this event. Finish code indicates whether this event is a piled up event (Finish Code = 1) or a single event (Finish Code = 0).

In word 1, EVTTIME\_LO is the lower 32-bit of the 48-bit event timestamp recorded by the signal processing FPGA. This 48-bit event timestamp is latched by either the leading edge channel fast trigger or CFD trigger.

In word 2, EVTTIME\_HI is the upper 16-bit of the 48-bit event timestamp. The result of the CFD computation (if enabled) varies slightly for the different ADC rates and precisions, see below.

In word 3, Event Energy is the 16-bit energy of the event computed by the Pixie-Net XL. The Trace Length is the length of the trace recorded for the event (in samples). If no trace is recorded, Trace Length will be 0. The trace out of range flag is an indicator showing whether or not the trace is out of the ADC range when the event is recorded. This could happen if the event pulse's amplitude is too big, causing the trace to be clipped over the upper limit of the ADC.

The 4-word fixed event header is followed by an optional 4-word block for raw energy sums and baselines (for debugging), an optional block of 8 QDC sums, and an optional block of 2 words for external time stamps.

Index	Data (32bit words)						Description			
0	[31]	[30:17]	[16:12]	[11:8]	[7:4]	[3:0]	Fixed 4-word event header, word #0			
	Finish Code (1 if piled up)	Event Length (32bit words)	Header Length (32bit words)	CratID	SlotID	Chan#				
1	[31:0]						Fixed 4-word event header, word #1			
	EVTTIME_LO[31:0]									
2	[31:16]			[15:0]			Fixed 4-word event header, word #2			
	CFD Result [15:0]			EVTTIME_HI[15:0]			See below for variant information			
3	[31]	[30:16]	[15:0]			Fixed 4-word event header, word #3				
	Trace Out-of-Range Flag	Trace Length (in samples or 16bit words)	Event Energy							
4	Energy sum – trailing [31:0]					Trailing energy sum				
5	Energy sum – leading [31:0]					Leading energy sum				
6	Energy sum – gap [31:0]					Gap energy sum				
7	Baseline [31:0]					Baseline value (NOT floating point)				
8	QDCSum0 [31:0]					QDC sum #0				
9	QDCSum1 [31:0]					QDC sum #1				
10	QDCSum2 [31:0]					QDC sum #2				
11	QDCSum3 [31:0]					QDC sum #3				
12	QDCSum4 [31:0]					QDC sum #4				
13	QDCSum5 [31:0]					QDC sum #5				
14	QDCSum6 [31:0]					QDC sum #6				
15	QDCSum7 [31:0]					QDC sum #7				
16	Ext_TS_Lo [31:0]					Lower 32-bit of 48-bit external clock timestamp (if WR: cycle count x4, i.e. units of 1ns)				
17	[31:16] not used, 0's	Ext_TS_Hi [15:0]				Upper 16-bit of 48-bit external clock timestamp (if WR: second count)				

Table 7-1: Pixie-16 list mode event header data structure (per Pixie-16 manual in 32 bit words).

Word #	Variable	Description
0	IDs	00-03 Channel ID 04-07 Slot ID 08-11 Crate ID 12-15 Header Length in 32bit words, low 4 bits
1	Lengths	00-00 Header Length in 32bit words, highest bit 01-14 Event Length in 32bit words 15-15 Finish Code (1 if piled up)
2	EVTTIME	EVTTIME LO[15:0] (units: 8ns)
3	EVTTIME	EVTTIME LO[31:16] (units: 8ns)
4	EVTTIME	EVTTIME HI[15:0] (units: 8ns)
5	CFD	CFD Result [15:0] invalid for DATA_FLOW = 4
6	Energy	Event Energy
7	Trace	00-14 Trace Length (in samples or 16bit words) 15-15 Trace Out-of-Range Flag
8	0x100 Esum 0x105 PSA	Energy sum – trailing [15:00] Raw PSA sum: base
9	0x100 Esum 0x105 PSA	Energy sum – trailing [31:16] Raw PSA sum: max
10	0x100 Esum 0x105 PSA	Energy sum – leading [15:00] Raw PSA sum: sum 0
11	0x100 Esum 0x105 PSA	Energy sum – leading [31:16] Raw PSA sum: sum 1
12	0x100 Esum 0x105 CFD	Energy sum – gap [15:00] Raw CFD info
13	0x100 Esum 0x105 CFD	Energy sum – gap [31:16] Raw CFD sum 1 low
14	0x100 BLavg 0x105 CFD	Baseline [15:00] Raw CFD sum 2 low, sum 1 high
15	0x100 BLavg 0x105 CFD	Baseline [31:16] Raw CFD sum 2 high
16	Ext_TS	Ext_TS_Lo [15:00]
17	Ext_TS	Ext_TS_Mi [31:16]
18	Ext_TS	Ext_TS_Hi [15:00]
19	reserved	
20+	ADC data	ADC Data [15:0]

Table 7-2: Pixie-16 style list mode event header data structure in 16 bit words.  
Words 8-15 vary for run type 0x100 and 0x105. 8 QDC sums may be inserted after word 15.

### 7.1.1.2 Waveforms

If trace recording is enabled, trace data will immediately follow the last word of the event header. Since raw ADC data points can be 16-bit, 14-bit or 12-bit numbers depending on the ADC variant that is used, two ADC data numbers are packed into one 32-bit word to improve the storage efficiency, as shown below. Since the event header could have variable length (4 to 18 words) depending on the selection of various output data options, the header length, event length and trace length that are recorded in the first 4 words of the event header should be used to parse each event data block and navigate through the output data file. For Pixie-Net XL variants with 12-bit ADC's, the upper 4 bits of each 16-bit ADC

data number are filled with 0's. For Pixie-Net XL variants with 14-bit ADC's, the upper 2 bits of each 16-bit ADC data number are filled with 0's.

Index	Data		Description
n	Last word of event header [31:0]		Last word of event header. The event header could be 4, 6, 8, 10, 12, 14, 16 or 18 words long
n+1	ADC Data #1 [15:0]	ADC Data #0 [15:0]	Packing of ADC Data #0 and #1
n+2	ADC Data #3 [15:0]	ADC Data #2 [15:0]	Packing of ADC Data #2 and #3
...	...	...	

Table 7-3: Pixie-16 list mode trace data structure.

#### 7.1.1.3 CFD results

Variant	Data			
<b>100, 125 MHz</b>	[31]	[30:16]		[15:0]
	CFD forced trigger bit	CFD Fractional Time[14:0] $\times 32768$		EVTTIME_HI[15:0]
<b>250 MHz</b>	[31]	[30]	[29:16]	[15:0]
	CFD forced trigger bit	CFD trigger source bit	CFD Fractional Time[13:0] $\times 16384$	EVTTIME_HI[15:0]
<b>500 MHz</b>	[31:29]		[28:16]	[15:0]
	CFD trigger source bits [2:0]		CFD Fractional Time[12:0] $\times 8191$	EVTTIME_HI[15:0]

Table 7-4: Word 2 of Event Header for different Pixie-16 variants

The CFD fractional time is the CFD trigger time computed by the Pixie-Net XL's ARM processor. It is multiplied by a power of two before being stored in the event header to report the fraction as an integer number. The power of 2 varies for the ADC digitization rate. The CFD fractional time will be 0 if CFD trigger is not enabled. For DATA\_FLOW = 3 this value is inserted into the FPGA data before sending it out as a UDP package. For DTA\_FLOW = 4, the ARM processor does not participate in the data processing and the result is invalid.

To get the actual CFD fractional time, the CFD Fractional Time [N:0] stored in the event header should be first divided by that power of two. The CFD forced trigger bit indicates whether the CFD trigger is forced (1) or not forced (0). A forced CFD trigger is needed to avoid that the Pixie-Net XL channel gets stuck in an infinite loop if the CFD threshold is set too high or no zero crossing is encountered after 32 clock cycles have passed after the local fast trigger point. If the CFD forced trigger bit is 1, the CFD fractional time will be zero.

As explained elsewhere, in 250 MHz Pixie-Net XL modules, the ADCs run at the full 250 MHz speed, but the FPGA logic runs at 125 MHz in the Signal Processing FPGA. That means at every 8 ns interval, two 4 ns ADC samples are processed in parallel.

However, the CFD logic in the 250 (or 500) MHz Pixie-Net XL still looks for the CFD zero crossing point at every 4 (or 2 ns) ADC sample. Therefore, when using the 8 ns clock to capture the CFD zero crossing point location, there are two possibilities of the CFD zero crossing: it could either occur at the first 4 ns of the 8 ns interval (odd cycle) or at the second 4 ns of the 8 ns interval (even cycle). The Pixie-Net XL outputs such “within- cycle” information using the CFD trigger source bits in the event header. Offline analysis programs can then know when the CFD zero crossing occurred and the exact CFD fractional time can be known (see section 12 or the Pixie-16 manual for details).

The raw CFD data for run type 0x105 can be used as follows to compute the CFD result shown in table 7-9:

```
cfdsrc      = (word_12>>1) & 0x0001
cfdfrc      = (word_12>>2) & 0x0001
cf dout1    = word_13 + (word_14 & 0x00FF)<<16
cf dout2    = (word_14 & 0xFF00)>>8 + word_15<<8
cf dout2    = 0x1000000 - cf dout2;           // convert to positive
ph = (double)cf dout1 / ( (double)cf dout1 + (double)cf dout2 )
```

To format the result as in the 0x100 list mode format, compute

```
if(250 MHz)
    cfd = (int)floor(ph*16384)
    cfd = (cf d&0x3FFF)
    cfd = cfd + (cf dsr c<<14)
    cfd = cfd + (cf dfrc<<15)
elseif(125 MHz)
    cfd = (int)floor(ph*32768)
    cfd = (cf d&0x7FFF)
    cfd = cfd + (cf dfrc<<15)
```

To format the result in ns, compute

```
cf d = ph
if(cf dsr c) cf d = cf d - 16
if(cf dfrc) cf d = 0
cf d = cf d * WFscale
```

where WFscale is 4 or 8 ns for 250 or 125 MHz sampling, respectively.

---

<sup>6</sup> In some cases it may have to be +1.

### 7.1.2 Modified Pixie-16 style binary list mode files (Run Type 0x110, 0x111)

*DATA\_FLOW 3,4 to UDP receiver (10G only)*

Run types 0x110 and 0x111 are designed exclusively to stream data out as UDP packages via the 10G interface. Use of the 10G interface requires a specific number of words in a package. Therefore, in Run Type 0x110 and 0x111, 3 header words are added compared to Run Types 0x100 and 0x105, respectively. All other definitions are unchanged. Note that the Header Length and Event Length do not include the 3 extra words

Word #	Variable	Description
0	EventHeadLen	Event Header length in 16 bit words (23)
1	ModuleType	A number identifying the Pixie module type, including ADC bits and rate.
2	RunType	A number identifying the data format
3	IDs	00-03 Channel ID 04-07 Slot ID 08-11 Crate ID 12-15 Header Length in 32bit words, low 4 bits
4	Lengths	00-00 Header Length in 32bit words, highest bit 01-14 Event Length in 32bit words 15-15 Finish Code (1 if piled up) <u>Header Length and Event Length do not count the first 3 words.</u> It is recommended to use _EventHeadLen+ Trace Length instead
5	EVTTIME	EVTTIME LO[15:0] (units: 8ns)
6	EVTTIME	EVTTIME LO[31:16] (units: 8ns)
7	EVTTIME	EVTTIME HI[15:0] (units: 8ns)
8	CFD	CFD Result [15:0] Invalid for DATA_FLOW = 4
9	Energy	Event Energy
10	Trace	00-14 Trace Length (in samples or 16bit words) 15-15 Trace Out-of-Range Flag
11	0x110 Esum 0x111 PSA	Energy sum – trailing [15:00] Raw PSA sum: max
12	0x110 Esum 0x111 PSA	Energy sum – trailing [31:16] Raw PSA sum: base
13	0x110 Esum 0x111 PSA	Energy sum – leading [15:00] Raw PSA sum: sum 0
14	0x110 Esum 0x111 PSA	Energy sum – leading [31:16] Raw PSA sum: sum 1
15	0x110 Esum 0x111 CFD	Energy sum – gap [15:00] Raw CFD info
16	0x110 Esum 0x111 CFD	Energy sum – gap [31:16] Raw CFD sum 1 low
17	0x110 BLavg 0x111 CFD	Baseline [15:00] Raw CFD sum 2 low, sum 1 high
18	0x110 BLavg 0x111 CFD	Baseline [31:16] Raw CFD sum 2 high
19	Ext TS	Ext TS Lo [15:00]
20	Ext TS	Ext TS Lo [31:16]
21	Ext TS	Ext TS Hi [15:00]
22	reserved	
23+	ADC data	ADC Data [15:0]

Table 7-5: list mode event header data structure in 16 bit words for Run Types 0x110 and 0x111.

Words 8-15 vary for run type 0x110 and 0x111. QDC sums are not supported

### 7.1.3 Pixie-4e style binary list mode files (Run Type 0x400)

*DATA\_FLOW 0,1,2 to SD card*

Run type 0x400 creates single-channel event records in binary format<sup>7</sup>, **LMdata.b00**. This format is compatible with the Pixie-4e. The file starts with a file header of 32 words. The 32 words (16 bit unsigned integer, low byte first) are:

Word #	Variable	Description
0	BlkSize	Block size (16-bit words)
1	ModNum	Module number
2	RunFormat	Format descriptor = Run Type
3	ChanHeadLen	Channel Header Length
4	CoincPat	Coincidence pattern
5	CoincWin	Coincidence window in 8ns clock ticks
6	MaxCombEventLen	Maximum length of traces plus headers from all 4 channels (in blocks)
7	BoardVersion	Module type and revision
8	EventLength0	Length of traces from channel 0 plus header (in blocks)
9	EventLength1	Length of traces from channel 1 (in blocks)
10	EventLength2	Length of traces from channel 2 (in blocks)
11	EventLength3	Length of traces from channel 3 (in blocks)
12	SerialNumber	Serial number of that module
13–31	unused	Reserved

Table 7-6: File header data format, total 32 words (16bit).

Following the file header, the single channel event records are stored in sequential order. Each event starts out with a channel header of 32 words. The 32 words (16 bit) are:

Word #	Variable	Description
0	EvtPattern	Hit pattern.
1	EvtInfo	Event status flags.
2	NumTraceBlks	Number of blocks of Trace data to follow the header
3	NumTraceBlksPrev	Number of blocks of Trace data in previous record (for parsing back)
4	TrigTimeLO	Trigger time, low word (in units of 1 ns)
5	TrigTimeMI	Trigger time, middle word (in units of 2^16 x 1 ns)
6	TrigTimeHI	Trigger time, high word (in units of 2^32 x 1 ns)
7	TrigTimeX	Trigger time, extra 8 bits (in units of 2^48 x 1 ns)
8	Energy	Pulse Height
9	ChanNo	Channel number
10	PSA Values	Amplitude A
11	PSA Values	reserved
12	PSA Values	Base B
13	PSA Values	QDC0
14	PSA Values	QDC1
15	PSA Values	Ratio R *1000
16–31	reserved	

Table 7-7: Event (channel) header for Run Type 0x400, total 32 words (16bit).

<sup>7</sup> XIA in the past made available a utility to convert this data format to the IEC63047 list mode standard. If of interest, please contact XIA for current availability.

The hit pattern is a bit mask, which tells which channels were recorded detected within the specified coincidence window plus some additional status information, as listed in table 7.1. The channel header may be followed by waveform data. An offline analysis program can recognize this by reading the number of waveform blocks from the NumTraceBlks word. The block size is defined in the file header.

Bit #	Description
<i>EvtPattern</i>	
0..3	If set, indicates that data for channel 0..3 have been recorded <sup>8</sup>
4..7	4: Logic level of FRONT panel input 5: Result of LOCAL acceptance test 6: Logic level of backplane STATUS line, 7: Logic level of backplane TOKEN line (= result of global coincidence test), see section 7
8..11	If set, indicates that channel 0..3 has been hit in this event <sup>8</sup> (i.e. if zero, energy reported is invalid or only an estimate)
12..15	If set, indicates that the GATE input of channel 0..3 has been high at time of fast trigger
<i>EvtInfo</i>	
0	Coincidence test result
1	Logic level of backplane VETO line
2	If set, indicates event is piled up
3	If set, indicates waveform FIFO full
4	If set, indicates this channel was hit (else the event was recorded based on distributed trigger)
5	If set, indicates this channel was out of ADC range at time of fast trigger
6	If set, indicates that the GATE input of this channel has been high at time of fast trigger
7	If set, indicates that at least one of the PSA sums overflowed.
8..14	reserved
15	If set, indicates a data transmission error has been detected for this event. Parts of header and waveform may be corrupted

Table 7-8: Event pattern and Event info in Run Type 0x400, total 32 bits.

---

<sup>8</sup> As event records are for a single channel at a time, only one bit in [0..3] is set. If there was a coincident pulse in any other channel, the corresponding hits in [8..11] are set. However, recording of those other channels follows those channels' rules. For example, if a channel is piled up it will only be recorded if pileup rejection is turned off. Event records thus may show coincidence patterns with more channels than actually being recorded.

### 7.1.4 PSA text data without waveforms (Run Type 0x401)

The .dt3 files created in Run Type 0x401 list energies, channel numbers, 48 bit time stamps, and 6 PSA values as tab delimited values, one event per line. (This is the same format as created with the Pixie Viewer AutoProcessListModeData option set to 3). See below for an example.

```
Module:      0
Run Type:    1025
Run Start Time (s):    33.912160
Event Channel TimeStamp Energy RT    Apeak Bsum   Q0 Q1 PSaval
0     0     16956079848  286    4369    8738    808    1361   2434    0
1     0     16956109724  899    4369    8738    739    628    65466   65535
...

```

Example of .dt3 file content

The headers have the following meanings:

TimeStamp	48 bit time stamp (2ns units)
Energy	Pulse Height
RT*	unused, reserved for rise time
Apeak*	peak amplitude
Bsum*	pre-trigger baseline sum
Q0*	first sum on rising edge of pulse
Q1*	second sum on rising edge of pulse
PSAval*	DSP computed ratio of Q sums

\* = only meaningful when PSA firmware variant is active

### 7.1.5 Modified Pixie-4e style binary list mode files (Run Type 0x410)

*DATA\_FLOW 3,4 to UDP receiver (10G only)*

Run types 0x410 creates single-channel event records in binary format. This runtype is designed exclusively to stream data out as UDP packages via the 10G interface. The 31+ words (16 bit unsigned integer, low byte first) are:

Word #	Variable	Description
0	EventHeadLen	Event Header length (31)
1	ModuleType	A number identifying the Pixie module type, including ADC bits and rate.
2	RunType	A number identifying the data format (0x410)
3	EvtPattern	Hit pattern.
4	EvtInfo	Event status flags.
5	NumTraceBlks	Number of blocks of Trace data to follow the header
6	reserved	(was: Number of blocks of Trace data in previous record, for parsing back)
7	TrigTimeLO	Trigger time, low word (in units of 1 ns)
8	TrigTimeMI	Trigger time, middle word (in units of $2^{16}$ x 1 ns)
9	TrigTimeHI	Trigger time, high word (in units of $2^{32}$ x 1 ns)
10	TrigTimeX	Trigger time, extra 8 bits (in units of $2^{48}$ x 1 ns)
11	Energy	Pulse Height
12	ChanNo	Channel number
13	PSA max (or ampl)	See section 12.2.3
14	(CFD)	
15	PSA base	
16	PSA Q0 (- base?)	
17	PSA Q1 (- base?)	
18	(PSA R)	
19	CFD raw	See section 7.1.1
20	CFD raw	
21	CFD raw	
22	CFD raw	
23	EXT_TS	
24	EXT_TS	
25	EXT_TS	
26	reserved	
27	reserved	
28	TransferInfo	15-8: multi-package event segment number 7-0: channel number
29	reserved	
30	reserved	
30+	ADC Data [15:0]	ADC data

Table 7-9: Channel header for Run Type 0x410, total 31 words (16bit).

To accommodate more than 4 channels, and noting that channel number is indicated in word 12, the EvtPattern and EvtInfo words have been modified as follows:

<b>Bit #</b>	<b>Description</b>
<i>EvtPattern</i>	
0..7	If set, indicates that channel 0..7 from this Kintex has been hit in this event
8..15	reserved
<i>EvtInfo</i>	
0	reserved
1	if set, TTCL logic approved the event
2	If set, indicates event is piled up
3	reserved
4	reserved
5	If set, indicates this channel was out of ADC range at time of fast trigger
6	If set, indicates that the front panel / VETO logic has been high at time of fast trigger
7..15	reserved

Table 7-10: Event pattern and Event info in Run Type 0x410, total 32 bits.

### 7.1.6 Gammasphere compatible binary list mode files (Run Type 0x411)

*DATA\_FLOW 3,4 to UDP receiver (10G only)*

Run types 0x411 creates single-channel event records in binary format. This runtype is designed exclusively to stream data out as UDP packages via the 10G interface. It is compatible to the Gammasphere data format. The 31+ words (16 bit unsigned integer, low byte first) are:

Word #	Variable	Description
0	Fixed	0xAAAA
1	Fixed	0xAAAA
2	USER_PACKET_DATA	Defined by user in settings.ini (Note: last 4 bits <i>may</i> indicate channel number)
3	GeoAddr + Packet Length	10-0: 31+ number of waveform samples 11: ID of Pixie-Net XL SFP port 15-12: MODULE_ID specified in settings.ini
4	TrigTimeLO	Trigger time, low word (in units of 1 ns)
5	TrigTimeMI	Trigger time, middle word (in units of $2^{16} \times 1$ ns)
6	TrigTimeHI	Trigger time, high word (in units of $2^{32} \times 1$ ns)
7	Types	3-0: Header Type (0xD) 9-4: reserved 10-15: header length (31)
8	CFD raw	
9	CFD raw	
10	CFD raw	
11	CFD raw	
12	BLavg	Baseline [15:00]
13	BLavg	Baseline [31:16]
14	Esum	Energy sum – leading [15:00]
15	Esum	Energy sum – leading [31:16]
16	Esum	Energy sum – trailing [15:00]
17	Esum	Energy sum – trailing [31:16]
18	Energy	Pulse Height
19	PSA max	
20	PSA base	
21	PSA sum0	
22	PSA sum1	
23	EXT_TS	
24	EXT_TS	
25	EXT_TS	
26	ModuleType	A number identifying the Pixie module type, including ADC bits and rate.
27	EvtInfo	Event status flags. See 0x410
28	Channel ID	Channel number
29	reserved	
30	reserved	
30+	ADC Data [15:0]	ADC data

Table 7-11: Channel header for Run Type 0x411, total 31 words (16bit).

### 7.1.7 Legacy Run Types

Run Type	File format	Wave-forms	Function	File extension	UDP output	Notes
0x104	binary	yes	startdaq	.bin	10G only	3 extra words to 0x100, for 10G block size
0x401	text	No	startdaq	.dt3	No	PSA
0x402	binary	Yes	acquire	.b00	No	Coincidences, compatible with Pixie-4e
0x404	binary	Yes	startdaq, udpena, udpdis	.bin	10G only	Extended header for 8x PSA sums
0x500	text	Yes	startdaq	.txt	No	General purpose, slow
0x501	text	No	startdaq	.dat	No	General purpose
0x502	text	No	startdaq	.dt2	No	PSA
0x503	text	No	coincdaq	.dt3	No	Coincidences

#### 7.1.7.1 List mode files with waveforms (Run Type 0x500)

The default filename for list mode files with waveforms is **LMdata.txt**. Data is in text format and organized as one value per line. The first 4 lines report run type and run start time information. After that, for each event there are 8 header lines and N waveform sample lines. (N can be computed from the TRACE LENGTH in  $\mu$ s specified in the ini file.)

Line	Value	Example
0	File header: module number	Module: 0
1	File header: run type	Run Type: 0x500
2	File header: start time per FPGA clock counter, in units of ns	Run Start Time Stamp (ticks) : 4
3	File header: start time per Linux clock, in units of s since epoch	Run Start Time (s) : 1477668839
4	Event header: Event number	0
5	Event header: Channel number	0
6	Event header: Hit pattern	0x110321
7	Event header: upper 32 bit of FPGA time stamp (in units of $2^{32} \times 1$ ns)	0
8	Event header: lower 32 bit of FPGA time stamp (in units of 1 ns)	26711904
9	Event header: energy	855
10	Event header: PSA result	0
11	Event header: CFD result	0
12	Event waveform: sample 0	539
13	Event waveform: sample 1	511
14	Event waveform: sample 2	525
	...	...

The hit pattern is a bit mask, which tells which channels were recorded detected within the specified coincidence window plus some additional status information, as listed in table 7.1.

Bit #	Description
0..3	If set, indicates that data for channel 0..3 have been recorded <sup>1</sup>
4..7	4: Logic level of FRONT panel input 5: Result of LOCAL acceptance test 6: reserved 7: reserved
8..11	If set, indicates that channel 0..3 has been hit in this event <sup>9</sup> (i.e. if zero, energy reported is invalid or only an estimate)
12..15	If set, indicates that the GATE input of channel 0..3 has been high at time of fast trigger
16	Coincidence test result
17	Logic level of backplane VETO line
18	If set, indicates event is piled up
19	If set, indicates waveform FIFO full
20	If set, indicates this channel was hit (else the event was recorded based on distributed trigger)
21	If set, indicates that the GATE input of this channel has been high at time of fast trigger
22	If set, indicates this channel was out of ADC range at time of fast trigger
23..30 4	Reserved
31	If set, indicates a data transmission error has been detected for this event. Parts of header and waveform may be corrupted

Table 7-12: Event pattern and Event info in Run Type 0x400, 0x500, 0x501, total 32 bits.

### 7.1.7.2 List mode files without waveforms (Run Type 0x501)

The default filename for list mode files without waveforms is **LMdata.dat**. Data consists of comma separated values and is organized as one event per line. The first 2 lines are names and values of run start information. Line 3 lists the column headers for the following event data. For example,

```
Module,RunType,Run_Start_ticks,Run_Start_sec,Unused1,Unused2
0,0x501,4,1477668787,--,--
No,Ch,Hit,Time_H,Time_L,Energy
0,0,0x110321,0,5148104,361
1,1,0x110322,0,5148104,373
2,0,0x110321,0,5960704,360
3,1,0x110322,0,5960704,372
```

lists in line 2 the module number, run type, and run start time in ns (FPGA) and s (Linux). Line 4 and beyond show the event number, channel number, hit pattern, upper and lower time stamp, energy. (See above for definition of these values).

### 7.1.7.3 List mode files with PSA (Run Type 0x502)

In run type 0x502, the data acquisition program creates list mode files without waveforms named **LMdata.dt2**. Similar to data in run type 0x501, it lists in the first 2 lines are names

---

<sup>9</sup> As event records are for a single channel at a time, only one bit in [0..3] is set. If there was a coincident pulse in any other channel, the corresponding hits in [8..11] are set. However, recording of those other channels follows those channels' rules. For example, if a channel is piled up it will only be recorded if pileup rejection is turned off. Event records thus may show coincidence patterns with more channels than actually being recorded.

and values of run start information. Line 3 lists the column headers for the following event data. For example,

```
Module,Run_Type,Run_Start_ticks,Run_Start_sec,Unused1,Unused2
0,0x502,-1107295228,1487638335,--,--
Event_No,Channel_No,Hit_Pattern,Event_Time_H,Event_Time_L,Energy,
Amplitude,CFD,Base,Q0,Q1,PSAvalue
0,0,0x110121,0,396065120,4972,695,0,122,766,289,377
1,0,0x110121,0,396136216,4496,554,0,124,693,305,440
2,0,0x110121,0,396342104,2367,593,0,122,652,287,440
3,0,0x110121,0,396342848,2367,253,0,123,346,172,497
4,0,0x110121,0,396422920,355,520,0,122,496,290,584
```

lists in line 2 the module number, run type, and run start time in ns (FPGA) and s (Linux). Line 4 and beyond show the event number, channel number, hit pattern, upper and lower time stamp, energy, and results of the PSA. (See above for definition of these values). In addition, run type 0x502 creates a file PSA.csv that contains the histogram data of PSA value vs energy (see below).

#### 7.1.7.4 Coincidence list mode files without waveforms (Run Type 0x503)

The default filename for list mode files without waveforms is **LMdata.dt3**. Data consists of comma separated values and is organized as one 4-channel event per line. The first 2 lines are names and values of run start information. Line 3 lists the column headers for the following event data. For example,

---

```
Module,Run_Type,Run_Start_ticks,Run_Start_sec,Unused1,Unused2
0,0x503,3204391957,1490636304,--,--
Event_No,Hit_Pattern,Event_Time_H,Event_Time_L,PPStime,Time0,Time1,Time2,Time3,Energy0,Energy1,Energy2,Energy3
1,0x11032F,0,515640,0,515472,515400,0,0,3654,3726,0,0
2,0x11032F,0,1171000,0,1170832,1170760,0,0,3656,3726,0,0
3,0x11032F,0,1826360,0,1826192,1826120,0,0,3659,3727,0,0
4,0x11032F,0,2481720,0,2481552,2481480,0,0,3655,3727,0,0
5,0x11032F,0,3137080,0,3136912,3136840,0,0,3655,3725,0,0
6,0x11032F,0,3792440,0,3792272,3792200,0,0,3660,3732,0,0
7,0x11032F,0,4447800,0,4447632,4447560,0,0,3657,3727,0,0
```

lists in line 2 the module number, run type, and run start time in ns (FPGA) and s (Linux). Line 4 and beyond show the event number, hit pattern, upper and lower time stamp of the event, and local time stamp (24 bit) and energy for each channel. (See above for definition of these values). It also reports a PPStime, which is the local time latched by an external trigger signal. The local time stamp is captured at the rising edge of the pulse in that channel, the event time stamp is latched when the data is recorded in all channels.

#### 7.1.7.5 Binary coincidence list mode files with waveforms (Run Type 0x402)

Run type 0x402 creates 4-channel event records in binary format, **LMdata.b00**. This format is compatible with the Pixie-4e. The file starts with a file header of 32 words, same as Run Type 0x400.

Following the file header, the 4-channel event records are stored in sequential order. Each event starts out with a channel header of 32 words. The 32 words (16 bit) are

Word #	Variable	Description
0	EvtPattern	Hit pattern.
1	EvtInfo	Event status flags.
2	NumTraceBlks	Number of blocks of Trace data to follow the header (all channels)

3	NumTraceBlksPrev	Number of blocks of Trace data in previous record (for parsing back)
4	TrigTimeHI	Event trigger time, high word
5	TrigTimeX	Event trigger time, extra 8 bits
6	Energy sum	Sum of channel energies
7	NumUserDataBlks	Number of blocks of user header data to follow
8	LocalTimeLO 0	Local trigger time, low word (ch. 0)
9	LocalTimeMI 0	Local trigger time, middle word (ch. 0)
10	Energy 0	Pulse Height (ch. 0)
11	NumTraceBlks 0	Number of blocks of Trace data to follow the header (ch. 0)
12	LocalTimeLO 1	Local trigger time, low word (ch. 1)
13	LocalTimeMI 1	Local trigger time, middle word (ch. 1)
14	Energy 1	Pulse Height (ch. 1)
15	NumTraceBlks 1	Number of blocks of Trace data to follow the header (ch. 1)
16	LocalTimeLO 2	Local trigger time, low word (ch. 2)
17	LocalTimeMI 2	Local trigger time, middle word (ch. 2)
18	Energy 2	Pulse Height (ch. 2)
19	NumTraceBlks 2	Number of blocks of Trace data to follow the header (ch. 2)
20	LocalTimeLO 3	Local trigger time, low word (ch. 3)
21	LocalTimeMI 3	Local trigger time, middle word (ch. 3)
22	Energy 3	Pulse Height (ch. 3)
23	NumTraceBlks 3	Number of blocks of Trace data to follow the header (ch. 3)
24	reserved	reserved
25	reserved	reserved
26	TrigTimeLO	Event trigger time, low word
27	TrigTimeMI	Event trigger time, middle word
28-31	reserved	reserved

Table 7-13: Channel header for Run Type 0x402, total 32 words (16bit).

#### 7.1.7.6 Binary list mode files with waveforms (Run Type 0x104)

Word #	Variable	Description
0	EventHeadLen	Event Header length (23)
1	ModuleType	A number identifying the Pixie module type, including ADC bits and rate.
2	RunType	A number identifying the data format
3	IDs	00-03 Channel ID 04-07 Slot ID 08-11 Crate ID 12-15 Header Length in 32bit words, low 4 bits
4	Lengths	00-00 Header Length in 32bit words, highest bit 01-14 Event Length in 32bit words 15-15 Finish Code (1 if piled up)
5	EVTTIME	EVTTIME LO[15:0]
6	EVTTIME	EVTTIME LO[31:16]
7	EVTTIME	EVTTIME HI[15:0]
8	UDP COUNT	Counter for UDP output packets [15:0]
9	Energy	Event Energy
10	Trace	00-14 Trace Length (in samples or 16bit words) 15-15 Trace Out-of-Range Flag
11	QDC0	00-15 QDC sum 0 [15:00]
12	QDC0	00-07 QDC sum 0 [23:16] 08-15 All zero

13	QDC1	00-15 QDC sum 1 [15:00]
14	QDC1	00-07 QDC sum 1 [23:16] 08-15 All zero
15	QDC2	00-15 QDC sum 2 [15:00]
16	QDC2	00-07 QDC sum 2 [23:16] 08-15 All zero
17	QDC3	00-15 QDC sum 3 [15:00]
18	QDC3	00-07 QDC sum 3 [23:16] 08-15 All zero
19	Ext TS	Ext TS Lo [15:00]
20	Ext TS	Ext TS Lo [31:16]
21	base	Averaged pre-trigger base [15:00]
22	max	Maximum of QDC samples [15:00]
23+	ADC data	ADC Data [15:0]

Table 7-14: List mode event header data structure (in 16 bit words) for 4 QDC words, base and maximum.

#### 7.1.7.7 Binary list mode files over 10G Ethernet with waveforms (Run Type 0x404 -- old)

Word #	Variable	Description
0	EventHeadLen	Event Header length (47)
1	ModuleType	A number identifying the Pixie module type, including ADC bits and rate.
2	RunType	A number identifying the data format
3	ADCRate	ADC rate in MSPS
4	ADCbits	ADC precision in bits
5	FWversion	A number identifying the firmware version
6	UDP_COUNT	Counter for UDP output packets [15:0]
7	EvtPattern	Reserved for hit pattern NYI
8	EvtInfo	Event status flags.
9	NumTraceBlks	Number of blocks (16bit x 32 words) of trace data to follow the header
10	reserved	unused
11	TrigTimeLO	Trigger time, low word (in units of 1 ns)
12	TrigTimeMI	Trigger time, middle word (in units of 2^16 x 1 ns)
13	TrigTimeHI	Trigger time, high word (in units of 2^32 x 1 ns)
14	TrigTimeX	Trigger time (reserved)
15	Energy	Pulse Height
16	ChanNo	Channel number
17	QDC0	00-15 QDC sum 0 [15:00]
18	QDC0	00-07 QDC sum 0 [23:16] 08-15 All zero
19	QDC1	00-15 QDC sum 1 [15:00]
20	QDC1	00-07 QDC sum 1 [23:16] 08-15 All zero
21	QDC2	00-15 QDC sum 2 [15:00]
22	QDC2	00-07 QDC sum 2 [23:16] 08-15 All zero

23	QDC3	00-15 QDC sum 3 [15:00]
24	QDC3	00-07 QDC sum 3 [23:16] 08-15 All zero
25	QDC4	00-15 QDC sum 4 [15:00]
26	QDC4	00-07 QDC sum 4 [23:16] 08-15 All zero
27	QDC5	00-15 QDC sum 5 [15:00]
28	QDC5	00-07 QDC sum 5 [23:16] 08-15 All zero
29	QDC6	00-15 QDC sum 6 [15:00]
30	QDC6	00-07 QDC sum 6 [23:16] 08-15 All zero
31	QDC7	00-15 QDC sum 7 [15:00]
32	QDC7	00-07 QDC sum 7 [23:16] 08-15 All zero
33	base	Averaged pre-trigger base [15:00]
34	max	Maximum of QDC samples [15:00]
35	Ext TS	Ext TS Lo [15:00] (in units on 1ns)
36	Ext TS	Ext TS Lo [31:16]
37	reserved	unused
38	reserved	unused
39	reserved	unused
40	reserved	unused
41	reserved	unused
42	reserved	unused
43	reserved	unused
44	reserved	unused
45	reserved	unused
46	reserved	unused
47+	ADC data	ADC Data [15:0]

Table 7-15: LEGACY list mode event header data structure (in 16 bit words) for 8 QDC words, base and maximum.

#### 7.1.7.8 Binary list mode files over 10G Ethernet without waveforms (Run Type 0x404 - new)

Word #	Variable	Description
0	EventHeadLen	Event Header length (323)
1	ModuleType	A number identifying the Pixie module type, including ADC bits and rate.
2	RunType	A number identifying the data format
3	ADCRate	ADC rate in MSPS
4	ADCbits	ADC precision in bits
5	FWversion	A number identifying the firmware version
6	UDP_COUNT	Counter for UDP output packets [15:0]
7	EvtPattern	Reserved for hit pattern NYI
8	EvtInfo	Event status flags.
9	NumTraceBlks	Number of blocks (16bit x 32 words) of trace data to follow the header
10	reserved	unused
11	TrigTimeLO	Trigger time, low word (in units of 1 ns)
12	TrigTimeMI	Trigger time, middle word (in units of 2^16 x 1 ns)
13	TrigTimeHI	Trigger time, high word (in units of 2^32 x 1 ns)
14	TrigTimeX	Trigger time (reserved)
15	Energy	Pulse Height
16	ChanNo	Channel number
17	QDC0	00-15 QDC sum 0 [15:00]
18	QDC0	00-07 QDC sum 0 [23:16] 08-15 All zero
19	QDC1	00-15 QDC sum 1 [15:00]
20	QDC1	00-07 QDC sum 1 [23:16] 08-15 All zero
21	QDC2	00-15 QDC sum 2 [15:00]
22	QDC2	00-07 QDC sum 2 [23:16] 08-15 All zero
23	QDC3	00-15 QDC sum 3 [15:00]

24	QDC3	00-07 QDC sum 3 [23:16] 08-15 All zero
25	QDC4	00-15 QDC sum 4 [15:00]
26	QDC4	00-07 QDC sum 4 [23:16] 08-15 All zero
27	QDC5	00-15 QDC sum 5 [15:00]
28	QDC5	00-07 QDC sum 5 [23:16] 08-15 All zero
29	QDC6	00-15 QDC sum 6 [15:00]
30	QDC6	00-07 QDC sum 6 [23:16] 08-15 All zero
31	QDC7	00-15 QDC sum 7 [15:00]
32	QDC7	00-07 QDC sum 7 [23:16] 08-15 All zero
33	base	Averaged pre-trigger base [15:00]
34	max	Maximum of QDC samples [15:00]
35	Ext TS	Ext TS Lo [15:00] (in units on 1ns)
36	Ext TS	Ext TS Lo [31:16]
37	reserved	unused
38	reserved	unused
39	reserved	unused
40	reserved	unused
41	reserved	unused
42	reserved	unused
43	reserved	unused
44	reserved	unused
45	reserved	unused
46	reserved	unused
46+		Repeat from word 7, 7 more times

Table 7-16: LEGACY list mode event header data structure (in 16 bit words) for 8 QDC words, base and maximum; 8 events per package.

Notes:

- CCSRA\_TRACEENA\_08 is ignored.

- Available in FW version 0x3A42 and later

## 7.2 Run Statistics Files

Run Statistics files contain comma separated values. There are two types of files: RS.csv and RATES.csv.

### 7.2.1 RS.csv

The first row lists the column headers. The columns are

ParameterCo	Name of controller parameter
Controller	Value of controller parameter
ParameterSy	Name of system parameter for FPHA #
System#	Value of system parameter
ParameterCh	Name of channel parameter
Channel#	Value of channel parameter for channel #

The units of the reported values are generally in seconds, nanoseconds, or counts per seconds. For example, the line

```
TOTAL_TIME,60.6966,RUN_TIME,60.6964,60.6964,COUNT_TIME, 0, 0, 0, 0,...
```

reports an acquisition TOTAL TIME of 60.6966s (time the controller was requesting data), RUN\_TIMES of slightly less for each FPGA (due to dead time effects) and a COUNT\_TIME of 0 for the listed channels (disabled)

### 7.2.2 RATES.csv

The first row lists the column headers. The columns are

Parameter	Name of controller parameter
Controller	Value of controller parameter

---

### 7.3 MCA Files

MCA files contain comma separated values. The first row lists the column headers (bin number and channel number). The following rows contain the values. For example,

bin,MC Ach0,MC Ach1,MC Ach2,MC Ach3 ...

0,575,563,0,0 ...

1,55,54,0,0

2,48,62,0,0

3,59,53,0,0

4,56,66,0,0

5,69,51,0,0

# 8 Controller (MZ) Registers Visible to Linux

The FPGA (PL) registers described below are used to apply settings to the pulse processing firmware and read back event data. This is generally handled by XIA's API functions and users do not need to understand this functionality in detail. The primary way to set registers is by specifying parameters in the .ini file and call XIA's API function `proggippi` to convert and apply as appropriate. XIA may change register addresses and bits in future code revision, but will try to keep the settings file format constant and compatible.

## 8.1 I/O Registers

**Address range is 0x000 – 0x03F. Can be read back to verify I/O. 16 bit each (single write ok)**

Register	Address	R/W	Description
CSRIN	0	R/W	Run Control Register bits 0 RunEnable (set to start DAQ run) connected to line E1 for both K7s 9 nLive (set to 1 to pause DAQ run) = !RunEnable
COINCPATTERN	1	R/W	Allowed coincidence pattern and other control bits 0..15 CoincPattern
I2C	2	R/W	Control the SDA and SCL lines 0 SDA 1 SCL 2 SDA ENA (SDA output enable)
OUTBLOCK (global) DEVICE_CS	3	R/W	Specifies address range for reads Bit 3 = select FPGA 0 for external I/O Bit 4 = select FPGA 1 for external I/O
HV_DAC	4	R/W	HV DAC value (16 bit) unused
SERIAL_IO	5	R/W	Value for offboard serial IO (16 bit) unused
AUX_CTRL	6	R/W	Aux Control bits for HW 0 pulser enable 1 red LED on/off 2 reserved 3 synchronize to PTP triggers 4 select main I2C 5 select DB0 I2C 6 select DB1 I2C 7 reserved 8 reserved 9 require WR time start/stop in PicoZed
FPGA_PROG	7	R/W	0 probg for FPGA boot
PLLSTART	8	W	Writing to this register issues a pulse to start programming the LMK PLL for ADC and FPGA processing clock Bits defined by CLK_CTRL: 0 LMK_SEL0: if 1, use WR as ADC clk source, else other LMK 1 LMK_SEL1 if 1, use WR as ADC clk source, else FPGA 1

Register	Address	R/W	Description
COUNTER_CLR	9	W	Writing to this register issues a pulse to clear runstats counters
RTC_CLR	10	W	Writing to this register issues a pulse to clear RTC time counter
	11	R/W	reserved
RUNCONTROL	12	R/W	0 LMRUN402 (coincidence mode) 1 MCARUN (no trace out mode) 2 CWGROUP (only 1 record per CW) 3 QSPI_MCA (set to read MCA data via QSPI stream, affects live if FIFO full)
FPGACONF	13	W	FPGA configuration data
REV	14	R/W	HW revision from PROM (written by bootfpga)
	15	R/W	Reserved
SERIAL DB0 A	16	R/W	Value for DAC serial IO (16 bit) ch.0
SERIAL DB0 B	17	R/W	Value for DAC serial IO (16 bit) ch.1
SERIAL DB0 C	18	R/W	Value for DAC serial IO (16 bit) ch.2
SERIAL DB0 D	19	R/W	Value for DAC serial IO (16 bit) ch.3
SERIAL DB1 A	20	R/W	Value for DAC serial IO (16 bit) ch.4
SERIAL DB1 B	21	R/W	Value for DAC serial IO (16 bit) ch.5
SERIAL DB1 C	22	R/W	Value for DAC serial IO (16 bit) ch.6
SERIAL DB1 D	23	R/W	Value for DAC serial IO (16 bit) ch.7
ExAddrForWrite	24	R/W	Write address for external I/O data write
ExDataWrite	25	R/W	Write data for external I/O data write
ExAddrForRead	26	R/W	Write address for external I/O data read
ExDataRead	27	R	Read data for external I/O data read
MCAdata0 low	28	R	Read MCA data (E) from FIFO, advance FIFO
MCAdata0 high	29	R	Read MCA data (channel etc) from FIFO 0-2 channel number 3 fippi ID 4 sign bit 5 pileup bit 6 rangebad 7 veto at time of fast trigger OR !TTCLaccept at time of energy filter done 8-11 coinc pattern ch. 0-3 12-15 fixed 0xA (or 0x2)
MCAdata1 low	30	R	Read MCA data (E) from FIFO, advance FIFO
MCAdata1 high	31	R	Read MCA data (channel etc) from FIFO Same as MCAdata0 high
<i>Read only registers</i>			
CSROUT_L	32	R	Run Status info bits 0 RunEnable (set to start DAQ run) 1 WR tm_link_up (1 = Ethernet link up for PZ) 2 SDA readback 3 WR tm_time_valid (1 = timecode is valid for PZ) 4 zdtfull 5 PTPenable required 6 reserved 7 MCAdataready0 PTPenabled 8 MCAdataready1 9 nLive 10 PSA licensed

Register	Address	R/W	Description
			11 VetoIn 12 MC Afifofull 13 ACTIVE (=RunEnable) 14 FPGA configuration DONE 15 FPGA configuration INIT
CSROUT_H	33	R	Currently fixed 0xBE00
reserved	34	R	Internal use
reserved	35	R	
HW_VERSION	36	R	HW ID number
SNUM	37	R	Serial number of the module
SYSTIME_L	38	R	Time since last boot (lower 16 bit) [lsb=1ns]
SYSTIME_M	39	R	Time since last boot (middle 16 bit)
TOTALTIME_L	40	R	Total run time (lower 16 bit) by controller
TOTALTIME_M	41	R	Total run time (mid 16 bit) [lsb=1ns]
TOTALTIME_H	42	R	Total run time (high 16 bit)
TOTALTIME_X	43	R	Total run time (extra 16 bit)
FW_VERSION	44	R	FW Revision number
WR_TIME	45		WR time (s) L
WR_TIME	46		WR time (s) M
WR_TIME	47		WR time (s) H
WR_NSTIME	48		WR cycle time (ns) L
WR_NSTIME	49		WR cycle time (ns) H
reserved	50-63	R	unused
<i>Write only registers</i>			
WR_START_TAI_L	32	W	Start and stop time during which run is enabled if CSR bit 1 is set. Same units as WR_TIME.
WR_START_TAI_M	33	W	
WR_START_TAI_H	34	W	
WR_STOP_TAI_L	35	W	Start and stop time during which run is enabled if CSR bit 1 is set. Same units as WR_TIME.
WR_STOP_TAI_M	36	W	
WR_STOP_TAI_H	37	W	
	38-63	W	unused

## 9 Pulse Processing (K7) Register Map

These registers are indirectly visible to Linux through read/write from/to controller registers

The architecture of the Pixie-Net XL implies certain restrictions and indirections in the host register I/O. There are 4 levels of addressing involved

**MicroZed (xillybus) address range (AMZ)** as described in section 8. These are direct read and write operations. 4 registers handle data/address read/write (AMZ = 0x18-0x1B).

**FPGA (K7) chip select:** FPGA 0 or FPGA 1 via OUTBLOCK register  
In AMZ=0x03 (OUTBLOCK), write 0x0 for MicroZed I/O, 0x04 for FPGA 0, 0x08 for FPGA1

**K7 pages (~channel):** There are pages of registers for the system and channel parameters; essentially each channel is a page (numbered as 0x10N for channel N), and the system registers is another page (number 0x000).

In AMZ = 0x18 (K7 address), write the FPGA's page address 0x03.

In AMZ = 0x19 (K7 write), write the page number e.g. 0x101 for ch.1.

**K7 register address:** In each page, 256 registers can be addressed. They are divided into 4 ranges: System I/O (0x0-3F), Channel I/O (0x40-7F), System O only (0x80-BF), Channel O only (0xC0-FF). Some addresses are unused. Specifying range and page are both necessary because it makes the address space more compact in the firmware. Note that the page number is ignored for writes to addresses lower than 0x40 (if page is only set in page 0, one could only change the page once)

Writes:

In AMZ = 0x18 (K7 address), write the address 0x0-FF

In AMZ = 0x19 (K7 write), write the value for that register

Reads:

In AMZ = 0x1A (K7 address), write the address 0x0-FF

In AMZ = 0x1B (K7 read), read the value for that register

### Notes:

- K7 register addresses **do not** increment after reads
- The address write for read both sets the address and moves the data from K7 to MZ (data bus = valid for read). It is essential. However, it is so slow that it (almost) not finishes before an immediate read from MZ (external data read). Therefore [for now] the read from MZ is executed twice, the second read gives valid data.
- K7 header (and trace) memory is now a FIFO, just always read 32 (or TraceLength) words from top (register DC)

## 9.1 Input/Output Registers

**System Address range is 0x000 – 0x03F. Channel address range is 0x40-0x7F with PAGE specified in 0x03. Can be read back to verify I/O. 16 bit each**

Use these to specify parameters that control the data acquisition

**1) set FPGA CS in controller register 0x03 (outblock)**

**2) write address for read (or write) to controller register 0x1A (or 0x18). See Address map below**

**3) read (or write) 16 bit numbers from (or to) controller register 0x1B (or 0x19)**

Example 1: write CSR of K7 0

Write 0x03: set bit 3      select K7 0

Write 0x18: 0x0      address for K7 CSR (0x0)

Write 0x19: 0x0200      CSR value, bit 9 set

Example 2: read trace memory of channel 2 of K7 0

Write 0x03: set bit 3      select K7 0

Write 0x18: 0x03      address for channel number

Write 0x19: 0x102      write channel number

Read 0x1B: tval      trace value

Register	Address	R/W	Description	Comment
CSRIN	0x00	R/W	Run Control Register bits 0 RunEnable (set to start DAQ run) 1 require WR time start/stop in Kintex 2 P4E_RUNSTATS 3 AUTO_UDP 4 AUTO_QSPI 5 HEADER_ENA 6 DM_CONTROL FP_TTCL_TRIG 7 TG_TEST_ENA 8 FP_COUNT 9 FP_VETO 10 FP_EXTCLR	Now nLive via E1  CountTime etc per P4e model Stream LM data via UDP and MCA data to MZ, no control through ARM Stream MCA data to MZ, no control through ARM (no UDP) Use header memory (set to 1 except for pure MCA runs with DATA_FLOW==5)  Allow DM to control UDP output (see SW triggering with DATA_FLOW==6) use front panel to start TTCL acceptance window  Enable test data generator in 10G firmware  count FP pulses as ext_ts, else local clock (or WR) use FP as VETO  use FP to clear ext_ts

Register	Address	R/W	Description	Comment
			11 FP_PEDGE 12 HEADER_LONG 13 SYNC_AT_START 14 TTCL_CLK_OUT 15 TTCL_CLK_SL	select rising/falling edge for count or clear  long headers in LM file  clear time stamps at run start time (live falling edge)  if 1, output TTCL clk from FPGA, else local clk  if 1, Kintex uses TTCL clk from jitter cleaner, else from TDF
COINCPATTERN	0x01	R/W	Allowed coincidence pattern and other control bits 0..15 CoincPattern	
ADCCTRL	0x02	R/W	0 swap 0-1 1 swap 2-3 2 swap 4-5 3 swap 6-7	ADC control bits. Currently only used for dual ADCs channel swapping (DB02, DB04, DB08)
PAGE	0x03	R/W	0..7 channel number 8 set to 1 to select channel pages, 0 for system page 9-15 reserved	
HOSTCLR	0x04	W	reset	Writing to this address resets counters, buffers, memories (SDRAM)
ADCSPI	0x05	R/W	Data for ADC control via SPI	
BITSLIP, ADC_DELAY	0x06	W	0-4: clk delay 8-12 data delay	Write starts bitslip (DB01) Write applies clk delay (DB04, DB06)
WR_START_L	0x07	R/W	Start time for comparison with WR time (tm_TAI seconds)	40 valid bits
WR_START_M	0x08	R/W		
WR_START_H	0x09	R/W		
WR_STOP_L	0x0A	R/W	Stop time for comparison with WR time (tm_TAI seconds)	40 valid bits
WR_STOP_M	0x0B	R/W		
WR_STOP_H	0x0C	R/W		
DEST_MAC_L	0x0D	R/W	Specify target/remote MAC for data transfers	
DEST_MAC_M	0x0E	R/W		
DEST_MAC_H	0x0F	R/W		
DEST_IP_L	0x10	R/W	Specify target/remote IP address for data transfers	
DEST_IP_H	0x11	R/W		
SOURCE_IP_L	0x12	R/W	Specify source IP address for data transfers	This should match the WR IP address in the WR PROM
SOURCE_IP_H	0x13	R/W		
CHECKSUM_SHORT	0x14	R/W	IPv4 Checksum for a (fixed length) data packet without waveforms	
CHECKSUM_LONG	0x15	R/W	IPv4 Checksum for a (fixed length) data packet with waveforms	
ENERGY	0x16	R/W	ARM computed energy value for UDP data packet (unused)	Obsoleted by internal energy computation
CFD	0x17	R/W	ARM computed CFD value for UDP data packet	
UDP_CTRL	0x18	R/W	0-7 TRACEBLOCKS Number of 32-sample waveform blocks in packet 8-11 PAYLOAD TYPE 0 short (no waveforms)	Writing this word starts the data transfer  If bit 10 is set, data is moved out of memory but not sent out as

Register	Address	R/W	Description	Comment
			1 long (waveforms) bit 10 set: reject event 12-15: CHANNEL ARM computed channel number for UDP data packet	UDP package (used in DATA_FLOW=3)
HDR_IDS	0x19	R/W	0-3 slot_id 4-7 crate id 8-11 header length 12-15 module id	0-7, 12-15 Replaces values in inforegister (17, 0x5C) since it's the same for all channels 8-11 is fixed to 10 (number of 32bit words in output header block for runtype 0x100)
UDP_PAUSE	0x1A	R/W	Minimum number of cycles to wait between UDP output packages	should take 4ms for UDP_PAUSE = 64K
PLLSPIA	0x1B	R/W	0-15 PLL SPI register address and control	Program PLL of WRclkDB (AD9516). Write to 0x1C starts transfer [Rev B only]
PLLSPID	0x1C	R/W	0-7 PLL SPI data	
DM_ACCEPT	0x1D	R/W	0 accept (1) or reject (0) data in SDRAM FIFO 1-15 unused	Write to 0x1D starts FIFO flush of 1 event. (DATA_FLOW==6 only)
DEST_PORT	0x1E	R/W	UDP port number	
SOURCE_PORT	0x1F	R/W	UDP port number	
SOURCE_MAC_L	0x20	W	Specify source MAC for data transfers	Ignored in 1G firmware
SOURCE_MAC_M	0x21	W		
SOURCE_MAC_H	0x22	W		
ADC_CHANNEL	0x23	W	0-7 channel number 8-15 adc_dt [7:0]	
ADC_INTERVAL	0x24	W	0-16 adc_dt [23:8]	Write starts ADC capture
SLEEP_TIMEOUT	0x25	W	Clocks disabled after 64K x 40 ns x this number	Set to 65535 to prevent any disabling
RUN_TYPE	0x26	W	Run type for event header in UDP transfers	
USER_PACKET_DATA	0x27	W	User data word for run type 0x411	
TTCL_SYNC_TIME_L	0x28	W	TTCL sync target time	Unused
TTCL_SYNC_TIME_M	0x29	W		
TTCL_SYNC_TIME_H	0x2A	W		
TTCL_APPR_WIN	0x2B	W	TTCL acceptance window length	
GROUPMODE_A,B	0x2C	W	Defines input to channels' group trigger	0 OR of local triggers 1 trigger from system (res.) 2 external trigger from front panel 3 coinc. of 2 channel pairs 4 reserved 5 reserved 6 beta gated OR of dynode triggers
GROUPMODE_C,D	0x2D	W		
GROUPMODE_E,F	0x2E	W		
GROUPMODE_G,H	0x2F	W		
GROUPMODE_FIP	0x30	W	0-7 Defines output to System FPGA trigger lines (reserved) 8-15 Defines gating options	Reserved
VETO_MODE				0 Front panel (see MCSRA bits for inversion and other options) 1 Veto from system (reserved) 2 stretched trigger from channel 6 AND 7 3stretched trigger from channel 6 AND 7, inverted 4 stretched trigger from channel 6 OR 7 5 stretched trigger from channel 6 OR 7, inverted

<b>Register</b>	<b>Address</b>	<b>R/W</b>	<b>Description</b>	<b>Comment</b>
GATE_LENGTH	0x31	W	0-7 length of gating window (reserved)	In clock ticks (8ns)
DYNODE_WINDOW			8-15 length of dynode window (reserved)	In clock ticks (8ns)
CSRinC	0x32	W	0 TRACE_LONG	transmit 8 UDP packages per event (for traces > 704 samples)
	0x33-0x3F	R/W ?	Reserved	do not use
0 a	0x40	W	0 halt 1 CCSRA_INVERT_05 2 CCSRA_VETOENA_06 3 CCSRA_EXTRIGSEL_01 4-6 decimation 7-13 SlowLen 14-15 lowPlusGapLen[1:0]	invert the incoming signal require veto unused SLOW_FILTER_RANGE ENERGY_RISETIME ENERGY_RISETIME, ENERGY_FLATTOP
0 b	0x41	W	0-4 SlowPlusGapLen [6:2]  5 CCSRA_CHANTRIGSEL_03 6 CCSRA_SYNCDATAACQ_04 7 CCSRC_GROUPTRIGSEL_02 8-15 FastLen	ENERGY_RISETIME, ENERGY_FLATTOP unused synchronize dead times share triggers TRIGGER_RISETIME
0 c	0x42	W	0-6 FastPlusGapLen  7-15 Threshold [8:0]	TRIGGER_RISETIME, TRIGGER_FLATTOP TRIGGER_THRESHOLD
0 d	0x43	W	0-6 Threshold [15:9] 7-12 CFDDelay 13 CSRC_CHANVETOSEL_03 14 CCSRC_MODVETOSEL_04 15 CCSRA_ENARELAY_14	TRIGGER_THRESHOLD CFD_DELAY Unused Unused reserved
1a	0x44	W	0-6 GapLen 7-15 RBDelayLen_SF [8:0]	ENERGY_FLATTOP ENERGY_RISETIME, ENERGY_FLATTOP
1b	0x45	W	0-2 RBDelayLen_SF [11:9] 3-15 PeakSep	ENERGY_RISETIME, ENERGY_FLATTOP
1c	0x46	W	0- 12 PeakSamp 13- 15 RBDelayLen_TF [2:0]	ENERGY_RISETIME, ENERGY_FLATTOP TRIGGER_RISETIME, TRIGGER_FLATTOP
1d	0x47	W	0-8 RBDelayLen_TF [11:3] 9-12 CFD_SCALE 13 CCSRA_GOOD_02 14 CCSRA_PILEUPCTRL_15 15 CSRC_INVERSEPILEUP_00	CFD scaling factor set to 0 to mark channels as bad reject piled up events capture only piled up events
2a	0x48	W	0-11 CHANTRIG_STRETCH 12 CCSRA_FTRIGSEL_00 13 CCSRA_TRACEENA_08 14 CCSRA_QDCENA_09 15 CCSRA_CFDMODE_10	Unused use local or distributed trigger record waveforms compute QDC sums enable CFD timing
2b	0x49	W	0 CCSRA_GLOBTRIG_11 1 CCSRA_CHANTRIG_13 2 CCSRC_PAUSEPILEUP_07 3 CCSRC_RBADDIS_06 4-15 VETO_STRETCH	unused unused Ignore closely following triggers Ignore out of range length of logic output signal
2c	0x4A	W	0-3 CHANNEL_ID 4 CSRC_CHECK_ELIMITS_08  5CCSRC_LOCAL_ENERGY_09	Channel number for LM data Enable test for energy within range Use local trigger for E filter capture even in group trigger mode

Register	Address	R/W	Description	Comment
			6 reserved 7 CCSRC_GATEADC_11 8 – 15 FASTTRIG_BACKLEN [7:0]	Used for test purposes Output trigger and gate info with ADC traces coincidence
2d	0x4B	W	0-3 FASTTRIG_BACKLEN [11:8] 4-15 EXTTRIG STRETCH	coincidence unused
3a	0x4C	W	Reserved DAC programming	
3b	0x4D	W	Reserved DAC programming	
5a	0x4E	W	0-8 FTRIGOUT_DELAY 9-15 unused	unused
5b	0x4F	W	Unused	
5c	0x50	W	0-8 EXTERN_DELAYLEN 9-15 TRACE_DELAY [6:0]	Input delay Pre-trigger delay
5d	0x51	W	0-3 TRACE_DELAY [9-7] 4-15 unused	Pre-trigger delay
6a	0x52	W	QDC0Len	QDCLen0
6b	0x53	W	QDCLen1	QDCLen1
6c	0x54	W	QDCLen2	QDCLen2
6d	0x55	W	QDCLen3	QDCLen3
7a	0x56	W	QDCLen4	QDCLen4
7b	0x57	W	QDCLen5	QDCLen5
7c	0x58	W	QDCLen6	QDCLen6
7d	0x59	W	QDCLen7	QDCLen7
13a	0x5A	W	CFD_THRESH	CFD_THRESHOLD
13b	0x5B	W	PSA_THRESH	PSA_THRESHOLD
17 Info a	0x5C	W	Unused	unused
17 Info b	0x5D	W	Unused	unused
17 Info c	0x5E	W	0-13 TRACE_LENGTH 14-15 unused	Full waveform length
17 Info d	0x5F	W	Unused	
22a	0x60	W	Reserved	Not decoded, do not use
22b	0x61	W	Reserved	Not decoded, do not use
	0x62	W	0-15 E coef C0 (low)	ENERGY_RISETIME, ENERGY_FLATTOP, TAU, BLAVG, BLcut  Coefficients for energy computation
	0x63	W	0-07 E coef C0 (high) 8-15 BLAVG	
	0x64	W	0-15 E coef Cg (low)	
	0x65	W	0-07 E coef Cg (high) 8-15 unused	
	0x66	W	0-15 E coef C1 (low)	
	0x67	W	0-07 E coef C1 (high) 8-15 BLcut/4 (*4)	
	0x68	W	0-15 lower energy limit	When E in range and enabled, issue logic pulse on MMCX output of length VETO_STRETCH
	0x69	W	0-15 upper energy limit	
	0x6A-7F	W	Reserved	Not decoded, do not use

## 9.2 Output Only Registers

Address range is 0x080 – 0x0FF. Read only, 16 bits

Use these during the data acquisition to get info of the current status

Register	Address	R/W	Description
CSROUT	0x80	R	Run Status info bits 0-7: mirror CSRin 8: WR tm_link_up (1 = Ethernet link up) 9: WR tm_time_valid (1 = WR timecode is valid) 10: Transfer to SDRAM in progress flag 11: Transfer to UDP in progress flag 12: SDRAM full flag 13: ACTIVE (run in progress) 14-15: reserved
SYSSTATUS	0x81	R	System status bits
MEM_I_CNT_L	0x82	R	Number of events moved into SDRAM
MEM_I_CNT_M	0x83	R	
MEM_O_CNT_L	0x84	R	Number of events moved out from SDRAM
MEM_O_CNT_M	0x85	R	
ADCFRAME	0x86	R	Frame pattern of ADC capture
DEBUG_K	0x87	R	Fixed 0xABCD
RUN_TIME_L	0x88	R	Run time (low)
RUN_TIME_M	0x89	R	Run time (middle)
RUN_TIME_H	0x8A	R	Run time (high)
	0x8B	R	reserved
FW_VERSION	0x8C	R	Version number of K7 firmware
WR_TM_TAI_L	0x8D	R	White Rabbit TAI part of the timecode (full seconds)
WR_TM_TAI_M	0x8E	R	
WR_TM_TAI_H	0x8F	R	
WR_TM_CYC_L	0x90	R	Fractional part of WR time, upshifted by 4, units 1ns
WR_TM_CYC_H	0x91	R	
DPM_a	0x92	R	DPM status [15:00]
DPM_b	0x93	R	DPM status [31:16]
DPM_c	0x94	R	DPM status [47:32]
DPM_d	0x95	R	DPM status [63:48]
PLL reg	0x96	R	Reserved for data from PLL or other SPI device
ADC_FIFO (low)	0x97	R	ADC value from FIFO (low). Read advances FIFO. Must specify ADC_CHANNEL and ADC_INTERVAL for FIFO first
ADC_FIFO (high)	0x98	R	ADC value from FIFO (high). Read advances FIFO. Must specify ADC_CHANNEL and ADC_INTERVAL for FIFO first
	0x99-0xBF	R	Unused
<i>Channel Specific Registers</i>			
8a	0xC0	R	Count time counter L
8b	0xC1	R	Count time counter M
8c	0xC2	R	Count time counter H

Register	Address	R/W	Description
8d	0xC3	R	reserved
9a	0xC4	R	ADC data
9b	0xC5	R	reserved
9c	0xC6	R	reserved
9d	0xC7	R	reserved
10a	0xC8	R	Baseline Lsum low
10b	0xC9	R	Baseline Lsum high
10c	0xCA	R	Baseline Tsum low
10d	0xCB	R	Baseline Tsum high
11a	0xCC	R	Baseline average low
11b	0xCD	R	Baseline average high
11c	0xCE	R	Baseline Gsum low
11d	0xCF	R	Baseline Gsum high , BL unlock
12a	0xD0	R	Fast Peak (NTRIG) counter L
12b	0xD1	R	Fast Peak (NTRIG) counter M
12c	0xD2	R	Fast Peak (NTRIG) counter H
12d	0xD3	R	reserved
	0xD4	R	NOUT counter L
	0xD5	R	NOUT counter M
	0xD6	R	NOUT counter H
	0xD7	R	reserved
	0xD8	R	Header FIFO [15:00]
	0xD9	R	Header FIFO [31:16]
	0xDA	R	Header FIFO [47:32]
	0xDB	R	Header FIFO [63:48], advance FIFO
	0xDC	R	Waveform FIFO [15:00]
	0xDD	R	Waveform FIFO [31:16]
	0xDE	R	Waveform FIFO [47:32]
	0xDF	R	Waveform FIFO [63:48], advance FIFO
	0xE0	R	BL lock
	0xE1	R	read to advance waveform FIFO by tracelength
	0xE2	R	read to advance Energy FIFO by 1
	0xE3	R	Energy FIFO [15:01], pileup flag[0]
	0xE4	R	NPPI counter L
	0xE5	R	NPPI counter M
	0xE6	R	NPPI counter H
	0xE7	R	reserved
	0xE8	R	FTDT counter L
	0xE9	R	FTDT counter M
	0xEA	R	FTDT counter H
	0xEB	R	reserved
	0xEC	R	GDT counter L
	0xED	R	GDT counter M
	0xEE	R	GDT counter H
	0xEF	R	reserved
	0xF0	R	GCOUNT counter L, NYI
	0xF1	R	GCOUNT counter M, NYI
	0xF2	R	GCOUNT counter H, NYI
	0xF3	R	Reserved
	0xF4	R	Current ICR
	0xF5	R	Current Out-of-Range
reserved	0xF6-0xFF	R	unused



# 10 Linux Configuration

## 10.1 Licensing Information

The Pixie-Net XL software includes several components that are licensed with the GNU general public license (GPL). For most of them, the software is unchanged, and the source code is provided on the Pixie-Net XL SD card with the corresponding GPL license files. A small portion of the software is modified by XIA and the modified source code is provided both on the SD card and on XIA's website. The source code for the Linux kernel built by XIA for the Pixie-Net XL and for the full collection of Ubuntu Linux programs is too large for the SD card or to host by XIA, but we can provide it upon request. XIA strives to keep those programs up to date and our own code compatible with the latest version, however, the best source of the programs may be their official repository.

List of key GPL programs:

- LinuxPTP                  source code on SD card or  
                                  at <http://linuxptp.sourceforge.net/>
- ptp-mii-tool              source code on SD card or  
                                  at <https://github.com/giftnuss/net-tools> (original mii-tool) or  
                                  at [files.xia.com](http://files.xia.com) (request access)
- Linux kernel              source code per request or  
                                  at <https://github.com/Xilinx/linux-xlnx>
- Ubuntu files              source code and configuration files on SD or  
                                  at <https://releases.linaro.org/ubuntu/images/>  
                                  developer/latest/linaro-vivid-developer-20151215-714.tar.gz  
                                  or per request
  - includes
    - Lighttpd
    - Samba
    - g++
    - i2c-tools

## 10.2 Software Information

The Pixie-Net XL Linux system is originally based on Xillinux (1.3), which is based on Ubuntu LTS 12.04. The latest distribution (2.0) can be downloaded from the Xillinux website (<http://xillybus.com/xillinux>). In what is distributed by XIA, the kernel and Linux OS have been upgraded to Xilinx kernel release 2017.4 and Ubuntu 18.04 LTS, respectively.

The SD card contains the Linux OS (not visible to Windows) in a Linux partition ("root") and 4 boot files (visible) in a small FAT partition ("boot"). Controller configuration updates of the 4 bootfiles have to be copied to that boot partition. Updates to the control software for data acquisition etc requires copying the latest code into the folder /var/www in the root partition.

### 10.2.1 Ubuntu 18

In the latest Pixie-Net XL Ubuntu 18 release, the main differences to previous versions with Ubuntu 15 are

- added support for ntfs drives
- removal of ROOT
- removal of remote desktop login
- login required for UART/USB terminal
- bugfix applied in kernel for transmission of large files

On Ubuntu 18, The following Linux applications are installed with apt-get install.

- lighttpd  
`/etc/lighttpd/lighttpd.conf` modified as in `/var/www/other_settings` to allow cgi and web operations.

Create `"/var/www/webopspasswords`

`webops:xia17pxn`

This file should be modified to change the password. For sensitive environments, change to encrypted passwords.

Create folder `/var/www/webops`

change owner to `www-data (=lighttpd)`

in `webops`, make links to cgi, ini, jpg, js, html files from `/var/www` by executing every line in `linklist.txt` from the command line.

- g++
- i2c-tools
- samba (see configuration file in `/var/www/other_settings`)
  - + sudo smbpasswd -a root xia17pxn
  - + sudo service smbd restart
- python-dev
- libboost-all-dev
- minicom
- ntfs-3g (with `fuse_fs` driver enabled in kernel)
- ntfs-config
- ethtool
- linuxptp

The desktop `xfce4` for graphical remote login and ROOT are **not** installed by default in Ubuntu 18 (ROOT is no longer an apt package).

### 10.2.2 Systemd startup routine to configure FPGAs

A `systemd` startup routine is called at boot time to configure the FPGAs and set the parameters (i.e. execute `./bootfpga` and `./progfippi`). The routine `pixie_boot.service` is located in `/etc/systemd/system/` and calls the shell

script `autoboot.sh` in `/var/www/`. Output from the script is captured in `autoboot.log`.

The script can be modified as necessary and executed from the command line or automatically at the next boot. The service file can be modified as well, but requires restarting the service with

```
systemctl disable pixie_boot.service
systemctl start pixie_boot.service
systemctl enable pixie_boot.service
```

A backup copy of `pixie_boot.service` is located in the folder `other_settings`

### 10.2.3 Other configurations

- Sync temperature  
For kernel 4.x, raw files are `/sys/devices/soc0/amba/f8007100.adc/iio:device0`, which is difficult to open/read from a program because of the colon.  
So we made a symbolic link :  
`ln -s /sys/devices/soc0/amba/f8007100.adc/iio\:device0/in_temp0_raw temp0_raw`  
to access it from `PixieNetCommon.c` and compute T in Celsius.
- To automatically give permission to all users to `/dev/uio0` (avoiding the initial `chmod` command):  
insert the following line in the file `/etc/udev/rules.d/10-local.rules`:  
`KERNEL=="uio0", MODE=="666"`  
and also in `/etc/udev/uio.rules` as it seems to overwrite the 10-local rules.

## 10.3 Other Settings

### 10.3.1 Static IP address

In some networks, it is required that the Pixie-Net has a static IP. This is usually configured by the “local network administrator” but here are some notes that may be helpful:

- Ubuntu 18  
As Ubuntu 18.04 uses netplan to manage the network, you can create a file ending in `.yaml` in the `/etc/netplan/` directory. As an example, see the `01-netplan.yaml` in the “other settings” folder on the Pixie-Net. You would need to modify the IP address/gateway and may also be necessary to add additional DNS.  
After that, you can use the command `netplan apply` to apply the changes, or you can reboot.
- Ubuntu 15  
Ubuntu15 configures static IP by modifying the file `/etc/network/interfaces`. An example is also located in the “other settings” folder on the Pixie-Net. When using static IP, uncomment the 4-11 lines in the file and make the corresponding changes.

# 11 Theory of Operation

## 11.1 Digital Filters for $\gamma$ -ray Detectors

Energy dispersive detectors, which include such solid state detectors as Si(Li), HPGe, HgI<sub>2</sub>, CdTe and CZT detectors, are generally operated with charge sensitive preamplifiers as shown in Figure 6.1 (a). Here the detector D is biased by voltage source V and connected to the input of preamplifier A which has feedback capacitor C<sub>f</sub> and feedback resistor R<sub>f</sub>.

The output of the preamplifier following the absorption of an  $\gamma$ -ray of energy E<sub>x</sub> in detector D is shown in Figure 6.1 (b) as a step of amplitude V<sub>x</sub> (on a longer time scale, the step will decay exponentially back to the baseline, see section 6.3). When the  $\gamma$ -ray is absorbed in the detector material it releases an electric charge Q<sub>x</sub> = E<sub>x</sub>/ $\epsilon$ , where  $\epsilon$  is a material constant. Q<sub>x</sub> is integrated onto C<sub>f</sub>, to produce the voltage V<sub>x</sub> = Q<sub>x</sub>/C<sub>f</sub> = E<sub>x</sub>/( $\epsilon$ C<sub>f</sub>). Measuring the energy E<sub>x</sub> of the  $\gamma$ -ray therefore requires a measurement of the voltage step V<sub>x</sub> in the presence of the amplifier noise  $\sigma$ , as indicated in Figure 11-1 (b). Scintillator detectors read out with a photomultiplier tube generate pulses in a different mechanism, but for the most part they can still be described as fast rise followed by exponential decay, so the processing described below equally applies.

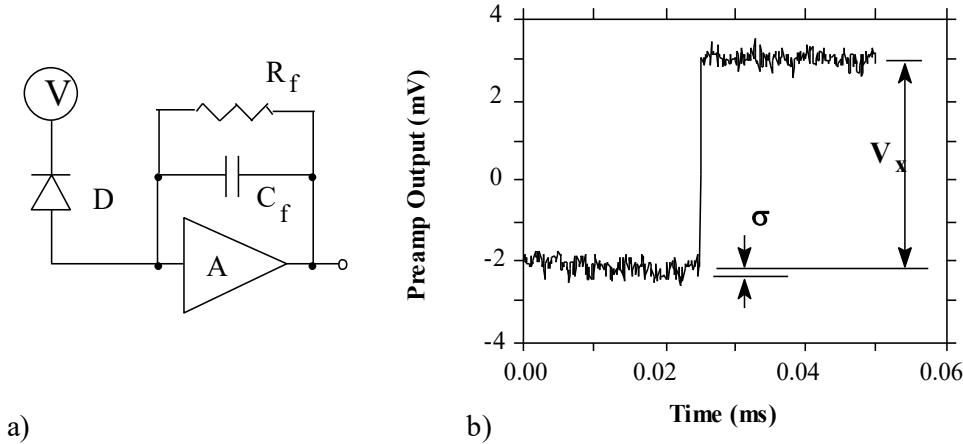


Figure 11-1: (a) Charge sensitive preamplifier with RC feedback; (b) Output on absorption of a  $\gamma$ -ray.

Reducing noise in an electrical measurement is accomplished by filtering. Traditional analog filters use combinations of a differentiation stage and multiple integration stages to convert the preamp output steps, such as shown in Figure 11-1 (b), into either triangular or semi-Gaussian pulses whose amplitudes (with respect to their baselines) are then proportional to V<sub>x</sub> and thus to the  $\gamma$ -ray's energy.

Digital filtering proceeds from a slightly different perspective. Here the signal has been digitized and is no longer continuous. Instead it is a string of discrete values as shown in Figure 11-2. Figure 11-2 is actually just a subset of Figure 11-1 (b), in which the signal was digitized by a Tektronix 544 TDS digital oscilloscope at 10 MSPS (mega samples per second). Given this data set, and some kind of arithmetic processor, the obvious approach to determining V<sub>x</sub> is to take some sort of average over the points before the step and subtract

it from the value of the average over the points after the step. That is, as shown in Figure 11-2, averages are computed over the two regions marked “Length” (the “Gap” region is omitted because the signal is changing rapidly here), and their difference taken as a measure of  $V_x$ . Thus the value  $V_x$  may be found from the equation:

$$V_{x,k} = - \sum_{i(\text{before})} W_i V_i + \sum_{i(\text{after})} W_i V_i \quad (1)$$

where the values of the weighting constants  $W_i$  determine the type of average being computed. The sums of the values of the two sets of weights must be individually normalized.

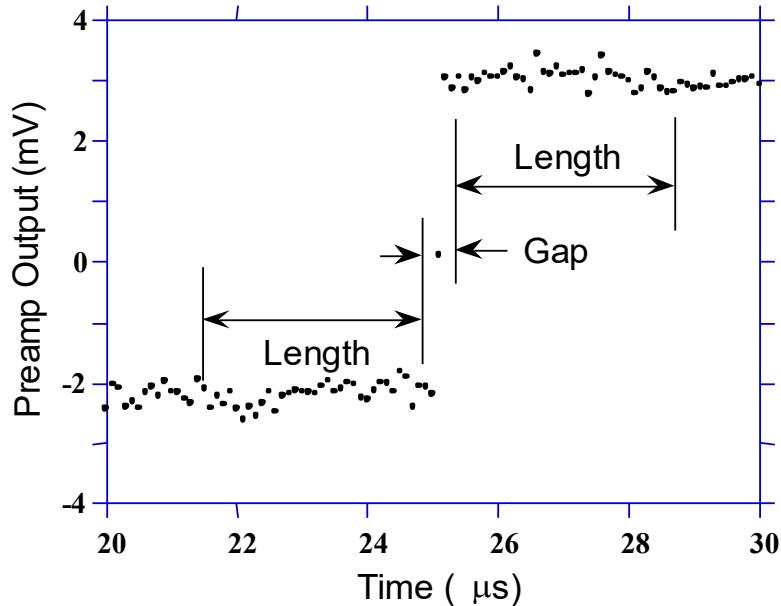


Figure 11-2: Digitized version of the data of Figure 6.1 (b) in the step region.

The primary differences between different digital signal processors lie in two areas: what set of weights  $W_i$  is used and how the regions are selected for the computation of Eqn. 1. Thus, for example, when larger weighting values are used for the region close to the step while smaller values are used for the data away from the step, Eqn. 1 produces “cusp-like” filters. When the weighting values are constant, one obtains triangular (if the gap is zero) or trapezoidal filters. The concept behind cusp-like filters is that, since the points nearest the step carry the most information about its height, they should be most strongly weighted in the averaging process. How one chooses the filter lengths results in time variant (the lengths vary from pulse to pulse) or time invariant (the lengths are the same for all pulses) filters. Traditional analog filters are time invariant. The concept behind time variant filters is that, since the  $\gamma$ -rays arrive randomly and the lengths between them vary accordingly, one can make maximum use of the available information by setting the length to the interpulse spacing.

In principle, the very best filtering is accomplished by using cusp-like weights and time variant filter length selection. There are serious costs associated with this approach however, both in terms of computational power required to evaluate the sums in real time and in the complexity of the electronics required to generate (usually from stored coefficients) normalized  $W_i$  sets on a pulse by pulse basis.

The Pixie-Net XL takes a different approach because it was optimized for high speed operation. It implements a fixed length filter with all  $W_i$  values equal to unity and in fact computes this sum afresh for each new signal value  $k$ . Thus the equation implemented is:

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

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

While this relationship is very simple, it is still very effective. In the first place, this is the digital equivalent of triangular (or trapezoidal if  $G \neq 0$ ) filtering which is the analog industry's standard for high rate processing. In the second place, one can show theoretically that if the noise in the signal is white (i.e. Gaussian distributed) above and below the step, which is typically the case for the short shaping times used for high signal rate processing, then the average in Eqn. 2 actually gives the best estimate of  $V_x$  in the least squares sense. This, of course, is why triangular filtering has been preferred at high rates. Triangular filtering with time variant filter lengths can, in principle, achieve both somewhat superior resolution and higher throughputs but comes at the cost of a significantly more complex circuit and a rate dependent resolution, which is unacceptable for many types of precise analysis. In practice, XIA's design has been found to duplicate the energy resolution of the best analog shapers while approximately doubling their throughput, providing experimental confirmation of the validity of the approach.

## 11.2 Trapezoidal Filtering in a Pixie Module

From this point onward, we will only consider trapezoidal filtering as it is implemented in a Pixie module according to Eqn. 6.2. The result of applying such a filter with Length  $L=1\mu s$  and Gap  $G=0.4\mu s$  to a  $\gamma$ -ray event is shown in Figure 6.3. The filter output is clearly trapezoidal in shape and has a rise time equal to  $L$ , a flattop equal to  $G$ , and a symmetrical fall time equal to  $L$ . The basewidth, which is a first-order measure of the filter's noise reduction properties, is thus  $2L+G$ .

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

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

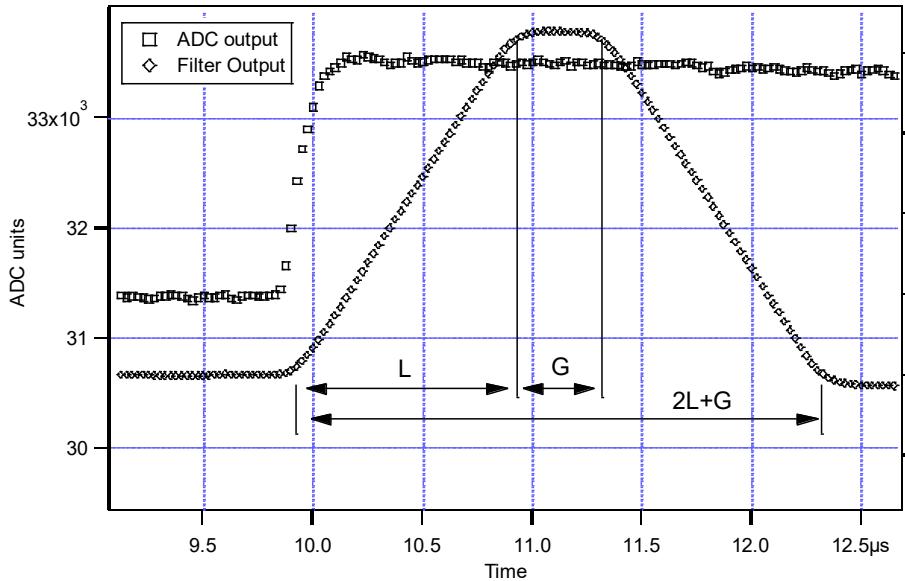


Figure 11-3: Trapezoidal filtering of a preamplifier step with  $L=1\mu s$  and  $G=0.4\mu s$ .

### 11.3 Baselines and Preamplifier Decay Times

Figure 11-4 shows an event over a longer time interval and how the filter treats the preamplifier noise in regions when no  $\gamma$ -ray pulses are present. As may be seen the effect of the filter is both to reduce the amplitude of the fluctuations and reduce their high frequency content. This region is called the *baseline* because it establishes the reference level from which the  $\gamma$ -ray peak amplitude  $V_x$  is to be measured. The fluctuations in the baseline have a standard deviation  $\sigma_e$  which is referred to as the *electronic noise* of the system, a number which depends on the rise time of the filter used. Riding on top of this noise, the  $\gamma$ -ray peaks contribute an additional noise term, the *Fano noise*, which arises from statistical fluctuations in the amount of charge  $Q_x$  produced when the  $\gamma$ -ray is absorbed in the detector. This Fano noise  $\sigma_f$  adds in quadrature with the electronic noise, so that the total noise  $\sigma_t$  in measuring  $V_x$  is found from

$$\sigma_t = \sqrt{(\sigma_f^2 + \sigma_e^2)} \quad (3)$$

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

With a RC-type preamplifier, the slope of the preamplifier is rarely zero. Every step decays exponentially back to the DC level of the preamplifier. During such a decay, the baselines are obviously not zero. This can be seen in Figure 11-4, where the filter output during the exponential decay after the pulse is below the initial level. Note also that the flat top region is sloped downwards.

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

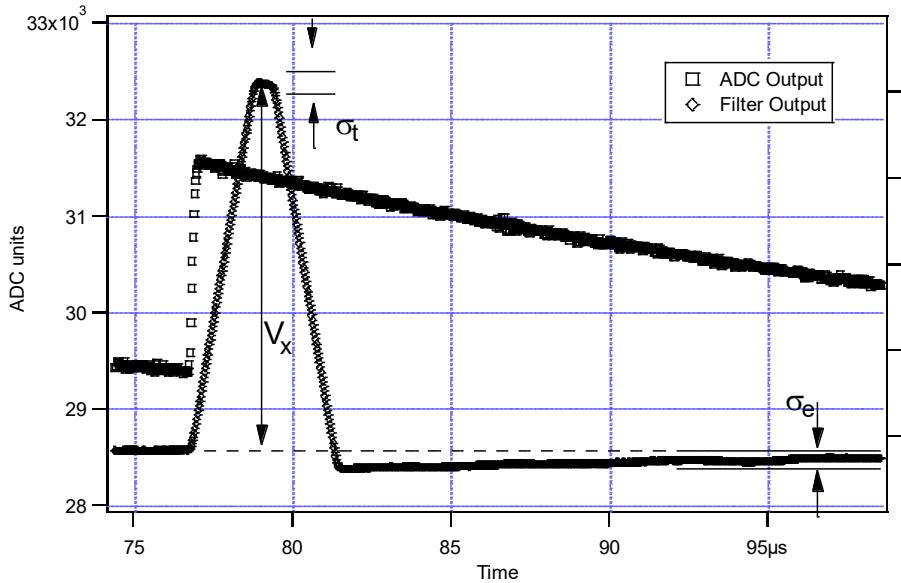


Figure 11-4: A  $\gamma$ -ray event displayed over a longer time period to show baseline noise and the effect of preamplifier decay time.

## 11.4 Thresholds and Pile-up Inspection

As noted above, we wish to capture a value of  $V_x$  for each  $\gamma$ -ray detected and use these values to construct a spectrum. This process is also significantly different between digital and analog systems. In the analog system the peak value must be “captured” into an analog storage device, usually a capacitor, and “held” until it is digitized. Then the digital value is used to update a memory location to build the desired spectrum. During this analog to digital conversion process the system is dead to other events, which can severely reduce system throughput. Even single channel analyzer systems introduce significant dead time at this stage since they must wait some period (typically a few microseconds) to determine whether or not the window condition is satisfied.

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

The peak detection and sampling in a Pixie module is handled as indicated in Figure 11-5. Two trapezoidal filters are implemented, a *fast filter* and a *slow filter*. The fast filter is used to detect the arrival of  $\gamma$ -rays, the slow filter is used for the measurement of  $V_x$ , with reduced noise at longer filter rise times. The fast filter has a filter length  $L_f = 0.1\mu s$  and a gap  $G_f = 0.1\mu s$ . The slow filter has  $L_s = 1.2\mu s$  and  $G_s = 0.35\mu s$ .

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

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

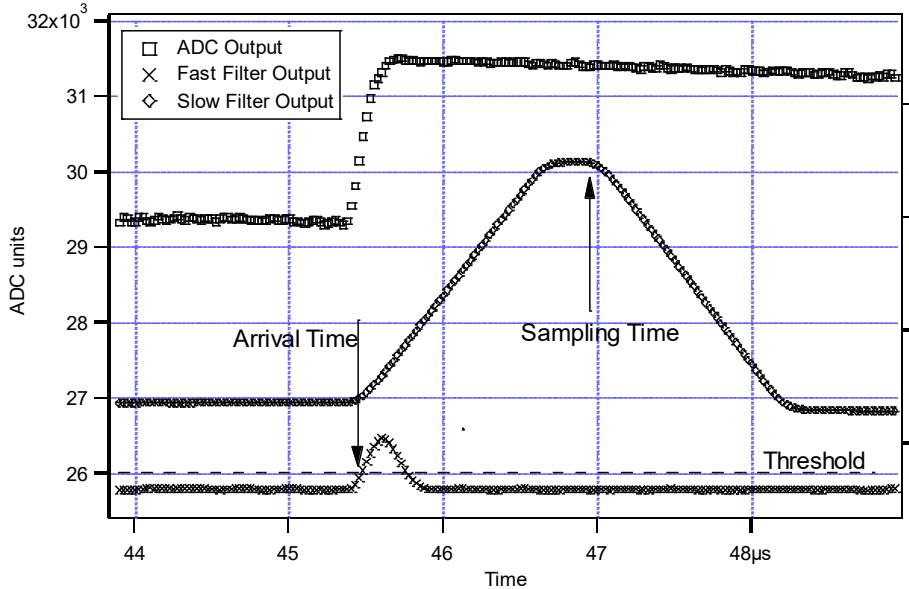


Figure 11-5: Peak detection and sampling in a Pixie module.

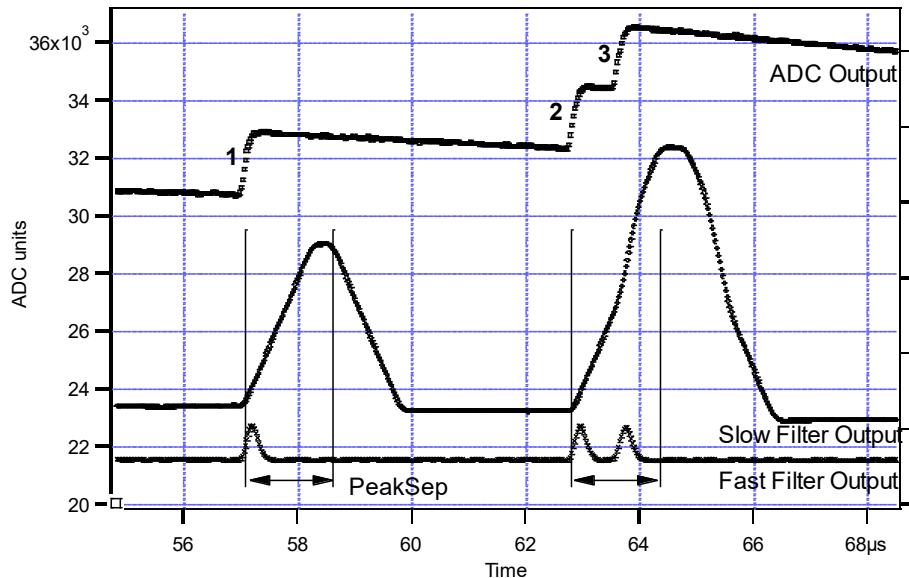


Figure 11-6: A sequence of 3  $\gamma$ -ray pulses separated by various intervals to show the origin of pileup and demonstrate how it is detected by the Pixie module.

The value  $V_x$  captured will only be a valid measure of the associated  $\gamma$ -ray's energy provided that the filtered pulse is sufficiently well separated in time from its preceding and succeeding neighbor pulses so that their peak amplitudes are not distorted by the action of the trapezoidal filter. That is, if the pulse is not *piled up*. The relevant issues may be understood by reference to Figure 11-6, which shows 3  $\gamma$ -rays arriving separated by various

intervals. The fast filter has a filter length  $L_f = 0.1\mu s$  and a gap  $G_f = 0.1\mu s$ . The slow filter has  $L_s = 1.2\mu s$  and  $G_s = 0.35\mu s$ .

Because the trapezoidal filter is a linear filter, its output for a series of pulses is the linear sum of its outputs for the individual members in the series. Pileup occurs when the rising edge of one pulse lies under the peak (specifically the sampling point) of its neighbor. Thus, in Figure 6.6, peaks 1 and 2 are sufficiently well separated so that the leading edge of peak 2 falls after the peak of pulse 1. Because the trapezoidal filter function is symmetrical, this also means that pulse 1's trailing edge also does not fall under the peak of pulse 2. For this to be true, the two pulses must be separated by at least an interval of  $L + G$ . Peaks 2 and 3, which are separated by less than  $1.0\mu s$ , are thus seen to pileup in the present example with a  $1.2\mu s$  rise time.

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

Because the fast filter rise time is only  $0.1\mu s$ , these  $\gamma$ -ray pulses do not pileup in the fast filter channel. The Pixie module can therefore test for slow channel pileup by measuring the fast filter for the interval PEAKSEP after a pulse arrival time. If no second pulse occurs in this interval, then there is no trailing edge pileup and the pulse is validated for acquisition. PEAKSEP is usually set to a value close to  $L + G + 1$ . Pulse 1 passes this test, as shown in Figure 6.6. Pulse 2, however, fails the PEAKSEP test because pulse 3 follows less than  $1.0\mu s$ . Notice, by the symmetry of the trapezoidal filter, if pulse 2 is rejected because of pulse 3, then pulse 3 is similarly rejected because of pulse 2.

## 11.5 Filter Range

To accommodate a wide range of energy filter rise times from tens of nanoseconds to tens of microseconds, the filters are implemented in the FPGA with different clock decimations (filter ranges). The ADC sampling rate is always 8ns (2ns or 4ns in 500 MSPS or 250 MSPS variants), but in higher clock decimations, several ADC samples are averaged before entering the energy filtering logic. In filter range 1,  $2^1$  samples are averaged,  $2^2$  samples in filter range 2, and so on. Since the sum of rise time and flat top is limited to 127 decimated clock cycles, filter time granularity and filter time are limited to the values listed in Table 6.1.

## 11.6 Data Flow

### 11.6.1 Data Capture Process

The data capture in the Pixie-Net XL starts with the detection of a rising edge in the detector signal. At the rising edge, when the trigger filter output goes above the trigger threshold, a *fast trigger* is generated. The *fast trigger* can be blocked by gating signals, for example an external digital VETO signal. At appropriate times after the fast trigger, the following data values are captured into temporary storage:

- Time stamps (immediately)
- Coincidence information (after  $\frac{1}{2}$  of the channel's coincidence window)
- Energy sums (after energy filter rise time + energy filter flat top)
- Final computed pulse height (energy)

- Pileup and out-of-range information (after energy filter rise time + energy filter flat top)
- Optional QDC and PSA sums at the end of the last sum
- Starting address of ADC waveform memory

and the (delayed) waveform data begins to flow into the waveform memory, for the user specified length of trace. When the last of these data capture processes is complete, the data is moved from temporary storage to a *header memory* internal to the FPGA. (Waveform data is not moved again.). This front end header memory can store several hundred events and the front end trace memory can store 8K samples.

### 11.6.2 Data Flow

Traditionally, when the front end memories are not empty, a flag is raised for the ARM processor (strictly speaking, the Linux DAQ program running on the ARM processor). Noticing this flag, the ARM processor reads one record and checks if it is to be recorded per the user defined pileup and coincidence conditions. If so, the ARM processor computes final energies, increments the MCA histogram, reads the list mode data from the FPGA and writes it to file. If the event is piled up or otherwise rejected, it is cleared from the front end memories without recording. When the front end memories are empty, the ARM processor reads baseline measurements and keeps a running average. This traditional mode is rather slow.

To increase the throughput, portions or all of the readout process can be done by the FPGA and data from the front end memories can be buffered in a large SDRAM emmory. This is controlled by the DATA\_FLOW parameter. In the fastest mode, DATA\_FLOW =4, the FPGA determines if an event is to be rejected, if not it is moved to SDRAM storage, and from there send out automatically as a UDP package, as detailed below.

The table below summarizes the DATA\_FLOW options.

DATA_FLOW	Function	LM data file on	RS, MCA file on	Runtype	DAQ function	LM values computed
0	ARM in full control	SD card	SD card	0x100, 0x105, 0x400, 0x401	startdaq	E, PSA, CFD
1	Baseline average from FPGA	SD card	SD card	0x100, 0x105, 0x400, 0x401	startdaq	E, PSA, CFD
2	E and baseline average from FPGA	SD card	SD card	0x100, 0x105, 0x400, 0x401	startdaq	E, PSA, CFD
3	UDP output	Remote PC	SD card	0x100, 0x105, 0x110, 0x111, 0x410, 0x411	startdaq	E, CFD
4	Auto UDP, pileup test in FPGA	Remote PC	SD card	0x100, 0x105, 0x110, 0x111, 0x410, 0x411	startdaq, udpena, udpdis	E
5	MCA only, pileup test in FPGA	n/a	SD card	0x301	mcadaq	E
6	UDP output with DM approval	Remote PC	SD card, (DM)	0x100	netdaq	E, CFD

For DATA\_FLOW=1, the FPGA computes a baseline average from the raw energy filter sums, reducing readout load for the ARM processor. For DATA\_FLOW=2, the FPGA also

computes the final pulse height value E so that the ARM does not need to read any baselines and raw sums at all. For DATA\_FLOW=3, the ARM only reads raw CFD values, computes the final CFD ratio and writes it back to the FPGA, and then directs the FPGA data to the Ethernet output, as a UDP data package that can be received and written to file by a remote PC. For DATA\_FLOW=4, the FPGA sends the UDP packages out automatically, without CFD result, and the ARM's role is limited to incrementing the MCA spectra and monitoring run statistics. If a DAQ is started with `udpena`, the ARM only starts the processing in the FPGA and does nothing more, so no files are created on the SD card until `udpdis` is executed, which stops the FPGA and saves the run statistics only to SD card. For DATA\_FLOW=5, no LM data is stored and a dedicated, simplified DAQ function is used (`mcadag`) to increment MCA and run statistics only. For DATA\_FLOW=6, the UDP data output is further subject to approval of an external “decision maker” (see software triggering, section 12.3).

The table below summarizes the DATA\_FLOW parameter restrictions.

DATA_FLOW	Tracelength	Trace delay	Trace Enable	Veto, out-of-range rejection	Pileup inspection options	NOUT counter
0	Multiple of 32 samples	Any	Any	Applied by FPGA	Applied by ARM	Counts pulses passing pileup test in FPGA
1	Multiple of 32 samples	Any	Any	Applied by FPGA	Applied by ARM	
2	Multiple of 32 samples	Any	Any	Applied by FPGA	Applied by ARM	
3	Multiple of 32 samples,	Any	Any	Applied by FPGA	Applied by ARM	
4	Multiple of 32 samples, All channels equal to ch.0	Any	All channels equal to ch.0	Applied by FPGA	Applied by FPGA	
5	Ignored	Ignored	disabled	Applied by FPGA	Applied by FPGA	
6	Multiple of 32 samples	Any	Any	Applied by FPGA	Applied by ARM	

## 11.7 Dead Time and Run Statistics

### 11.7.1 Definitions

Dead time in the Pixie-Net XL data acquisition can occur at several processing stages. For the purpose of this document, we distinguish three types of dead time (described below), each with a number of contributions from different processes.

Please note: There is a conceptual difference between momentary dead time (associated with a pulse) and cumulative dead time (sum of dead time contributions during an acquisition). Their relation is not trivial.

Live time is often used to describe the portion of the overall time during which the system was not dead. However, since dead time can occur on several levels, this term is prone to misunderstandings and not used here.

### 11.7.1.1 Dead time associated with each pulse

#### 1. Filter dead time

At the most fundamental level, the energy filter implemented in the FPGA requires a certain amount of pulse waveform (the “filter time”) to measure the energy. Once a rising edge of a pulse is detected at time  $T_0$ , the FPGA computes three filter sums using the waveform data from  $T_-$  (a energy filter rise time before  $T_0$ ) to  $T_1$  (a flat top time plus filter rise time after  $T_0$ ), see section 11.4 and figure 11-7. If a second pulse occurs during this time, the energy measurement will be incorrect. Therefore, processing in the FPGA includes pileup rejection which enforces a minimum distance between pulses and validates a pulse for recording only if no more than one pulse occurred from  $T_0$  to  $T_1$ . Consequently, each pulse creates a dead time  $T_d = (T_1 - T_0)$  equal to the filter time. This dead time, simply given by the time to measure the pulse height, is unavoidable unless pulse height measurements are allowed to overlap (which would produce false results).

Assuming randomly occurring pulses, the effect of dead time on the output count rate is governed by Poisson statistics for paralyzable systems with pileup rejection<sup>10</sup>. This means the output count rate  $OCR$  (valid pulses) is a function of filter dead time  $T_d$  and input count rate  $ICR$  given by

$$OCR = ICR * \exp(-ICR * 2 * T_d), \quad (4)$$

which reaches a maximum  $OCR_{max} = ICR_{max}/e$  at  $ICR_{max} = 1/(2*T_d)$ . Simply speaking, the factor 2 for  $T_d$  comes from the fact that not only is an event  $E_2$  invalid when it falls into the dead time of a previous event  $E_1$ , but  $E_1$  is rejected as piled up as well. This filter dead time is accumulated in the SFDT counter in each processing channel.

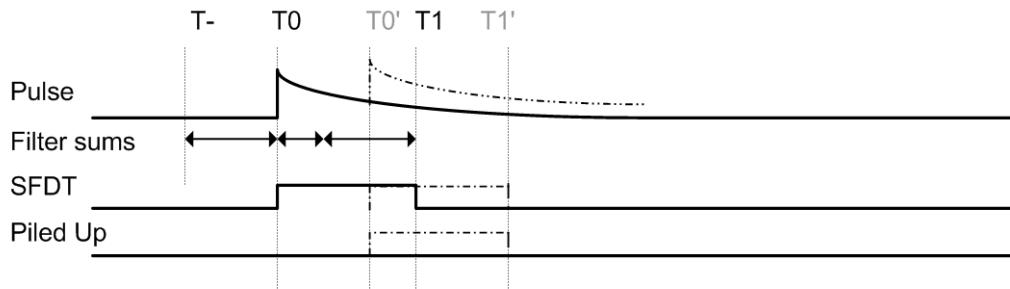


Figure 11-7: Filter dead time. A pulse arriving at  $T_0$  will incur slow filter dead time (for energy measurement) until  $T_1$ . At  $T_1$ , the pileup status is latched – for a single pulse, it is logic low and the event is accepted. A second pulse arriving at  $T_0'$  will extend the dead time and cause the pileup status to be logic high. Unless pileup rejection is disabled, both events are rejected.

#### 2. Fast trigger dead time (FTDT)

A second type of dead time only affects the trigger filter. Triggers are issued when the trigger filter output goes above the trigger threshold set by the user. However, the trigger filter output will remain above threshold for a finite amount of time, depending on the length of the trigger filter and the rise time of the input signal. During this time, no second

<sup>10</sup> G. Knoll, Radiation and Measurement, J Wiley & Sons, Inc, 2000, chapters 4 and 17.

trigger can be issued<sup>11</sup>. Therefore triggers are not counted during this time, and when computing the input count rate, the time lost has to be taken into account. FTDT is thus purely a correction for the computation of the input count rate.

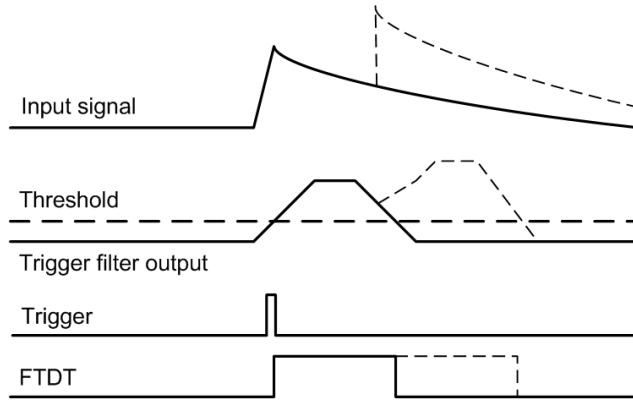


Figure 11-8: Fast Trigger Dead Time (FTDT). A second pulse is not detected if the trigger filter output is still above threshold.

### 3. Other

In the Pixie-Net XL, up to 500 events (and/or total 8Ki waveform samples) are buffered in the FPGA. Thus new events are accepted while captured ones are read out and processed further. If the buffers fill up, the channel pauses acquisition and stops the count time counter.

#### 11.7.1.2 Dead time associated with external conditions

There are three dead time effects that originate from outside the trigger/filter FPGA. The first two have the effect of stopping the Pixie-4 Express count time counter, the last is counted separately.

---

<sup>11</sup> The MAXWIDTH parameter can be used to define a maximum acceptable time over threshold and thus to reject events piled up “on the rising edge”.

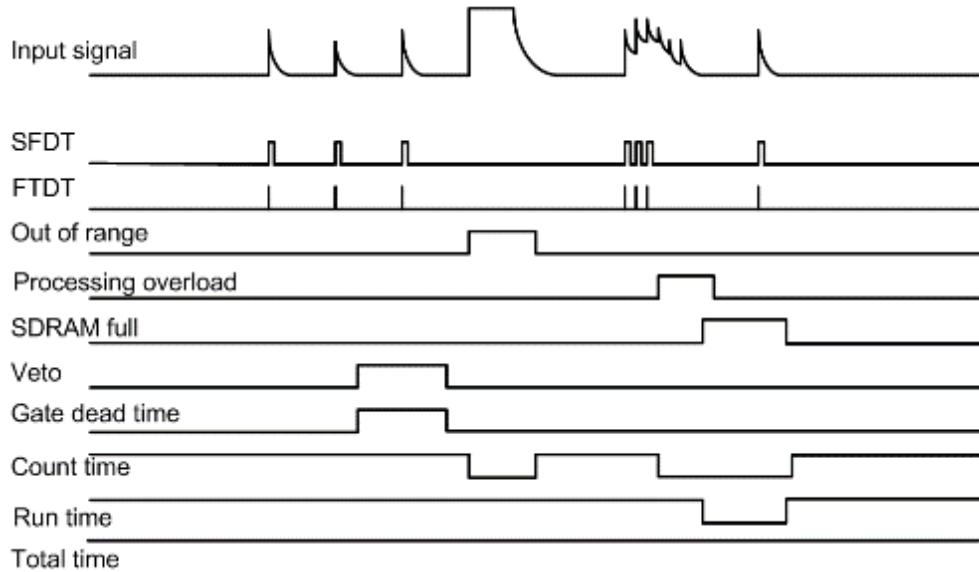


Figure 11-9: The count time counter is stopped when the signal is out of range and when events are rejected because of a processing backlog (e.g. local buffer memory full or file write process busy). SFDT and FTDT are only counted when the count time is on. The gate dead time is counted in a separate counter, but also only when the count time is on. Run time and total time are always on unless the run is stopped (see below).

### 1. Signal out of range

When detector gains or offsets drift, or an unusual large pulse is generated in the detector, the analog input of the ADC may go out of range. In this condition, the FPGA can not accumulate meaningful filter sums and thus is considered dead. This condition persists during the actual out-of-range time and several filter times afterwards until the bad ADC samples are purged from filter memory. The count time counter is stopped during the out-of-range condition because no triggers can be issued and no pulses are counted.

### 2. On-board pulse processing limit

The on-board pulse processing by the ARM processor computes the pulse height (energy) from raw energy filter sums, which is then stored in list mode memory and/or binned into spectrum memory. In the Pixie-Net XL, the computation itself takes only a few cycles, but there is significant readout and other overhead. In List mode with nonzero waveforms, the limit is strongly dependent on the length of the captured waveform. Bursts of pulses may still exceed the processing rate momentarily, fill the buffers, and so prevent the channel from acquiring more data. Thus the count time counter is stopped during such buffer full (processing overload) conditions.

### 3. Gate or Veto

If an external signal prohibits acquisition using the *Gate* or *Veto* signals, the channel is also dead (disabled on purpose). The appropriate way to count *Gate* or *Veto* dead time may depend on the experiment. The Pixie-Net XL counts both gate time and number of gate pulses. Please contact XIA for details.

#### 11.7.2 Count time and dead time counters

The Pixie-Net XL firmware has been optimized to reduce the dead time as much as possible, and a number of counters measure the remaining dead times as well as the number

of counts to provide information for dead time correction. The result of these counters is stored in the following output variables:

### **TOTAL TIME**

The TOTAL TIME is an attempt to measure the real laboratory time during which the Pixie module was requested to take data. It essentially counts the time from the command to start a data acquisition to the command to end it. The TOTAL TIME includes the time spent for run start initialization and host readout. However, since it is based on the Zynq internal clock and only updated periodically, it may not be as precise as a “laboratory wall clock” over long time spans.

### **RUN TIME**

The RUN TIME variable tracks the time during which the Pixie-Net XL was “switched on” for data acquisition. It’s currently identical to the TOTAL TIME

### **COUNT TIME**

The COUNT TIME is counted in the FPGA independently for each channel and measures the time the channel is ready for acquisition. The COUNT TIME counter starts when the ARM processor finished all setup routines at the beginning of a run, omits the times the ADC signal is out of range, each channel's local 500-event buffer is full, and ends when the ARM encounters an end run condition. Internally, the “counter on” signal is called LCE. It is thus the time during which triggers are counted and can cause recording (or pile up) of data, the best available measurement of the time the channel was active. The difference between COUNT TIME and TOTAL TIME can be used to determine how long the local 500-event buffers were full and waiting for readout or other events prevented the channel from data taking (e.g. out of range).

### **FTDT (fast trigger dead time)**

The fast trigger dead time counts the time the trigger filter is unable to issue triggers because the trigger filter output is already above threshold (and can not recognize a second pulse). It does not include the time triggers have been “paused” for a short time after a first trigger (an advanced user option to suppress double triggering), because the concept is that all triggers occurring during the pause are counted as only one trigger. When computing the input count rate, one should divide the number of triggers counted (FASTPEAKS) by the difference (COUNT TIME – FTDT) since triggers are not counted during FTDT.

### **SFDT (slow filter dead time)**

The slow filter dead time counts the time new triggers will not lead to the recording of new data. This is the time the pileup inspection is taking place and the summation of energy filter sums is in progress (section 11.7.1.1). In case pileup inspection is inverted or disabled, there is no contribution to SFDT.

### **GDT (*GATE* dead time)**

The dead time from *Veto* /GFLT is counted separately from SFDT for each channel. As mentioned above, the use of these signals may depend on the application.

In the current firmware, the time during which GDT is counted depends on several user options on signal source and polarity. The source options result in a signal GCE to be counted, the polarity selects whether to count while GCE is high or low, as listed in the following table.

Use Veto	Gate Mode	GCE	Count @ Fall	GDT incremented
0	0	( <i>Veto</i> OR <i>Gate</i> *) AND LCE	0	GCE high
1	0	<i>Gate</i> * AND LCE	0	GCE high

0	1	(Veto OR Gate*)	0	GCE high
1	1	Gate*	0	GCE high
0	0	(Veto OR Gate*) AND LCE	1	GCE low
1	0	Gate* AND LCE	1	GCE low
0	1	(Veto OR Gate*)	1	GCE low
1	1	Gate*	1	GCE low

\* possibly shaped and delayed

For the case that the Veto input is used for a GFLT-type validation pulse, it may be more useful to work with the number of pulses issued. They can be counted by using the *Veto* input as the source for GATE PULSEs, which are counted in the variable GCOUNT.

### 11.7.3 Count Rates

Besides the count time and dead times, the Pixie-Net XL counts the numbers of triggers in each channel, NTRIG, the number of valid single channel events, NUMEVENTS, and the number of valid pulses stored for each channel, NOUT. To accommodate dead time correction for pileup even in cases where events are not recorded for other reasons (e.g. not matching coincidence or veto requirements), a counter NPPI counts the number of locally triggered events passing pileup inspection. In addition, it counts the number of gate pulses for each channel, GCOUNT. FASTPEAKS and GCOUNT are inhibited when the COUNT TIME counter is not incrementing. NUMEVENTS and NOUT by nature only count events captured when the COUNT TIME counter is incrementing.

Count rates are then computed as follows:

Input count rate	ICR = NTRIG / (COUNT TIME – FTDT)
Event rate	ER = NUMEVENTS / RUN TIME
Channel output count rate	OCR = NOUT / COUNT TIME
Channel Pass Pileup Rate	PPR = NPPI / COUNT TIME
Gate count rate	GCR = GCOUNT / COUNT TIME

Users are free to use the reported values to compute rates and time better matching their preferred definitions.

Notes:

- Output pulse counters are updated whenever an event has been processed; input, gate and all time counters are updated every ~7ms. Therefore reading rates at random times, e.g. clicking *Update* in the Pixie Viewer, might return slight inconsistencies between input rates and output rates. At the end of the run, all rates are updated and these effects should disappear.
- NOUT is counted for each event a channel is processed no matter if the channel had a valid hit or not. Thus a channel that is processed in “group trigger” mode may have an output count rate even though its input count rate is zero.
- Since COUNT TIME counters are paused when the header memory is full, the input and output count rates should be considered as “rates while active” as opposed to actual rates per elapsed lab time. For input count rates, this is the more intuitive case, since the detector will not stop generating pulses when the channel becomes inactive due to a full buffer and the input count rate should closely correspond to the detector’s rate. For output count rates, it is a matter of perspective – should it mean the total number of counts per acquisition lab time or the number of counts processed

while the Pixie module is taking date? The former would produce unreasonably low count rates when e.g. the signal goes out of range periodically, since it will not account for the duty cycle of the signal source. The latter would produce unreasonably high rates if the system is near its processing limit and often paused full buffers, though it will better reflect the pileup rejection statistics. The choices made in the current firmware select the latter case, but by multiplying the output count rate with COUNT TIME / TOTAL TIME, the former can be recovered.

#### 11.7.4 Dead time correction in the Pixie-Net XL

Historically, dead time correction in analog systems relied on the system dead time measurements taken directly from the acquisition system and the recorded output count rate. For example, a peak sensing ADC module might output a “dead time” signal during the several microseconds it would require to capture the peak value. Thus reconstruction of true count rate required the knowledge of dead times associated with various stages of acquisition and the subsequent mathematical modeling to tie this quantity to the input rate. The classic paralyzable and non-paralyzable models of pulse acquisition do exactly that. For example, the dead time from a non-paralyzable ADC conversion process simply “takes away” active counting time ( $T_d$  for each output count) and so one can use the classical model of  $OCR = ICR/(1+T_d*ICR)$ , derive<sup>12</sup>  $OCR/ICR = (real\ time - dead\ time)/(real\ time)$ , and solve for  $ICR$  as a function of measured  $OCR$ , real time and dead time.

In the Pixie-Net XL, the input count rate is measured directly with the trigger filter, and so the system dead time bears only theoretical or diagnostic value. For any measurements where accurate determination of true (source) counts are required (activity measurements), the empirical ratio  $ICR/OCR$  is the only really unbiased quantity for dead time correction. No matter what the actual dead time the acquisition process incurs on the system, the  $ICR/OCR$  ratio applied to any region of interest in the energy spectrum correctly reconstructs the true counts in this region, assuming a random source so that pulses are lost with equal probability in each region or in time. **For cases where events are added by group trigger or removed by coincidence or veto requirements, PPR should be used instead of OCR.**

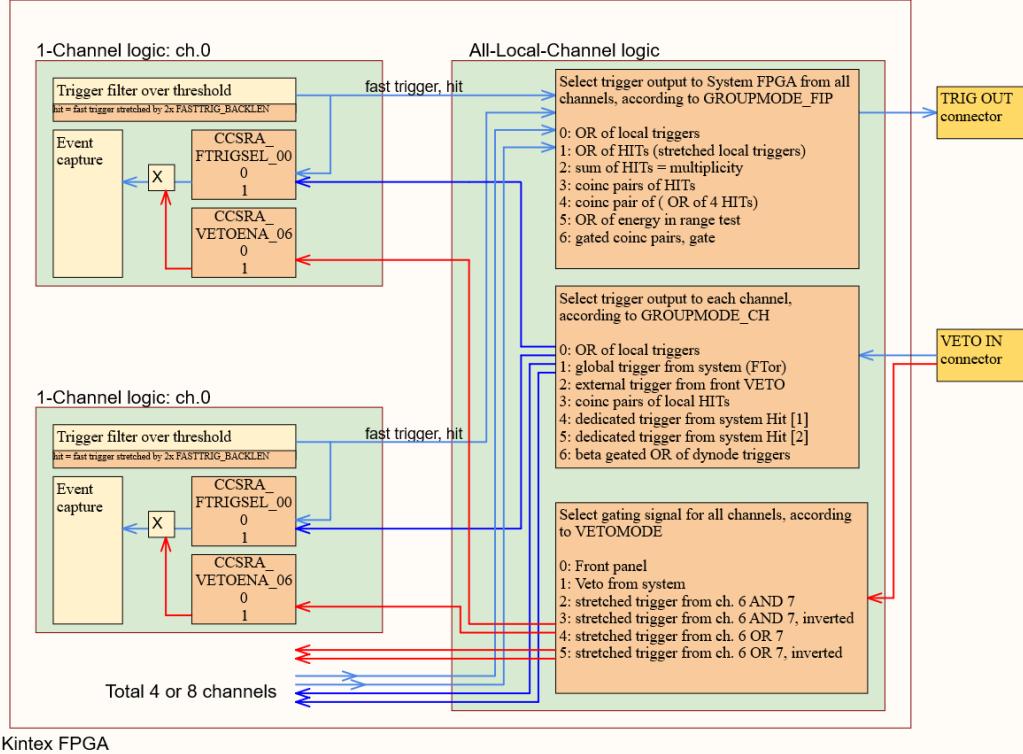
Please also see a related application note: XIA Pixie-4e Dead Time Correction

---

<sup>12</sup>  $OCR(1+T_d*ICR) = ICR$  can be written as  $OCR = ICR(1-OCR*Td)$ . The measured cumulative dead time DT is  $DT = OCR*Td*RT$ . Therefore  $OCR/ICR = RT-DT/RT$ .

## 11.8 Trigger Distribution

Every pulse processing channel of the Pixie-Net XL can share its local trigger with other channels. This trigger distribution is implemented in the following way:



Each channel generates a “local fast trigger” when the trigger filter goes over the user defined threshold. In a pure local trigger mode (parameter `CCSRA_FTRIGSEL_00 = 0`), this signal is used for the event capture of this channel and the remaining trigger distribution logic is unused. In distributed (or group) trigger mode, triggering uses the local fast trigger’s connection to a common “all-local-channel” logic block that can combine signals from the 4 or 8 channels within an FPGA. The common logic block has two functions:

1. Generate a trigger output signal from all channels, with the choices defined by `GROUPMODE_FIP`:
  - 0: build an OR of local triggers. This is a 1-cycle pulse
  - 1: stretch the local triggers by  $2 \times \text{FASTTRIG\_BACKLEN}$  (=hit), then build an OR
  - 2: add the hit signals into a 4-bit multiplicity sum (Pixie-16 x1/x2 only)
  - 3: make a pulse when 2 neighboring hit signals are high (Pixie-16 x1/x2 only)
  - 4: make a pulse when any hit of ch. 0-3 AND any hit of ch. 4-7 are high
  - 5: build an OR of the “energy in range” test for all channels
  - 6+: additional custom modes
 The trigger signal is output on the TRIG OUT connector on the Pixie-Net XL rear panel. (Internal jumpers must be set appropriately, for Kintex 0 only.) For choices 2 and 3, which produce 4 output bits, the result is currently only available on internal test points.
2. Generate a trigger for pulse processing for each channel (an alternate to the local trigger named “group trigger”), with the choices defined by `GROUPMODE_CH`:
  - 0: build an OR of local triggers

- 1: use trigger from system logic (Pixie-16 x1/x2 only)
- 2: make a pulse from the signal at the VETO IN front panel connector. Use MCSRA\_FPPEDGE\_07 to choose rising (1) or falling (0) edge
- 3: make a pulse when 2 neighboring hit signals are both high
- 4+: additional custom modes  
CCSRA\_FTRIGSEL\_00 must be 1 to use the group trigger for event capture in the channel.

Notes:

The pattern of hit signals, latched at time FASTTRIG\_BACKLEN after a trigger, is also part of the list mode event information, independent of the trigger settings. This allows offline selection of coincident events even when running in local trigger mode.

When triggering a channel with another channel or with an external signal, the energy computation will likely not be correct (the external trigger is not related to the rising edge of the local signal). Set CCSRC\_LOCAL\_ENERGY\_09 to 1 to use the local trigger for the energy computation in group trigger mode.

## 11.9 Gating

### 11.9.1 Gating the Fast Trigger

As described in section 11.8, the fast trigger is the key signal to capturing data for pulse processing. For purposes of gating unwanted triggers, the selected fast trigger can be blocked by a VETO signal if CCSRA\_VETOENAL\_06 = 1. The source of the VETO signal is selected by the VETOMODE parameter (applies to all channels). Choices are

- 0: Front panel input (requires configuration of the MMCX as input)
- 1: A signal from the System FPGA (Pixie-16 x1/x2 only)
- 2+: additional custom modes

Following traditional Pixie-16 handling of the gating, a VETO signal has the following effects

- Does suppress the generation of valid fast triggers, so
  - no event record generated
  - no deadtime for readout
  - no contribution to output counts
  - no contribution to passed pileup counts
- Does not suppress counting of input counts
- Does not suppress generation of a fast trigger output signal for coincidence purposes
- Does not suppress the pileup inspection logic, i.e. a vetoed pulse will still flag a closely following non-vetoed pulse as piled up
- The time VETO is blocking pulses is counted in a “gate dead time” counter.

### 11.9.2 Gating the Event Capture

It is sometimes not possible to provide a gating signal at the time of fast trigger. For example, external logic may use the fast trigger output signal(s) or signals from other devices to make an accept/reject decision that is valid a certain time after the fast trigger from a single detector. In some applications, this has been called a Global Second Level trigger (GSLT). The TTCL logic can be used to apply gating at the time the energy sums are latched, several microseconds past the fast trigger. Even without a TTCL adapter board, the front panel VETO connector can be routed to the input of the TTCL logic. A rising edge on the front panel input (after optional inversion) is stretched by the TTCL approval window and if that signal is logic high when the energy filter sums are latched, the pulse is considered acceptable. While the length of the TTCL approval window is programmable, the delay to match the energy filter must be generated by the external logic. The effects of the TTCL acceptance are as follows:

- Does suppress recording of the event, similar to the pileup test
  - no event record generated
  - smaller deadtime for (partial) readout
  - no contribution to output counts
- Does not suppress the generation of fast triggers
- Does not suppress generation of a fast trigger output signal for coincidence purposes
- Does not suppress the pileup inspection logic i.e. a rejected pulse will still flag a closely following pulse as piled up
- Does not suppress contribution to the passed pileup counts

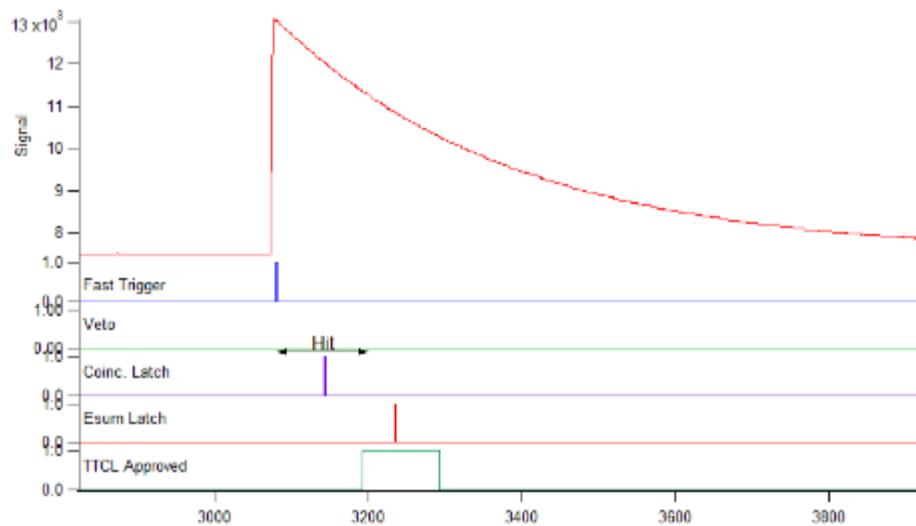
### 11.9.3 Gating Notes

Both Veto and TTCL logic are affected by the programmable input delay EXTERN\_DELAYLEN that acts like a delay cable in front of the Pixie module.

To assist in the setup of the delays, ADC traces can be reported with integrated information for triggers and gates. If CCSRC\_GATEADC\_11 is 1, the following bits of the untriggered ADC traces are replaced:

Bit	Signal
0	TTCL approved
1	latch to capture energy sums
2	latch to capture coincident information
3	veto
4	local fast trigger

ADC traces can be decoded to show these signals separately



Note that the “hit” signal indicating that this channel saw a pulse stays high 2x as long as FASTTRIG\_BACKLEN, the time from fast trigger to coincidence trigger, and can be captured by other channels with their local (or distributed) coincidence capture latch.

Since the latches and triggers are stretched to be only 4 clock cycles long, it is recommended to make the ADC sampling interval (XDT) at most 0.032us to ensure the signals are not missed in the waveforms.

# 12 Hardware and Firmware Special Functions

This sections describe hardware and firmware variants of the Pixie-Net XL. Typically, they are purchased separately as a licensed add-on to the standard hardware, firmware, and software. Please contact XIA for details.

## 12.1 White Rabbit Time Synchronization

The White Rabbit (WR) time synchronization is a standard feature of the Pixie-Net XL hardware and firmware. It is not required for standalone operation of a single Pixie-Net XL. However, it can be used to synchronize the time stamps between two or more Pixie-Net XL or between a WR master and a Pixie-Net XL. The Pixie-Net XL thus reports list mode data files where event time stamps are given in date/time of the WR clock master, to sub-nanosecond precision.

### 12.1.1 Setup

By default, the Pixie-Net XL is configured as a WR clock slave on ports SFP 0,1 or Z. This requires a WR slave compatible SFP module to be used in SFP 1 (**the blue one**). The optical fiber is then connected to the SFP module of a WR master (**the purple one**), for example a WR-LEN, a WR switch, or a Pixie-Net XL configured as WR master.

The “LINK” LED should turn on when the Pixie-Net XL is properly booted and connected to an Ethernet host, not necessarily with WR capabilities. The LED does not indicate WR synchronization status.

In the “10G+WR” option, ports SFP 0,1 and the FPGAs associated with them operate on the clock controlled by SFP Z and its associated FPGA.

The WR shell UART interface of the WR FPGA is connected to the UART port of the Pixie-Net XL’s Zynq controller<sup>13</sup>. This can be used to change the WR configuration of the Pixie-Net XL, for example IP numbers, or to change from WR clock slave to WR clock master (Remember to change SFP module also). In the Pixie-Net XL terminal window, type

```
minicom
```

to open the UART session. At the `wrc` prompt, enter commands as described in the WR user manual, section 4 and appendix A. Commonly used commands are

<code>ptp stop</code>	best to stop synchronization before making changes
<code>mode slave</code>	change to WR clock slave mode
<code>mode master</code>	change to WR clock master mode
<code>ptp start</code>	restart synchronization after making changes
<code>gui</code>	see status information in a text GUI. Use “ESC” to exit

Master/slave mode, MAC address, IP address etc can be saved on the on-board WR PROM for automatic configuration of the WR core after booting. Exit minicom by typing `<ctrl> a q` . This does not stop the WR core operations.

<sup>13</sup> More complicated in the 10G+WR and PZ option

### 12.1.2 Data Acquisition

With the WR core active and connected to a WR clock master, the Pixie-Net XL FPGA operates on a clock that is synchronized in phase and frequency to the clock master and has access to the WR time/date of the master. The synchronized clock is used for the ADCs and pulse processing (with suitable multiplication/division of frequency, if necessary). The WR time/date is used in the firmware and software as follows

- In the software (startdaq), if the parameter WR\_RUNTIME\_CTRL is set to 1 in settings.ini, startdaq reads the WR time/date prior to starting the data acquisition, computes the target start and stop time (at 10s intervals) and writes it to the FPGA. The data acquisition is then enabled as usual.
- In the firmware, if the parameter WR\_RUNTIME\_CTRL is set to 1 in settings.ini, the FPGA will only take data while the WR time is between target start and stop time
- In the firmware, list mode events always record the WR time/date in the “external time stamp” field. The “trigger time” is reset at the WR start time and increments in sync with the WR clocks.

To facilitate data acquisition with multiple Pixie-Net XL modules, a number of Linux shell scripts have been developed for booting, data acquisition, and file transfer from/to a host PC. They require the correct Ethernet IP of the Zynq controller as a parameter hard coded in the files.

- wrsetup\_2x.sh  
This function goes through the steps of booting FPGAs, applying parameters, and adjusting some settings. It needs to be executed once after powerup
- wrdaq\_2x.sh  
This function asks for the requested run time, changes the settings file, and calls ./startdaq on both Pixie-Net XL.
- wrcollect\_2x.sh  
This function copies the binary list mode, MCA, and run statistics files from the Pixie-Net XL to the host PC (local directory) and renames them with \_[IPaddr].
- wrhalt\_2x  
This function shuts down Linux on two Pixie-Net XL units.

By using these scripts, it is no longer necessary to log on to each Pixie-Net XL and type commands manually. However, the modules must be set up for automatic SSH with a public key (see notes in wrsetup\_2x.sh).

Note: It is also possible to specify the WR start time via the web API as part of the DAQ start. In this case, the remote program has to first get the current WR time/date and add a sufficient offset to accommodate the time required for the web write transaction.

### 12.1.3 Output Data

The WR time/date is reported in the List Mode data as generated by the ./startdaq function:

- In run type 0x400, the time/date is reported in previously unused fields of the channel header:

- A 32 bit number  $TS_{low}$  reported in words 18 and 19 of the channel header is 16 times the number of clock cycles in the current WR second, i.e.  
 $TS_{low} = CH[18] + CH[19]*65536$  (in ns)  
For the WR clock operating at 62.5 MHz, or 16ns per cycle, the result  $TS_{low}$  is in units of 1 ns.
- A 16 bit number  $TS_{high}$  reported in word 20 of the channel header is the lower 16 bit of the current WR second (nominally since 1970)  
 $TS_{high} = CH[20]$  (in s)
- Word 21 of the channel header is reserved for higher bits of the WR second, currently all zero
- In run type 0x100, the time/date is reported in the “external time” fields of the event header (32bit format):
  - A 32 bit number  $TS_{low}$  reported in words 16 of the event header is 16 times the number of clock cycles in the current WR second, i.e.  
 $TS_{low} = EH[16]$  (in ns)  
For the WR clock operating at 62.5 MHz, or 16ns per cycle, the result  $TS_{low}$  is in units of 1 ns.
  - A 16 bit number  $TS_{high}$  reported in word 17 of the event header is the lower 16 bit of the current WR second (nominally since 1970)  
 $TS_{high} = EH[17]$  (in s)
  - The upper 16 bit of event header word 17 is reserved for higher bits of the WR second, currently all zero

#### 12.1.4 Ethernet Data Output

The WR logic implements an Ethernet data interface in addition to the clock synchronization. This can be used to output list mode data from the FPGA to a remote data receiver, without the need of the data flowing through the Zynq processor with slow I/O. At the time of writing, the Ethernet data output is using the UDP protocol, with one event per UDP package. A host PC or equivalent device is required to receive and store the UDP data. This PC can be the same as the one used to communicate with the Zynq controller or a different one; however the Zynq control connects to the Pixie-Net XL via 10/100/1000 Ethernet (copper) and the UDP data output connects through a WR switch via 1Gbps fiberoptic to each of the Pixie-Net XL’s FPGA SFP ports.

The WR Ethernet data output logic currently only supports list mode data format 0x100. For debug or diagnostic purposes, there are different modes of Zynq involvement in the data flow, which is specified by the DATA\_FLOW parameter in the settings file (see section 11.6). DATA\_FLOW=4 is the recommended mode for normal purposes.

On the receiving PC, a simple program can receive UDP packages and store them to disk. Example programs for Windows and Linux are provided in the Pixie-Net software release package.

The WR logic requires the destination MAC address and source/destination IP addresses and ports to be specified in the settings file, for each of the 2 FPGAs. The destination can be the same for both FPGAs. Source MAC is taken from the WR PROM, which is programmed through the WR UART interface.

## 12.2 Pulse Shape Analysis

A variety of radiation detectors have the ability to discern between different types of radiation, such as neutrons and gamma rays, by creating different pulse shapes for different types of interactions. In addition, phoswich detectors, consisting of multiple layers of different scintillators, produce different pulse shapes depending on which layer absorbs the radiation. A suitable pulse shape analysis (PSA) can detect such differences, which then can be used for particle identification. The Pixie-Net XL firmware includes such PSA functions.

### 12.2.1 Overview

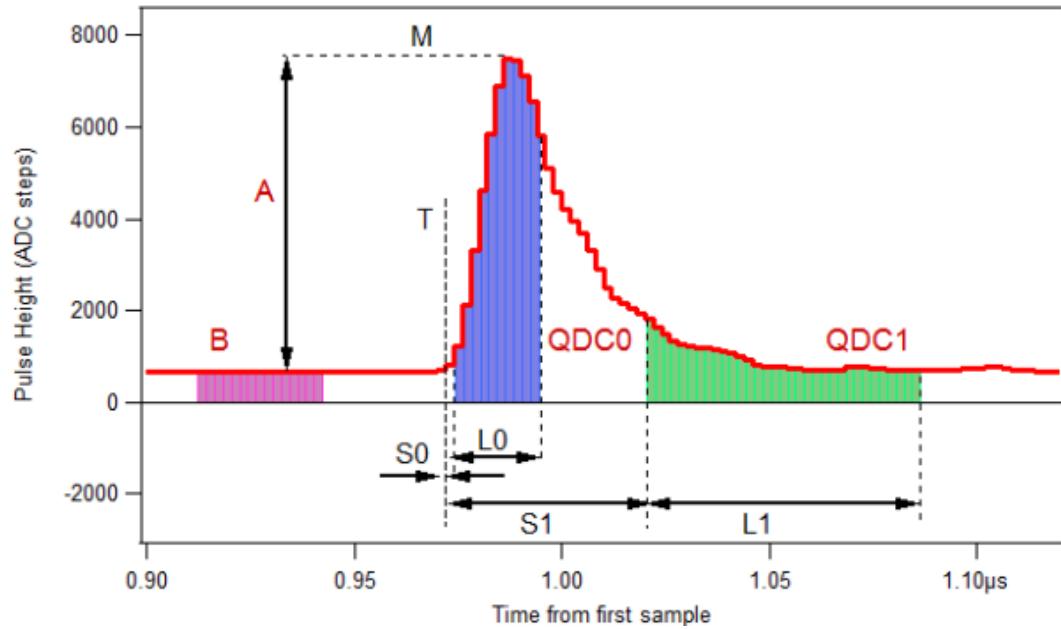


Figure 12-1: The PSA quantities and input parameters

The approach used by the Pixie-Net XL is a digital version of the Charge Comparison Method, where two sums over characteristic regions of the pulse are accumulated and a suitable ratio expresses the difference in pulse shape. This is an established method used in a variety of applications, well suited for online processing due to its simplicity. In this case, the functions compute baseline average sum  $B$ , amplitude  $A$ , and two sums  $QDC0$  and  $QDC1$  over characteristic areas of the pulse, as shown in Fig.1. The first 8 samples of the waveform are summed and normalized to obtain  $B$ . The maximum sample  $M$  in the waveform is located, then subtracted by  $B$  to obtain  $A$ . Four input parameters ( $L0$ ,  $L1$ ,  $S0$ ,  $S1$ ), specified prior to the data acquisition, define the length and delay of the sums  $QDC0$  and  $QDC1$  relative to the trigger  $T$ , as shown in Fig. 1. The sums  $QDC0$  and  $QDC1$  are baseline subtracted by  $B$  (scaled according to  $L0$  and  $L1$ , respectively). Another return value, or PSA ratio  $R = QDC1/QDC0$  is computed as the final result to differentiate pulse types. (Other ratios or combinations can be implemented on request). These data, plus the timestamp  $TrigTime$  and overall pulse height  $E$  computed with a trapezoidal filter, are written into the list mode data stream, followed by optional storage of the full waveform for each event.

### 12.2.2 PSA Input Parameters

PSA input parameters are specified in the settings file. For each channel, the parameter QDC0\_LENGTH corresponds to L0, QDC1\_LENGTH corresponds to L1, QDC0\_DELAY corresponds to S0, and QDC1\_DELAY corresponds to S1. In addition, the Boolean QDC\_DIV8 can be set to 1 for long sums, where the result of QDC0 or QDC1 may overflow its 16-bit output variable. The parameter PSA\_THRESHOLD is used to detect the rising edge of a pulse. The threshold applies to both sums. QDC\_DIV can be 4 or 32 and divides all sums by this factor (to keep within the range of a 16-bit number).

The sum lengths and delays can be set individually, with the following limitations:

- QDC1 must finish last (i.e. S1+L1 > S0+L0)
- QDC1 must finish within the energy filter (for pileup rejection)
- L0, L1 must be between 2 and 60, and a multiple of 2
- L0+S0, L1+S1 must be between 0 and 250
- The difference of S0 and S1 must be a multiple of 2 (e.g S0 = 1, S1 = 17)

In the current settings files, there are 8 sets of QDC length and delay, but only 0 and 1 are used. The others are only used in Pixie-16-style QDC sums (see Pixie-16 manual).

### 12.2.3 PSA Return Values

The ARM processor writes the PSA return values to the list mode data stream. The PSA values are placed into words 10-15 of the channel header in run type 0x400 as described above. Table 11-1 lists the mapping of the values to the channel header locations. To accommodate fractions for the ratio R, the number recorded in the file is 1000\*R.

For text output in Run Type 0x401, 0x500 and 0x502, see section 7.

Word #	Variable	Description
0	EvtPattern	Hit pattern.
1	EvtInfo	Event status flags.
2	NumTraceBlks	Number of blocks of Trace data to follow the header
3	NumTraceBlksPrev	Number of blocks of Trace data in previous record (for parsing back)
4	TrigTimeLO	Trigger time, low word
5	TrigTimeMI	Trigger time, middle word
6	TrigTimeHI	Trigger time, high word
7	TrigTimeX	Trigger time, extra 8 bits
8	Energy	Pulse Height
9	ChanNo	Channel number
10	PSA Values	Amplitude A
11	PSA Values	reserved
12	PSA Values	Base B
13	PSA Values	QDC0
14	PSA Values	QDC1
15	PSA Values	Ratio R *1000
16-32	reserved	

Table 11-1: Channel header data format for PSA data acquisition in Run Type 0x400.

### 12.2.4 Reduced General Purpose Functionality

None

## 12.3 Constant Fraction Timing (CFD)

The following CFD algorithm is implemented in the signal processing FPGA of the 100 MHz and 250 MHz modules. Assume the digitized waveform can be represented by data series  $Trace[i]$ ,  $i = 0, 1, 2, \dots$ . First the fast filter response (FF) of the digitized waveform is computed as follows:

$$FF[i] = \sum_{j=i-(FL-1)}^i Trace[j] - \sum_{j=i-(2*FL+FG-1)}^{i-(FL+FG)} Trace[j] \quad (12-1)$$

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

$$CFD[i + D] = FF[i + D] * (1 - w/8) - FF[i] \quad (12-2)$$

Where  $D$  is called the CFD delay length and  $w$  is called the CFD scaling factor ( $w=0, 2, \dots, 7$ ).<sup>14</sup>

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

$$f = \frac{CFDout1}{CFDout1 - CFDout2} \quad (12-3)$$

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

$$CFD = \frac{CFDout1}{CFDout1 - CFDout2} \times N \quad (12-4)$$

Where  $N$  is scaling factor, which equals to 32768 for 100 MHz modules and 16384 for 250 MHz modules, respectively. This result however is not available in UDP streaming modes (DATA\_FLOW 3 or 4). Instead, it has to be computed by the receiving program (or offline).

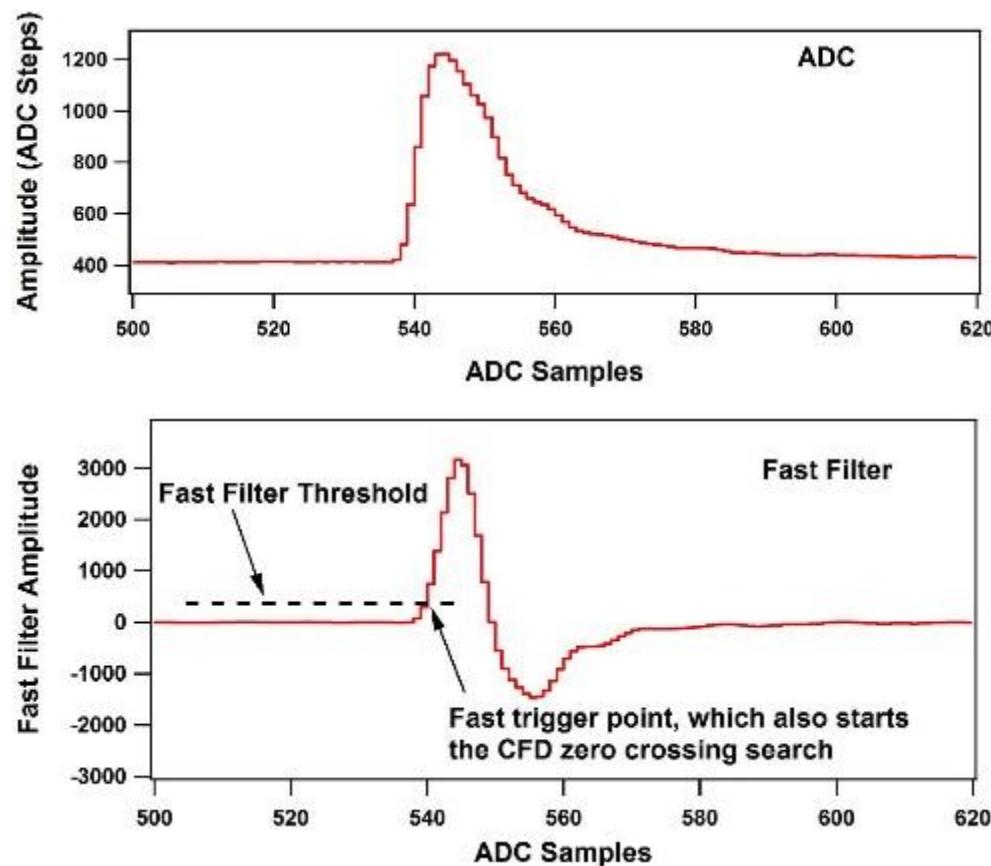
Figure 12-2 shows a sample ADC trace, its fast filter response and its CFD response, respectively. The top figure shows a raw ADC trace. After computing the fast filter response on the raw ADC trace using Equation 12-1, the fast filter response is compared against the fast filter threshold as shown in the middle figure. The ADC sample where the fast filter response crosses the fast filter threshold is called the fast trigger point, which also starts the search for the CFD zero crossing point. The CFD response is computed using Equation 12-2 and is shown in the bottom figure (for actual implementation in the firmware, the fast filter response  $FF$  is delayed slightly before being used for computing the CFD response so that there are sufficient number of CFD response points to look for the zero crossing point after the fast trigger). To prevent premature CFD trigger as a result of the noise in the CFD response before the actual trigger, a DSP parameter called  $CFDThresh$  is used to suppress those noise-caused zero crossing. However, if a zero crossing point cannot be found within a certain period after the fast trigger (typically 32 clock cycles), e.g., due to unnecessarily high  $CFDThresh$ , a forced CFD Trigger will be issued and a flag will be set in an event header word to indicate that the recorded CFD time for this event is invalid. However, the event will still have a valid timestamp which is

<sup>14</sup>  $w=1$  is not supported. It is built from 3 fractions of 2 ( $1/2 + 1/4 + 1/8$ ) and therefore needs more FPGA resources

latched by the fast filter trigger when fast filter crosses over the trigger threshold. The aforementioned CFD parameters correspond to the following processing parameters.

Table 12-1 Corresponding Parameters for the CFD Parameters

CFD Parameters	Parameters in the settings file	Units
FL	TRIGGER_RISETIME	$\mu\text{s}$
FG	TRIGGER_FLATTOP	$\mu\text{s}$
Fast Filter Threshold	TRIGGER_THRESHOLD	
D	CFD_DELAY	8 ns cycles
W	CFD_SCALE (valid values: 0, 2, 3, ... 7)	1-w/8
CFD Threshold	CFD_THRESHOLD	



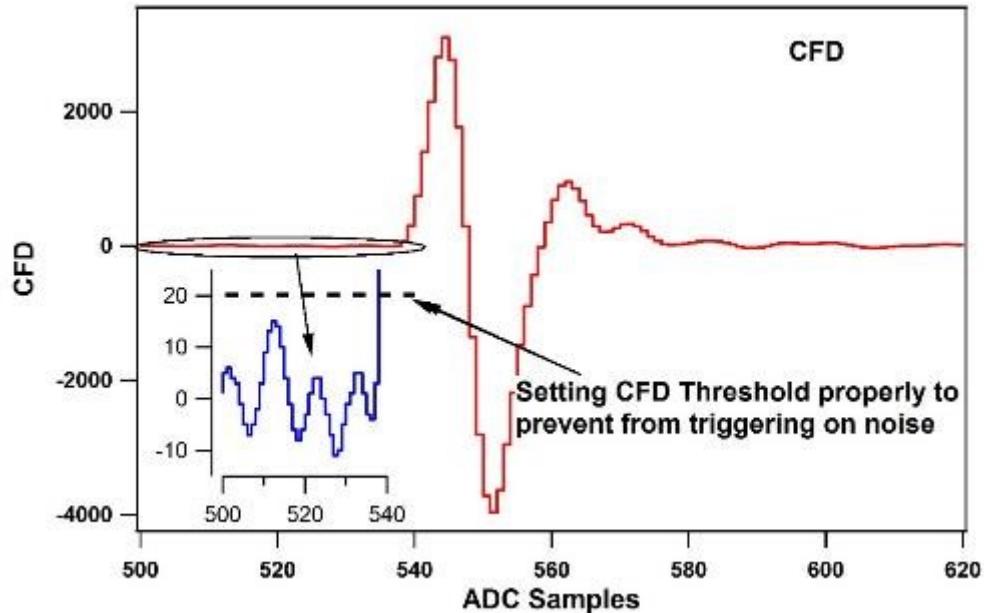


Figure 12-2: Illustration of the Pixie-16 CFD algorithm

In the 250 MHz Pixie-Net XL modules, the event timestamp is counted with 125 MHz clock ticks, i.e., 8 ns intervals, and two consecutive 250 MHz ADC samples are captured in one 8 ns interval as well. The CFD trigger also runs at 125 MHz, but the CFD zero crossing point is still reported as a fractional time between two neighboring 250 MHz ADC samples, which are processed by the FPGA in one 125 MHz clock cycle. However, the CFD zero crossing point could be in either the odd or even clock cycle of the captured 250 MHz ADC waveforms. Therefore, the firmware outputs a "CFD trigger source" bit in the output data stream to indicate whether the CFD zero crossing point is in the odd or even clock cycle of the captured 250 MHz ADC waveforms.

The CFD results are computed as 4 raw words in the FPGA. In Run Types and Data Flow modes where the DSP/ARM handles the readout, a final CFD result (ratio) is computed, but overall throughput is reduced. A run type may report the raw words or the final result or both, see table below. If waveforms are recorded, they can be used for offline CFD analysis in any Run Type.

Run Type	Data Flow	Online CFD (final ratio)	Online CFD (4 FPGA words)	Offline CFD (waveforms)
0x100	0,1,2,3	Yes	No	Yes
0x100	4	No	No	Yes
0x105	0,1,2,3	Yes	Yes	Yes
0x105	4	No	Yes	Yes
0x110	3	Yes	Yes	Yes
0x110	4	No	No	Yes
0x111	3	Yes	Yes	Yes

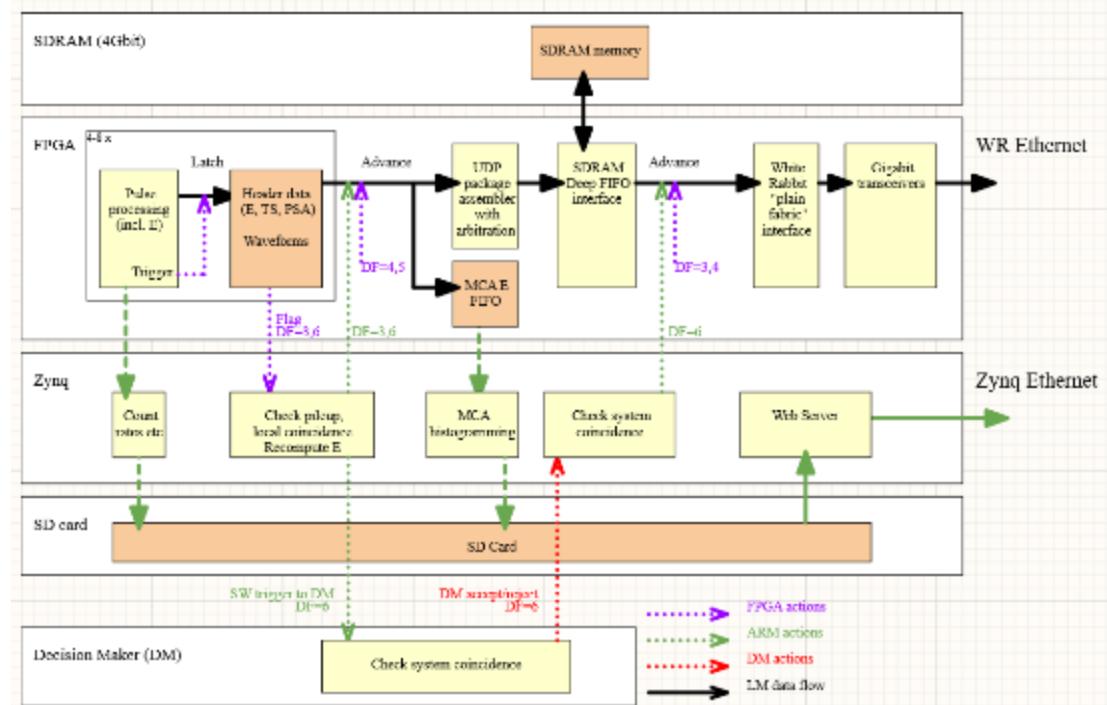
0x111	4	No	Ye	Yes
0x400	0,1,2	Yes	No	Yes
0x401	0,1,2	Yes	No	Yes
0x410	3	Yes	Yes	Yes
0x410	4	No	Yes	Yes
0x411	3,4	No	Yes	Yes

## 12.4 Software Triggering (discontinued)

### 12.4.1 Pixie-Net XL Firmware

Software triggering is a new concept in the Pixie-Net XL. The idea is that external, hard wired clock and trigger signals can be replaced by software communication. The purpose of external *clock* distribution is to ensure timestamps and other data are synchronized in multiple independent DAQ units. In the Pixie-Net XL, this functionality is provided by the White Rabbit clock synchronization. The purpose of external *trigger* distribution is usually to reduce data collected in a DAQ channel by applying coincidence or multiplicity requirements. (Another purpose is triggering acquisition in channels that do not see local pulses, which is not yet implemented.)

The software triggering concept of the Pixie-Net is outlined in the data flow block diagram below. The pulse processing FPGA, for each channel, latches header data and waveforms when it detects a detector pulse in local FPGA memory. The header data includes energy filter sums, time stamps, CFD/PSA data and also the pulse height E computed by the FPGA. When data is in the header memory, a flag is raised for the Zynq processor (DATA\_FLOW=3 or 6), which applies a local coincidence test before advancing the data to the next memory buffer stage, a large external SDRAM, and a local FPGA buffer for MCA histogramming, or rejecting the event altogether. (For DATA\_FLOW = 4 or 5, the data is automatically advanced to the next memory stage by FPGA logic). In DATA\_FLOW = 6, the Zynq processor also sends a software trigger message to the external DM. The DM collects trigger messages from multiple Pixie-Net XL DAQ modules, then sends and accept/reject message to the Pixie-Net XL. The Zynq processor then advances the data from the SDRAM to the WR Ethernet output or flushes it out of the memory. (In DATA\_FLOW = 3 or 4, the SDRAM data is always advanced to the WR Ethernet output by FPGA logic as all decisions have already been made locally. DATA\_FLOW = 5 is used for MCA-only acquisition, no LM data is stored). To accommodate latencies in the DM message exchange, the accept/reject messaged passed on by the Zynq to the FPGA are also buffered in a small FIFO.



List mode data is therefore first buffered in front end buffers holding ~500 events to give the Zynq time to make local coincidence decisions. It is further buffered in the SDRAM memory to give the DM time for system-wide coincidence decisions. Monitoring data is accumulated on the Zynq's SD card for local storage and readout through a web server. (In diagnostic modes DATA\_FLOW <3, list mode data is also stored on the SD card.)

#### 12.4.2 Decision Maker Software

(insert description here)

## 12.5 TTCL Compatible Clock and Trigger Distribution

A HW version that supports TTCL I/O is under development. It requires a modified clocking board with dedicated SFP connectors.

(under development)

## 12.6 QDC

Eight QDC sums, each of which can have different lengths, are computed in the Signal Processing FPGA of a Pixie-Net XL module for each channel and the sums are written to the list mode output data stream if the user requests so. The recording of QDC sums starts at the waveform point which is *Pre-trigger Trace Length* or *Trace Delay* earlier than the trigger point, which is either the CFD trigger or channel fast trigger depending on whether or not CFD trigger mode is enabled. The eight QDC sums are computed one by one continuously, but they are not overlapping. The recording of QDC sums ends when the eight intervals have all passed.

Either PSA or QDC sums can be enabled, not both. PSA sums are more precisely located at the rising edge of the pulse; QDC sums can extend over much longer intervals.

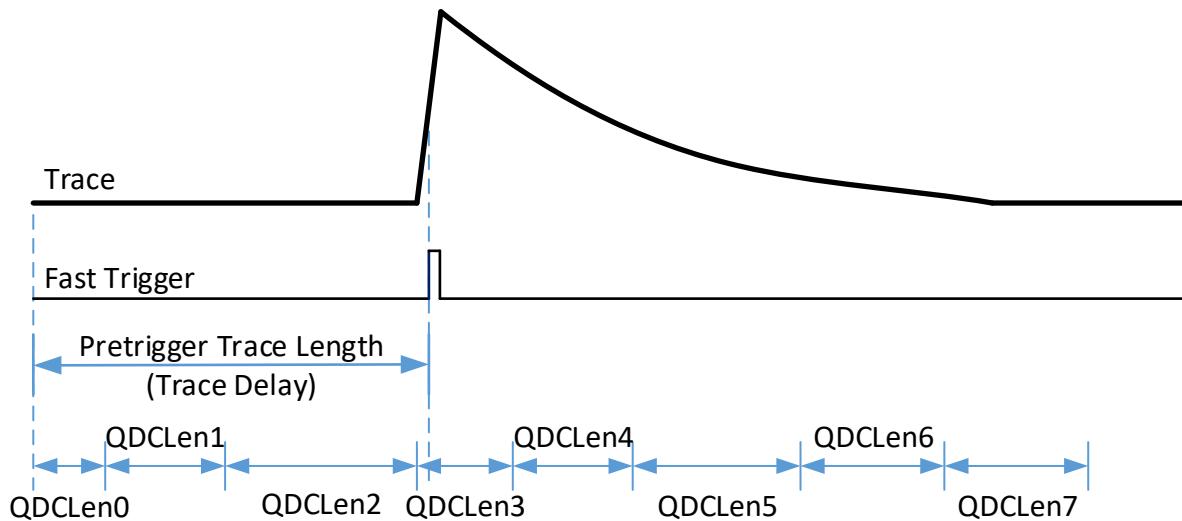


Figure 12-3: The 8 QDC sums of a triggered event.

# 13 Hardware Information

## 13.1 Access to the PCB hardware

To access the internal jumpers, ADC daughtercard switches, or clocking daughterboards please follow these steps:

- Remove power and USB connections
- Remove the 4 hex screws on the front panel (with SFPs) and the 4 hex screws on the rear panel (with power).
- Detach rear panel, then disconnect the fan cable connector (blue wire: left),
- Slide the board assembly out of the box towards the front, keeping the front panel snapped into the SFP connectors
- Locate the jumpers at the bottom side of the (green) motherboard.
- Reverse steps for re-assembly.

## 13.2 Jumpers

### 13.2.1 Revision A Modules

Pixie-Net XL Revision A has the following internal jumpers

- **JP400, JP401**  
When a shunt is connecting pin 1 and pin 2 in each jumper (position “=”), the rear panel PMOD pinout is identical to the MicroZed PMOD  
When a shunt is connecting both pin 1 and another shunt is connecting both pin 2 (position “||”), the rear panel PMOD pinout has pins 2 and 3 swapped. This is useful for some UART PMOD devices, e.g. the Digilent GPS module.  
**Currently used exclusively for WR UART**
- **JP622 (A, B)**  
When a shunt is placed on JP622 in position “loc”, the PTP Ethernet PHY is operating on a local clock. In position “sys” it is operating on the clock controlled by the WR slave logic.
- **JP800**  
When a shunt is placed on JP800 in position “in” the MMCX connector “ptpclk” is used as an input. In position “out” it is an output of the clock controlled by the PTP or WR logic (PTP 1 or SFP 1).
- **JP911, JP912**  
A shunt on these jumpers controls the input to the main clock tree:
  - in position “EXT”, the MMCX connector “ptpclk” is used (see also JP800)
  - in position “PTP”, the PTP synchronized clock is used (PTP 1)
  - in position “LOC”, the WR controlled local clock oscillators are used (SFP 1)

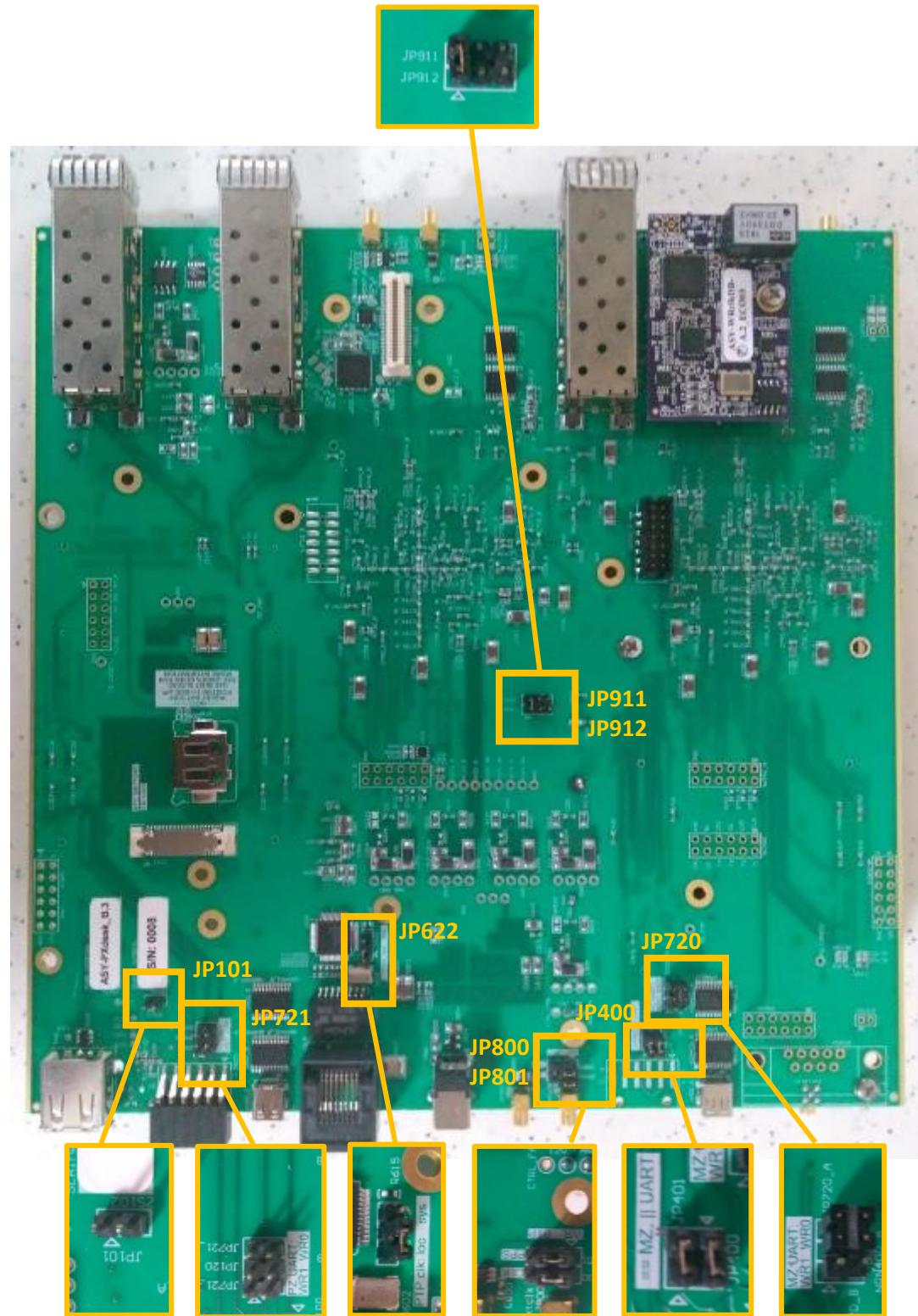
### 13.2.2 Revision B and C Modules

Pixie-Net XL Revision B has the following internal jumpers on the main board

- **JP400/401**  
**default:** “==”  
When a shunt is connecting pin 1 and pin 2 in each jumper (position “=”), the rear panel PMOD pinout is identical to the MicroZed PMOD  
When a shunt is connecting both pin 1 and another shunt is connection both pin 2 (position “||”), the rear panel PMOD pinout has pins 2 and 3 swapped. This is useful for some UART PMOD devices, e.g. the Digilent GPS module. **Currently used exclusively for WR UART**
- **Rev. B only: JP622 (A, B)**  
**default:** “loc”  
When a shunt is placed on JP622 in position “loc”, the PTP Ethernet PHY is operating on a local clock. In position “sys” it is operating on the clock controlled by the WR slave logic.
- **JP800**  
**default:** “WR”  
When a shunt is placed on JP800 in position “WR” the MMCX connector J800 “CLK OUT” outputs a clock synchronized by White Rabbit (typically 10 MHz), or a custom clock from the FPGA K7\_1 in custom designs.  
Rev B: In position “PTP”, J800 outputs the on-board slow clock as defined by JP911/912  
Rev C: In position “PZ”, J800 outputs the on-board slow clock as defined by JP911/912
- **JP801**  
**default:** “WR”  
When a shunt is placed on JP801 in position “WR” the MMCX connector J801 “PPS” outputs a pulse per second signal synchronized by White Rabbit or a custom trigger signal from the FPGA K7\_1 in custom designs.  
Rev B: In position “PTP”, J801 outputs a trigger signal generated by the PicoZed PTP PHY.  
Rev C: In position “PZ”, J801 outputs a trigger signal generated by the PicoZed.
- **JP911/912**  
**default:** “Kintex 1” (Rev. B: pin with triangle mark)  
A shunt on these jumpers controls the input to the main slow clock tree (25 MHz):
  - in position “PZ” the output clock from the PicoZed controller is used (optional White Rabbit synchronization). (Rev B: output from PTP Ethernet PHY.)
  - in position “External” (middle), the MMCX connector J802 “CLK IN” is used, expecting a 25 MHz clock.
  - in position “Kintex 1”, the output clock from the Kintex FPGA 1 (near MicroZed) is used (optional White Rabbit synchronization)
- **JP420/720\_A/B: “MZ UART”**  
**default:** WR1 (for Rev B: WR0)  
Connect middle pins pair to the pair on the right or left. A shunt on these jumpers connects the MicroZed’s UART port to the White Rabbit UART from FPGA K7\_0 or K7\_1. “WR1” means connecting to the FPGA closer to the MicroZed (connected to SFP 1). “WR0” connects to the FPGA connected to SFP 0. (Rev B units have WR0 and WR1 labels swapped)
- **JP120/721\_A/B: “PZ UART”**  
**default:** none  
Connect middle pins pair to the pair on the right or left. A shunt on these jumpers connects the PicoZed’s UART port to the White Rabbit UART from FPGA K7\_0 or K7\_1.
- **JP121/122: “PZ UART to PMOD”**  
**default:** none  
Connect PicoZed’s PS UART pins to two for the PL pins that are part of the PMOD connector.
- **JP101**  
**default:** none

JP101 is not a jumper, just an optional connector to plug into the UART from the PZ USB/UART port normally used for the root serial port terminal

Note: do not use the PMOD UART for external devices if jumpers JP720/721 or JP121/122 connect the UART to the White Rabbit UART.



---

## 13.3 EEPROMs

The Pixie-Net XL has several EEPROMS to store setup information

- Each of the two ADC daughtercards has an EEPROM + Thermometer chip. These chips are read during `./progfippi` and `./bootfpga` to get information on the daughterboard type and temperature. The information is programmed by XIA and should not be modified.
- The PXdesk main board has an EEPROM + Thermometer chip. This chip is read during `./progfippi` and `./bootfpga` to get information on the main board type, serial number, and temperature. The information is programmed by XIA and should not be modified.
- Each of the White Rabbit clock daughterboards has an EEPROM to store White Rabbit related configuration information, including SFP calibration data, MAC and IP addresses. They are programmed via the White Rabbit UART interface, which is connected to the Zynq controller (minicom).<sup>15</sup> Only one of these EEPROMs can be accessed at a time, use jumpers JP720/721 to switch. See the White Rabbit manual for more information.
- The Zynq controller has additional non-volatile memory that configures Linux or u-boot options. This memory can be modified through the Zynq serial interface when connected via USB/UART (not via SSH). Power cycle with the USB/UART terminal open and connected, and press any key within 1-3 seconds to enter the u-boot configuration utility. Then, for example to change the Zynq's MAC address, type at the u-boot prompt

```
set ethaddr 22:11:22:33:44:55
```

```
saveenv
```

then power cycle again.

---

## 13.4 LEDs

The Pixie-Net XL has several LED indicators:

- Each SFP port has a “link” LED (orange). These indicate the presence of an Ethernet connection, but sometimes also only the presence of an SFP module.
- A green “POWER” LED indicates the Pixie-Net is turned on and booted.
  - Blinks if powered but not booted
  - Constantly on when powered and booted ok
- An orange “ACTIVE” LED indicates that data acquisition is in progress
- A solid red “ERROR” LED indicates that the internal buffer for MCA increments is full (or turned on by the DAQ routine for some other reason). This usually means that the Pixie-Net XL can temporarily not process any more data until the buffer is emptied.

The list mode SDRAM buffer and the UDP output pipeline currently do NOT contribute to the “full” status.

A slow blinking red “ERROR” LED indicates that the FPGAs are in sleep mode to save power.

---

<sup>15</sup> More complicated in the PZ 10G+WR option

- The “LAN 1” Ethernet connector has built-in LEDs for link and activity.

## 13.5 Dimensions

The external dimensions of the Pixie-Net XL are

$$W = 202.4\text{mm},$$

$$H = 77.4 \text{ mm},$$

$$D = 193.8 \text{ mm}$$

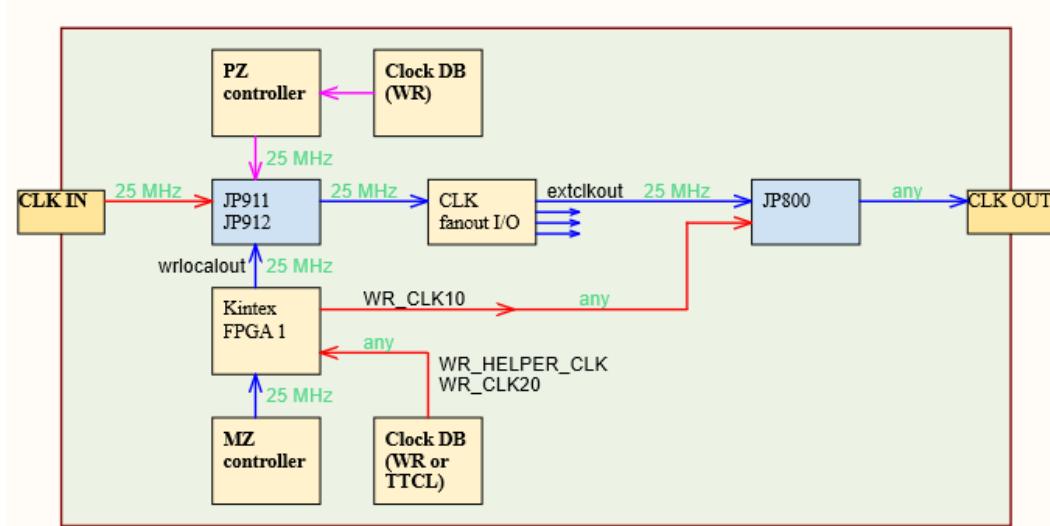
Allow extra clearance in D for connectors extending out beyond the enclosure (and cables plugging into them).

Mechanical drawings are available upon request.

## 13.6 Clocks

### 13.6.1 Clock Inputs and Outputs

The Pixie-Net XL has a clock input connector (MMCX “CLK IN”) and a clock output connector (MMCX CLK OUT). The configuration options are illustrated in the drawing below. Clock sources are in bold.

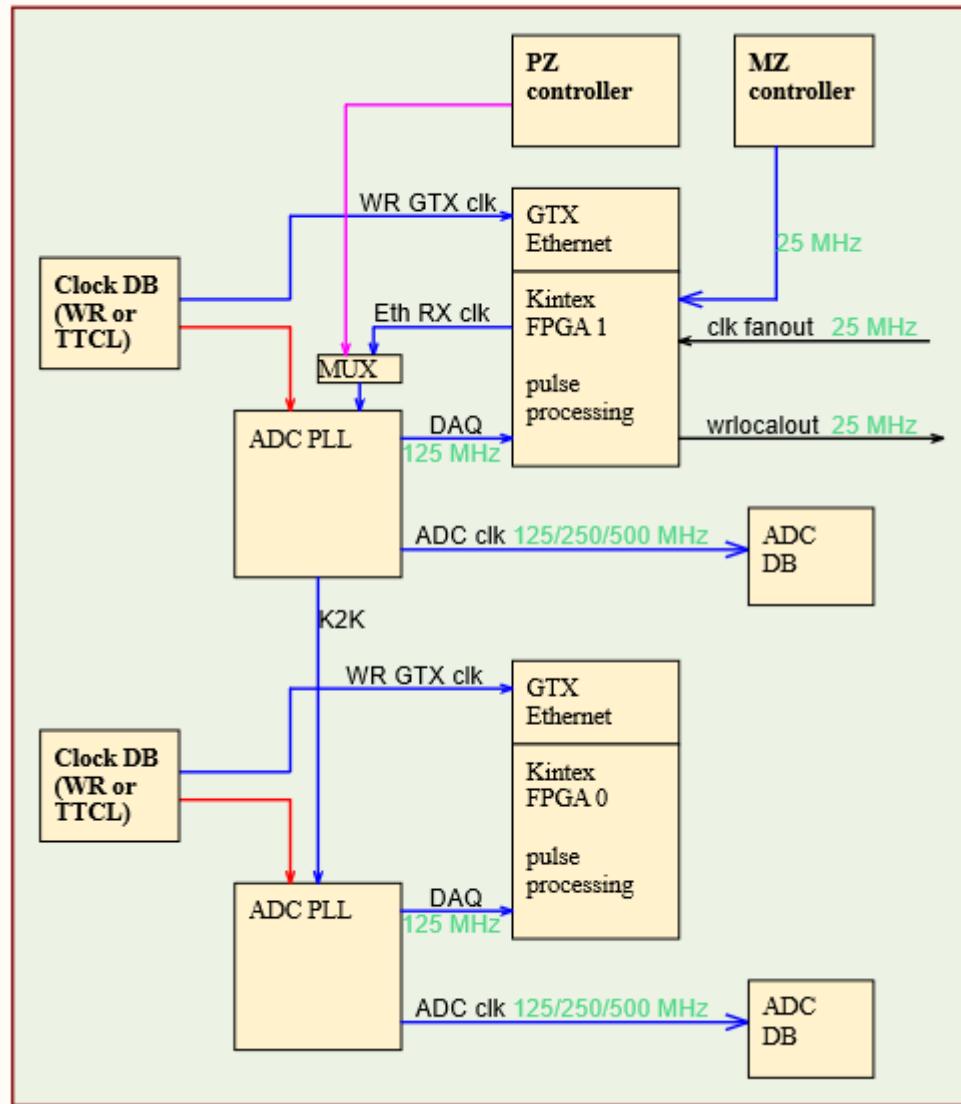


The clock input must be 25 MHz. Jumpers JP911 and JP912 select the source for the clock fanout circuitry; either the external CLK IN, the Kintex pulse processing FPGA 1, or the PicoZed controller. The Kintex FPGA does not generate a clock itself, but passes through the clock generated by the MicroZed controller or the clocking daughterboard. Firmware can be modified to modify the frequency output by the FPGA. The Kintex FPGA can additional output an arbitrary frequency that bypasses the clock fanout. For White Rabbit time synchronization that frequency defaults to 10 MHz. Jumper JP800 selects between the 25 MHz fanout clock or the Kintex arbitrary clock. The default clock connections are shown in blue.

Two of the outputs of the fanout clock goes back to the Kintex FPGAs. This allows pulse processing to be synchronized with the clock provided on CLK IN, and daisy-chaining multiple Pixie-Net XL units.

### 13.6.2 Pulse Processing Clock Options

The Pixie-Net XL processes signals from the ADCs clocked at 125 or 250 MHz with an internal 125 MHz clock. (For ADCs running faster than the processing clock, multiple ADC samples are processed in each FPGA clock cycle). Both FPGA processing clock and ADC clocks are regenerated by a phase locked loop circuit on the board (ADC PLL). The clock source for the ADC PLL is one of several options:



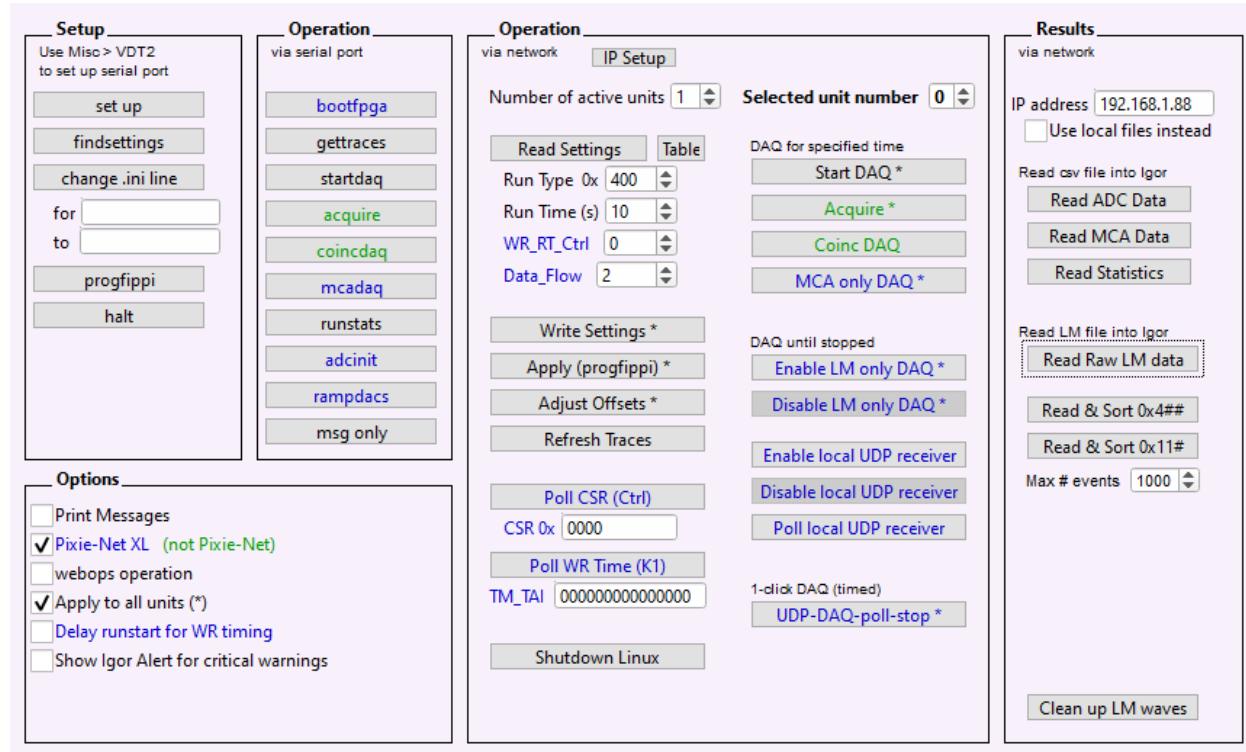
- **MicroZed controller (default, blue path in diagram)**  
An internal clock of the MicroZed controller is used by the Kintex pulse processing FPGA to generate a 125 MHz clock for the ADC PLL. The ADC PLL for Kintex 1 generates a 125 MHz clock for the ADC PLL associated with Kintex 0
- **Clock daughterboard**  
The clocking daughterboard always provides the 156.25 or 125 MHz clock for the Kintex Ethernet output. In addition, it can provide a clock alternative for the ADC PLL. This allows a more direct path for clock synchronization implemented on the daughtercard, for example the TTCL interface or the White Rabbit oscillators.

- **PicoZed controller**  
The PicoZed controller can provide another clock alternative for the ADC PLL. This bypasses the Kintex FPGA and may be of interest for users adding code to the PicoZed.
- **Other**  
Kintex 1 can use the signal from the clock fanout circuitry instead of the 25 MHz MicroZed clock. This allows the ADC PLL to be synchronized with the external clock input or other sources in the MicroZed or PicoZed controllers (see above).

## 13.7 Switching Between Board Configurations

	10G	10G + WR (PicoZed)	1G +WR	10G + TTCL
<b>WRclkDB model</b>	10G	10G + FB for PicoZed	FB	TTCL adapter
<b>PROM MB ("PCB_VERSION")</b>	0xA###	0xD###	0xB###	0xC###
<b>CLK_CTRL</b> (power cycle after change)	0	0?	3	0 (rev A Si bug) 3 (rev B)
<b>Zynq controller</b>	MicroZed or Z-turn	PicoZed	MicroZed or Z-turn	MicroZed or Z-turn
<b>Zynq HW_VERSION</b>	MZ = 0xA### ZT = 0xE###	PZ = 0xD###	MZ = 0xA### ZT = 0xE###	MZ = 0xA### ZT = 0xE###
<b>WR UART</b>	n/a	Connect external to PMOD 0	Default jumpers, use MZ minicom	n/a
<b>Kintex SFP</b>	10G	10 G	1G WR	10G
<b>PZ SFP</b>	n/a	1G WR	n/a	n/a
<b>TTCL SFP</b>	n/a	n/a	n/a	TTCL
<b>Source IP and MAC for LM data</b>	Settings.ini	Settings.ini	Settings.ini and PROM on WRclkDB	Settings.ini

# 14 Igor Pro GUI



## 14.1 Introduction and Setup

A graphical user interface based on Igor Pro is intended to simplify operation of the Pixie-Net and Pixie-Net XL. The GUI provides a panel with control buttons to execute API functions on the Linux OS, manages key parameter settings, and displays results.

The interface requires the following “installation” steps:

1. Install Igor Pro version 8 or higher on the user’s control PC (“the PC” thereafter).
2. Copy the Igor extension VDT2.xop (by Wavemetrics) from the “More Igor Extensions” folder to the “Igor Extensions” folder. Make sure to get the correct version for 32 vs 64 bit Igor application.
3. Extract the zip file from XIA containing Igor experiment and procedure files into any folder on the PC.
4. (Optional) From that folder, copy udp\_xop##.xop to the “Igor Extensions” folder if the UDP receiver functionality is to be used within Igor. Make sure to get the correct version for 32 vs 64 bit Igor application

Open the GUI program by double clicking on Pixie.pxp. This will open the Igor application with the PIXIE-NET XL panel shown above. (If not, go to top menu **XIA** and select **Pixie-Net XL panel** or type F2).

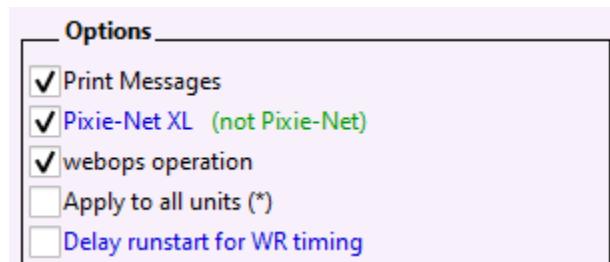
The following assumes that the Pixie-Net (XL) is powered up, the autoboot routine executed successfully to configure FPGAs and parameters, and the IP address is known.

## 14.2 Operation via web API (Recommended)

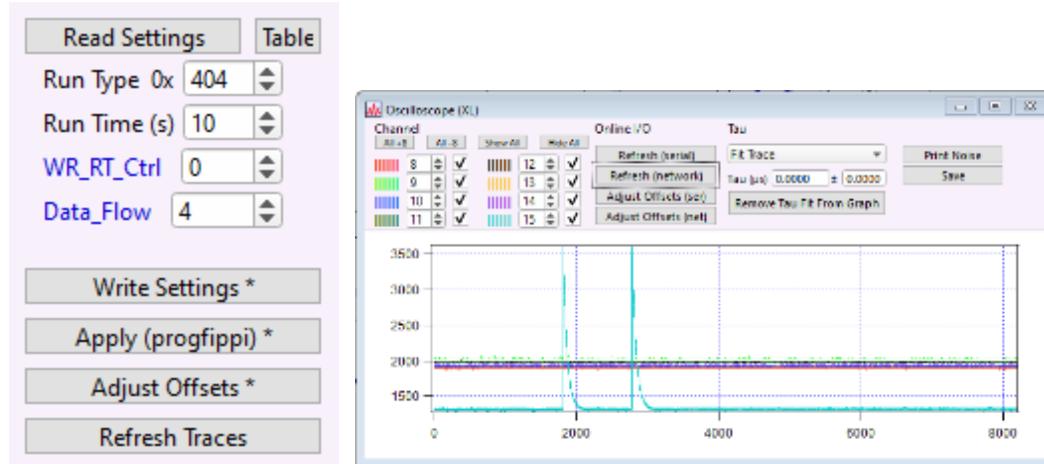
The web API operation uses the center block of controls in the PIXIE-NET XL panel. Operation is largely equivalent to the ADCsetup and ADCcontrol webpages; the same web API function are called, only from Igor instead of from the browser. Note that the communication target is the Pixie-Net XL folder /var/www/webops.



The interface currently supports up to 4 Pixie-Net devices. Click on [IP setup] to open a table to specify the IP addresses, user names and passwords. Then choose the number of units in use [Number of active units] and [Select the unit number] to communicate with.

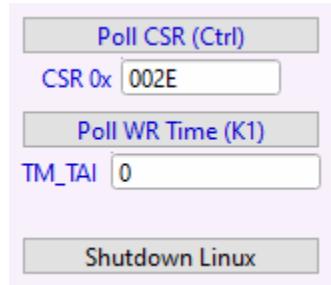


Some operations, such as writing parameters or starting a DAQ can be applied to all units. These operations are marked with \* but the checkbox [**Apply to all units**] must be checked. Furthermore, make sure to check [**webops operation**] unless operating in serial port mode and [**Pixie-Net XL**] unless using a Pixie-Net. (Blue text indicates a control is only relevant for Pixie-Net XL, green text only for Pixie-Net.) When the [**Print Messages**] box is checked, the full server response and other information is printed in the Igor history window.



Clicking the [**Read Settings**] button reads a number of key parameters<sup>16</sup> from the settings file (from `/var/www/webops/settings.ini`). The values are displayed in a table and in the fields below the button. Values can be edited in the table and fields, then written back to the settings file by clicking [**Write Settings**]. This will change the settings file, but values are only applied to the pulse processing FPGAs after clicking [**Apply (progfippi)**].

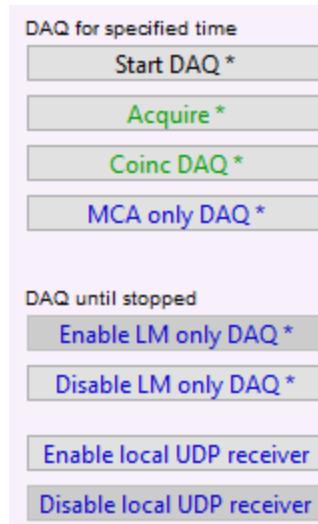
A common settings change is to adjust the DC offsets of the input, which can be performed by clicking [**Adjust Offsets**]. The function is rather slow to execute. While it changes the offsets in the Pixie-Net XL, it does not change the settings file automatically (to avoid overwriting previous values with something worse). Instead, verify the offsets are ok by clicking [**Refresh Traces**] and opening the Oscilloscope panel (top menu **XIA > Oscilloscope XL** or F3). Then manually update the values in the parameter table with the values printed in the Igor history window, click [**Write Settings**] and [**Apply**].



For the Pixie-Net XL, additional status information can be obtained with the [**Poll CSR**] and [**Poll WR Time**] button. Details are described elsewhere.

Before powering down the Pixie-Net XL, it is strongly recommended to shut down the Linux OS. This is not executed directly from Igor (Security concerns in the web API) but the [**Shutdown Linux**] button opens a Windows command window with an ssh session for root, in which (after specifying the password) the system can be shut down by typing `halt`.

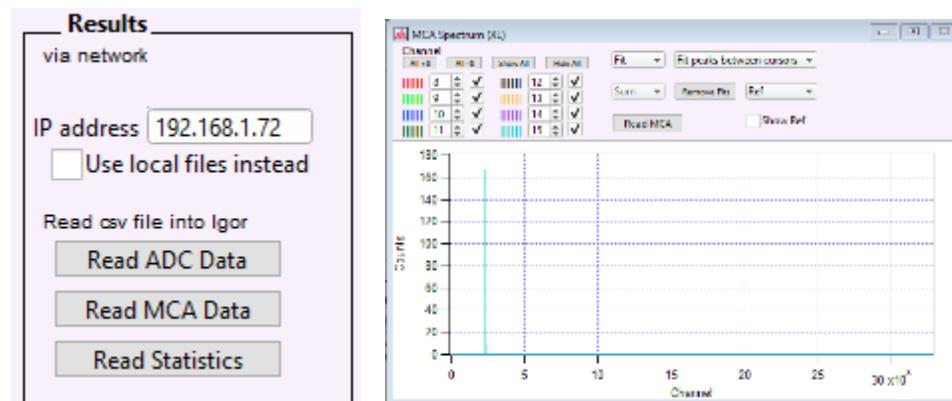
<sup>16</sup> Parameters not included in the “key parameters” must be edited in the ini file. XIA welcomes suggestions to choose additional “key parameters”.



Data acquisition can be started with the DAQ buttons. For the top 4 buttons, the button text matches the API functions, please see descriptions in section 4. Note that these DAQ functions run for the specified RUN TIME, and Igor will wait for the DAQ to finish.

The lower 4 DAQ buttons correspond to the API functions `udpena` and `udpdis`. This simplified run mode starts a list mode only DAQ, where the Linux OS is not involved. The [**Enable LM only DAQ**] simply turns on the DAQ in the FPGAs, and [**Disable LM only DAQ**] turns it off. No function is running on the Linux OS and therefore the web API function call returns immediately.

Since the list mode data is streamed out as UDP packages in this mode, a receiver program has to capture the UDP packets and write them to file. If the file “`udp_xop##.xop`” is correctly installed for Igor to use, the buttons [**Enable local UDP receiver**] and [**Disable local UDP receiver**] can be used to start and stop the receiver program within the xop. Otherwise, the standalone program `udp_receive.exe` can be manually started from a Windows command line (or on a different PC altogether).



During and after a DAQ, the files created on the Pixie-Net’s SD card can be read into Igor using the buttons in the results block. *Note: The IP address shown here is only valid if the webops option is not selected.* Clicking any of the [**Read ...**] buttons will retrieve the files `ADC.csv`, `MCA.csv`, or `RS.csv` from the Pixie-Net XL SD card and display them in the OSCILLOSCOPE, MCA SPECTRUM, or RUN STATISTICS TABLE, respectively. The [**Use local**

**[files instead]** option can be used to select the files manually in a file open dialog instead of reading from the remote SD card.

---

## 14.3 Operation via Serial Port

If the Pixie-Net XL is connected to the PC with the UART/USB serial port, the controls in the upper left SETUP and OPERATION blocks can be used instead of web API calls. The button text matches the API function name, please see descriptions in section 4. Essentially, clicking a button is equivalent to typing the button name into the terminal window.

To change parameters in the ini file (located in /var/www), specify the parameter name in the **[for \_\_\_\_]** control field, and the values for all channels in the **[to \_\_\_\_]** field.

As in the web API operation, files created on the Pixie-Net's SD card can be read into Igor using the buttons in the results block. However, you must specify the **[IP address]** and files are from the folder /var/www.

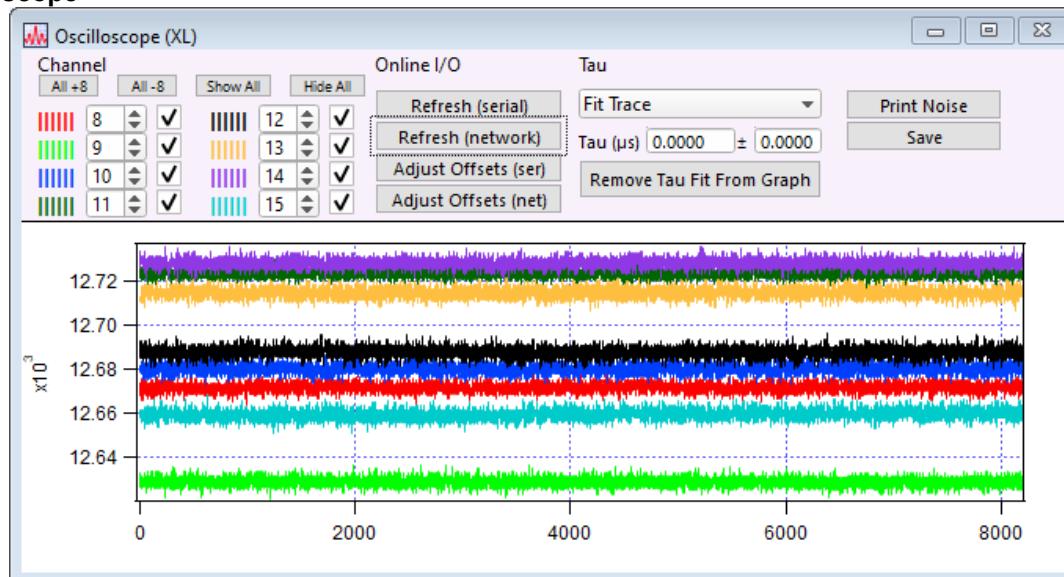
---

## 14.4 Results

The Pixie-Net XL generates 4 data files, which can be viewed in dedicated plots and tables:

Data	File	Plot/Table
ADC traces	ADC.csv	OSCILLOSCOPE
MCA spectra	MCA.csv	MCA SPECTRUM
Run statistics	RS.csv	RUN STATS TABLE
List mode data	LMdata.bin/b00/dat/dt2	LIST MODE TRACES

#### 14.4.1 Oscilloscope

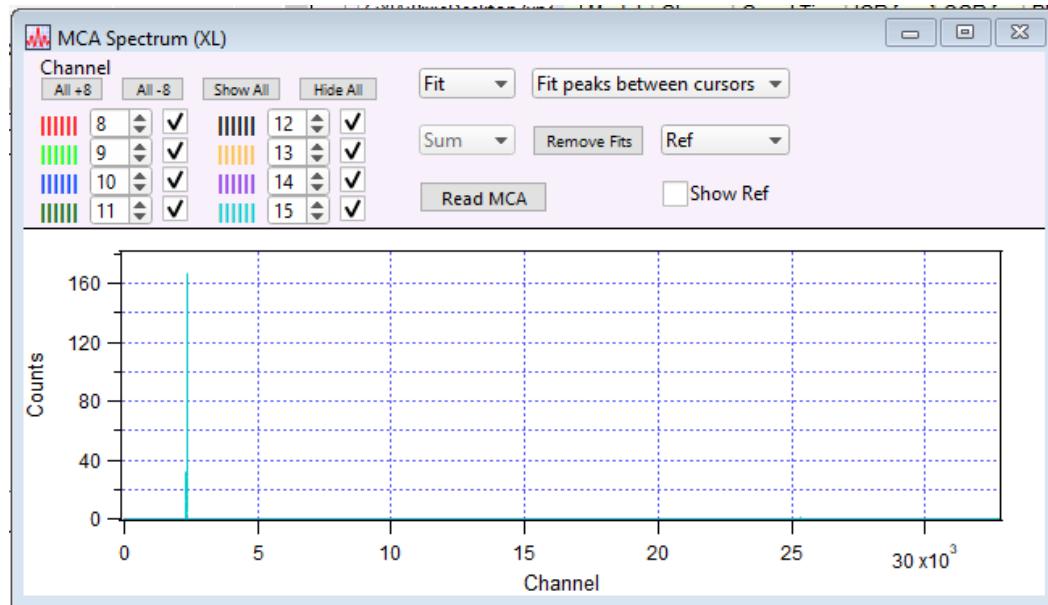


ADC traces are displayed in the OSCILLOSCOPE. It shows 8 channels in a common plot. Checkboxes in the upper left can be used to show/hide a channel. To accommodate configurations with more than 8 channels, the physical channels can be assigned arbitrarily to display channels. Display channels are always numbered 0 (red) to 7 (cyan), but the control field net to the checkbox specifies the physical channel. Buttons above the check boxes can be used to show/hide all, or add/subtract 8 for all display channels.

The control buttons duplicate the [Refresh] and [Adjust offset] buttons in the main PIXIE-NET XL panel, for web API (network or serial communication, respectively. Pulses in the display channels can be fitted to exponential decay, which can help to estimate the TAU parameter for the settings file. Further options allow to print the noise (std deviation) of each trace in the Igor history window, and to save the ADC data to a file.

#### 14.4.2 MCA Spectrum

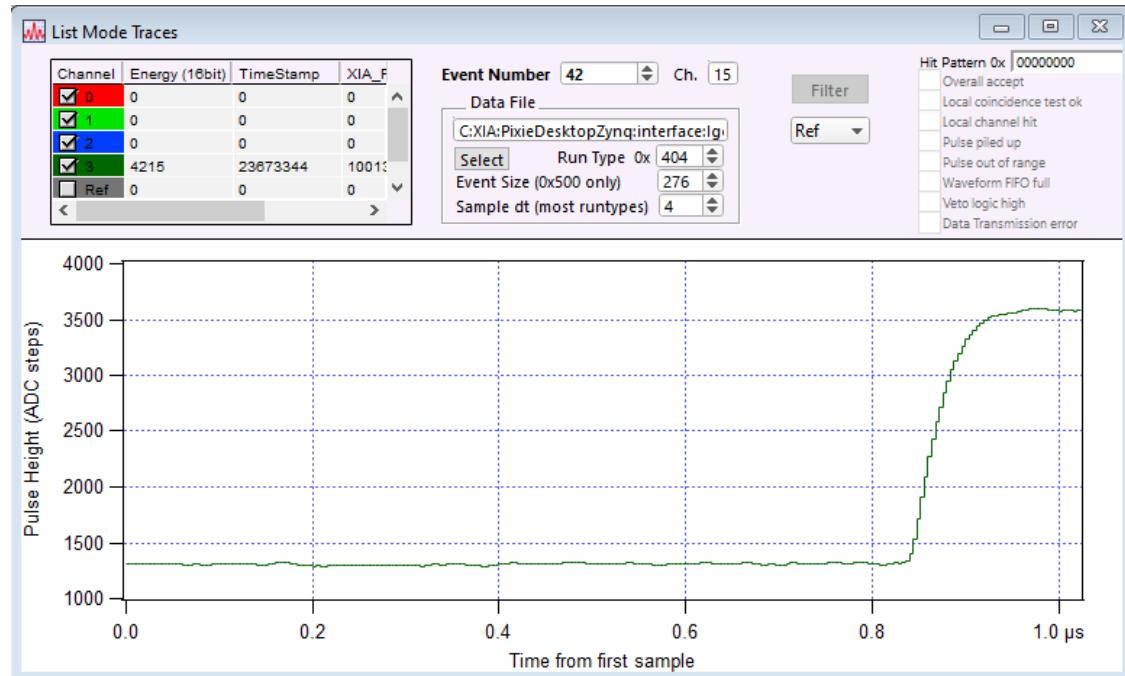
MCA spectra are displayed in the MCA SPECTRUM plot. Selection of the channels to display is identical to the OSCILLOSCOPE. Display channels can be fitted with a Gaussian to determine peak resolution. The [Read MCA] button duplicates the button in the PIXIE-NET XL panel.



### 14.4.3 Run Statistics Table

The data from RS.csv is displayed in a large table. For parameter descriptions, see sections 8 and 9.

#### 14.4.4 List Mode Traces



The List Mode Traces plot can be used to view individual pulses from the list mode data files. The file can be selected in a dialog with the [Select] button; then the event number can be selected with the [Event Number] control. Detection of the run type is (usually) automatic.

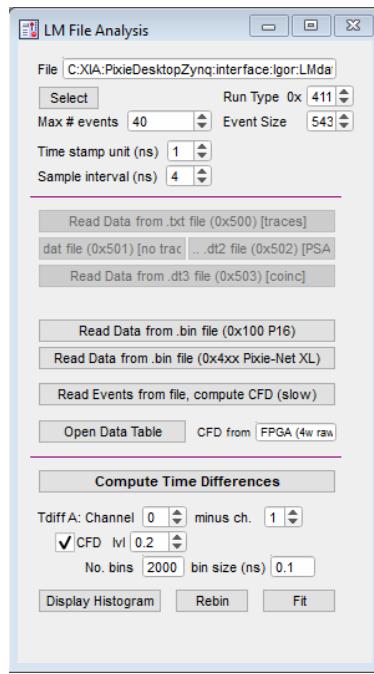
The display, for traditional reasons, displays 4 channels. For systems with more than 4 physical channels, physical channels  $\geq 4$  are mapped into display channels 0-3. Display channel is physical channel modulo 4. The physical channel is shown in the [Ch. \_\_] field. The panel also shows a number of data extracted from the event hit patterns and event info records; this is still only partially implemented.

#### 14.4.5 List Mode Data Analysis (under development)

The LM File Analysis panel (top menu XIA > List Mode File Analysis) serves to view LM data files in tables and compute CFD time differences.

The first section specifies the file to analyze. Any list mode data file generated by the Pixie-Net XL or Pixie-Net is valid. The code supports formats from Pixie-16, Pixie-4e, Pixie-Net and Pixie-Net XL which all have different "run type" ID numbers. While you can type a file name/path in the field and "select" one with a dialog, usually the buttons in the next section prompt you for the file to select in a new dialog. Therefore the field with the file/path is mostly for information. The "Run Type" and "Event Size" fields are updated when processing the file. "Time stamp" and "Sample interval" units are likewise updated when processing the file (most file types). The most important field is the "Max # events", which limits the events to process and show in tables and plots, to save time and memory with large files.

The second section has the buttons to read in the file and sort it into a table. The first few are disabled (darker gray) for unsupported run types from the Pixie-Net. The "Read Data... 0x100 P16" is for data in Pixie-16 style formats. The "Read Data ... Pixie-Net XL" is for binary Pixie-Net, Pixie-4e, and Pixie-Net XL formats. The underlying function simply takes the data from the file and organizes it in a table - columns for hit pattern, channel number, energy, time stamps etc. While the Pixie-Net XL has 16 channels, in the table



there are only 4 columns for energy, time stamps, and CFD<sup>17</sup>. The 16 channels are mapped into those columns as channel 0,4,8,12 -> energy0, energy 1,5,9,13 -> energy 1 etc.

You can open the table or bring it to front with the "Open data table" button. The CFD value is computed from the 4 raw values in the LM file (see above) and shown in units of ns

While the "Read Data ..." buttons read in the whole file and then sort the values into the table, the "Read Events from file, compute CFD ..." button reads one event at a time. This is similar to clicking through events in the "List Mode Traces" panel. For each event, Igor computes the CFD from the waveforms and timestamps and this computed value goes into the CFD0/1/2/3 fields in the table. As an indicator of that, the text in "CFD from" changes from "FPGA" to "Igor, cfd = xxx". That 'cfid' is the "lvl" value specified below in the third section. It is the constant fraction level for Igor to find the timing for, so e.g. for 0.2 is the 20% level of the full pulse amplitude.

The third section includes the button "Compute Time Difference" to compute LocTimeX-LocTimeY+CFDX-CFDY and put the result into column TdiffA of the table. If you uncheck the "CFD" checkbox, it computes LocTimeX-LocTimeY only. X and Y are specified in the "channel" fields. The other controls are for histogramming and fitting: You can specify the number of bins and the width (or size) of each bin. Clicking "Compute Time Difference" will also create the histogram with current bin settings; the histogram can be rebinned with modified bin settings using the "Rebin" button without recomputing the time differences. The histogram can be opened or brought to front with the "Display Histogram" button. Placing the cursors around a peak allows the peak to be fitted for time resolution with the "Fit" button, and the fit results to be printed in the Igor history Window.

## 14.5 Web API coding example

The source code for the Igor GUI is part of the GUI distribution. Simply open the procedure window "PixieNetXL.ipf" and view the code as an application example. Actual communication with the Pixie-Net is contained in the function "PixieNet\_WebIO". Buttons in the Pixie-Net XL panel are serviced by the function "PNXL\_ButtonControl".

While a number of Igor-specific functions are used to assemble and parse the strings for web communication, we believe equivalent functions exist in C, C++ or other programming environments, which should make porting the code relatively straightforward. Please contact XIA if you need support.

<sup>17</sup> A remainder from the days when there were only 4 channels in a Pixie board, but it saves some memory for Igor

# 15 UDP Receiver Program

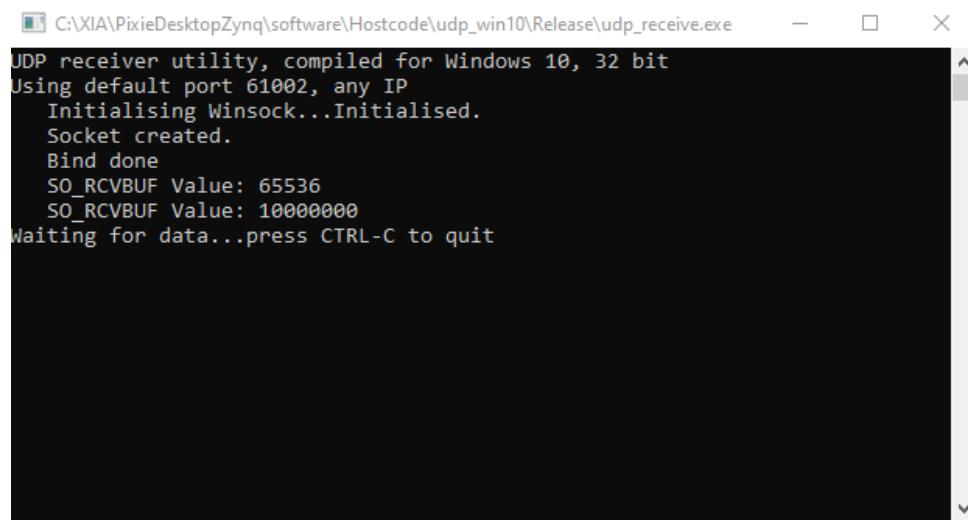
## 15.1 UDP Setup

The Pixie-Net XL is designed to send list mode data out as UDP packages via the SFP Ethernet ports connected to the pulse processing FPGA. To capture this data and save it to disk, XIA provides a very basic UDP receiver utility. Three variants are supported:

1. Windows 10 (udp\_receive.exe)
2. Linux (udp-recv)
3. Igor (udp\_xop64.xop)

The UDP receiver utilities can be downloaded from XIA's website. Please see accompanying instructions for installation and/or compiling these programs; on most systems it should be possible to simply copy to a local folder and execute them from a cmd or terminal window.

For data acquisition, first start the UDP receiver on the receiving PC, then the DAQ on the Pixie-Net XL. The UDP receiver will listen to UDP packages and write them to a local file. The file formats are described in section 7. End the UDP receiver with <ctrl>-C when the Pixie-Net XL DAQ is done.



```
C:\XIA\PixieDesktopZynq\software\Hostcode\udp_win10\Release\udp_receive.exe
UDP receiver utility, compiled for Windows 10, 32 bit
Using default port 61002, any IP
    Initialising Winsock...Initialised.
    Socket created.
    Bind done
    SO_RCVBUF Value: 65536
    SO_RCVBUF Value: 10000000
Waiting for data...press CTRL-C to quit
```

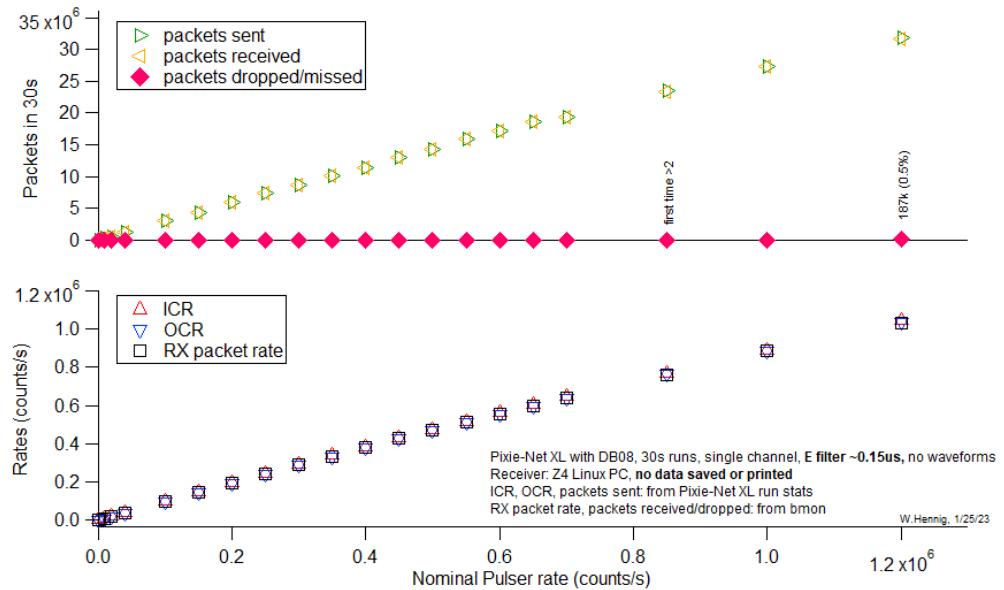
For the process to work properly, please note the following requirements

- Ensure a valid Ethernet connection from Pixie-Net XL SFP port to receiving PC.  
We recommend using a MikroTik 10G switch that also can “convert” to 1G Ethernet for standard PCs and has an utility to monitor network traffic
- Ensure source and destination MAC and IP are correct  
For the “1G” Ethernet option, source MAC and IP addresses are stored on a local PROM accessible via the WR UART, destinations in the settings file.  
For the “10G” Ethernet options, all MAC and IP addresses are stored in the settings file.
- Ensure the receiving PC allows the UDP receiver program access through any firewalls

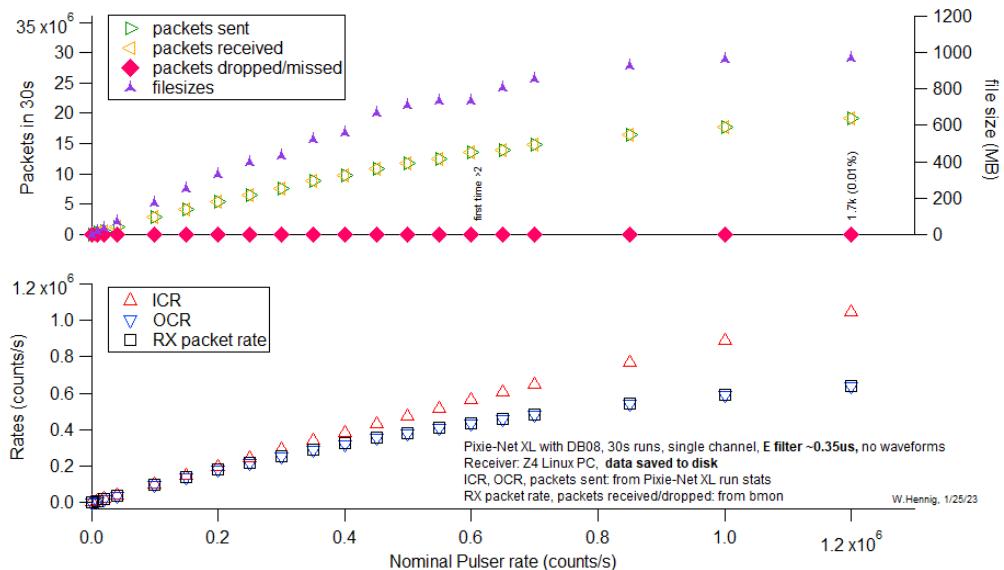
- Make sure UDP\_PAUSE is large enough to limit the maximum data rate within the capabilities of the network. This is especially important when funneling one or more 10G output streams into a 1G port, and for long traces ( $>672$  samples) that are broken up into 8 UDP packages.

## 15.2 UDP performance

The following plots describe throughput performance from a single SFP port under different conditions.

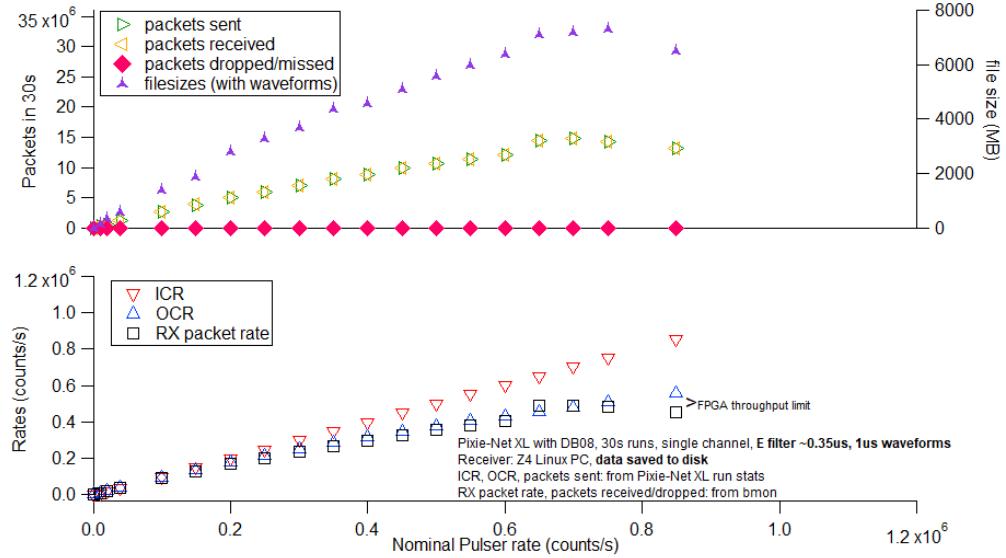


Using very short energy filters (to not limit DAQ from pileup) and not saving data to file, all incoming pulses are processed and transmitted to the receiver PC without dropping packets up to ~800,000 counts/s



With longer energy filters, some of the input counts are rejected as piled up, so the output count rate (OCR) drops below the input count rate (ICR). All packets are received and

saved to file up to an ICR of ~600,000 counts/s, and less than 0.01% are dropped up to ICR ~1,200,000 counts/s. Throughput to file is ~640,000 pulses/s.



When also saving waveforms, packets contain more data. The FPGA's data throughput limit is about 240 MB/s (current firmware) so that the RX packet rate<sup>18</sup> drops below OCR at an ICR of ~750,000 counts/s for a max throughput to file of ~480,000 pulses/s with 1μs waveforms.

Note: These results are specific to PC's architecture, network cards, operating system etc, and the network architecture. They are only intended as a typical example, not as a throughput specification.

<sup>18</sup> RX = received in PC