Pixie-Net

User Manual

Version 1.08

April 4, 2017

Hardware Revision: A

Software Revision: 1.08

Firmware Revisions:
0x0108 (standard)
0x1108 (PSA)

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

Contents

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

Observe all of these precautions to ensure your personal safety and to prevent damage to either the Pixie module or equipment connected to it.

Power Source

The Pixie-Net 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

User Adjustments/Disassembly

To avoid personal injury, and/or damage, always disconnect power before accessing the Pixie module's interior. There are no switches or jumpers for user configuration inside the box, however, there are debug push button and headers experienced users may want to use.

Detector and Preamplifier Damage

The Pixie module can provide +5V power and a 0..5V control voltage, intended for active PMT bases. The +5V power comes up immediately when the Pixie is connected to power, the control voltage is <u>undefined</u> until the Linux system is booted and basic configuration has been applied.

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

Voltage Ratings

Signals on the analog inputs (gold SMA connectors) must not exceed ± 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 these units. These modules are warranted against all defects for one (1) year. Please contact the factory or your distributor before returning items for service.

Linux Passwords

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

Linux Backup

The Pixie-Net 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 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 MERCHANTABILITYOR FITNESS FOR A PARTICULAR PURPOSE. XIA'S RESPONSIBILITY TO REPAIR OR REPLACE DEFECTIVE PRODUCTS IS THE SOLE AND EXCLUSIVE REMEDY PROVIDED TO THE CUSTOMER FOR BREACH OF THIS WARRANTY. XIA LLC AND ITS VENDORS WILL NOT BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IRRESPECTIVE OF WHETHER XIA LLC OR THE VENDOR HAS ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Contact Information:

XIA LLC

31057 Genstar Rd.

Hayward, CA 94544 USA

Telephone:	(510) 401-5760
Downloads:	http://support.xia.com
Hardware Support:	support@xia.com
Software Support:	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 **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and choose **Options** from the sub menu.
Bold	Bold text denotes items that you must select or click on in the software, such as menu items, and dialog box options.	...click on the **MCA** tab.
[Bold]	Bold text within [ ] denotes a command button.	**[Start Run]** 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 **Tools»Options**.
Italics	Italic text denotes a new term being introduced , or simply emphasis	*peaking time* refers to the length of the slow filter. ...it is important first to set the energy filter Gap so that SLOWGAP to *at least one unit greater than* the preamplifier risetime...
<Key> <Shift-Alt-Delete> or <Ctrl+D>	Angle brackets denote a key on the keybord (not case sensitive). A hyphen or plus between two or more key names denotes that the keys should be pressed simultaneously (not case sensitive).	<W> indicates the W key <Ctrl+W> represents holding the control key while pressing the W key on the keyboard
Bold italic	Warnings and cautionary text.	***CAUTION:*** *Improper connections or settings can result in damage to system components.*
CAPITALS	CAPITALS denote DSP parameter names	SLOWLEN is the length of the slow energy filter
SMALL CAPS	SMALL CAPS are used for panels/windows/graphs in the GUI.	…go to the MCADISPLAY panel and you see…

1 Introduction

The Pixie-Net is a Zynq System on Module (SoM) combined with an ADC input board. The Zynq is a combination of an FPGA (Programmable Logic, PL) with an ARM processor (Processing System, PS). The PL captures the ADC data and applies digital pulse processing. The PS runs a basic Linux OS with gcc, webserver, etc; it has USB and Ethernet peripherals and 1GB of memory.

1.1 Pixie-4 Express Features

- Designed for
    - high precision γ-ray spectroscopy with HPGe detectors,
    - timing with fast scintillators (NaI, $LaBr_3$, etc),
    - pulse shape analysis to extract time, position, and/or particle type in segmented or strip detectors, phoswich detectors, or neutron detectors
    - coincidence acquisition
- 12-16 bit, 125-500 MSPS ADC, 2-8 channels
  Standard versions: 12 bit, 250 MSPS
- Programmable gain and input offset.
- Programmable pileup inspection criteria include trigger filter parameters, threshold, and rejection criteria.
- Triggered synchronous waveform acquisition across channels and modules.
- Simultaneous amplitude measurement and pulse shape analysis for each channel.
- On-board MCA memory
- Configurable digital inputs and outputs
- Embedded Linux environment, acting as a standalone PC with built-in SD card drive, USB host, GB Ethernet, webserver, GPIO
- Open source DAQ software

1.2 Specifications (4 channel 12/250)

Front Panel I/O	
Signal Input (4x)	4 analog SMA inputs. Programmable input impedance: 50Ω and 10kΩ. Input range: After termination and attenuation, up to ±1.25V 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 below for analog gain values)
Pulser Output	1 MMCX coaxial connector Periodic, exponentially decaying reference pulser. Programmable on/off
Control Voltage Output	1 MMCX coaxial connector 0..5V programmable via DAC
Rear Panel I/O	
Logic Input/Output	General Purpose I/O connected to programmable logic: 1 MMCX coaxial connector input only default use: Veto signal to suppress event triggering 2.5V logic, 5V compatible 1 micro HDMI connector with 13 GPIO signals (not video) single ended or differential can be used as clock input factory set to 5V (default) or 2.5V logic.
Power	12V DC in. AC adapter requirements: 12V, 18W, Barrel Plug 2.1mm I.D. x 5.5mm O.D., center positive 5V DC out MMCX coaxial connector Usage: typical ~10W, peak ~14W
PMOD	12-pin 0.1” connector for 8 I/O signals, 3.3V power, GND Compatible with PMOD I/O modules (I2C, GPIO, serial, wifi, GPS, …)
Embedded Processing	
Processor	Xilinx Zynq
Data Interfaces	10/100/1000 Ethernet USB 2.0 (host) USB-UART
Memory	1 GB of DDR3 SDRAM 128 Mb of QSPI Flash 16 GB micro SD card (removable)
Operating System	Linux (Xillinux – based on Ubuntu LTS 12.04) Operates from Linux partition on SD card

Digital Controls	
Gain	Coarse gain: 250 MSPS variant: 2 gain settings: 2 and 5 Other variants: please contact XIA Digital gain: Arbitrary gain factor for channel matching
Offset	DC offset adjustment from –1.25V to +1.25V, in 65535 steps.
Shaping	Trapezoidal filter with peaking times 0.048 to 63.4 µs Adjustable flat top to eliminate ballistic deficit effects
Trigger	Digital trapezoidal trigger filter with adjustable threshold. Rise time and flat top set independently.
Coincidence	Programmable coincidence window: 40 to 1016 ns Reject unwanted hit patterns of the 4 channels
Data collection	MCA number of bins 1Ki to 32Ki Waveform lengths and pre-trigger delay List mode data format (text, binary, content)
Data Outputs	
Spectrum	1024-32768 bins per channel, 32 bit deep (4.2 billion counts/bin). Additional memory for sum spectrum for clover detectors.
Statistics	Real time, counting time, filter dead time, input and throughput counts.
List mode event data	Pulse height (energy), time stamps, pulse shape analysis results, waveform data (up to 4Ki samples) and ancillary data like hit patterns.

Table 1-1. Specifications for the Pixie-Net

1.3 System Requirements

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

1.3.1 Drivers and Software

The Pixie-Net 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 provide the Pixie Viewer, based on Wavemetrics' Igor Pro. (Igor version 6.22 or higher is required). Alternative interfaces are under development (LabView, ROOT (under Linux), MATLAB, etc).

1.3.2 Detector Signals

The Pixie-4 Express 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 ± 3.5V. Exceptions apply for certain attenuation and termination settings (see Appendix).

1.3.3 Power Requirements

The Pixie-4 Express consumes roughly 14 W, requiring the following currents from the AC adapter:

12V up to 1.5 A

1.3.4 Connectors and Cabling

The Pixie-Net uses SMA connectors for the analog inputs from the detectors. SMA to BNC adapter cables are provided with the module.

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

A micro HDMI connector is used for 13 additional digital inputs and outputs, but not video. HDMI cables are widely available and not provided with the module

1.4 Software and Firmware Overview

The DAQ software of the Pixie-Net consists of a small set of C programs (API functions) applying settings to the PL and reading data from the PL. The API functions are called from the Linux command line or as CGI scripts from a web page. 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 is edited by the user to adjust parameter settings.

Firmware code for the PL on-board pulse processing functions is loaded to the PL 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, 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 automatically boots the FPGA configuration and starts the Linux OS. There are several ways for a user to connect to the Pixie-Net Linux OS:

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

The typical procedure for setup would be to first install drivers for the serial port on a USB master PC (2.2), then to power up the Pixie-Net (2.1). Log on via serial port terminal, find the Pixie-Net's IP address, and execute a few setup programs (2.5). After that, the Pixie-Net can be operated through terminal, web interface and/or SMB.

2.1 Power

To power up the Pixie-Net, simply connect the 12V DC power plug from the AC adapter. The center pin must be positive and the adapter must be rated for 18W or more. *At time of writing, several outputs and internal settings are undefined after powerup, so it is recommended to log in quickly via serial port or ssh and apply the settings.*

To power down the Pixie-Net, first shut down the Linux OS (type `halt`), then remove the 12V DC power plug (and UART cable).

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. Installation consists of the following steps:

1. Download and extract/install Silicon Labs CP210x USB-to-UART driver, see microzed.org/sites/default/files/documentations/CP210x_Setup_Guide_1_2.pdf
2. Download and install Tera Term (or other suitable terminal program). See http://ttssh2.osdn.jp/
3. Connect USB cable between Pixie-Net and PC and power up the Pixie-Net (see 2.1) (PC: any USB port, Pixie-Net: port labeled UART)
4. From the Silicon Labs installation, run CP210xVCPInstaller_x64.exe to create a COM port
   - Find the new COM port's number in device manager
   - The COM port assignment is device specific, so this has to be repeated every time switching to a different Pixie-Net unit. For the same Pixie-Net, this is a one time operation.
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
   - Currently, no login credentials are required for Linux login via serial port

On a Linux system, it is possible to use the Minicom utility as for serial port I/O. 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 SSH login

Tera Term or another suitable program can be used to log in via the network once the Pixie-Net 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`
- log on to router, see list of connected devices
- just try 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's IP address. Default ID/PW is root/xia17pxn (*for now*).

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

2.4 Web Interface

Any web browser can be used to log in via the network once the Pixie-Net 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`
- log on to 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's IP address as the web address in your browser. Any browser should work; XIA uses Firefox and does not test for compatibility with other browsers. No password is required for the Pixie-Net home page.

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

2.5 SMB (Samba)

The Pixie-Net 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 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\MZwww` in the Windows Explorer location bar, where `192.168.1.xxx` is the IP Address of the Pixie-Net found as above and `MZwww` 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 (*for now*).

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)

2.6 Required Initial Linux Commands

Once logged on via the Linux terminal, the following steps **must** 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
2. Adjust permissions: `chmod 777 /dev/uio0`
   This allows cgi functions called via the web interface to access the FPGA register space
3. Apply settings to FPGA: `./progfippi`
4. Automatically set a few basic parameters (and resolve random channel swapping):
   `./findsettings`
   This should be performed with the detector connected, with the correct polarity and termination setting.

2.7 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. Type `more /sys/devices/amba.0/f8007100.ps7-xadc/temp` to get the temperature of the Zynq chip. 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      1281' settings.ini
   ```
   This replaces (-i = in the file) the line containing the string "RUN_TYPE" with the text "RUN_TYPE   1281".
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]

3 Pixie-Net Operation

Basic operation of the Pixie-Net 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.

3.1 Adjust Settings

The FPGA settings and 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 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 patterns), and channel (e.g. offset). A condensed settings file, *settings.ini*, contains the most important parameter and **will override the defaults**. The current method to change settings is to edit *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 certain 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
MCSRB_TERM01_01	If 1, channel 0 and 1 inputs are terminated with 50 Ohm, else high-Z
MCSRB_TERM23_02	If 1, channel 2 and 3 inputs are terminated with 50 Ohm, else high-Z
RUN_TYPE	0x301 for MCA only, 0x400 and 0x500-502 for various list mode runs
CCSRA_INVERT_05	If 1, invert incoming signal
ANALOG_GAIN	Currently limited to 2 or 5
DIG_GAIN	Arbitrary gain factor for energies in MCA spectra and list mode events
VOFFSET	DC offset
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 400 ADC steps. 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 steps for a 12 bit ADC). The decay time must be measured and specified as the parameter TAU. This is best verified by opening/refreshing the Pixie-Net 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 of 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 channels 0/1 and 2/3.

3.2 Data Acquisition

1. In the terminal, type `./startdaq` or `./acquire` to start run with current settings.
   The screen will be updated with print statements of the runtime
2. In the browser, navigate to MCA page (“view spectra”) or Run Statistics page (“view run statistics”) under DAQ Monitoring.
   Refresh these pages with browser button to see updates during DAQ

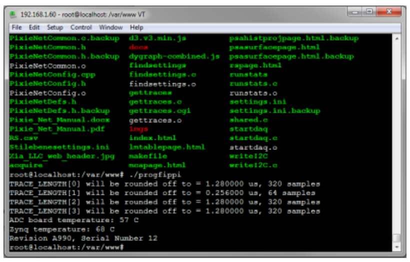
3. < wait > *Currently the only way to stop is ctrl-c*

4. When the DAQ finishes in the terminal, the final MCA, the run statistics, and the list mode data files have been created. *Currently, the filenames are fixed to MCA.csv, RS.csv and LMdata.dat/.txt* for `./startdaq` while `./acquire` allows arbitrary list mode file names.
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 `\\<Pixie Net IP>\MZwww`, the data files can be copied or opened by Windows tools and programs.

3.3 User Interface Options

There are a variety of interface options to operate the Pixie-Net. They range from basic terminal and file I/O to plug-ins for data acquisition or data analysis software. Fundamentally, the Pixie-Net 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

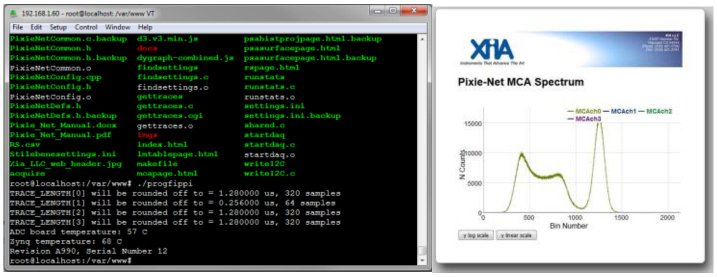
3.3.1 Terminal



At the most basic level, users can log in to the Pixie-Net 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’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.

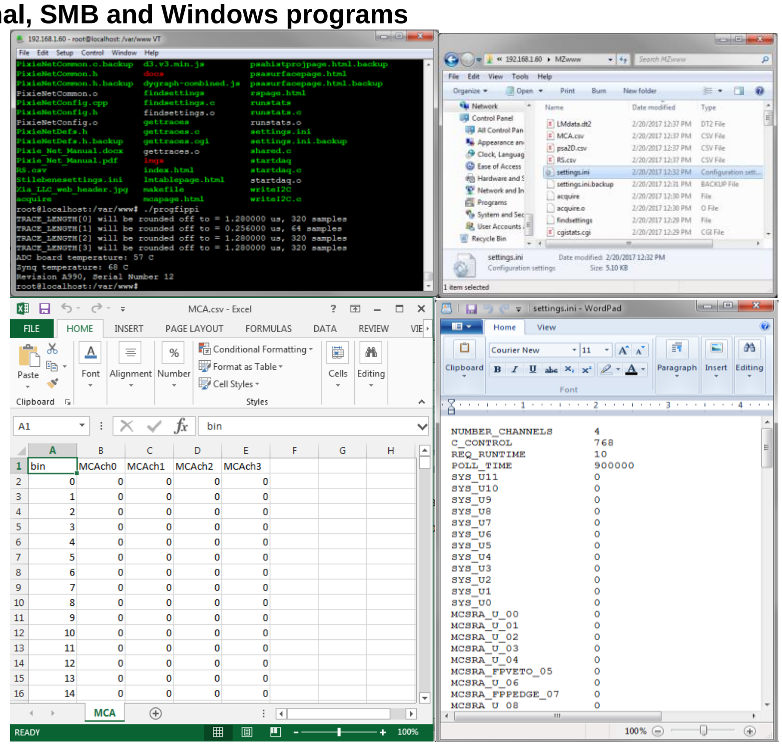
3.3.2 Terminal and Webpages



For direct graphical feedback, users can log in via terminal as in the preceding section, but view results as a webpage. The Pixie-Net 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.

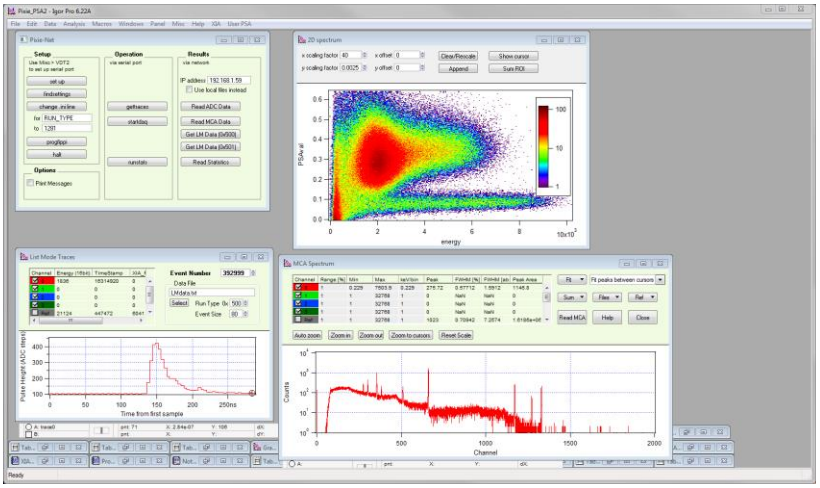
3.3.3 Terminal, SMB and Windows programs



With SMB file sharing, Windows programs can be used to directly view, edit, and analyze settings and results. 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

3.3.4 GUI plug-in



Many data acquisition and analysis program can communicate with serial ports, external files and web servers. Examples include Igor Pro, MATLAB, LabVIEW, and so on. It is therefore possible to develop graphical user interfaces with buttons and plots in such a program. In the picture above, Igor Pro's "very dumb terminal" function is 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 webpage and extracts and displays the data. For example, a "refresh" button in the Igor Pro "oscilloscope" display issues the command `./gettraces` and reads/displays ADC.csv. A "start run" button issues `./startdaq`, and so on. The data can then further be processed with user specific functions, for example the offline pulse shape analysis 2D histogram visible in the upper right of the plot.

Such plugins are currently under development, please contact XIA for more information.

3.4 Optimizing Parameters

Optimization of the Pixie-Net'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 2µs, 4µs, 8µs, 11.2µs, 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 1.2µs. For a small detector (20% efficiency) a flat top of 0.8µs is a good choice. For larger detectors flat tops of 1.2µs and 1.6µs 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 0.853µs, 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 τ (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 τ is especially important at high count rates where pulses overlap more frequently. If τ is off the optimum, peaks in the spectrum will broaden, and if τ is very wrong, the spectrum will be significantly blurred.

A first rough estimate of τ can be obtained from the ADC traces. Fine tuning of τ can be achieved by exploring small variations around the estimated value (± 1-2 µs). This is best done at high count rates, as the effect on the resolution is more pronounced. The value of τ found through this way is also valid for low count rates. Manually enter τ, take a short run, and note the value of τ 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 into “FPGA units”, e.g. number of samples instead of us
- Apply limits and dependencies, if not ok, give feedback and abort
- Read data from PROM and verify the current version of the Pixie-Net hardware is supported by the software
- Write to FPGA registers 0x010-0x04F to apply settings
  Channel Reg2 has to be written once with “halt” bit set, once cleared
- Toggle I2C lines (0x002) to set gain etc. [May be moved to FPAG at some point]
- Issue DSP_CLR and RTC_CLR (0x008, 0x00A)

Arguments: fixed to “settings.ini”

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

4.2 gettraces, cgitraces

Read untriggered ADC data: 2K samples, 4 channels. Gettraces writes to a local file ADC.csv, cgitraces prints a webpage with the ADC data to std out, which can be piped into the webserver.

Functions performed

- (TODO Possibly read/apply xdt setting from .ini file for sampling period)
- Read ADC register in PL, write to file

Arguments: (possibly ini file name)

Output: ADC.csv

Notes: For the cgi program to access the PL, its device space must be set to r+w permission to all(?) users: `chmod 777 /dev/uio0`. The cgi program requires presence of the web page template adcpage.html in the same folder.

4.3 findsettings

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

Functions performed

- Resolve random channel swapping between channels 0/1 and 2/3.
- Ramp DACs and determine DAC setting for a baseline at 10% of the full ADC range
  (Note: Pulse polarity has to be set correctly for this function to work properly)

- TODO Acquire baselines and determine Tau
- TODO more?

Arguments: (possibly ini file name)

Output: prints found values for offset, tau, etc

4.4 Runstats, cgistats

Read output (run statistics) registers and write to file. Also performed as part of startdaq. The separate function is used for development, setup, and diagnostics, for example the run statistics include module revision and serial number, temperature, ICR and OOR. runstats writes to a local file ADC.csv, cgistats prints a webpage with the run statistcs data to std out, which can be piped into the webserver.

Functions performed

- Read Runstats registers in PL
- TODO Read I2C info from PROM, (eventually move I2C I/O inside PL)
- write to file

Arguments: none

Output: RS.csv

4.5 startdaq

Start list mode run with or without waveforms (0x400, 0x500, 0x501, 0x502) or MCA run (0x301). Pulse shape analysis data is acquired in run types 0x400 and 0x502 for modules licensed for that function.

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 function `acquire` is more sophisticated and has higher throughput.

Functions performed

- Parse .ini file, extract parameters
- Compute coefficients for E reconstruction etc
- Set bits in reg 0x000 to start run
    - RunEnable -> 1 (an overall enable of the run)
    - nLive -> 0 (can be used to temporarily pause acquisition (rarely used))
- Monitor DAQ
    - Poll EVSTATS (register 0x101), if data ready:
read event data from register 0x1N#,
compute E,
read waveforms from register 0x30N,

write to LM file (if LM run),
increment MCA
    - Periodically save MCA to file (4Ki bins)
    - Periodically read runstats from register 0x2## and write to file
    - Periodically save PSA 2D histogram data to file
    - Periodically read BL data from register 0x1N#, 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, PSA.csv, LM file (text only)

4.6 acquire

Start list mode run with or without waveforms (0x400), coincidence list mode run (0x402), or MCA run (0x300). A more sophisticated and faster version of `startdaq`. Saves binary list mode data compatible with Pixie-4e.

Functions performed

- Parse .ini file, extract parameters
- Compute coefficients for E reconstruction etc
- Set bits in reg 0x000 to start run
    - RunEnable -> 1 (an overall enable of the run)
    - nLive -> 0 (can be used to temporarily pause acquisition (rarely used))
- Monitor DAQ
    - Poll EVSTATS (register 0x101), if data ready:
      read event data from register 0x1N#,
      compute E,
      read waveforms from register 0x30N,
      write to LM file (if LM run),
      increment MCA
    - Periodically save MCA to file (4 Ki bins)
    - Periodically read runstats from register 0x2## and write to file
    - Periodically read BL data from register 0x1N#, compute BLavg
- Stop when time is up
- Save final run statistics and MCA to file (32 Ki bins)

Arguments: optional filename for LM file, optional filename for .ini file with parameters, other file names fixed

Output: MCA.csv, RS.csv, LM file (binary)

4.7 coincdaq

Start list mode run in coincidence mode without waveforms (0x503). Coincidence more runs build 4-channel event records that reduce or eliminate the need to parse through list mode data and find records close in time. 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 function `acquire` is more sophisticated and has higher throughput.

Functions performed

- Parse .ini file, extract parameters
- Compute coefficients for E reconstruction etc
- Set bits in reg 0x000 to start run
    - RunEnable -> 1 (an overall enable of the run)
    - nLive -> 0 (can be used to temporarily pause acquisition (rarely used))
- Monitor DAQ
    - Poll EVSTATS (register 0x101), if data ready from ALL channels:
      read event data from register 0x1N#,
      compute E,
      read waveforms from register 0x30N,
      write to LM file,
      increment MCA
    - Periodically save MCA to file (4Ki bins)
    - Periodically read runstats from register 0x2## and write to file
    - Periodically read BL data from register 0x1N#, compute BLavg
- Stop when time is up
- Save final run statistics, and MCA to file (32Ki bins)

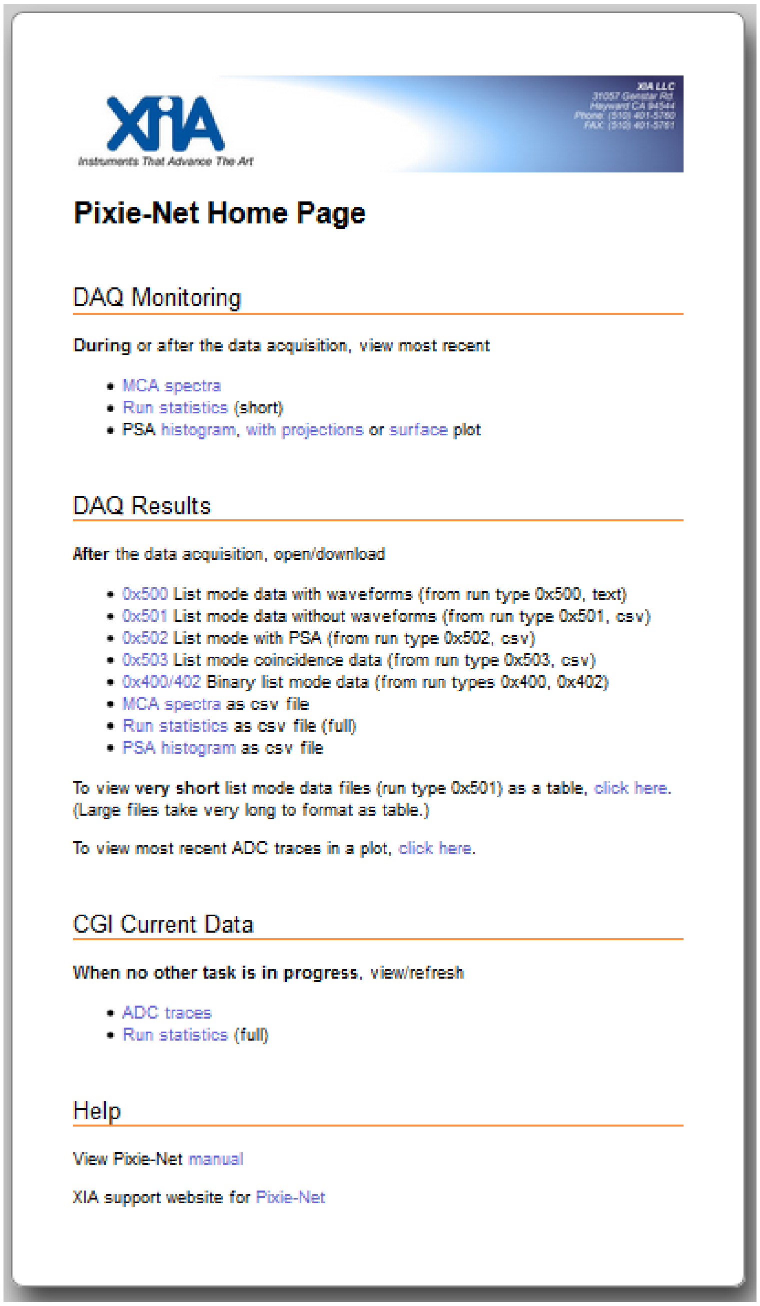
Arguments: none (file names are fixed)

Output: MCA.csv, RS.csv, LM file (text only)

5 Web Pages

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

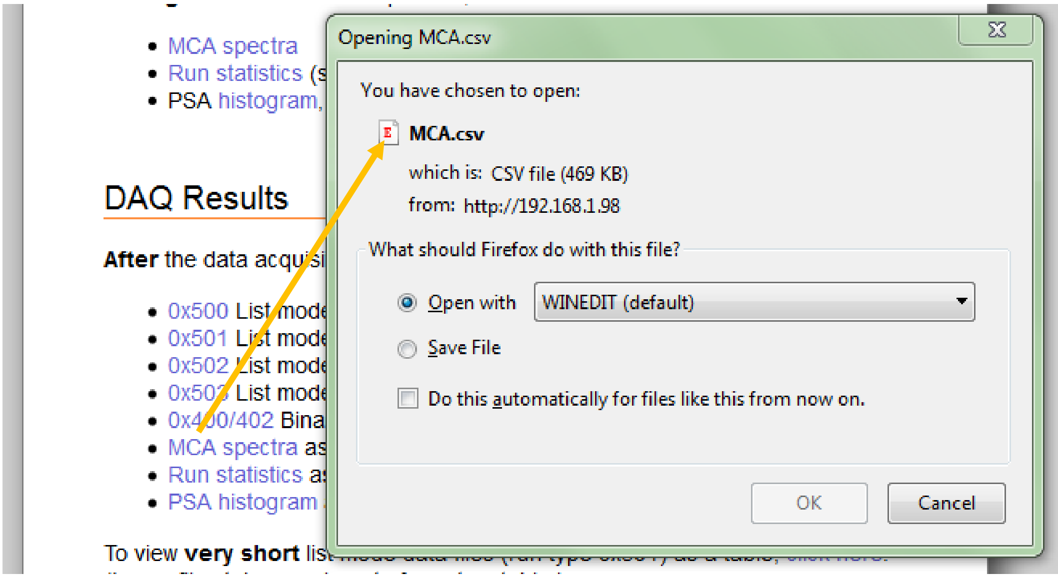
5.1 index.html



This is the home page for the Pixie-Net.

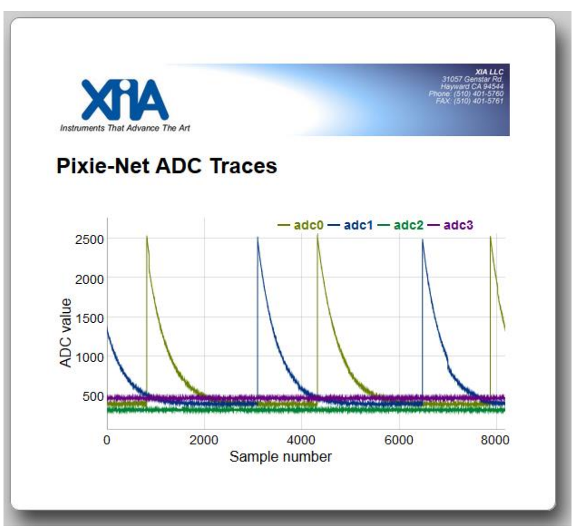
Under **DAQ Monitoring**, there are links to view (in the browser) the csv files from the current or most recent data acquisition for MCA spectra and run statistics.

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.



Under **CGI links**, procedures are executed directly on the Pixie-Net, their output generating webpages for ADC traces and other results. This combines the two steps of i) executing a function on the Pixie-Net terminal to create data and ii) refreshing the appropriate webpage to view the results.

5.2 adcpage.html

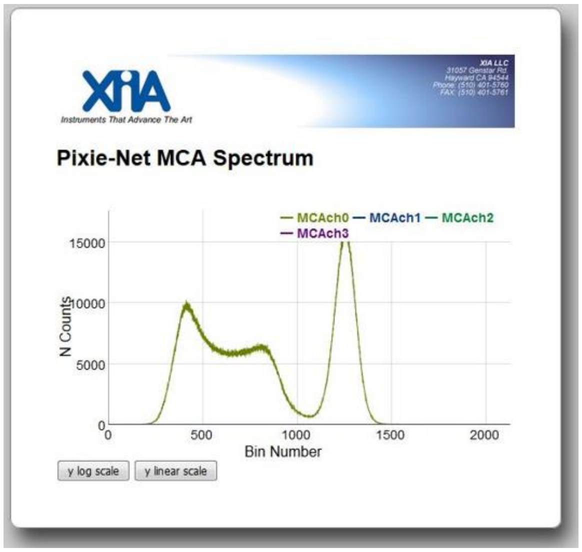


The adcpage contains a chart showing untriggered wavefroms read from the Pixie-Net's ADCs. Internally, this is a dygraph javascript that links to the file ADC.csv. The csv file has to be generated or refreshed by executing `gettraces` on the Pixie-Net terminal.

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

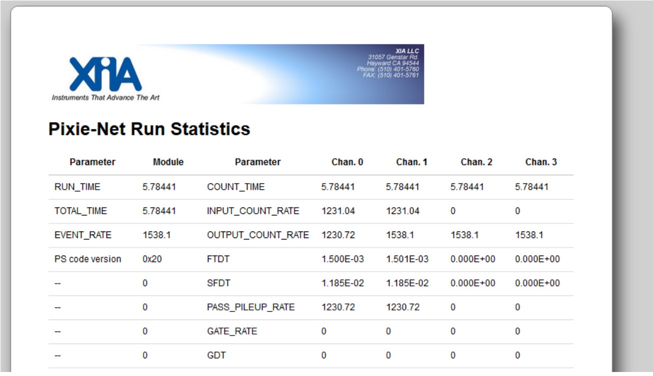
An equivalent page reading and displaying the data can be generated by a cgi script from the web browser when no data acquisition is in progress (section 5.1, **CGI links**).

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 file MCA.csv. The file is generated and periodically refreshed by executing `startdaq` or `acquire` on the Pixie-Net terminal.

5.4 rspage.html



Parameter	Module	Parameter	Chan. 0	Chan. 1	Chan. 2	Chan. 3
RUN_TIME	5.78441	COUNT_TIME	5.78441	5.78441	5.78441	5.78441
TOTAL_TIME	5.78441	INPUT_COUNT_RATE	1231.04	1231.04	0	0
EVENT_RATE	1538.1	OUTPUT_COUNT_RATE	1230.72	1538.1	1538.1	1538.1
PS code version	0x20	FTDT	1.500E-03	1.501E-03	0.000E+00	0.000E+00
--	0	SFDT	1.185E-02	1.185E-02	0.000E+00	0.000E+00
--	0	PASS_PILEUP_RATE	1230.72	1230.72	0	0
--	0	GATE_RATE	0	0	0	0
--	0	GDT	0	0	0	0

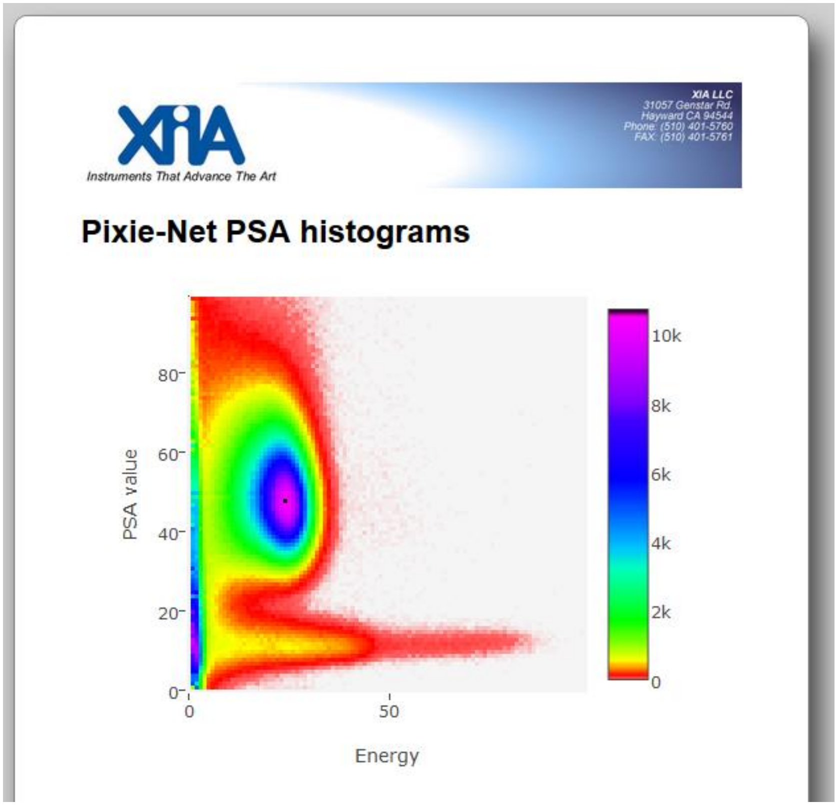
The rspage contains a table showing run statistics from the current or most recent data acquisition. Internally, this is a d3 javascript that links to the file RS.csv. The file is generated and periodically refreshed by executing `./startdaq` or `./acquire` on the

Pixie-Net terminal. It can also be generated outside a data acquisition by executing `./runstats` on the Pixie-Net terminal (actual run statistics will not change, but a real time counter increments and some hardware and status information may be of interest).

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 (section 5.1, **CGI links**).

Units in the page (and the underlying csv file) are in seconds or counts/s for the first ~8 lines, then raw internal units (counts and ns).

5.5 psahistpage.html

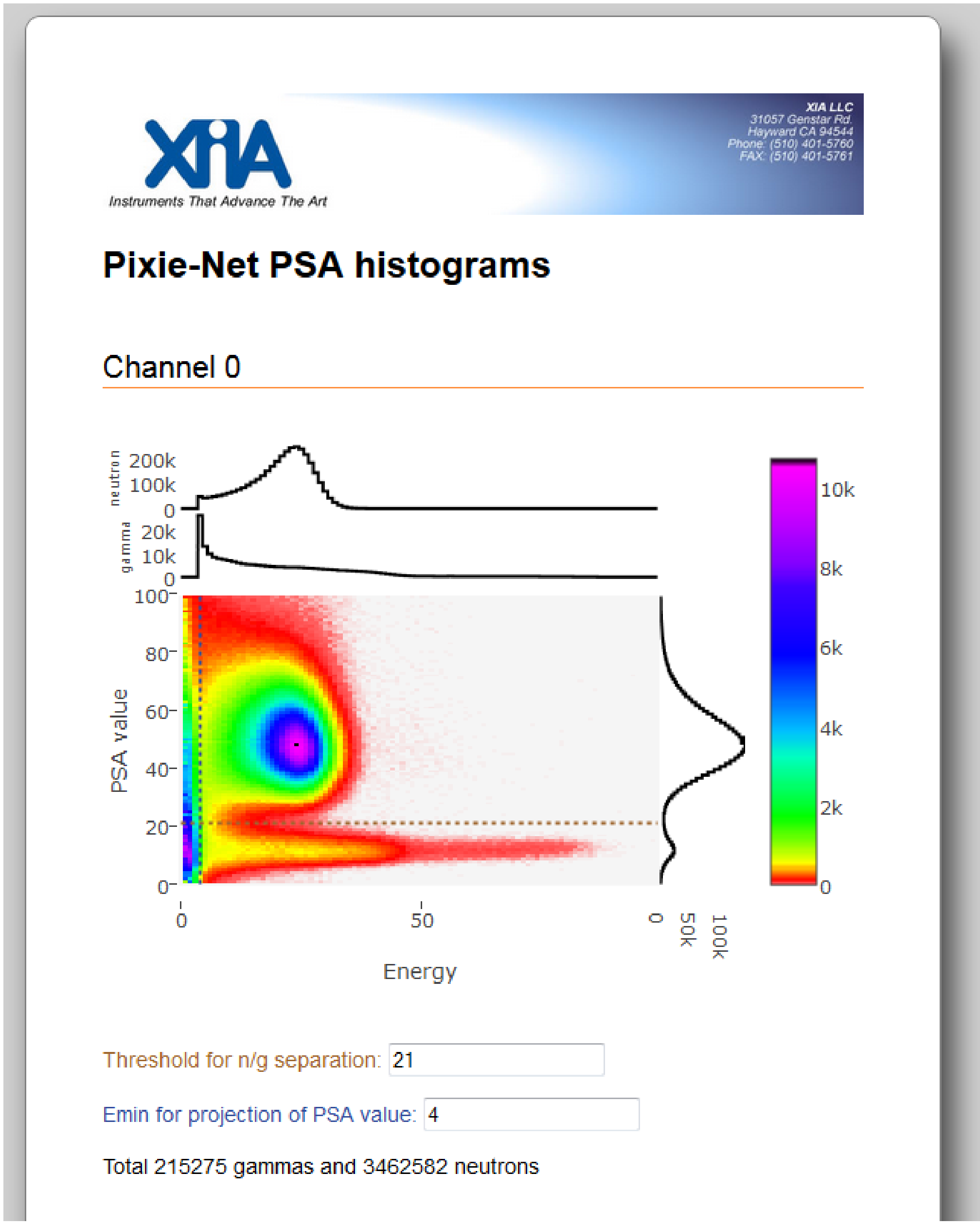


The psahistpage shows results of the pulse shape analysis (PSA) as a 2D histogram. The PSA, as described in section 11.8, computes two sums over characteristic regions of the detector pulse, and calculates the ratio (PSA value) of the two sums. The PSA value is a condensed characteristic of the pulse shape, and can be used with suitable detectors to distinguish event or particle types, for example gammas and neutrons or gammas and alphas.

In the psahistpage, the PSA value (y) is plotted against the pulse energy E (x), and each pixel is colored by intensity. Typically neutrons and gammas fall into two separate branches in the plot.

Internally, this is a plotly javascript that links to the file PSA.csv. The file is generated and periodically refreshed by executing `startdaq` on the Pixie-Net terminal. Hovering over the plot makes visible options to zoom and save. The histogram has 100 bins in each direction.

5.6 psahistprojpage.html

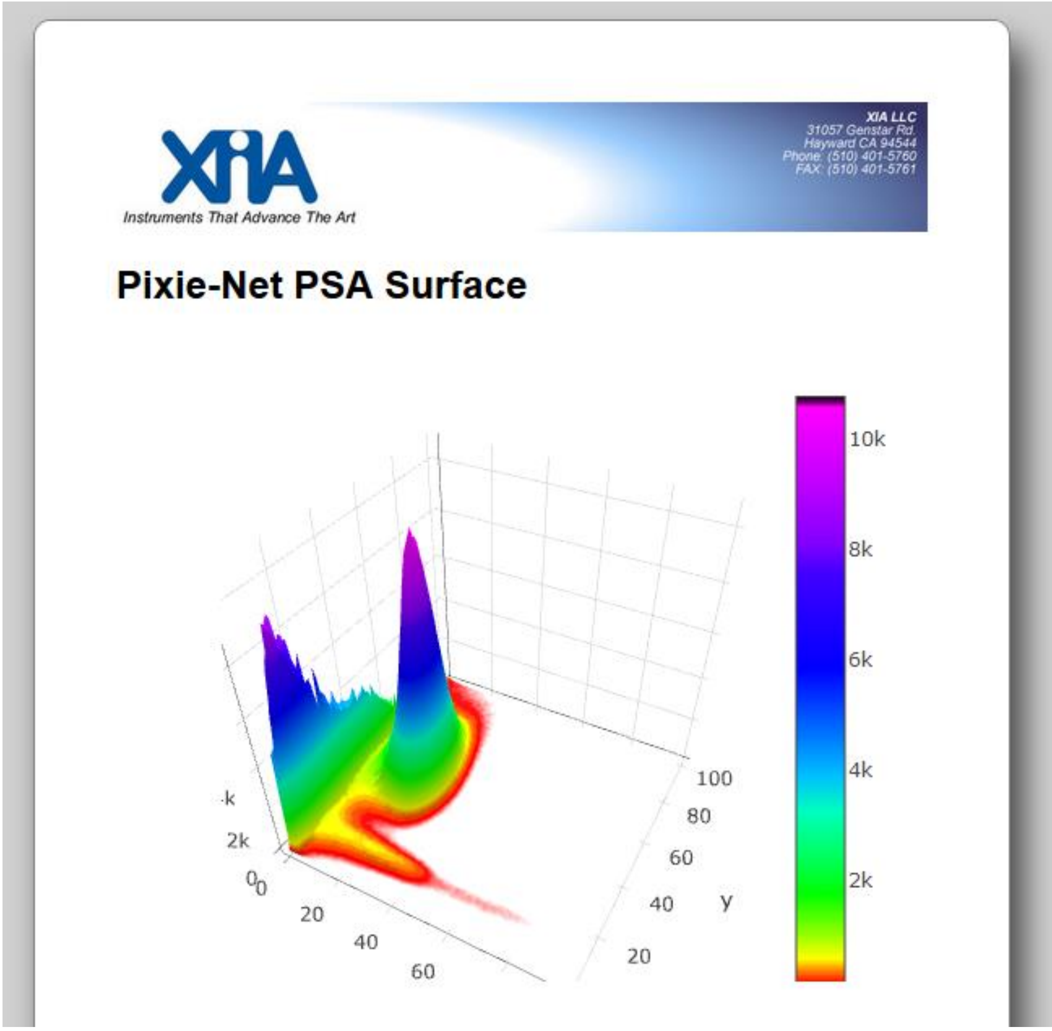


The psahistprojpage shows the same data as the psahistpage, but additionally computes

- a projections to the y axis for all bins > Emin
- a projection to the x axis for all bins > n/g threshold
- a projection to the x axis for all bins < n/g threshold
- the number of gammas and neutrons (E >Emin, PSA value >/< n/g threshold)

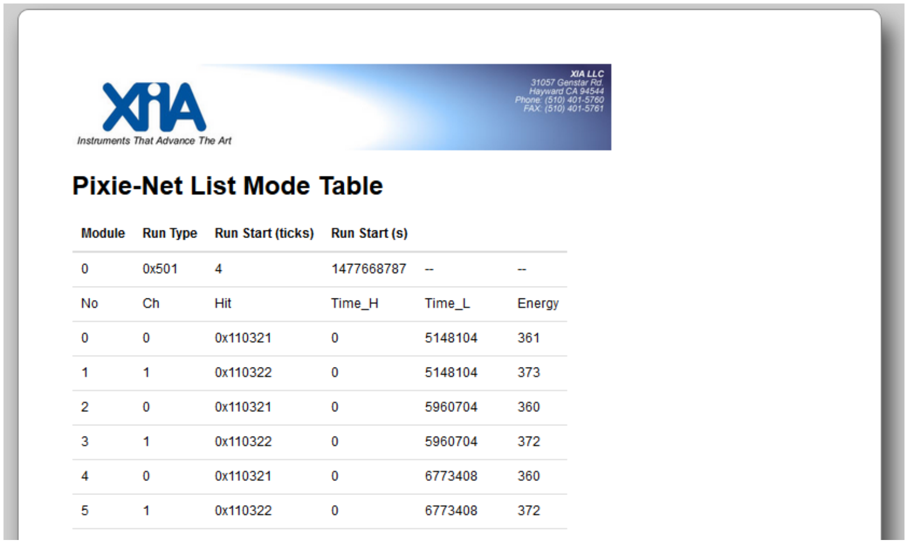
Emin and n/g threshold can be entered below the plot. Clicking the browser's refresh button will recalculate the projections.

5.7 psasurfacepage.html



The psasurfacepage shows the same data as the psahistpage, but as a 3D surface plot instead of a 2D histogram. Hovering over the plot makes visible options to rotate and pan the plot.

5.8 lmtablepage.html



XIA LLC
31057 Genstar Rd.
Hayward CA 94544
Phone: (510) 401-5760
FAX: (510) 401-5761

Instruments That Advance The Art

Pixie-Net List Mode Table

Module	Run Type	Run Start (ticks)	Run Start (s)		
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
4	0	0x110321	0	6773408	360
5	1	0x110322	0	6773408	372

The lmtablepage contains a table showing the results from a list mode run of run type 0x501, which generates a csv table with hit pattern, time stamps, and energy (but no waveforms). The first two lines of the table list run type and start time.

The time stamps units are nanoseconds. The energies reported are in internal units, calibration to keV is required.

Note: This page will load very slowly for large data files. Download of the data file and analysis in a text editor or spreadsheet program is recommended.

6 Parameters in the Settings Files

The ini files contains 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 system. 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.) 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 65536 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**
NUMBER_CHANNELS	4	Number of channels
C_CONTROL	Bits, 0x0-FFFF	unused
REQ_RUNTIME	**Seconds, 5..2^32**	**Requested run time in (full) seconds**
POLL_TIME	Tbd, 100..2^32	Number of internal polling loops between updates of csv output files. In the order of microseconds
SYS_U11		unused
SYS_U10		unused
SYS_U9		unused
SYS_U8		unused
SYS_U7		unused
SYS_U6		unused
SYS_U5		unused
SYS_U4		unused
SYS_U3		unused
SYS_U2		unused
SYS_U1		unused
SYS_U0		unused

6.2 Module Parameters

Parameter Name	Units Limits	Description
MCSRA_CWGROUP_00	**0 or 1**	**In coinc mode runs (0x503 or 0x402), save only one record per coincidence window**
MCSRA_U_01	0 or 1	Module control bit A.1, unused
MCSRA_U_02	0 or 1	Module control bit A.2, unused
MCSRA_U_03	0 or 1	Module control bit A.3, unused
MCSRA_U_04	0 or 1	Module control bit A.4, unused
MCSRA_FPVETO_05	0 or 1	Module control bit A.5, if 1, enables MMCX input as global Veto, NYI
MCSRA_U_06	0 or 1	Module control bit A.6, unused
MCSRA_FPPEDGE_07	0 or 1	Module control bit A.7, toggles active edge for front panel pulse counter, NYI
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
MCSRB_U_00	0 or 1	Module control bit B.0, unused
MCSRB_TERM01_01	**0 or 1**	**Module control bit B.1, if 1, termination for ch.0 and 1 is 50 Ohm, else “high impedance”**
MCSRB_TERM23_02	**0 or 1**	**Module control bit B.2, if 1, termination for ch.2 and 3 is 50 Ohm, else “high impedance”**
MCSRB_U_03	0 or 1	Module control bit B.3, unused
MCSRB_PDCH0_04	0 or 1	Module control bit B.4, power down ADC driver for ch.0, NYI
MCSRB_PDCH1_05	0 or 1	Module control bit B.5, power down ADC driver for ch.0, NYI
MCSRB_PDCH2_06	0 or 1	Module control bit B.6, power down ADC driver for ch.0, NYI
MCSRB_PDCH3_07	0 or 1	Module control bit B.7, power down ADC driver for ch.0, NYI
MCSRB_U_08	0 or 1	Module control bit B.8, unused
MCSRB_U_09	0 or 1	Module control bit B.9, unused
MCSRB_U_10	0 or 1	Module control bit B.10, unused
MCSRB_U_11	0 or 1	Module control bit B.11, unused
MCSRB_U_12	0 or 1	Module control bit B.12, unused
MCSRB_U_13	0 or 1	Module control bit B.13, unused
MCSRB_U_14	0 or 1	Module control bit B.14, unused
MCSRB_U_15	0 or 1	Module control bit B.15, unused

COINC_PATTERN_0000	**0 or 1**	**If 1, record events with no channel hit. Useful only for external triggers**
COINC_PATTERN_0001	**0 or 1**	**If 1, record events with hitpattern ch 3210 = #### (#### are the last 4 digits of the parameter name) For example, if COINC_PATTERN_0011=1, events with pulses in both channel 0 and 1 will be recorded (if rising edges occur within coincidence window length) If multiple COINC_PATTERN_#### are 1, events matching any of the patterns are recorded. For example if COINC_PATTERN_0111=1 and COINC_PATTERN_1011=1 and COINC_PATTERN_1101=1 and COINC_PATTERN_1110=1, pulses are recorded if exactly 3 channels have a pulse, but no matter which.**
COINC_PATTERN_0010	**0 or 1**	
COINC_PATTERN_0011	**0 or 1**	
COINC_PATTERN_0100	**0 or 1**	
COINC_PATTERN_0101	**0 or 1**	
COINC_PATTERN_0110	**0 or 1**	
COINC_PATTERN_0111	**0 or 1**	
COINC_PATTERN_1000	**0 or 1**	
COINC_PATTERN_1001	**0 or 1**	
COINC_PATTERN_1010	**0 or 1**	
COINC_PATTERN_1011	**0 or 1**	
COINC_PATTERN_1100	**0 or 1**	
COINC_PATTERN_1101	**0 or 1**	
COINC_PATTERN_1110	**0 or 1**	
COINC_PATTERN_1111	**0 or 1**	
COINCIDENCE_WINDOW	**µs, 0.040-4.088**	**Coincidence window length**
RUN_TYPE	**0x301, 0x400, 0x402, 0x500, 0x501, 0x502, 0x503**	**MCA only run List mode run with waveforms (binary .b00) Coincidence list mode run (binary .b00) 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)**
FILTER_RANGE	**1..6**	**Decimation/averaging for energy filter See note 1**
ACCEPT_PATTERN	Bits, 0x0020	Controls what type of event to accept (MODULEPATTERN)
SYNC_AT_START	0..1	Reset timers at runstart
HV_DAC	0..5V	Voltage on HV control output
SERIAL_IO	Bits, 0x0-FFFF	Bit pattern for offboard serial data
AUX_CTRL	Bits, 0x0-FFFF	Bit pattern for pulser, LED, etc Bit0 : pulser enabled
MOD_U7		Unused
MOD_U6		Unused
MOD_U5		Unused
MOD_U4		Unused
MOD_U3		Unused
MOD_U2		Unused
MOD_U1		Unused
MOD_U0		Unused

6.3 Channel Parameters

Parameter Name	Units Limits	Description
CCSRA_GROUP_00	**0 or 1**	**Channel control bit A.0, if 1, respond to distributed group triggers, not local triggers. Ignored (always 1) in Run Types 0x403, 0x503**
CCSRA_U_01	0 or 1	Channel control bit A.1, unused
CCSRA_GOOD_02	**0 or 1**	**Channel control bit A.2, if 0, channel will not be processed**
CCSRA_U_03	0 or 1	Channel control bit A.3, unused
CCSRA_TRIGENA_04	**0 or 1**	**Channel control bit A.4, if 1, enable trigger (local and distributed)**
CCSRA_INVERT_05	**0 or 1**	**Channel control bit A.5, if 1, ADC data is inverted before processing (for falling edge pulses)**
CCSRA_VETO_REJLO_06	0 or 1	Channel control bit A.6, if 1, reject events when global Veto is low
CCSRA_U_07	0 or 1	Channel control bit A.7, unused
CCSRA_U_08	0 or 1	Channel control bit A.8, unused
CCSRA_NEGE_09	0 or 1	Channel control bit A.9, if 1, allow negative numbers as result of energy computation, NYI
CCSRA_U_10	0 or 1	Channel control bit A.10, unused
CCSRA_U_11	0 or 1	Channel control bit A.11, unused
CCSRA_GATE_REJLO_12	0 or 1	Channel control bit A.12, if 1, reject events when channel-specific GATE signal is low
CCSRA_U_13	0 or 1	Channel control bit A.13, unused
CCSRA_U_14	0 or 1	Channel control bit A.14, unused
CCSRA_U_15	0 or 1	Channel control bit A.15, unused
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_VETO_REJHI_00	0 or 1	Channel control bit C.0, if 1, reject events when global Veto is high
CCSRC_GATE_REJHI _01	0 or 1	Channel control bit C.1, if 1, reject events when channel-specific Gate signal is high
CCSRC_GATE_FROMVETO_02	0 or 1	Channel control bit C.2, if 1, use global Veto as the input for this channel's Gate logic
CCSRC_PILEUP_DISABLE_03	0 or 1	Channel control bit C.3, if 1, disable pileup rejection
CCSRC_RBAD_DISABLE_04	0 or 1	Channel control bit C.4, if 1, disable rejection of out-of-range events
CCSRC_PILEUP_INVERT_05	0 or 1	Channel control bit C.5, if 1, accept only pulses that are piled up
CCSRC_PILEUP_PAUSE_06	0 or 1	Channel control bit C.6, if 1, disable pileup inspection for 32 clock cycles after trigger. For ringing input signals.
CCSRC_GATE_FEDGE_07	0 or 1	Channel control bit C.7, if 1, count Gate pulses on falling edge
CCSRC_GATE_STATS_08	0 or 1	Channel control bit C.8, if 1, run statistics are in GATE mode, only counting while GATE in on
CCSRC_VETO_FEDGE_09	0 or 1	Channel control bit C.9, if 1, count Veto pulses on falling edge
CCSRC_GATE_ISPULSE_10	0 or 1	Channel control bit C.10, if 1, logic to re-pulse incoming Gate signal with specified GATE_WINDOW is enabled
CCSRC_U_11	0 or 1	Channel control bit C.11, unused
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_CPC2PSA_14	0 or 1	Channel control bit C.14, | if 1, report gate pulse count as PSA value of list mode record
CCSRC_GATE_PULSEFEDGE_15	0 or 1	Channel control bit C.15, if 1, start pulse GATE_WINDOW at falling edge of Gate input signal
ENERGY_RISETIME	**µs, 0.032 .. 62.976**	**Energy filter rise time See Note 1) and section 3.4.1**
ENERGY_FLATTOP	**µs, 0.048 .. 62.976**	**Energy filter flat top See Note 1)) and section 3.4.1**
TRIGGER_RISETIME	**µs, 0.016 .. 0.480**	**Trigger filter rise time See note 2)) and section 3.4.2**
TRIGGER_FLATTOP	**µs, 0.048 .. 0.0488**	**Trigger filter flat top See note 2) and section 3.4.2**
TRIGGER_THRESHOLD	**(ADC steps) 0..4096**	**Trigger threshold See note 3 and section 3.4.2**

ANALOG_GAIN	**2 or 5**	**Gain with switches/relays/VGAs** **Values may vary for different HW variants**
DIG_GAIN	**Positive float**	**Digital gain adjustment factor.** **Measured ADC amplitude is multiplied with this factor before histogramming and reporting in list mode file.** **Binning artefacts can appear with very small or very large numbers**
VOFFSET	**V** **-1.25..1.25**	**Analog offset** **Used to compensate detector baseline offsets**
TRACE_LENGTH	**µs,** **0..16**	**Captured waveform length**
TRACE_DELAY	**µs,** **0..16**	**Pre-trigger delay**
PSA_START	µs, 0..16	Start for PSA in waveform, reserved
PSA_END	µs, 0..16	End for PSA in waveform, reserved
BINFACTOR	1-16	MCA binning factor: divide measured pulse height by by 2^N before binning
TAU	**µs** **positive float**	**Preamplifier decay time** **See section 3.4.3**
BLCUT	Positive int	Threshold for bad baseline measurements See section 3.4.4
XDT	µs	Sampling interval in untriggered traces, NYI
BASELINE_PERCENT	0-99	Target offset for baseline, nominally in percent, NYI
PSA_THRESHOLD	ADC steps, 0..2044	Threshold in CFD and PSA,
INTEGRATOR	0..2	Filter mode: 0-trapezoidal, 1-gap sum integral, NYI 2-ignore gap sum, NYI
GATE_WINDOW	µs, 0..2.04	Coincidence window with gate
GATE_DELAY	µs, 0..2.04	Delay of external gate signal
COINC_DELAY	µs, 0..1.02	Delay of ADC signal before coincidence test, equivalent to a cable delay
BLAVG	65535.. 65528 and 0	Baseline averaging See section 3.4.4
QDC0_LENGTH	**samples** **2..62**	**Length of PSA sum** **See section 11.8**
QDC1_LENGTH	**samples** **2..62**	**Length of PSA sum** **See section 11.8**
QDC0_DELAY	**samples** **0..252**	**Delay of PSA sum relative to trigger point** **See section 11.8**

QDC1_DELAY	**Samples 0..252**	**Delay of PSA sum relative to trigger point** **See section 11.8**
QDC_DIV8	**0..1**	**If 1, divide PSA sums by an extra factor 8 (to avoid overflow for long sums)** **See section 11.8**
MCA2D_SCALEX	**Positive float <655**	**The energy (max 64Ki) is divided by this factor before binning into the 2D PSA histogram (max 100 bins)**
MCA2D_SCALEY	**Positive float <655**	**The PSA value (max 64Ki) is divided by this factor before binning into the 2D PSA histogram (max 100 bins)**
PSA_NG_THRESHOLD	Positive float	Threshold for PSA value to distinguish neutrons and gammas, NYI
CHAN_U3		Unused
CHAN_U2		Unused
CHAN_U1		Unused
CHAN_U0		Unused

Notes

1. Energy filter rise time and flat top depend on Filterrange. Higher filter ranges have allow longer filter times but with increased coarseness. Maximum combined length is 126 x 0.008µs* 2^FR

Filter range	Filter granularity	max. T_{rise}+T_{flat}	min. T_{rise}	min. T_{flat}
1	0.016µs	2.032µs	0.032µs	0.048µs
2	0.032µs	4.064µs	0.064µs	0.096µs
3	0.064µs	8.128µs	0.128µs	0.192µs
4	0.128µs	16.256µs	0.256µs	0.384µs
5	0.256µs	32.512µs	0.512µs	0.768µs
6	0.512µs	65.024µs	1.024µs	1.536µs

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

2. Trigger filter rise time and flat top maximum combined length is 63 x 0.008µs
3. The internal triggering compares the output of the trigger filter (technically an area) with (TRIGGER_THRESHOLD * TRIGGER_RISETIME / 8), which must be <1024. 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 currently six types of list mode data acquisition, with waveforms and without, binary or text, and special purposes. The following table gives an overview of their differences. It is quite straightforward to modify the code in e.g. startdaq to generate application specific output files.

Run Type	File format	Wave-forms	Function	File extension	Notes
0x400	binary	Yes	startdaq, acquire	.b00	General purpose, fast, compatible with Pixie-4e
0x402	binary	Yes	coincdaq	.b00	Coincidences, compatible with Pixie-4e
0x500	text	Yes	startdaq	.txt	General purpose, slow
0x501	text	No	startdaq	.dat	General purpose
0x502	text	No	startdaq	.dt2	PSA
0x503	text	No	coincdaq	.dt3	Coincidences

7.1.1 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 µs 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 ns)	0
8	Event header: lower 32 bit of FPGA time stamp (in ns)	26711904
9	Event header: energy	855
10	Event header: <reserved>	0
11	Event header: <reserved>	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[1]
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[1] (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	Reserved
31	If set, indicates a data transmission error has been detected for this event. Parts of header and waveform may be corrupted

Table 7-1: Event pattern and Event info in Run Type 0x400, 0x500, 0x501, total 32 bits.

7.1.2 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).

[1] 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.

7.1.3 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 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).

7.1.4 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.

7.1.5 Binary list mode files with waveforms (Run Type 0x400)

Run type 0x400 creates single-channel event records in binary format, **LMdata.b00**. This format is compatible with the Pixie-4e. XIA provides a utility to convert this data format to the upcoming IEC63047 list mode standard. 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 7-2: 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
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--15	PSA Values	
16--31	reserved	

Table 7-3: Channel header for Run Type 0x400, total 32 words (16bit).

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.

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

Run type 0x404 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 7-4: Channel header for Run Type 0x402, total 32 words (16bit).

7.2 Run Statistics Files

Run Statistics files contain comma separated values. The first row lists the column headers. The columns are

`ParameterM` Name of module parameter

`Module` Value of module parameter

`ParameterC` 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 full definition and explanation of the values, please see sections 8.3 and 9.3.

For example, the line

```
TOTAL_TIME,5.78441,INPUT_COUNT_RATE,1231.04,1231.04,0,0
```

reports an acquisition TOTAL TIME of 5.78441s and an input rate of 1231.04 counts/s for channel 0 and channel 1.

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,MCAch0,MCAch1,MCAch2,MCAch3
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
```

7.4 PSA Files

PSA files, generated in Run Type 0x502, contain comma separated values. The first row lists the column headers, i.e. PSA value bin numbers – 100 bins per channel numbered from 0 to 399. The first column lists the energy bin number – 100 bins numbered from 0 to 99.

```
,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 …
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
1,0,927,5030,4657,4735,5798,6045,6773
2,0,4801,3505,4590,4712,5995,6901,769
3,0,1993,1978,2165,2370,2984,3618,425
4,0,496,577,642,795,1084,1318,1474,15
5,0,208,234,255,356,509,626,692,716,7
6,0,138,137,199,250,337,435,512,528,5
7,0,78,98,140,205,258,336,402,479,545
8,0,49,69,110,155,240,303,403,467,506
9,0,42,51,77,143,206,309,349,437,514,
...
```

8 Module 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 go in depth to understand this functionality. The primary way to set registers is by specifying parameters in the .ini file and call XIA's API function `progfippi` 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.

FPGA (PL) I/O registers are organized into 4 blocks. All registers are 32bit wide. All writeable registers are in block 0. To read registers, the output block number has to be written to address 0x003; this sets the upper digit of the read addresses. For example, writing 2 to the OUTBLOCK register (0x003) selects read address range 0x200-0x2FF.

8.1 Input Registers

Address range is 0x000 – 0x00F. Can be read back to verify I/O

Use these to specify parameters that control the data acquisition

Register	Address	R/W	Description
CSRIN	0x000	R/W	Run Control Register bits 0 RunEnable (set to start DAQ run) 9 nLive (set to 1 to pause DAQ run)
COINCPATTERN	0x001	R/W	Allowed coincidence pattern and other control bits 0..15 CoincPattern 16 LMRUN402 (coincidence mode) 17 MCARUN (no trace out mode) 18 CWGROUP (only 1 record per CW)
I2C	0x002	R/W	Control the SDA and SCL lines 0 SDA 1 SCL 2 SDA ENA (SDA output enable)
OUTBLOCK	0x003	R/W	Specifies address range for reads If 0: 0x000-0x04F If 1: 0x100-0x14F If 2: 0x200-0x29F If 3: 0x300-0x303
HV_DAC	0x004	R/W	HV DAC value (16 bit)
SERIAL_IO	0x005	R/W	Value for offboard serial IO (16 bit)
AUX_CTRL	0x006	R/W	Aux Control bits for HW 0 pulser enable 1 LED red on/off (NYI)
ADC_CTRL	0x007	R/W	Controls certain aspects of ADC operation 0 swap channel 0/1 data streams 1 swap channel 2/3 data streams
DSP_CLR	0x008	W	Writing to this register issues a dspclr pulse (processing init)
COUNTER_CLR	0x009	W	Writing to this register issues a pulse to clear runstats counters
RTC_CLR	0x00A	W	Writing to this register issues a pulse to clear RTC time counter

	0x00C- 0x00F	R/W	Reserved

8.2 Event Registers

Address range is 0x100 – 0x10F. Read only

Use these during the run to get info of the current status

Register	Address	R/W	Description
CSROUT	0x100	R	Run Status info bits 0 RunEnable (set to start DAQ run) 2 SDA readback 4 zdtfull 9 nLive 10 PSA enabled 11 VetoIn 13 ACTIVE (=RunEnable) 15..31 debug
EVSTATS	0x101	R	EVSTATS (Event status information) 0 DataReadyA (if 1, there is data in channel's ZDT buffer) 1 DataReadyB 2 DataReadyC 3 DataReadyD
reserved	0x102- 0x10F	R	
PPSTIME	0x103	R	Current PPStime (local time latched with external trigger)
reserved	0x104	R	
EVTIME_L	0x105	R	Event time stamp M, L (mode 0x402, 503)
EVTIME_H	0x106	R	Event time stamp X, H (mode 0x402, 503)
EVPPS	0x107	R	Event PPS time (mode 0x402, 503)

8.3 Run Statistics Registers

Address range is 0x200 – 0x21F. Read only

Used for run statistics and other output values. Sometimes two 16bit words per address

Unit “ticks” means 2ns clock ticks

Index in runstats para-meter list	PL address	units	Parameter name	Description
0	0x200	Bits	L CSROUT H CSROUT	See above, addr 0x100
1	0x201	ns	L SYSTIME_L H SYSTIME_M	Time since last STC reset (lower 32 bit)
2	0x202	ns	L RUNTIME_L H RUNTIME_M	RunTime for current/last run (lower 32 bit)
3	0x203	ns	L RUNTIME_H H RUNTIME_X	RunTime for current/last run (upper 32 bit)
4	0x204	ns	L TOTALTIME_L H TOTALTIME _M	TotalTime for current/last run (lower 32 bit)
5	0x205	ns	L TOTALTIME _H H TOTALTIME _X	TotalTime for current/last run (upper 32 bit)
6	0x206		L NUMEVENTS_L H NUMEVENTS _M	Number of events in current/last run (lower 32 bit)
7	0x207		L NUMEVENTS _H H NUMEVENTS _X	Number of events in current/last run (upper 32 bit)
8	0x208		L EHL H BHL	Event Header Length Buffer Header Length
9	0x209		L FIFOLENGTH H CHL	Length of waveform capture FIFO Channel Header Length
10	0x20A		L FIPREVISION H SYSREVISION	FW Revision numbers
11	0x20B		L SERIAL_NUMBER H --	Serial number of the module
12-31	0x20C – 0x21F		reserved	

Output values derived from the run statistics parameters are

1. EVENT_RATE = NUMEVENTS / RUNTIME

9 Channel 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 go in depth to understand this functionality. The primary way to set registers is by specifying parameters in the .ini file and call XIA's API function `progfippi` 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.

Channel I/O registers are organized into 4 blocks. All registers are 32bit wide. All writeable registers are in block 0. To read registers, the output block number has to be written to address 0x003; this sets the upper digit of the read addresses. For example, writing 2 to the OUTBLOCK register (0x003) selects read address range 0x200-0x2FF.

9.1 Input Registers

Address range is 0x0N0 – 0x0NF, N=1..4. Can be read back to verify I/O

Use these to specify parameters that control the data acquisition

Register	Address	R/W	Description
0	0x0N0	R/W	CCSRA, CCSRC: 0 GROUPTRIG (local or dist. trigger) 1 reserved 2 GOOD (enables triggers) 3 reserved 4 TRIGENA (trigger enabled) 5 INVERT (ADC input polarity) 6 VETOENA (GFLT required) 7 reserved 8 reserved 9 NEGE (NYI) 10 CFD TIME (NYI) 11 reserved 12 GATE_ENA (enable GATE rejection) 13 LOCALTIME (channel time stamp source) 14 ESTIMEATE_E (NYI) 15 reserved 16 VETO_INV (invert GFLT/Veto) 17 GATE_INV (polarity for TrigCtrl) 18 VETO2GATE (use GFLT/Veto for Gate) 19 PILEUP_DISABLE 20 RBAD_DISABLE (disable Rangebad) 21 PILEUP INVERT 22 PILEUP_PAUSE 23 GATE_EDGEINV (select r/f edge for GATE) 24 GATE_STATS (gate statistics, NYI) 25 GDT_ALLOW (ignored) 26 GATE_NOPULSE (use input directly, no edge) 27 reserved 28 GateIsBit0 29 GATE_OUT (to FP after invert/repulse etc)
1	0x0N1	R/W	0-6 SL (Slow Length)

			8-14 SLSG (Slow length + gap) 16-22 SG (slow filter gap for BL pileup) 24-30 RDEL2-8 (delay for clearing Rangebad, bits 0,1=0)
2	0x0N2	R/W	0-5 FL (Fast Length) 8-13 FLFG (Fast length + gap) 16-25 THRE (trigger threshold) 26-29 DEC (filter decimation) 31 HALT (stop filters)
3	0x0N3	R/W	0-6 PEAKSAM (sample slow filter for E sum) 13-25 PEAKSEP (pileup inspection period) 26+ GAIN
4	0x0N4	R/W	0-15 DAC value
5	0x0N5	R/W	0-8 User Delay/4 (pretrigger trace) 29 NONZERO_TRACE
6	0x0N6	R/W	0-9 TRACELENGTH/4 16-23 coincwindow 24-31: PSATH (PSA threshold/4)
7	0x0N7	R/W	0-7 GATEWINDOW (stretch length of GATE pulse) 8-14 GATEDELAY (input delay for GATE)
8	0x0N8		Reserved for ADC programming
9	0x0N9	R/W	0-7 CoincDelay
10	0x0NA	R/W	0-4 QDC0length (length) 5 QDCDIV0 6-7 QDCshift (shift within a group of 4 samples) 8-14 QDC0delay (length+delay) 15 QDC0correct (correct for 4 sample coarseness) 16-20 QDC1length 21 QDCDIV1 24-30 QDC1delay 31 QDC1correct
11-15	0x0NB- 0x0NF	R/W	Reserved

9.2 Event Registers

Address range is 0x1N0 – 0x1NF, and 0x30N, N=1..4. Read only

Use these during the run to read event data

Register	Address	R/W	Description
EVDATAA	0x1N0	R	Hitpattern
EVDATAB	0x1N1	R	Event or local time stamp M, L (local TS in mode 0x402, 0x503 is lower 24 bits * 256)
EVDATAC	0x1N2	R	Event time stamp X, H
PSAA	0x1N3	R	PSA value
PSAB	0x1N4	R	PSA value or gate pulse counter
CFDA	0x1N5	R	CFD values
CFDB	0x1N6	R	CFD values
EVDATAD	0x1N7	R	Lsum
EVDATAE	0x1N8	R	Tsum
EVDATAF	0x1N9	R	Gsum, read advances event buffers and increments NOUT
REJECT	0x1NA	R	Read advances event buffers without incrementing NOUT
BLDATAL	0x1NB	R	Lsum for BL avg
BLDATAT	0x1NC	R	Tsum for BL avg

BLDATAG	0x1ND	R	Gsum for BL avg
	0x1NE	R	reserved
ADC	0x1NF	R	ADCvalue

Register	Address	R/W	Description
WF0	0x300	R	Waveform FIFO channel 0
WF1	0x301	R	Waveform FIFO channel 1
WF2	0x302	R	Waveform FIFO channel 2
WF3	0x303	R	Waveform FIFO channel 3

9.3 Run Statistics Registers

Address range is 0x2[2N]0 – 0x2[2N+1]F, N=1..4. Read only

Used for run statistics and other output values.

Index in runstats parameter list	PL address	Units	Parameter name	Description
0	0x2[2N]0		L OOR	Out of range fraction
1	0x2[2N]1		L ICR	Input count rate
2	0x2[2N]2	ns	L COUNTTIME_L H COUNTTIME _M	Count Time
3	0x2[2N]3	ns	L COUNTTIME_H H COUNTTIME_X	
4	0x2[2N]4		L NTRIG_L H NTRIG_M	Number of triggers (P4e: FastPeaks)
5	0x2[2N]5		L NTRIG_H H NTRIG_X	
6	0x2[2N]6	ns	L FTDT_L H FTDT_M	Fast trigger dead time
7	0x2[2N]7	ns	L FTDT_H H FTDT_X	
8	0x2[2N]8	ns	L SFDT_L H SFDT_M	Slow filter dead time*
9	0x2[2N]9	ns	L SFDT_H H SFDT_X	
10	0x2[2N]A		L GCOUNT_L H GCOUNT_M	Number of gate pulses*
11	0x2[2N]B		L GCOUNT_H H GCOUNT_X	
12	0x2[2N]C		L NOUT_L H NOUT_M	Number of output counts
13	0x2[2N]D		L NOUT_H H NOUT_X	
14	0x2[2N]E	ns	L GDT_L H GDT_M	Gate dead time*
15	0x2[2N]F	ns	L GDT_H H GDT_X	

16	0x2[2N+1]0		L NPPI_L H NPPI_M	Number of counts passing pileup inspection*
17	0x2[2N+1]1		L NPPI_H H NPPI_X	
18-31	0x2[2N+1]2- 0x2[2N+1]F		Reserved	

Properties marked with * are omitted in the PSA version of the firmware

Output values derived from the run statistics parameters are

1. OUTPUT_COUNT_RATE = NOUT / COUNTTIME
2. INPUT_COUNT_RATE = NTRIG / (COUNTTIME-FTDT)
3. PASS_PILEUP_RATE = NPPI / COUNTTIME
4. GATE_RATE = GCOUNT / COUNTTIME

10 Linux Configuration

The Pixie-Net Linux system is based on Xillinux (1.3), which is based on Ubuntu LTS 12.04. The initial distribution can be downloaded from the Xillinux website (http://xillybus.com/xillinux). In what is distributed by XIA, the initial setup steps as described in the Xillinux documentation have been applied, including copying the Linux OS image on an SD card and increasing the "disk space" to 16GB.
(http://xillybus.com/downloads/doc/xillybus_getting_started_zynq.pdf)

The SD card contains mainly the Linux OS (not visible to Windows) and 4 boot files (visible) in a small FAT partition. FPGA configuration updates have to be copied to that partition.

Notes for the particular configuration of Xillinux for Pixie-Net:

- Xillybus demo source files are in /root/xillybus
- By default, Xillinux knows only about "main & restricted" Linux apps. Therefore we modified /etc/apt/sources.list and uncommented the "universe" entry to get more apps
- Installed Lighttpd web server
  The default webpage is in /var/www. Need to modify /etc/lighttpd/lighttpd.conf as follows: add `mod_cgi` under server.modules and

  ```
  $HTTP["url"] =~ "/cgi-bin/" {
          cgi.assign = ( "" => "" )
  }
  cgi.assign      = (
          ".cgi"  => ""
  )
  ```
- sudo apt-get install i2c-tools
- sudo apt-get install g++
- install boost libraries 1.61 in /usr/local and compiled all binaries

- Installed Samba
  Following Ubuntu help:" How to Create a Network Share Via Samba …"[2]

About This Guide

In this text, I teach how to create a network share via Samba using the CLI (Command-line interface/Linux Terminal) in an uncomplicated, simple and brief way targeting Windows users.

Contents

Procedures

All commands must be done as root (precede each command with 'sudo' or use 'sudo su').

1. Install Samba

```
sudo apt-get update
sudo apt-get install samba
```

2. Set a password for your user in Samba

```
sudo smbpasswd -a <user_name>
```

```
Note: Samba uses a separate set of passwords than the standard Linux system accounts
(stored in /etc/samba/smbpasswd), so you'll need to create a Samba password for yourself.
This tutorial implies that you will use your own user and it does not cover situations
involving other users passwords, groups, etc...
```

```
Tip1: Use the password for your own user to facilitate.
```

```
Tip2: Remember that your user must have permission to write and edit the folder you want to
share.
Eg.:
sudo chown <user_name> /var/opt/blah/blahblah
sudo chown :<user_name> /var/opt/blah/blahblah
```

```
Tip3: If you're using another user than your own, it needs to exist in your system
beforehand, you can create it without a shell access using the following command :
sudo useradd USERNAME --shell /bin/false

You can also hide the user on the login screen by adjusting lightdm's configuration, in
/etc/lightdm/users.conf add the newly created user to the line :
hidden-users=
```

3. Create a directory to be shared

```
mkdir /home/<user_name>/<folder_name>
```

4. Make a safe backup copy of the original smb.conf file to your home folder, in case you make an error

```
sudo cp /etc/samba/smb.conf ~
```

5. Edit the file "/etc/samba/smb.conf"

```
sudo nano /etc/samba/smb.conf
```

```
Once "smb.conf" has loaded, add this to the very end of the file:

[<folder_name>]
path = /home/<user_name>/<folder_name>
valid users = <user_name>
read only = no
```

```
Tip: There Should be in the spaces between the lines, and note que also there should be a
single space both before and after each of the equal signs.
```

6. Restart the samba:

```
sudo service smbd restart
```

7. Once Samba has restarted, use this command to check your smb.conf for any syntax errors

```
testparm
```

8. To access your network share

```
    sudo apt-get install smbclient
    # List all shares:
    smbclient -L //<HOST_IP_OR_NAME>/<folder_name> -U <user>
    # connect:
    smbclient //<HOST_IP_OR_NAME>/<folder_name> -U <user>
```

To access your network share use your username (<user_name>) and password through the path "smb://<HOST_IP_OR_NAME>/<folder_name>/" (Linux users) or "\\<HOST_IP_OR_NAME>\<folder_name>\" (Windows users). Note that "<folder_name>" value is passed in "[<folder_name>]", in other words, the share name you entered in "/etc/samba/smb.conf".

```
Note: The default user group of samba is "WORKGROUP".
```

Source

1. http://www.hardcode.nl/archives_147/article_548-samba-quick-setup-on-ubuntu-1004.htm

How to Create a Network Share Via Samba Via CLI (Command-line interface/Linux Terminal) - Uncomplicated, Simple and Brief Way! (last edited 2015-10-06 20:46:34 by r-2buntu-d)

[2] https://help.ubuntu.com/community/How%20to%20Create%20a%20Network%20Share%20Via%20Samba%20Via%20CLI%20%28Command-line%20interface/Linux%20Terminal%29%20-%20Uncomplicated,%20Simple%20and%20Brief%20Way!

11 Theory of Operation

11.1 Digital Filters for γ-ray Detectors

Energy dispersive detectors, which include such solid state detectors as Si(Li), HPGe, HgI_2, 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_f and feedback resistor R_f.

The output of the preamplifier following the absorption of an γ-ray of energy E_x in detector D is shown in Figure 6.1 (b) as a step of amplitude V_x (on a longer time scale, the step will decay exponentially back to the baseline, see section 6.3). When the γ-ray is absorbed in the detector material it releases an electric charge $Q_x = E_x/\varepsilon$, where ε is a material constant. Q_x is integrated onto C_f, to produce the voltage $V_x = Q_x/C_f = E_x/(\varepsilon C_f)$. Measuring the energy E_x of the γ-ray therefore requires a measurement of the voltage step V_x in the presence of the amplifier noise σ, 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.

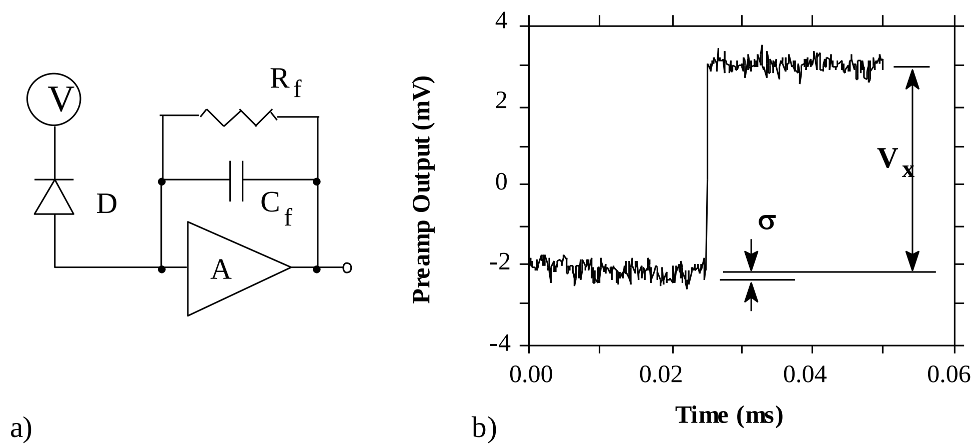


Figure 11-1: (a) Charge sensitive preamplifier with RC feedback; (b) Output on absorption of a □-ray.

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_x and thus to the γ-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_x 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:

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

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.

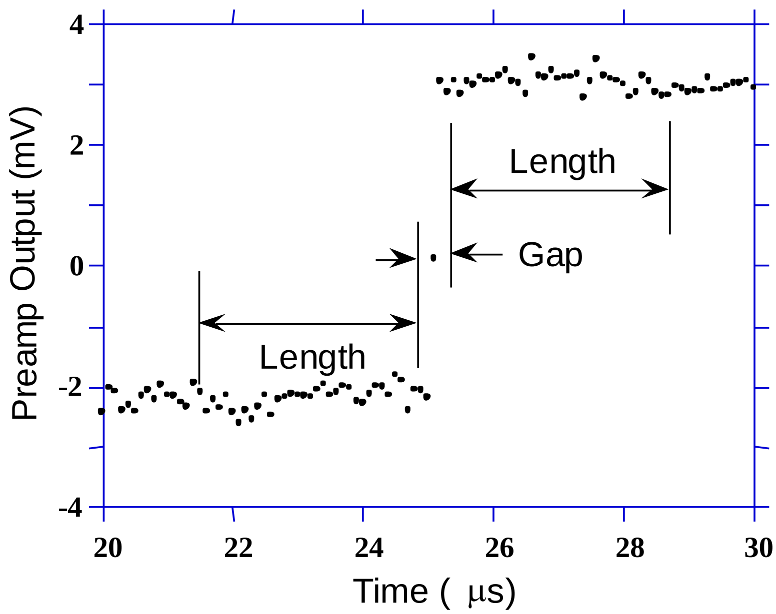


Figure 11-2: Digitized version of the data of Figure 6.1 (b) in the step region.

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 γ-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 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:

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

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=1μs and Gap G=0.4μs to a γ-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.

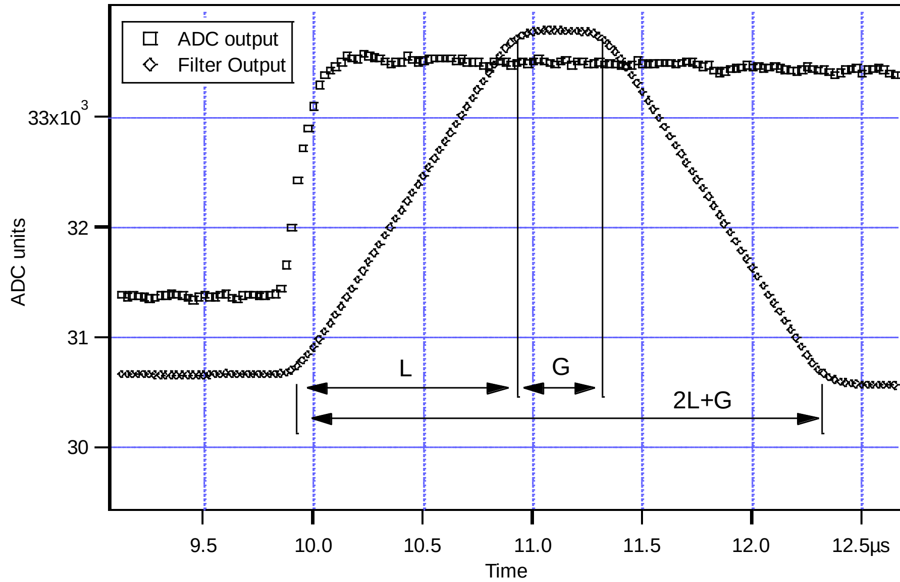


Figure 11-3: Trapezoidal filtering of a preamplifier step with L=1µs and G=0.4µs.

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 γ-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 γ-ray peak amplitude V_x is to be measured. The fluctuations in the baseline have a standard deviation σ_e which is referred to as the *electronic noise* of the system, a number which depends on the rise time of the filter used. Riding on top of this noise, the γ-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 γ-ray is absorbed in the detector. This Fano noise σ_f adds in quadrature with the electronic noise, so that the total noise σ_t in measuring V_x is found from

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

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

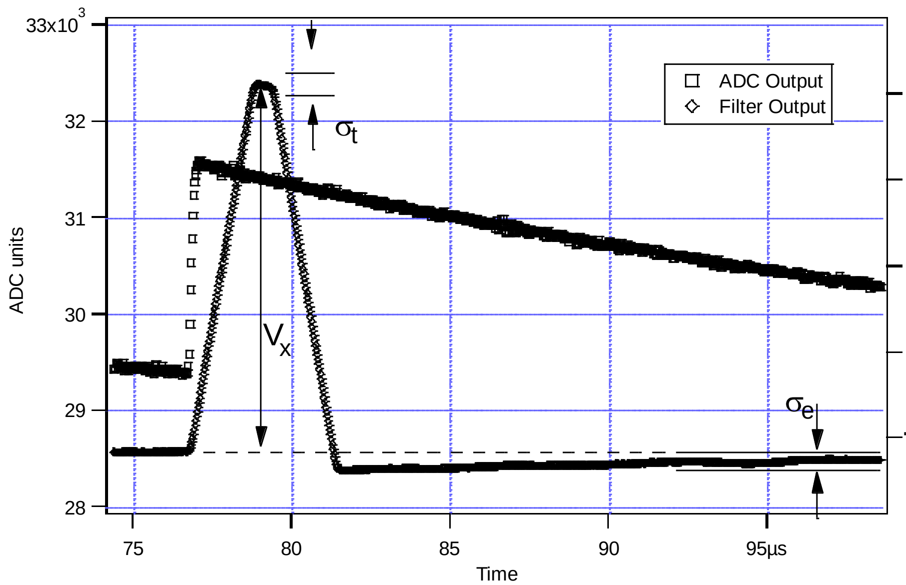


Figure 11-4: A γ-ray event displayed over a longer time period to show baseline noise and the effect of preamplifier decay time.

11.4 Thresholds and Pile-up Inspection

As noted above, we wish to capture a value of V_x for each γ-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, 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 γ-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 = 0.1µs and a gap G_f =0.1µs. The slow filter has L_s = 1.2µs and G_s = 0.35µs.

The arrival of the γ-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).

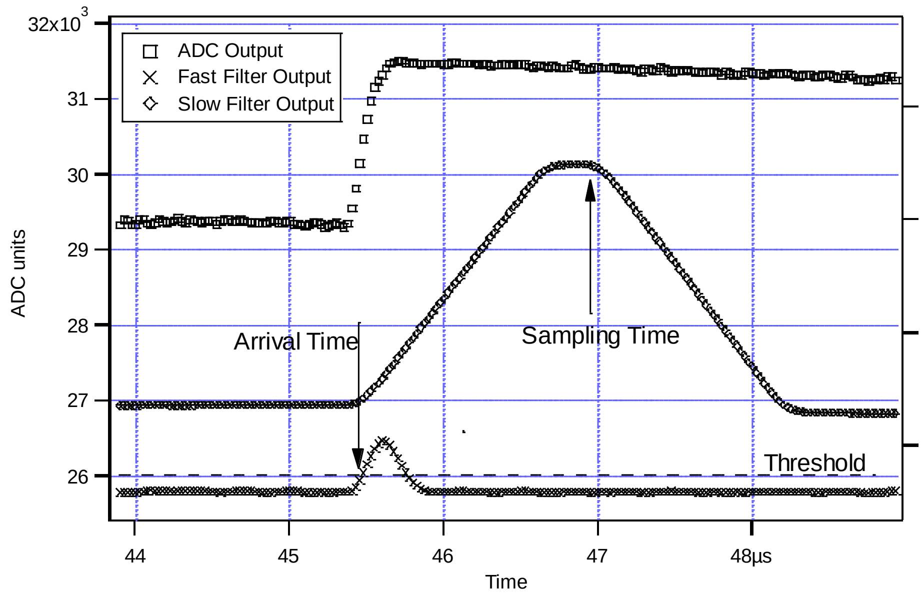


Figure 11-5: Peak detection and sampling in a Pixie module.

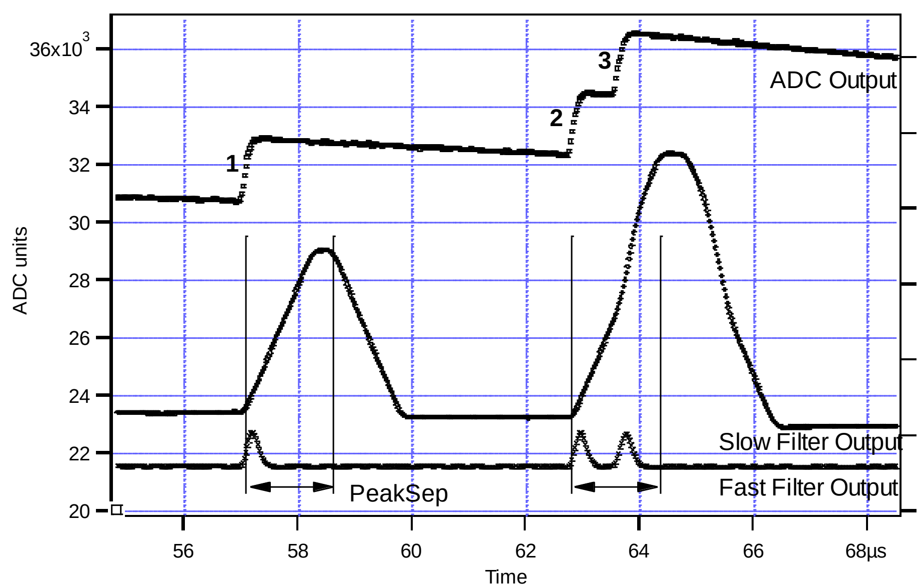


Figure 11-6: A sequence of 3 γ-ray pulses separated by various intervals to show the origin of pileup and demonstrate how it is detected by the Pixie module.

The value V_x captured will only be a valid measure of the associated γ-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 γ-rays arriving separated by various intervals. The fast filter has a filter length L_f = 0.1µs and a gap G_f =0.1µs. The slow filter has L_s = 1.2µs and G_s = 0.35µs.

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 µs, are thus seen to pileup in the present example with a 1.2 µs 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 µs, these γ-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 µs. 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 Capture Process

The data capture in the Pixie-Net is based on the principle that for every detected rising edge, one record is assembled from the continuously running processes for waveform capture and energy filters. As some of the processes are not finished by the time of the rising edge, input data or capture signals are delayed appropriately. For example, incoming ADC data is delayed for the waveform capture by the user specified pre-trigger delay. The signal to capture energy filter sums is sent through a delay line of length (energy filter rise time plus energy filter flat top) to capture the output after filtering.

Consequently, for every rising edge, the following information is latched into front end buffers:

- 56 bit time stamp of latch signal
- 32 bit time stamp of last rising edge in this channel
- Energy filter sums for last rising edge in this channel
- Pileup inspection flags
- Coincidence flags
- Starting address of waveform memory

and the (delayed) waveform data begins to flow into the waveform memory, for the user specified length of trace. The front end buffers hold 500 such records and the waveform memory holds 8Ki samples.

When the front end buffers are not empty, a flag is raised for the ARM processor. On 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, and reads and writes the list mode data to file. If the event is piled up or otherwise rejected, it is cleared from the front end buffer without recording.

Closely following rising edges still capture one record per edge, with the limitation of one record per 1/8 of a decimated clock cycle in filter range 3 and higher. If such events are piled up, the energy will be not a valid measure of the pulse height and waveforms may overlap from pulse to pulse, but some of the information in the record may still be useful for offline re-analysis.

11.7 Dead Time and Run Statistics

11.7.1 Definitions

Dead time in the Pixie-Net 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 T0, the FPGA computes three filter sums using the waveform data from T- (a energy filter rise time before T0) to T1 (a flat top time plus filter rise time after T0), 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 T0 to T1. Consequently, each pulse creates a dead time Td = (T1 – T0) 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[3]. This means the output count rate OCR (valid pulses) is a function of filter dead time Td and input count rate ICR given by

$$OCR = ICR * \exp(-ICR * 2 * Td), \qquad (4)$$

which reaches a maximum $OCR_{max} = ICR_{max}/e$ at $ICR_{max} = 1/(2*Td)$. Simply speaking, the factor 2 for Td comes from the fact that not only is an event E2 invalid when it falls into the dead time of a previous event E1, but E1 is rejected as piled up as well. This filter dead time is accumulated in the SFDT counter in each processing channel.

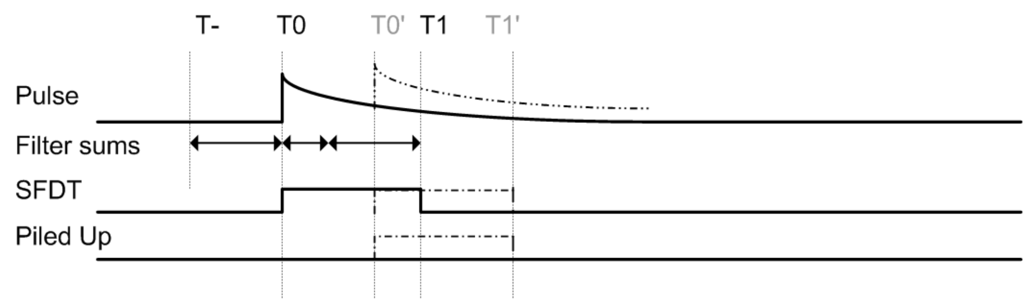


Figure 11-7: Filter dead time. A pulse arriving at T0 will incur slow filter dead time (for energy measurement) until T1. At T1, the pileup status is latched – for a single pulse, it is logic low and the event is accepted. A second pulse arriving at T0' will extend the dead time and cause the pileup 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 trigger can be issued[4]. 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.

---

[3] G. Knoll, Radiation and Measurement, J Wiley & Sons, Inc, 2000, chapters 4 and 17.

[4] 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".

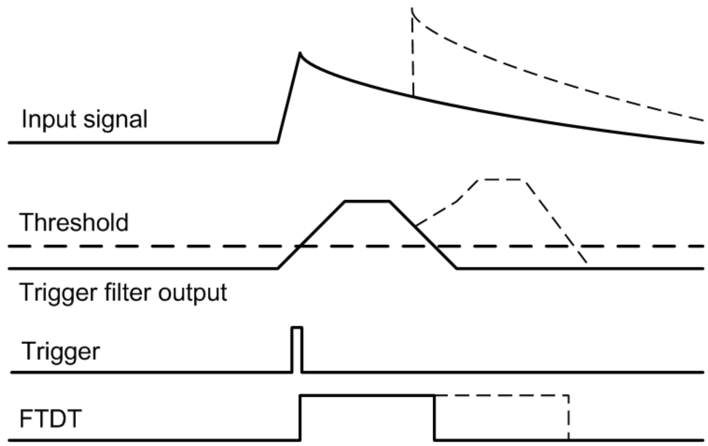


Figure 11-8: Fast Trigger Dead Time (FTDT). A second pulse is not detected if the trigger filter output is still above threshold.

3. Other

In the Pixie-Net, 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 <u>Dead time associated with external conditions</u>

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.

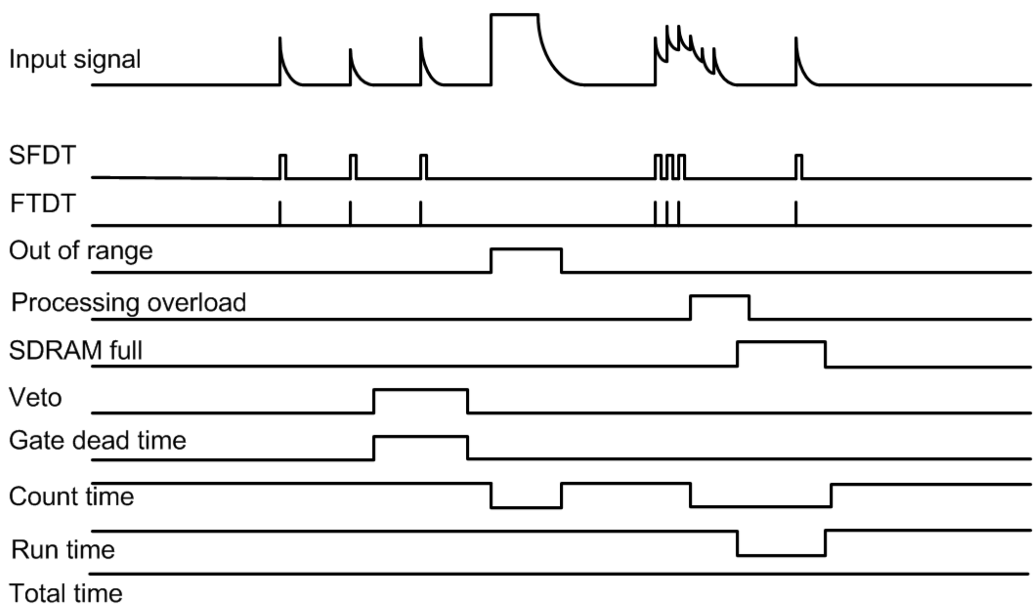


Figure 11-9: The count time counter is stopped when the signal is out of range and when events are rejected because of a processing backlog (e.g. local buffer memory full or file write process busy). SFDT and FTDT are only counted when the count time is on. The gate dead time is counted in a separate 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, 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 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 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 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	(*Veto* OR *Gate**) AND LCE	0	GCE high
1	0	*Gate** 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 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 local 500-event buffers are 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

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 (Td for each output count) and so one can use the classical model of OCR = ICR/(1+Td*ICR), derive[5] OCR/ICR = (real time – dead time)/(real time), and solve for ICR as a function of measured OCR, real time and dead time.

---

[5] OCR(1+Td*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.

In the Pixie-Net, 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.**

In the Pixie-Net firmware design, the counter SFDT attempts to independently account for the system dead time. SFDT counts the time during which the pileup rejection will reject this and any subsequent pulse that are too close in time. Essentially it measures – cumulative for all pulses – the time from the first trigger until a new trigger is again allowed, extended by any trigger during the interval. This is a paralyzable dead time with pileup rejection, closely matching the classical model. SFDT correctly measures the *time* during which pulses are not recorded, but unless simplified to the assumptions in the classical model, it is not trivial to compute from that the *number* of pulses lost. The detailed mathematical treatment is beyond the scope of this writing.

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

11.8 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 firmware includes such PSA fucntions (enabled for thus licensed units).

11.8.1 Overview

The approach used by the Pixie-Net 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.

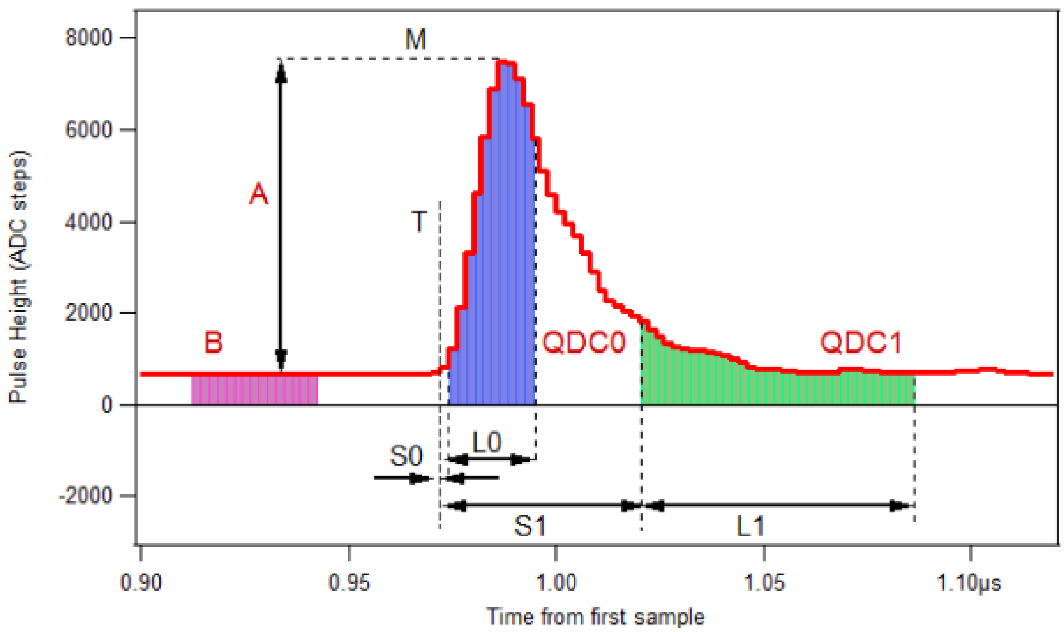


Figure 11-10: The PSA quantities and input parameters

An additional function of the PSA firmware is to compute a constant fraction timing value for the rising edge. This is illustrated in Figure 11-11. Having found the maximum of the pulse, the firmware logic computes the CFD level as 50% of the amplitude, and finds the two closest points to that level, cfdlow and cfdhigh. The CFD time of arrival x is then computed by the DSP through interpolation between cfdhigh and cfdlow:

x /1 = (cfdhigh – cfdlow) / (CFD level – cfdlow)

In the Pixie-Net, time stamps are latched by local or distributed triggers issued at the rising edge of a pulse. To accommodate delayed pulses in coincidence systems, the *CFD time* is captured at the end of the user specified coincidence window (1/2 of the "Window width" specified in the Pixie Viewer). The reported *CFD time* value is the time from the interpolated CFD level crossing to the end of the coincidence window. Therefore one can compute [in units of 2ns sampling intervals] the fractional (subsampling) time of arrival x from cfdlow

x = 1 – frac((*CFD time*)/256)

and the absolute time of the CFD crossing is

CFD crossing = *TrigTime* – (*CFD time*)/256+ ½ Window width

For example, for a user specified coincidence Window width of 400ns, the Pixie module may report an event with a *TrigTime* of 1234 and a *CFD time* of 12832. Then one can compute

x = 1-frac(50.125) = 0.875 (or 1.75ns)

and

CFD crossing = 1234 [x 2ns] – (12832/256) [x 2ns]+ ½ x 400ns

= 2567.75 ns after last reset of the time stamp counter

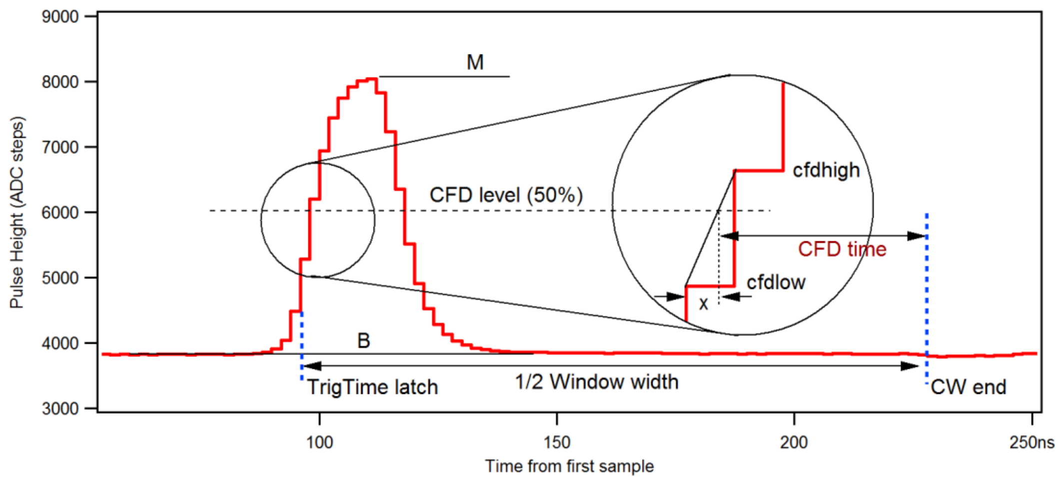


Figure 11-11: CFD timing definitions

Restrictions for the CFD:

- The CFD level is fixed to 50%
- The rising edge, from 50% level to maximum, must be less than 12 samples (24ns)
- If more than one pulse occurs within the coincidence window, results are invalid
- To measure time-of-arrival differences between channels, all channels must be in group trigger mode so data is captured on the same trigger.
- As *CFD time* has a maximum value of 512 ns, the coincidence Window width must be less than 1024ns

11.8.2 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_TESHOLD is used to detect the rising edge of a pulse. The threshold applies to both sums and is also used for the CFD maximum detection.

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

- QDC1 must finish last (i.e. S1+L1 > S0+L0)
- L0, L1 must be between 2 and 120, and a multiple of 2
- L0+S0, L1+S1 must be between 0 and 254
- The difference of S0 and S1 must be a multiple of 2 (e.g S0 = 1, S1 = 17)

Two additional input parameters are used by the ARM C code, which accumulates a 2D histogram of the PSA data. This histogram plots PSA value R vs energy E. both R and E are 16 bit numbers which can range from 0 to 65536. The histogram has 100 bins[6] in each

---

[6] The 2D histogram can be recreated offline from list mode data with arbitrary number of bins. 100 was chosen for online binning to keep processor load and website rendering speed within reason. The number is a compile parameter and can be modified by experienced users if necessary.

direction. Therefore R and E must be divided by a scaling factor to map the value to a bin. These factors are

- MCA2D_SCALEX, i..e bin x = E / MCA2D_SCALEX
- MCA2D_SCALEY, i..e bin y = R / MCA2D_SCALEY

To see the full range, MCA2D_SCALEX/Y should be 655. In practice, a smaller value is often useful to “zoom in” to a particular range.

11.8.3 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. To accommodate fractional samples for the CFD time, the number recorded in the file is 256* CFD time (1 LSB = 3.91ps). Divide the reported value by 256 to match units with the Trigger time (1ns).

For text output in Run Type 0x502, see section 7.1.3

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	CFD time *256
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.

11.8.4 Reduced General Purpose Functionality

Due to resource limitations in the PL, the following output parameters are not computed in the PSA variant of the firmware:

- No gate pulse count in the event record
- No GDT counters in the channel run statistics
- No NPPI or PPR counters in the channel run statistics
- No SFDT counters in the channel run statistics
- No GCOUNT or GATE_RATE counters in the channel run statistics
- No CSFDT and CCT counters in the module run statistics

and the following functions are simplified or removed

- No delay and window for external gates
- No offboard serial I/O
- No coincidence mode acquisition (0x503, 0x402) with PSA and vice versa

These parameters and functions can be reinstated as necessary if other trade-offs are made.