

PIXIE-4 Online Help

Version 2.00, July 2009

XIA LLC

31057 Genstar Road
Hayward, CA 94544 USA

Phone: (510) 401-5760; Fax: (510) 401-5761
<http://www.xia.com>



Disclaimer

Information furnished by XIA is believed to be accurate and reliable. However, XIA assumes no responsibility for its use, or for any infringement of patents, or other rights of third parties, which may result from its use. No license is granted by implication or otherwise under the patent rights of XIA. XIA reserves the right to change the DGF product, its documentation, and the supporting software without prior notice.

- **Getting Help for the Pixie-4**

There are several ways to get help for the Pixie-4. You can use IGOR's built-in help browser to access the Pixie-4 specific help file by selecting Help -> Help Topics from the top menu bar. Choose "Pixie4-Help" in the popup menu on the left, and select the appropriate help topic from the list on the right.

Each Pixie-4 Run Control Panel also has a "Help" button, which directly displays the help topic for that panel. In the help topics, click on [blue underlined links](#) to jump to cross references. [**bold blue bracketed text**] refers to buttons or control fields in the control panels.

- **Getting Started**

Preparations

1. If you have a remote controller, first install the driver software for the controller itself. Unless directed otherwise by the manufacturer of the controller, this can be done with or without the controller and Pixie-4 modules installed in the host computer and/or chassis. If the modules are installed, ignore attempts by Windows to install drivers for the modules until step 7.
NI controllers come with a multi-CD package called "Device Driver Reference CD". For simplicity it is recommended to install the software on these CDs in the default configuration.
2. Unless already installed, power down the host computer, install the controller in both the host computer and chassis, and power up the system again (chassis first).
3. Windows will detect new hardware (the controller) and should find the drivers automatically. Verify in Window's device manager that the controller is properly installed and has no "resource conflicts".
4. Install Igor Pro
5. Install the Pixie-4 software provided by XIA
6. Unless already installed, power down the host computer, install the Pixie-4 modules in the chassis, and power up the system again (chassis first).
7. Windows will detect new hardware (the Pixie-4 modules) and should find the drivers automatically. If not, direct it to the "drivers" directory in the Pixie-4 software distribution installed in step 5. Verify in Window's device manager that the modules are properly installed as "PLX Custom (OEM) PCI 9054 Boards" and have no "resource conflicts". Currently, the driver must be version 5.2. The previously used driver version 4.1 will identify the modules as "Custom (OEM) PCI 9054 Boards" without the "PLX"
8. Find *Pixie4.pxp* in the installed folder and double-click it to open the Pixie-4 Viewer.

- **Initial Startup**

When the Pixie Viewer has been loaded, the *Pixie Start Up Panel* should be prominently displayed in the middle of the desktop. It will prompt you to do the following:

Select PXI chassis type

First select the PXI chassis you are using. Currently supported are standard 4-, 5-, 6- or 8-slot chassis, the National Instruments 14-slot or 18-slot chassis with 3 PCI bus segments, and the 14-slot 6U chassis designed for Pixie-16 modules by XIA. (The National Instruments 14-slot chassis has the same PCI bus configuration as the 18-slot chassis, only fewer slots)

Specify the Pixie-4 modules:

Next select the number of Pixie-4 modules in the system. Then specify the PXI slot number in which each module resides.

Finally, click [**Start Up System**]. (If you want to try the software without a chassis or modules attached, click on [**Offline Analysis**].) In the IGOR history window, a message will show if the modules have been initialized successfully.

You will now see the *Main* control panel that links to all other control panels. Its controls are organized in three groups: Setup, Run Control, and Results. In the Setup group, the [**Start System**] button opens the *Pixie Start Up Panel* in case you need to reboot the modules. The [**Open Panels**] popup menu leads to four panels where parameters and acquisition options are entered. To get started, select *Parameter Setup* in this menu, which will open (or bring to front) the [Parameter Setup](#) panel. For most of the actions the Pixie Viewer interacts with one Pixie-4 module at a time. The number of that module is displayed in the *Main* control panel and at the top right corner of the [Parameter Setup](#) panel (inside the [**Module**] control).

Proceed with the steps below to configure your system.

1. At the bottom of the [Parameter Setup](#) panel, click on the [**Oscilloscope**] button

The [Oscilloscope](#) is a graph that shows the untriggered signal input. Click [**Refresh**] to update the display. The pulses should fall between about 1600 and 15000 on the left axis. If no pulses are visible or if they are cut off above 16384 or below 0, click [**Adjust Offsets**] to automatically set the DC offset. There is a control called [**Offset %**] on the Oscilloscope which can be used to set the target DC offset for each channel (in % of the full range). If the pulse amplitude is too large to fall in the display range, decrease the [**Gain**]. Since the offsets might drift, for example after changes in input count rate, it is useful to leave the display open and check the offsets once in a while.

2. In the [Energy tab](#) of the [Parameter Setup Panel](#), input an estimated preamplifier RC decay time for [**Tau**] then click on [**Auto Find Tau**] to determine the actual Tau value for the current channel of the current module. Repeat this for other channels if necessary. The Tau finder works best for a Tau value from 20 μ s to 200 μ s at moderate count rates.
3. At the bottom of the [Parameter Setup Panel](#), click on [**save**] to save the system parameters found so far. You can save the settings into either an existing settings file or a new file.
4. Click on the [Run Control](#) tab, set [**Run type**] to 0x301 MCA Mode, [**Poll time**] to 1 second, and [**Run time**] to 30 seconds or so, then click [**Start Run**]. After the run is complete, go to the [Main](#) panel and select *MCA spectrum* from the [**Open Panels**] popup menu in the *Results* group. The [MCA Spectrum](#) shows the MCA histograms for all four channels. You can deselect other channels while working on only one channel. After defining a range in the spectrum with the cursors and setting the fit option to [**fit peaks between cursors**], you can apply a Gauss fit to the spectrum by selecting the channels to be fit in the [**Fit**] popup menu. You can also enter the fit limits using the [**Min**] and [**Max**] fields or by specifying a [**range**] around the tallest peak or the peak with the highest energy. To scale the spectrum in keV, enter the appropriate ratio in the field [**keV/bin**].

If you are not getting a nice-looking spectrum, you may need to adjust some settings such as filter rise time and flat top etc. Refer to the User's Manual for details.

- **Main Panel**

The *Main* control panel links to all other control panels. At the top of the panel is a control to select the current [**Module**]. Its other controls are organized in three groups: Setup, Run Control, and Results.

In the *Setup* group, the [**Start System**] button opens the *Pixie Start Up Panel* in case you need to

reboot the modules. The **[Open Panels]** popup menu leads to four panels where parameters and acquisition options are entered:

[Parameter Setup Panel](#)
[Oscilloscope](#)
[ChassisRegisterPanel](#)
[AllFilesPanel](#)

In the *Run Control* group, **[Run type]** popup menu is used to choose the run type, described in more detail under [Run Control](#). The **[Start Run]** button starts a data acquisition, the **[Stop Run]** button stops it. You can also enter the total [Run Time] and the [Number of Spills] to be acquired in list mode runs. For more run control options, see the [Run Control](#) tab of the [Parameter Setup Panel](#).

In the *Results* group, the **[Update]** button refreshes the **[Event rate]** and **[Input Count Rate]** shown below. The **[Open Panels]** popup menu leads to five panels and graphs that show the data acquired in more detail:

[MCA Spectrum](#)
[List Mode Traces](#)
[List Mode Spectrum](#)
[Run Statistics](#)
[File Series](#)

- **Parameter Setup Panel**

The *Parameter Setup* panel is use to set all parameter for one particular module, except for those related to the analog gain and offset, which are located in the [Oscilloscope](#). At the upper right of the panel is a control to select the current **[Module]**. Using the control tabs, you can define parameters in the following areas:

[Trigger](#)
[Energy](#)
[Waveform](#)
[Gate](#)
[Coincidence](#)
[Advanced](#)
[Run Control](#)

It also has several buttons on the bottom of the panel:

[Start System] can be used to re-initialize the modules
[Oscilloscope] opens the [Oscilloscope](#) graph
[Copy] opens the [CopyPanel](#) to copy settings between channels/modules
[Extract] opens the [ExtractPanel](#) to extract settings from file
[Load] loads settings from file
[Save] saves settings to file
[More] will show additional controls on the control panel
[Help] will open this help file, at the entry corresponding to the currently active tab
[About] will show software and hardware version information

- **Trigger**

The *Trigger* tab of the *Parameter Setup* panel sets parameters that control the operation of the trigger filter. If not all described controls are visible, click on the **[More]** button at the bottom of the panel.

Trigger Filter

In the Filter group you can set the [[Rise Time](#)] and [[Flat Top](#)] for the trigger filter of each channel. The units of time are μs . For a detailed description of filter operation please refer to the User's Manual. The trigger filter is always operated at the full ADC sampling rate. Its rise time can be varied between 27ns and 840ns. Its flat top is valid between 0ns and 813ns. The trigger filter will most often use a flat top comparable with the average signal rise time. In applications with very short rise times a flat top of zero will give the best pileup rejection performance.

The trigger [[Threshold](#)] corresponds to $\frac{1}{4}$ of the pulse height in ADC steps, e.g. with a threshold of 20, triggers are issued for pulses above 80 ADC steps as shown in the [Oscilloscope](#). This relation is true if the trigger filter rise time is large compared to the pulse rise time and small compared to the pulse decay time. A pulse shape not meeting these conditions has the effect of raising the effective threshold. For very slow rising pulses, increase the trigger filter [[Flat Top](#)] as much as possible. The threshold value is scaled with the trigger filter rise time, therefore it is not limited to integer numbers. The threshold should be set just above the noise level of the signal. For a modeled behavior of the trigger, you can open the [Filter Display](#) from the [List Mode Traces](#) graph or the [ADCFilterDisplay](#) from the [Oscilloscope](#) that show trigger filter and threshold computed from acquired waveforms using the current settings.

Trigger Options

Trigger options include checkboxes to control the following:

[[Enable trigger](#)]

You can switch on or off any channel's ability to contribute to the trigger.

[[Respond to group triggers only](#)]

This checkbox controls if a channel uses its own trigger or instead the distributed "group trigger" for the acquisition of waveforms, timestamps, and optionally energies. When not checked, only the triggers generated in this particular channel are used. When checked, the distributed triggers from other channels or modules are used (also may include this channel, if enabled).

Triggers are always distributed within a module: Any channel with "trigger enabled" will issue a trigger for distribution, and any channel set to "group trigger" will respond to the distributed trigger, including those issued by itself. The controls in the [ChassisRegisterPanel](#) determine how trigger signals are distributed between modules.

[[Good channel](#)]

Only channels flagged as good will be read out. This setting has no bearing on the channel's capability to issue a trigger. There can be a triggering channel whose data are discarded. Channels not marked as good will not be included in the automatic offset adjustment or the computation of the minimum required coincidence window.

[[Read always](#)]

Check this box if you want a good channel to be read out even if it did not report a hit. If operated in group trigger mode you will always get a valid waveform. This way you can collect waveforms not biased by trigger requirements. If the signal on the channel crossed the trigger threshold (even if not trigger enabled) you also get a valid energy and timestamp. A checkbox in the [Energy](#) tab allows to get an estimate of the energy even when the signal does not cross the trigger threshold.

[[Local Timestamp](#)]

Set this checkbox to record individual timestamps for all channels in group trigger mode, rather than the identical time stamps based on the (last) group trigger.

Normally, in acquisitions with shared group triggers, all channels record the (identical) timestamp of the (last) group trigger for this event. Since waveforms are captured based on the group trigger, this ensures that within a data record, the traces and timestamps are correlated. However, if no waveforms

are recorded, there is no time-of-arrival information of possible delays between channels. Setting the checkbox preserves the time difference information. See the user manual for details.

Note that since only trigger enabled channels latch time stamps, if the box is checked, the recorded timestamp is only valid for trigger enabled channels where the signal crossed the threshold.

- **Energy**

The *Energy* tab of the *Parameter Setup* panel sets parameters that control the operation of the energy filter and the pulse height reconstruction in the DSP. If not all described controls are visible, click on the [\[More\]](#) button at the bottom of the panel.

Energy Filter

In the Filter group you can set the [\[Rise Time\]](#) and [\[Flat Top\]](#) for the energy filter of each channel. The units of time are μs . For a detailed description of filter operation please refer to the User's Manual.

There are several filter [\[Filter Ranges\]](#) for the energy filter; with a granularity of $0.027\mu\text{s}$ for filter range 1 (1-bit decimation), $0.053\mu\text{s}$ for filter range 2 (2-bit decimation), and so on.

Employing a trapezoidal filter avoids the kind of ballistic deficit that occurs when a finite rise time signal is used in conjunction with a Gaussian shaper. The energy filter flat top time should thus be a little larger than the longest rise time expected. The output of the energy filter is sampled one decimated clock cycle before the end of the flat top, plus the signal arrival may jitter by up to one decimated clock cycle with respect to the decimated clock. You should therefore make the flat top two notches longer than the signal rise time.

The sum of energy filter rise time and flat top cannot exceed 127 decimated clock periods. If you type in a rise time or flat top value that violates this limit, the Pixie Viewer will adjust it accordingly.

Energy Computation

To compute the energy from the filter sums, the DSP needs to know the decay time [\[Tau\]](#) of the preamplifier. To set and measure the decay time, enter an estimated value then click on the [\[Auto Find\]](#) button. You can also enter a known good value directly in the control. The RC calibration needs to be performed only once for a given preamplifier. The result is then stored in the parameter database, and can be saved in the settings file by clicking on the [\[Save\]](#) button. A manual fit to find the decay time is possible in the [Oscilloscope](#).

The [\[Integrator\]](#) variable controls the event energy reconstruction:

- 0: Normal code.
- 1: Use energy filter gap sum only. This is equivalent to simply integrating over the pulse. It is useful for scintillator applications where event energies can be derived by setting the energy filter flat top long enough to cover the whole scintillation pulses. Tau is ignored in this mode.
- 2: Ignore energy filter gap sum when reconstructing event energy. This is useful for step pulses whose amplitude is the difference between the high and low steps.
- 3-5: Same as 1, but the energy is multiplied by a factor 2, 4, or 8, respectively. This is useful for very fast pulses that do not have a lot of area under the peak and thus will show up only at the very low end of the spectrum.

Energy Options

Energy options include checkboxes to control the following:

[Estimate energy if not hit]

If a channel is read out even if it did not report a hit (see [\[Read always\]](#) in the [Trigger](#) tab), its energy

is normally reported as zero since there was no valid local trigger to capture the value of the energy filter. However, as the energy filter is computed continuously, checking this box will capture the filter value based on the (last) group trigger distributed within the module or over the backplane. This might be useful for channels with occasional very small pulses (below the threshold), or possibly to capture energy estimates on piled up pulses.

Note that since the timing of the group trigger is not precise with respect to the non-triggering pulse, the energy reported is only a rough estimate. It might help to set the flat top time to a large value to make the capturing of the energy filter less time sensitive.

[Allow energy < 0]

Normally, energies computed as negative are set to zero. Typically such pulses are close to the noise and the pre-trigger sum is larger than the post-trigger sum. In some applications pulses may be bipolar and it may be useful to preserve the negative value. Checking this box will prevent the processing from setting the energy to zero.

Energy Action Buttons

Clicking [Optimize Tau] opens a panel that allows automatic scanning of all Tau values within the scanning limit specified by the user, examining the energy resolution at each Tau, and picking the optimal Tau value which gives the best energy resolution. Results of this optimization, including the Tau value and energy resolution, are stored in an output file whose name is specified by the user in the beginning of the run. Another file which has the same name as the output file but with a different extension (*.tmp) is used to store step-by-step intermediate results.

Before starting the optimization, all conditions listed on the *Auto Optimization of Decay Time* panel should be fulfilled first. This is to ensure that valid MCA spectra will be produced during the optimization and Gauss Fit of energy peaks on the MCA spectra will generate meaningful results. The four channels of a Pixie-4 module can be used to optimize four different [Decay Time Limits] at the same time when the same detector signal is split and input into the four channels. This would speed up the scanning of a large range of Tau values. You could change the [Decay Time Limits] of each channel by changing the [Channel] control on the left upper corner of the Pixie-4 Run Control panel. At any time, the Pixie-4 module used to carry out this optimization is the current module, i.e. the module indicated by the [Module] control on the left upper corner of the Pixie-4 Run Control panel. So if you want to use a different module, change the [Module] number before you set the [Decay Time Limits] for each channel.

Click on [Optimize Filter] to open the *Auto Optimization of Energy Filter panel*. An optimization of energy filter settings is carried out by scanning all possible combinations of energy filter rise and flat top within their respective limits specified by the user, examining the energy resolution at each combination, and picking the optimal combination which gives the best energy resolution. Results of this optimization, including energy filter rise time, flat top and energy resolution, are stored in an output file whose name is specified by the user in the beginning of the run. Another file which has the same name as the output file but with a different extension (*.tmp) is used to store step-by-step intermediate results.

Again, before starting the optimization, all conditions listed should be fulfilled first. This is to ensure that valid MCA spectra will be produced during the optimization and Gauss Fit of energy peaks on the MCA spectra will generate meaningful results. The four channels of a Pixie-4 module can be used to optimize four different Auto Scanning Limits at the same time when the same detector signal is split and input into the four channels. This would speed up the scanning of a large range of energy filter rise time and flat top. You could change the Auto Scanning Limits of each channel by changing the [Channel] control on the left upper corner of the *Pixie4 Run Control* panel. At any time, the Pixie-4 module used to carry out this optimization is the current module, i.e. the module indicated by the [Module] control on the left upper corner of the *Pixie-4 Run Control* panel. So if you want to use a different module, change the [Module] number before you set the Auto Scanning Limits for each

channel.

Click on [[Scan Settings](#)] to open the [File Series Scan](#) panel. This panel allows scanning of filter parameters and Tau in a series of files.

- **Waveform**

The *Waveform* tab of the *Parameter Setup* panel sets parameters that control the operation of the waveform capture and pulse shape analysis (PSA). If not all described controls are visible, click on the [[More](#)] button at the bottom of the panel.

Trace Capture

The [[Trace Length](#)] and [[Trace Delay](#)] values entered in this group of controls, both in units of μs , govern the waveform acquisition. [[Trace Length](#)] is the total length of the acquired waveform, which can be set independently of the parameters in the [Energy and Trigger Filter](#). You can use the delay parameter to move the trigger point within the trace: [[Trace Delay](#)] measures the trigger time with respect to the beginning of the recorded trace. The trace lengths are up to $13.6\mu\text{s}$ for each channel and can be specified in 13.3ns increments.

If event dead time is a concern, the trace length should be set as short as possible. In particular, if only energies and time stamps are of interest, the trace length can be set to zero.

Pulse Shape Analysis

[[PSA Start](#)] and [[PSA End](#)] specify the trace range for pulse shape analysis (PSA). Currently the Pixie-4 supports two types of PSA: XIA_PSA and USER_PSA. XIA_PSA reports the signal arrival time by measuring the time when the trace reaches a preset percentage level ("constant fraction") of its magnitude. The percentage threshold can be set defined in [[CFD Threshold](#)]. To reduce processing time, the XIA_PSA is only performed when the checkbox [[Compute CFD time](#)] is checked. The computed value is the arrival time after the start of the acquired waveform in units of $1/256$ th of an ADC sampling interval. [[PSA Start](#)] and [[PSA End](#)] should be set to include the rising edge of the trace.

- **Gate**

The *Gate* tab of the *Parameter Setup* panel sets parameters that control the gating of the data acquisition with external signals. If not all described controls are visible, click on the [[More](#)] button at the bottom of the panel. See also the user manual for a detailed description of the Gate and Veto operation.

GFLT (Veto)

We define Veto as a signal distributed to all modules and channels, but each channel may be individually enabled to require or ignore this signal. The signal is distributed over the backplane; the front panel MMCX connector "DSP-OUT" on the Pixie-4 can be used to input an external signal to the backplane if the module is enabled to do so in the [ChassisRegisterPanel](#).

Veto is active during the validation of a pulse (after pileup inspection), an energy filter rise time plus flat top after the rising edge. With suitable external logic, the decision to veto a pulse can be made from information obtained at the rising edge of the pulse (e.g. multiplicity from several channels) and therefore this function is also called Global First Level Trigger (GFLT).

If the [[required](#)] checkbox is set, this channel requires the GFLT (Veto) signal to be present to record event data. Present here means the Veto voltage has to be 0V on the backplane line or the Pixie-4 front panel input. If the [[inverted](#)] checkbox is set, the polarity of the GFLT (Veto) signal is inverted, i.e. the channel requires the Veto backplane line to be absent 3.3V to record events.

Gate

We define GATE as a dedicated signal for each individual channel. It is active near the rising edge of the pulse, e.g. to suppress a detector pulse with a coincident pulse from a BGO shield.

To accommodate delays between the detector signal and the GATE, a rising edge on the GATE input starts a counter of length [[Gate Window](#)]. A time [[Gate Delay](#)] after a trigger from the Pixie-4 trigger filter (local or distributed) the status of the counter is latched. If still counting, the GATEBIT is high. If the counter finished already, the GATEBIT is low.

The GATEBIT is recorded in the list mode data (upper 4 bits of the hit pattern word). If the [[required](#)] checkbox is set, this channel requires the GATEBIT to be low to record event data. If the [[inverted](#)] checkbox is set, this channel requires the GATEBIT to be high to record event data. If the [[invert edge](#)] checkbox is set, the counter is started on the falling edge of the GATE input.

The Pixie-4 has no front panel input for GATE signals, but they can be passed to the module through the backplane by a neighboring PDM module to the left of the Pixie-4. Alternatively, if the [[uses GFLT input](#)] box is checked, the GFLT (Veto) input is used as the source for the GATE signal.

- **Coincidence**

The *Coincidence* tab of the *Parameter Setup* panel sets parameters that control coincidence acquisition within a module. For coincidence between modules, refer to the [ChassisRegisterPanel](#). If not all described controls are visible, click on the [[More](#)] button at the bottom of the panel.

Allowed Hit Pattern

In this section, you can select the Coincidence Pattern that defines which channels have to contribute to an event to make it acceptable. For example, you might be only interested in events in which only a single channel was hit, or in any event in which more than two channels contributed. Below is a description for functions within a module.

For each event, the Pixie-4 first reads which channels contributed before reading any further data. This Hit Pattern is a 4 bit number, e.g. (0001) if only channel 0 contributed or (1111) if all channels contributed. By setting the checkboxes you can require that an event must match one of the selected hit patterns to be accepted for further processing. Any or all hit patterns can be selected; if none is selected no events will be accepted, if all are selected any event will be selected.

The hit pattern (0000) can be useful when sharing triggers between modules: Traces etc will be recorded in the module for any channel marked as [[Read Always](#)] in the [Trigger](#) tab even if there is no hit in the module itself.

Setting the checkboxes will automatically modify the 16bit coincidence pattern shown above the checkboxes where it can also be edited directly (type "0x####" to enter a hex number ####).

An example shall illustrate this feature. Assume a single module connected to 4 detectors which observe a Na-22 source, emitting back to back 511keV gamma-rays from positron annihilation. Channels 0 and 1 are connected to one pair of back to back detectors and channels 2 and 3 are connected to a second pair of back to back detectors. You are interested only in gammas from positron annihilation. Thus a coincidence in channel 0 and 1 or a coincidence in channel 2 and 3 is required. If all 4 channels were in coincidence, that would be fine too. So, the acceptable hit patterns would be (0011), (1100) and (1111), where the right most digit indicates channel 0 and the left most is for channel 3. To achieve the desired behavior, you have to select the three acceptable hit patterns by checking the appropriate boxes, and deselect all other hit patterns by not checking their boxes.

Note the difference between "hit" and "trigger": A "hit" occurs if a pulse goes over threshold and passes pileup inspection. "Hits" can not be disabled. A "trigger" means there is a "hit" plus a request for event processing is sent to the module's DSP unit. "Triggers" can be disabled on a channel by

channel basis in the [Trigger](#) tab. Channels will be recorded if they have a hit or their [\[Read always\]](#) checkbox is set, but only "hit" channels will have valid energy and timing information.

Coincidence Window

Each channel with a pulse above threshold, whether trigger enabled or not, contributes to the hit pattern the moment the pulse is validated as not piled up (i.e. an energy filter time after rising edge of pulse). However, if the channel is not trigger enabled, its contribution to the hit pattern is cleared after a fixed delay (~400ns). This is to ensure hits don't linger for channels that are not read out.

The hit pattern is read for comparison with the coincidence pattern about 66ns after validation of the first pulse that is trigger enabled. If several channels contribute to an event, the coincidence window -- the time period in which (delayed) channels can contribute to the hit pattern -- is thus ~66ns. This window can be increased by entering a number in the [\[Window width\]](#) control field to accommodate larger delays between channels, as may be required by the physics of the experiment up to a value of about 0.86ms.

Note that a difference in energy filter times between channels will cause even channels with simultaneous pulses to contribute to the hit pattern at different times, since the pile up inspection takes different amounts of time. The software thus calculates the required additional window width to compensate for any such difference and displays it in the [\[Required by energy filters\]](#) field. The popup menu below controls how this required minimum is applied:

[\[Increase for E filter, never decrease\]](#)

at each change of [\[Window width\]](#) or the energy filter times, ensures that the [\[Window width\]](#) is at least the required minimum. If a larger number is entered in [\[Window width\]](#) or if filter times are changed which would lower the required minimum, keep the **larger** value.

Use this mode to modify the coincidence window as required by the experiment, but remember to verify its value after changes in the energy filter times. In particular, when energy filter times are increased one channel after the other, the increase in the first channel will also increase the [\[Window width\]](#) but the increase in the last channel (making filter times equal again) will **not decrease** the [\[Window width\]](#).

[\[Keep at required E filter width\]](#)

At all times keep the [\[Window width\]](#) at the required minimum. If a larger or smaller number is entered, it is ignored and changed back to the required minimum. Use this mode if there are no coincidence requirements from the experiment and differences in the energy filter should be accommodated, but unnecessary large values should not linger as in the first mode.

[\[Disregard E filter requirement\]](#)

In this mode, the energy filter requirement is ignored and any value can be entered for the [\[Window width\]](#). The entered value is not modified by the software. Use this mode if channels are independent so there is no need to wait for delays between channels, or if a specified [\[Window width\]](#) is deemed sufficient and should not be modified by the software.

Notes:

- 1) Any added coincidence window width will increase the time required to process an event and thus reduce the maximum count rate.
- 2) Pulses occurring *during the readout of data* (after the end of the coincidence window) are lost. The readout may take several microseconds, longer if waveforms are to be recorded.
- 3) The cut off at the end of the coincidence window is precise to within 13.3ns

Clover Addback

To support 4-fold clover detectors, there is a further option to sum energies from individual channels in events with more than one hit and bin them into an addback spectrum. This function is enabled with the checkbox [[Sum channel energies for addback MCA](#)]. The Pixie-4 module will thus generate 4 channel MCA and one addback MCA at the same time.

If clover addback is selected, there is the further option to omit those events with more than one active channel from the individual channel

This clover addback is supported for the accumulation of spectra in MCA mode and in list mode, but the list mode data does not contain the sum energy (only the channel energies).

- **Advanced**

The *Advanced* tab of the *Parameter Setup* panel sets advanced parameters. No controls are visible unless the [[More](#)] button at the bottom of the panel is clicked.

- **Histogram Control**

This section shows the parameters controlling the operation of the MCA increment in Pixie-4 memory. Energy values are calculated to 16-bit fixed-point numbers. This would correspond to a 64k spectrum. To map the full energy range into the available 32k spectrum, one has to combine bins. At minimum, 2 bins have to be combined into one, so the [[Binning Factor](#)] has to be set to 1 (combining 2^1 bins). Higher binning factors can be useful for low count rate or low resolution applications.

If you want to see a certain range of the spectrum at higher resolution you can enter a [[Minimum Energy](#)]. This will discard all energies below the minimum and start binning the spectrum from the minimum energy (Bin 0 = Emin).

To disable the incrementing of MCA histograms, uncheck the [[Enable](#)] box.

- **Pileup Inspection**

In this group, the pileup inspection can be modified. Normally, events are not recorded if a pulse is followed by a second pulse in less than the sum of energy filter rise time and energy filter flat top.

If the [[Disable](#)] box is checked, the pileup inspection is disabled, i.e. all pulses are allowed. The energy is only computed once for any number of pulses following each other closer than the above limit; energy filter sums are latched with respect to the first pulse. The result of the energy computation will contain contributions from both pulses and thus be not equal to the energy in either pulse (and likely not equal to their sum energy)

If the [[Invert](#)] box is checked, the pileup inspection is inverted, i.e. only piled up pulses are recorded. The energy is computed as described above.

If the [[Pause](#)] box is checked, triggers are ignored for ~400ns after a first pulse. This may be useful in systems where the detector outputs a signal with ringing or overshoots, which would cause the Pixie-4 to trigger twice on the same pulse and thus reject it as piled up. (This effect is usually more pronounced in larger pulses, and thus causes a loss of counts at the high end of the spectrum). The computed energy is likely to be correct, since the ringing is not a second pulse with random energy. However, this option will make the rejection of real pileup less effective.

If the [[Ignore out of range](#)] box is checked, events are accepted even if the signal goes out of range, meaning the detector output is beyond the range of the ADC input (after gain adjustment).

- **Run Control**

The *Run Control* tab of the *Parameter Setup* panel sets parameters that control the data acquisition. If

not all described controls are visible, click on the [\[More\]](#) button at the bottom of the panel.

Run Control Group

This section sets run type, run time, and polling time

[Run Type]

This popup menu is used to set the run type to one of the following modes:

List Mode

List mode is the general data acquisition run. Waveforms, energies and time stamps are collected on an event-by-event basis. The data is stored in various formats (see section 3.6 of the user manual for details):

0x100	full event data (9 words), plus waveforms
0x101	full event data (9 words), no waveforms
0x102	compressed event data (4 words), no waveforms
0x103	compressed event data (2 words), no waveforms

Fast List Mode

Fast list mode is no longer supported

MCA Mode

MCA mode puts all modules into a typical spectrum-only acquisition mode in which there are no list-mode data required. The event data is not stored in the output buffer, but only used to calculate the energy for incrementing the spectrum. Runs end after the time specified in [\[Run Time\]](#) counts down to zero.

[Poll Time]

The polling time indicates the time interval at which the Pixie-4 Viewer checks if the run in the selected modules has ended. If so, runs are stopped in all modules, if they have not stopped already, and the data are read out.

[Run Time]

This variable is used to indicate the total run time for MCA runs or the overall timeout limit for list mode runs. During a run, it counts down the remaining time. List mode runs end when the [\[Number of Spills\]](#) is reached or the [\[Run Time\]](#) is down to zero, whichever comes first.

Output File

You can choose a [\[Base name\]](#) and a [\[Run number\]](#) in order to form an output file name. The run data will be written to files whose name is composed of both (i.e. base####.xxx). The run number is automatically incremented at the end of each run if you select [\[Auto increment run number\]](#) on the [Data Record Options](#) panel, but you can change it manually as well. Data are stored in files in either the MCA folder if the run is a MCA run or the PulseShape folder if the run is a List Mode run. These files have the different extension as described below.

".bin"

For list mode runs, buffer data are stored in a file with name extension ".bin". This is a binary file consisting of 16bit unsigned integers.

".dat"

For list mode runs, a summary of event data is stored in a file with name extension ".dat". This is an ASCII file. For long list mode runs with many spills, this file can become much larger than the binary file. If you do not need the ASCII data, you can disable the ".dat" file by unchecking the [\[Auto process list mode data ...\]](#) in the [Data Record Options](#) panel.

".mca"

For both list mode runs and MCA runs, MCA spectrum data are stored in a file with name extension ".mca" if you select [[Auto store spectrum ...](#)] on the [Data Record Options](#) panel. This is a binary file consisting of 32bit unsigned integers. It includes MCA data for all modules, ordered from channel 0 of module 0 to channel 3 of the last module

".set"

Module settings are stored in a file with name extension ".set" after each run if you select [[Auto store settings ...](#)] on the [Data Record Options](#) panel. Run statistics can also be extracted from this file. This is a binary file consisting of 16bit unsigned integers. It is equivalent to the settings files saved from the [Settings](#) tab.

".ifm"

For both list mode runs and MCA runs, run statistics information are stored in a file with the extension ".ifm" if you select [[Auto store statistics ...](#)] in the [Data Record Options](#) panel.

List Mode Spill Settings

[Number of Spills]

In list mode runs, since available memory limits the amount of data that each module can store, the data has to be read out from time to time. Each such readout is called a spill. For a longer run in list-mode, you can request several spills. For example, if you request a run with 10 spills, you will get 10 times the memory data. At start of the first run all previous run history is cleared, e.g. MCA memory and run and live time information. The next nine sub-runs are started with a Resume Run command (without interaction with the user), which leaves previous run information intact. Run times and live times and spectra in MCA memory are thus incremented from all 10 spills, and the list mode data from all 10 spills is appended to the same file.

Even if set to zero, there will be at least one spill. List mode runs end when the [[Number of Spills](#)] is reached or the [[Run Time](#)] is down to zero, whichever comes first.

[Soil Timeout]

This variable sets the timeout for each individual spill. During a run, it counts down the remaining time. If it reaches zero, the current spill is ended by the host, and a new spill starts.

[Events / Buffer]

List mode data is first accumulated in a buffer in internal DSP memory. This variable indicates the preset number of list mode events the Pixie-4 module will store in its buffer for each run. The number is calculated automatically if you change the run type.

The automatically computed value ($\text{Events/Buffer} = \text{MAXEVENTS}$) is the maximum "safe" number of events. That is, given the maximum length of an event (all "good" channels contributing), in any case at least MAXEVENTS events will fit in the output buffer. MAXEVENTS can be decreased by the user if desired. MAXEVENTS can be set to zero, to disable the halting at a preset number. This makes the acquisition more efficient: if MAXEVENTS takes into account 4 "good" channels per event, but only few multi-channel events occur in the acquisition, the buffer will only be filled up to about 1/4 when MAXEVENTS is reached. Setting Events/Buffer to zero will always fill the buffer as much as possible. It is ignored in MCA runs.

[1, 32, 16/16 Buffers per Spill]

List mode data is first accumulated in a buffer in internal DSP memory. There are two options to use the larger external memory on the Pixie-4 to store several buffers and read them out in a fast block read. These options reduce readout dead time (because the block read is faster).

If the radio button [[1 buffer per spill](#)] is selected, the external memory is not used at all and the module stores only one buffer in internal memory. The total number of events will be [[Events / Buffer](#)] times the [[Number of Spills](#)]. The readout deadtime will be higher in this mode.

If the radio button [**32 buffers per spill**] is selected, the module stores 32 data buffers in external memory, which then are read out in a fast block read. Each readout counts as one "spill". The total number of events will be 32 times the [**Events / Buffer**] times the [**Number of Spills**].

If the radio button [**16/16 buffers**] is checked, the module stores 16 data buffers in the first half of the external memory, which then are available for read out in a fast block read while another 16 buffers are stored in the second half, and so on. Each readout of 16 buffers counts as one "spill". The total number of events will be 16 times the [**Events / Buffer**] times the [**Number of Spills**]. The modules continue to take data until the number of spills is reached and the host stops the acquisition.

External memory readout is not supported for Rev. B modules.

Synchronization

The first check box asks if all runs should [**Simultaneously start and stop**] in all modules. In almost all multi-module systems this will be the case and the box should be checked.

Synchronization signals are distributed over a PXI backplane line. If a Pixie-4 module is present in the crate, but not part of the current data acquisition setup, it might inhibit the synchronization setup. For setups with more than 7 modules in chassis with PCI bridges, make sure the [**Trigger share mode**] in the [ChassisRegisterPanel](#) is set to 2 for all module except the left most module, which should be set to 3. There must be no gaps between the modules in this case.

If you also want all timers in all modules to be reset to zero with the start of the next data acquisition run, click the box [**Synchronize clocks**]. For this feature to be useful all Pixie-4 modules should be operating from the same master clock as described in the user's manual.

Normally, once clocks are synchronized, they will stay synchronized until modules are power cycled. Therefore the checkbox is cleared at the end of the run, preserving time correlation between subsequent runs. In some cases it may be beneficial to resynchronize the clocks in every new run (i.e. reset the clock to zero for every new data file). To do so, check the [**every new run**] checkbox.

Start Run

After setting all parameters, you can start a run to take data. During the run, the [**Run Time**] control shows the remaining time. If you select multiple spills for list mode runs, the number of spills will also count down during the run.

For list mode runs, when the first module reaching the preset maximum number of events stops its run, it will also stop the runs in all other modules if Module Synchronization is enabled. Then the data buffer of each Pixie-4 module will be read out and saved into a file. If more than one spill is requested, the run will resume in all modules.

When the [**Run Time**] control counts down to 0, the Pixie-4 Viewer will issue a run stop command to stop the run in all modules, followed by readout of data to file.

Stop Run

If you want to stop a run before it finishes by itself, you can click on this button to manually stop it. This will end runs in all modules and read out and save the data.

Data Record Options

This panel gives you several options for automating tasks after or during a run. Most are checked by default to ensure all data are saved for each data run.

[[Auto increment run number](#)] will increase the run number in the file name of the data files to avoid overwriting of files.

In addition, if this option is checked, Igor will test at the beginning of a run if the output file already exists (testing the .bin file for list mode runs and the .mca file for MCA runs). If the file exists, Igor will increment the run number and try again; after 20 tries it will add "_new" to the base file name and continue. This feature is intended to avoid overwriting previous files when Igor only remembers a run number from an earlier run, for example after a PC crash.

[[Auto store spectrum ...](#)] will store the spectrum data automatically after a run as a binary .mca file.

[[Auto store settings ...](#)] will store the run parameters automatically after a run in binary format (same as settings file). Run statistics can also be extracted from this file.

[[Auto process list mode data ...](#)] will extract the energies and timestamps from the binary data and save them in a ".dat" file in ASCII format. This may take a very long time for large data sets.

[[Auto store statistics ...](#)] will store the run statistics and run start/stop date and time automatically after a run in ASCII format in an .ifm file.

In addition, you can choose to automatically [[Update MCA every N seconds](#)] during MCA runs or every N spills during List mode runs. "N" can be selected in the variable control field. Run statistics will be updated at the same time. You can also manually update the MCA spectrum by clicking the [[Update](#)] button in the [MCA Spectrum](#) display.

Further, you can choose to automatically store data in [[New Files every N spills](#)] during List mode runs or every N seconds during MCA mode runs. "N" can be selected in the variable control field. Run statistics will be updated at the same time. This feature can be used to break up output data into several smaller files; since it will take some time to save the data, N should be set to a rather large value, i.e. at least several hundred spills or several thousand seconds.

Note that a *new* run will be started after saving, i.e. all run statistics will be reset, MCA spectra will be cleared, and clocks resynchronized if the [[in every run](#)] checkbox is set in the [Synchronization](#) section in the [Run](#) Tab. Thus enabling this feature is equivalent to manually starting several individual runs with N spills.

[[Do not parse list mode file ...](#)] will disable the automatic parsing of the list mode file after the run to build a list of pointers to individual events. This is required for Igor to display events in the [List Mode Traces](#) graph, but may take a long time for large files. If disabled, you have to manually load the file in the [List Mode Traces](#) graph before viewing traces and/or energies.

[[Delay runstart ...](#)] will cause Igor to wait in a loop until the current Windows time matches the date and time entered in the corresponding field. [[Force run stop ...](#)] will force the run to end when the current Windows time matches the date and time entered in the corresponding field. Both functions are based on a text string comparison and thus require the date and time to be entered in exactly the right format. To assist finding this format, clicking on the [[Date/Time](#)] button will print the current date and time in the Igor history window.

To abort a run start delay, click on the "Abort" button in the bottom left corner of the Igor window, then click [[Stop Run](#)].

- **Oscilloscope**

= [Calibrate](#) -> Oscilloscope

The Oscilloscope shows 8192 untriggered ADC samples from the input for each channel. The time between samples can be set using the [[dT](#)] variable. The display is updated through its [[Refresh](#)] button. Any DC offset of the preamplifier signal has to be compensated for in order to bring the DC-

coupled input into the ADC range. The exact DC value has no bearing on the acquired spectrum and its origin, which is always at zero. The DC-adjustment is used only to ensure that the signals to be measured fall comfortably into the ADC range. When clicking the [[Adjust Offsets](#)] button, the Pixie-4 Viewer will set the DC offset to a percentage of the full ADC range specified in the [[Offset %](#)] control. Gain and offset can be changed manually with the [[Gain V/V](#)] and [[Offset V](#)] controls.

If pulses begin with a falling edge, toggle the [[invert](#)] checkbox. The core of the trigger/filter FPGA can only trigger on a rising edge. If the checkbox is set, the FPGA will invert the signal before processing it for triggers. You may have to [[Adjust Offsets](#)] again to set the offset properly after inversion.

The offset calibration must be performed with the preamplifiers connected to the Pixie-4 inputs and with both the preamplifier power and detector HV switched on. One should also repeat the offset calibration each time measurement conditions change in any major way, e.g., when the count rate changes greatly. All such changes may influence the DC offset value of the preamplifier signal.

A Tab control to the right contains additional controls:

[[Buttons](#)]

This tab contains the [[FFT Display](#)] button, which opens the [FFTDisplay](#) used to analyze the noise spectrum of the acquired trace; the [[Filters](#)] button which opens the [ADCFilterDisplay](#) used to view the effect of the digital filters applied to the ADC traces; and the [[Caputre](#)] button which repeats the action of the [[Refresh](#)] button until a pulse is detected (useful for low count rates).

[[Tau](#)]

This tab contains controls to [Manually Fit Decay Time](#)

[[Rates](#)]

This tab displays the current input count rate [[ICR](#)] and the current fraction of time the signal is out of range [[Out of Range](#)]. These values are updated in the DSP every ~2-3ms if a run is in progress or not. Their precision is in the order of 5-10%, or 50 cps.

FFTdisplay

You can analyze the noise spectrum in the trace captured in the Oscilloscope, by observing the Fourier transform of the signal. For best results, remove any source from the detector and only regard traces without actual events. The chart shows a plot of amplitude vs. frequency. The plot is calibrated such that a sine wave with 100 ADC units amplitude (200 units peak-to-peak) will show up with an amplitude of 100. To convert a noise floor measurement into ADC units/sqrt(Hz) use the variable FFTbin displayed at the top of the chart, which tells the width of each frequency bin in the Fourier spectrum. The conversion from amplitudes to rms ADC units/sqrt(Hz) is accomplished by multiplying with $1/\sqrt{2 \cdot \text{FFTbin}}$. Now, observe that an ADC unit corresponds to $61\mu\text{V}$. Using the known gain of the Pixie-4 you can convert the noise into an input noise voltage density measured in $\text{V}/\sqrt{\text{Hz}}$. Or, given a particular energy calibration, the noise density can be expressed as $\text{eV}/\sqrt{\text{Hz}}$.

If you click on the [[Apply Filter](#)] button, you can see the effect of the energy filter simulated on the noise spectrum.

ADCFilterDisplay

This graph shows the ADC trace for the selected channel and the response of the (slow) energy filter and the (fast) trigger filter together with an estimate of the trigger threshold. The display is updated through its [[Refresh](#)] button.

Notes:

1. For best representation of the filters, the filter lengths should be an integer multiple of the ADC

sampling interval.

2. Only a simple trapezoidal difference filter is shown for the energy filter. In the actual pulse height calculation, corrections are applied that take into account the decay of previous pulses, the contribution during the flat top time, and long term baseline effects.

Manual Fit Decay Time

Manual Tau Fitting is done on a channel-by-channel basis. First set the cursors on the trace to fit. If possible, some of the baseline after the pulse should be included in the fit region. Use "Ctrl-I" to show the cursors at the bottom of the graph if they are not already visible. Clicking into the [\[Fit Trace\]](#) popup menu lets you select the channel to fit. The result is shown in the field [\[Tau\]](#) below. If acceptable, use the [\[Accept\]](#) popup menu to assign the fit result to a channel.

Clicking [\[Remove Tau Fit ...\]](#) will remove the fit function and residuals from the graph.

- **CopyPanel**

This panel can be used to copy parameter settings from one module to another. The source module and channel are selected at the top of the panel. The parameters to be copied are organized into list box in the left-hand column. The right-hand column shows the destination channels and modules for the copy operation. The Items to copy shown on the Copy Panel and the actual variables to be copied are listed below.

Items	Actual variables to be copied
[Gain]	Gain [V/V]
[Offset]	Offset [V] and base percent
[Filter]	Energy Filter Rise Time and Flat Top, Baseline Cut
[Trigger]	Trigger Filter Rise Time and Flat Top, Trigger Threshold
[FIFO]	Trace Length, Delay, dT [μ s], PSA Start, PSA End, CFD Threshold
[ChanCSR]	Channel CSRA, Channel CSRB, Channel CSRC
[Coinc.]	Coincidence Pattern, Coincidence Window
[MCA]	Cut-Off Energy, Binning Factor
[TAU]	Tau [μ s]
[Integrator]	Integrator
[ModCSR]	All module coincidence settings and backplane options except front panel GFLT

After selecting source, destination and parameters, click on the [\[Copy\]](#) button to execute the copy operation.

- **ExtractPanel**

This panel can be used to extract parameter settings from a file to selected modules and channels. The source file is specified at the top of the panel. Click on the [\[Find\]](#) button to locate the source file. Parameters to be extracted and destination modules or channels are selected in the same manner as in the [CopyPanel](#). Click the [\[Extract\]](#) button to execute the operation.

- **AllFilesPanel**

This panel gives you access to the underlying files of the Pixie-4 software. Usually, these files are already loaded in the memory of the Pixie-4 Viewer. You only have to change these files when you receive updates from XIA.

The directory locations are specified as complete (not relative) search paths: the DSP Path for the DSP code; and the FPGA Path for the trigger/filter FPGA configuration. Use a colon (:) as the separator between drive name, directory, and subdirectories. Do not use backslashes (\). For example use "D:\XIA\data" rather than "D:\XIA\data".

Full path and file name should be limited to 80 characters.

- **MCA Spectrum**

The MCA Spectrum displays the histograms accumulated in the Pixie-4 modules. The channels to be displayed can be selected with checkboxes within the parameter/result table. Gray fields indicate input parameters, purple fields indicate results. The [Sum] checkbox adds or removes the clover adback spectrum.

The [Fit] menu allows you to make Gaussian fits to peaks in the histograms. The fit range can be set by the cursors, the [Min] and [Max] fields in the table, or by specifying a percentage range around the tallest or highest energy peak. The [Fit] menu starts a fitting routine for one or all channels. The routine takes into account a constant background term, though its result is not displayed. Results that are displayed include the peak position, the number of counts in the peak, and the relative and absolute full width at half max (FWHM). For best results include some of the background in the fit.

To calibrate the energy scale, you can enter a scaling factor [keV/bin] in the table as well. This simply changes the label on the x axis.

The [Sum] menu gives you three options of summing the spectrum. You can either sum the range defined by the [Min] and [Max] fields without any corrections, sum the range and subtract the background, or sum the entire MCA. Summing is performed on all four channels at the same time. The calculated Sum is displayed in the [Peak Area] field, and the bin with the largest number of counts within the sum range is displayed in the [Peak] field.

The [Files] popup menu allows to store individual spectra and read back stored spectra from disk. There are three operations:

[Save MCA to Igor text file] will save the current MCA (4 channels) as a scaled wave with some added comments in a text file. This can be useful to import data into other applications

[Read MCA to Igor text file] will read back a Igor text file saved as above.

[Extract MCA from binary file] will read MCA data for the current module from a binary file, such as the

.mca file automatically saved at the end of the run. The file is assumed to be in 32bit unsigned integer format.

You can [Update] the MCA during a run at any time. The [MCA source] field displays where the MCA was last read from, e.g. from the module's memory or a specific file.

The [Zoom] buttons above the graph can be used as shortcuts to zooming operations with the mouse. The [Reset Scale] button resets the scaling of the MCA to 1/bin.

- **List Mode Traces**

After a list mode run has finished, the acquired waveforms and event data can be displayed on an event-by-event basis in the List Mode Traces panel. The most recently acquired data file will be searched for the event requested in the [Trace number] field. The display will show the traces from

the selected module, and the associated time stamps, energies and PSA values for those channels that reported a hit in this event. Also shown are the hit pattern and event timestamp. Traces and energies are scaled as 16-bit numbers. Note that the ADC traces shown in the [Oscilloscope](#) are raw 14-bit numbers, i.e. ADC traces have values divided by 4 compared to list mode traces.

Other displays show the [\[Event times\]](#) and the [\[Hitpattern\]](#). The last 4 bits of the hit pattern mark the channels recorded (bits 0-3 for channel 0-3) and bits 8-11 mark the channels reporting a hit (bits 8-11 for channel 0-3). Bit 5 shows the result of the local coincidence test, bit 6 the level of the "Status" line, and bit 7 the result of the global coincidence test. Bits 4, 12-15 are used for the channel GATEBIT.

In order to display traces from an earlier experimental run one needs to change the Data File name by entering it directly in the [\[Data File\]](#) control or clicking the [\[Find\]](#) button.

To see the response of the energy and trigger filters for the current event, click on [\[Digital Filter\]](#), which opens the [Filter Display](#). This window is mainly intended for diagnostic purposes. For information how to use the Pixie-4 for more detailed pulse shape analysis, please contact XIA.

Filter Display

This graph is used to see the simulated response of the energy and trigger filters in an event. You can browse the leading edge trigger filter response and the energy filter response of individual events. The latter requires a trace length of at least twice the peaking time plus the gap time to be displayed. The trace is shown in red. The trigger filter is shown in blue, and the energy filter is shown in green.

Note that only a simple difference filter is displayed for the energy filter, without the corrections for baseline and decay of the pulse used in the energy calculations.

- **List Mode Spectrum**

Pulse height spectra can be reconstructed from list mode data stored on the disk. The file shown in the [\[Data File\]](#) field will be processed and the resulting histograms will be displayed for the selected Pixie-4 module. Use [\[Read\]](#) after changing the data file to process the new data, and [\[Histo\]](#) to update the displayed spectrum. The full spectrum length is equal to 64k channels. Use [\[No. of bins\]](#) and [\[Delta E\]](#) settings to compress the spectrum such that it fits the display. Hint: use 8000 and 4 to see the full range of data, and then adjust these numbers to zoom into the range of interest. The number of bins and the delta E variables are kept in memory for each channel individually. Be sure to select the channel of interest prior to changing these variables. Use the mouse to zoom in on peaks of interest.

The [\[Fit\]](#) menu allows you to make Gaussian fits to peaks in the histograms. The fit range can be set channel by channel in the [\[Min\]](#) and [\[Max\]](#) fields, or by placing cursors on the spectrum with the mouse. The [\[Fit\]](#) menu starts a fitting routine for one or all channels. The routine does take a constant background term into account, though its value is not displayed. The fit results that are displayed include the peak position, the number of counts in the peak, and its relative and absolute full width at half maximum (FWHM), calculated from the Gaussian fit. For best results be sure to extend the fit range to cover some of the constant background.

The [\[Zoom\]](#) buttons above the graph can be used as shortcuts to zooming operations with the mouse. The [\[Reset Scale\]](#) button resets the scaling of the MCA to 1/bin.

- **Run Statistics**

The run statistics for all modules and channels are displayed in the *All Run Statistics* panel. It also shows the time and date of run start and stop and the source where the statistics were last read from.

The information can be refreshed during a run by clicking the [[Update](#)] button, it can be saved to or read from an .ifm file with the [[Files](#)] menu.

The "DAQ Fraction" is an estimate of the fraction of the lab time a module is taking data. As described above, "Run Time" and "Live Time" measure only the time a module was actively taking data, but not the time to setup a run or the time a module is waiting for readout by the host. Thus "Run Time" is always smaller than the elapsed "Lab time" (i.e. the difference between date/time of run start and run stop). "DAQ Fraction" is defined as "Run Time" / (Current date/time - run start date/time) *100 and thus gives an estimate of the time lost for readout etc. (After a run is finished, the run stop time replaces the current time)

Note that "Run Time" is read from the module, while the date/time is obtained from the host computer to a precision of 1s. "DAQ Fraction" is therefore not a precise number.

In list mode runs, the time to read out one spill is about 0.030s per module. The DAQ Fraction will drop significantly if the time to fill the memory approaches this value. To maximize the DAQ Fraction, a) run in 32 buffer/spill mode, b) make sure the polling time is significantly smaller than the fill time, c) define only those channels actually used as "good", and d) minimize the amount of waveform captured or run in a compressed list mode to store more events in the same amount of memory.

For example, with 4 channels active in mode 0x103, the memory will hold $744 \times 32 = 23,808$ events, so at a count rate of 23kcps, the memory will fill in about 1s and be read out in less than 0.1s, which results in a DAQ Fraction of more than 91%.

The [[History](#)] button opens a plot in which the event rate for each module is plotted. "Current" event rates are calculated every time the [[Update](#)] button is clicked or an automatic update is performed if the [[Track Event rates ...](#)] box is checked. In the calculation, the "current" event rate is determined as (counts since last update) / (time since last update).

- **File Series Scan**

The File Series Scan panel is used to set up a series of acquisition runs in which the settings are modified. One file is saved for each setting, creating a file series that can be processed in the [File Series](#) panel.

A control field [[Filter Range](#)] is repeated from the [Energy](#) tab. In three groups of controls, you can set the start, end, and step size for varying the energy filter rise time, the energy filter flat top, and the decay time Tau. If the step size is zero, that parameter will not be varied.

Two buttons assist in setting up the initial conditions: [[Set Parameters to Start](#)] sets the current values of the energy filter and Tau to the start value defined in the *File Series Scan* panel. If you omit to click this button, the file series will begin with the current value; this is useful to resume a file series. [[Set Scan Run Conditions](#)] will set the checkboxes in the [Data Record Options](#) panel to the values required for the scan, and set the run time to the total time required (interval N in the [Data Record Options](#) panel times the number of settings).

At the bottom of the panel, the button [[Start Scan](#)] starts the file series. This is a different button from the standard Start Run button, because it is starting a run which is modifying parameters. All the updates during a run work the same as in a standard run, though; and the run can be stopped with the standard Stop Run button. When the run is complete, click on the [[File Series](#)] button to open the [File Series](#) panel for analysis

- **File Series**

The *File Series Results* panel is used to process a series of data files with consecutive run numbers. There are control fields to enter the [[BaseName](#)] as in the [Run Control](#) tab of the *Parameter Setup* panel and the [[Start](#)] and [[End](#)] end run number. These numbers are inclusive, i.e. Start = 1 and End =

10 processes files 1..10.

Clicking [[Parse Files](#)] will read spectra from each file and perform a Gauss fit with the settings in the [MCA Spectrum](#). The results of the fit are plotted in the graph section. The individual spectra are also accumulated into a total spectrum (for each channel), which at the end of processing replace the spectra shown in the [MCA Spectrum](#).

In addition, the energy filter settings and the value for the decay time are read for each file from the associated .ifm file and also plotted in the graph. This can be used to find the optimum settings together with the function to scan filter settings described under [File Series Scan](#).

- **ChassisRegisterPanel**

The Chassis Setup Panel controls several system-wide parameters.

Backplane Options

Trigger share mode

As described for the [Trigger](#) tab of the *Parameter Setup* panel, if group trigger operation is requested, each channel will respond to a distributed trigger rather than its own local trigger. Triggers are always distributed within a module: Any channel that is enabled to trigger will contribute to the distributed trigger, and any channel in group trigger mode will (only) respond to the distributed trigger, including those issued by itself.

In addition, triggers can be distributed over the PXI backplane between modules. The field [[Trigger share mode](#)] controls this distribution: There are currently 4 modes:

In mode = 0, the module is disconnected from the backplane. Group triggers are only distributed between the 4 channels of the module.

In mode = 1, the module is connected to the backplane's wired-OR lines. This means that any module in mode = 1 issues triggers on common backplane lines, and every module in mode = 1 responds to these triggers.

Modes 2 and 3 are used in large chassis with more than 8 slots, where backplane lines do not support a wire-OR across PCI segment boundaries. A module in mode = 2 receives triggers from the right neighbor, builds an OR with its own triggers, and sends them to the left neighbor. The leftmost module thus receives a "daisy-chain OR" of triggers from all modules and should be set to mode = 3 to put these distributed triggers on shared backplane lines (PXI_TRIG0, PXI_TRIG1). All modules in mode 2 or 3 respond to triggers on these lines (i.e. at the same time). Note that the backplane has to be configured to drive PXI_TRIG0 and PXI_TRIG1 from the leftmost segment to the other segments. In the XIA 18-slot chassis, this drive direction is hardwired. For National Instrument chassis, use the NI PXI explorer to configure the chassis backplane.

Notes:

Since other backplane lines are used for other Pixie-4 functions (e.g. PXI_TRIG3 for run synchronization), it is generally a good idea to configure the backplane to drive all PXI_TRIGx lines from the leftmost segment to the other segments. Also, for proper functioning of the run synchronization use mode 2 and 3 in systems with more than 7 Pixie-4 modules even if not sharing triggers.

Rev. B modules are always connected to the backplane (equiv. mode = 1).

Front Panel Input Settings

One option to connect an external GFLT (Veto) signal to the Pixie-4 system is to use the MMCX front panel connector on a Pixie-4 module. The GFLT signal is distributed via the PXI backplane to all modules, and - if enabled in the [Gate](#) tab of the *Parameter Setup* panel - a channel only records

events where the GFLT signal is present. If the [[Use front panel for GFLT input](#)] checkbox is set, the current module will internally connect the front panel input labeled "DSP-OUT" to drive the GFLT backplane line. Only one device may issue the signal to the backplane. If the checkbox is set for one module, the function will automatically be disabled for all other modules. However, it is the user's responsibility to ensure that no other device in the PXI chassis issues signals on this line. The Fron panel input signal must be a LVTTTL signal, i.e. 0 = 0V, 1 = 3.3V.

A second option for the MMCX front panel input is to configure it to contribute to the STATUS line on the backplane. This is a wire-OR line connected to all modules in a backplane segment. If the MMCX input is high, the module will pull down the wire-OR line (this constitutes a logic 1 for this line). The state of the STATUS line can be recorded during event processing (see below)

PDM control

For making coincidence decisions between modules, it is recommended to use XIA's logic and preamp power module, the PXI-PDM. It is meant to reside in slot 2. The PXI-PDM has no PCI interface, and thus is programmed by a Pixie-4 module in the neighboring slot. To enable a module to program the PXI-PDM, the checkbox [[Module programs PDM to immediate left](#)] has to be set. The Pixie-4 module then sends the [[PDM control pattern](#)] to the PDM.

The PXI-PDM receives the Pixie-4's hit pattern through the PXI Star Trigger lines. A module can be set to not send out its hit pattern by unchecking the box [[Send local hit pattern to PDM in slot 2](#)]. The PDM assumes a default (0000).

Note that due to the architecture of the PXI Star Trigger, a module in slot 2 must not send signals on this backplane pin. Therefore for a module in slot 2 this checkbox is automatically cleared. (The module should not be in slot 2 in the first place because the PDM has to be there to receive the hit patterns)

The coincidence decision in the PDM is determined from the full 48bit hit pattern of the modules in slots 3-14 and the [[PDM control pattern](#)] as listed below. This decision list is implemented in the PDM's firmware with no claim to cover all cases. Please contact XIA to request additional cases or to obtain verilog source code to write custom PDM firmware.

Events are accepted if MC ==1. Logic operators are AND = &, XOR = ^, OR = |, NOT = !. Operators in front of an array hit[a:b] or {a,b,c,d} are unitary reductions to a single bit (e.g. &{a,c,b} = a AND b AND c). Variables are defined as follows:

```
multi_all      = multiplicity of all channels (0 ... 48 in any particular event)
multi_req1     = multiplicity requirement 1.
                 its value is 1 for PDMcontrol[0:3] = 1,4,7; 2 for 2,5,8; 3 for 3,6,9; 0 otherwise
multi_req2     = multiplicity requirement 2.
                 its value is equal to PDMcontrol[0:3]
MxCy          = Hit bit of module x, channel y. Module 0 is in slot 3.
payy          = pair yy, counting from ch. 0 and 1 of Module 0. Channels are ANDed
poyy          = pair yy, counting from ch. 0 and 1 of Module 0. Channels are ORed
pxyy          = pair yy, counting from ch. 0 and 1 of Module 0. Channels are XORed
hitNs         = an array of all channel N hit bits e.g. hit0s = (M0C0,M1C0,M2C0,...)
Msum_all      = number of modules active in event
Msum_g1       = number of modules in group (module 0 ... module 5) active in event
```

case (PDM control pattern)

```
// test and debug, overall multiplicity
16'h0000: MC <= 0; // always low
16'h0001: MC <= 1; // always high
```

// overall multiplicity

```
16'h001X: MC <= (multi_all >= multi_req2); // multi_req2 defined above, X = 1...F
16'h002X: MC <= (multi_all == multi_req2); // X = 1...F
```

```

// single module tests -- can actually be done in P4 with "local contributes to global" option
16'h0100: MC <= (MOC0 & !MOC1 & !MOC2 & !MOC3); // 1 specific channel hit, no others
16'h0101: MC <= (MOC0 & MOC1 & !MOC2 & !MOC3); // 2 specific channels hit, no others
16'h0102: MC <= (MOC0 & MOC1 & MOC2 & !MOC3); // 3 specific channels hit, no others
16'h0103: MC <= (MOC0 & MOC1 & MOC2 & MOC3); // 4 specific channels hit
16'h0104: MC <= (MOC0
                    ); // 1 specific channel hit, others don't matter
16'h0105: MC <= (MOC0 & MOC1
                    ); // 2 specific channels hit, others don't matter
16'h0106: MC <= (MOC0 & MOC1 & MOC2
                    ); // 3 specific channels hit, others don't matter

16'h0111: MC <= ( (multi00 >= multi_req1) ); // 1 or more
16'h0112: MC <= ( (multi00 >= multi_req1) ); // 2 or more
16'h0113: MC <= ( (multi00 >= multi_req1) ); // 3 or more
16'h0114: MC <= ( (multi00 == multi_req1) ); // 1
16'h0115: MC <= ( (multi00 == multi_req1) ); // 2
16'h0116: MC <= ( (multi00 == multi_req1) ); // 3

// 2 module tests
16'h0200: MC <= ( (MOC0 & !MOC1 & !MOC2 & !MOC3) | (M1C0 & !M1C1 & !M1C2 & !M1C3) );
// 1 specific channel hit in either module, no others
16'h0201: MC <= ( (MOC0 & MOC1 & !MOC2 & !MOC3) | (M1C0 & M1C1 & !M1C2 & !M1C3) );
// 2 specific channels hit in either module, no others
16'h0202: MC <= ( (MOC0 & MOC1 & MOC2 & !MOC3) | (M1C0 & M1C1 & M1C2 & !M1C3) );
// 3 specific channels hit in either module, no others
16'h0203: MC <= ( (MOC0 & MOC1 & MOC2 & MOC3) | (M1C0 & M1C1 & M1C2 & M1C3) );
// 4 specific channels hit in either module, no others

16'h0204: MC <= ( (MOC0
                    ) | (M1C0
                    ) );
// 1 specific channel hit in either module, others don't matter
16'h0205: MC <= ( (MOC0 & MOC1
                    ) | (M1C0 & M1C1
                    ) );
// 2 specific channels hit in either module, others don't matter
16'h0206: MC <= ( (MOC0 & MOC1 & MOC2
                    ) | (M1C0 & M1C1 & M1C2
                    ) );
// 3 specific channels hit in either module, others don't matter
16'h0207: MC <= ( (MOC0 & !MOC1 & !MOC2 & !MOC3) ^ (M1C0 & !M1C1 & !M1C2 & !M1C3) );
// 1 specific channel hit in either, not both modules, no others
16'h0208: MC <= ( (MOC0 & MOC1 & !MOC2 & !MOC3) ^ (M1C0 & M1C1 & !M1C2 & !M1C3) );
// 2 specific channels hit in either, not both modules, no others
16'h0209: MC <= ( (MOC0 & MOC1 & MOC2 & !MOC3) ^ (M1C0 & M1C1 & M1C2 & !M1C3) );
// 3 specific channels hit in either, not both modules, no others
16'h020A: MC <= ( (MOC0 & MOC1 & MOC2 & MOC3) ^ (M1C0 & M1C1 & M1C2 & M1C3) );
// 4 specific channels hit in either, not both modules, no others

16'h020B: MC <= ( (MOC0
                    ) ^ (M1C0
                    ) );
// 1 specific channel hit in either, not both modules, others don't matter
16'h020C: MC <= ( (MOC0 & MOC1
                    ) ^ (M1C0 & M1C1
                    ) );
// 2 specific channels hit in either, not both modules, others don't matter
16'h020D: MC <= ( (MOC0 & MOC1 & MOC2
                    ) ^ (M1C0 & M1C1 & M1C2
                    ) );
// 3 specific channels hit in either, not both modules, others don't matter

16'h0211: MC <= ( (multi00 >= multi_req1) | (multi01 >= multi_req1) ); // 1 or more in either module
16'h0212: MC <= ( (multi00 >= multi_req1) | (multi01 >= multi_req1) ); // 2 or more in either module
16'h0213: MC <= ( (multi00 >= multi_req1) | (multi01 >= multi_req1) ); // 3 or more in either module
16'h0214: MC <= ( (multi00 == multi_req1) | (multi01 == multi_req1) ); // 1 each in either module
16'h0215: MC <= ( (multi00 == multi_req1) | (multi01 == multi_req1) ); // 2 each in either module

```

```

16'h0216: MC <= ( multi00 == multi_req1 ) | ( multi01 == multi_req1 ); // 3 each in either module

16'h0221: MC <= ( ( multi00 >= multi_req1 ) ^ ( multi01 >= multi_req1 ) ); // 1 or more in either, not both modules
16'h0222: MC <= ( ( multi00 >= multi_req1 ) ^ ( multi01 >= multi_req1 ) ); // 2 or more in either, not both modules
16'h0223: MC <= ( ( multi00 >= multi_req1 ) ^ ( multi01 >= multi_req1 ) ); // 3 or more in either, not both modules
16'h0224: MC <= ( ( multi00 == multi_req1 ) ^ ( multi01 == multi_req1 ) ); // 1 in either, not both modules
16'h0225: MC <= ( ( multi00 == multi_req1 ) ^ ( multi01 == multi_req1 ) ); // 2 in either, not both modules
16'h0226: MC <= ( ( multi00 == multi_req1 ) ^ ( multi01 == multi_req1 ) ); // 3 in either, not both modules

16'h0230: MC <= ( ( pa00 | pa01 ) | ( pa02 | pa03 ) ); // one or more pairs out of 4 pairs ; a pair is two channels hit
16'h0231: MC <= ( ( pa00 | pa01 ) ^ ( pa02 | pa03 ) ); // one or more pair in one module, not both
16'h0232: MC <= ( ( pa00 ^ pa01 ) | ( pa02 ^ pa03 ) ); // exactly one pair in one or both modules
16'h0233: MC <= ( ( pa00 ^ pa01 ) ^ ( pa02 ^ pa03 ) ); // exactly one pair in one, not both module

16'h0234: MC <= ( ( po00 | po01 ) | ( po02 | po03 ) ); // one or more pairs out of 4 pairs ; a pair = at least one of two channels
hit
16'h0235: MC <= ( ( po00 | po01 ) ^ ( po02 | po03 ) ); // one or more pair in one module, not both
16'h0236: MC <= ( ( po00 ^ po01 ) | ( po02 ^ po03 ) ); // exactly one pair in one or both modules
16'h0237: MC <= ( ( po00 ^ po01 ) ^ ( po02 ^ po03 ) ); // exactly one pair in one, not both module

// 3 module tests
16'h0300: MC <= ( ( M0C0 & !M0C1 & !M0C2 & !M0C3 ) | ( M1C0 & !M1C1 & !M1C2 & !M1C3 ) |
                ( M2C0 & !M2C1 & !M2C2 & !M2C3 ) ); // 1 specific channel hit in either module, no others
16'h0301: MC <= ( ( M0C0 & M0C1 & !M0C2 & !M0C3 ) | ( M1C0 & M1C1 & !M1C2 & !M1C3 ) |
                ( M2C0 & M2C1 & !M2C2 & !M2C3 ) ); // 2 specific channels hit in either module, no others
16'h0302: MC <= ( ( M0C0 & M0C1 & M0C2 & !M0C3 ) | ( M1C0 & M1C1 & M1C2 & !M1C3 ) |
                ( M2C0 & M2C1 & M2C2 & !M2C3 ) ); // 3 specific channels hit in either module, no others
16'h0303: MC <= ( ( M0C0 & M0C1 & M0C2 & M0C3 ) | ( M1C0 & M1C1 & M1C2 & M1C3 ) |
                ( M2C0 & M2C1 & M2C2 & M2C3 ) ); // 4 specific channels hit in either module, no others

16'h0304: MC <= ( ( M0C0 ) | ( M1C0 ) | ( M2C0 ) );
           // 1 specific channel hit in either module, others don't matter
16'h0305: MC <= ( ( M0C0 & M0C1 ) | ( M1C0 & M1C1 ) | ( M2C0 & M2C1 ) );
           // 2 specific channels hit in either module, others don't matter
16'h0306: MC <= ( ( M0C0 & M0C1 & M0C2 ) | ( M1C0 & M1C1 & M1C2 ) |
                ( M2C0 & M2C1 & M2C2 ) ); // 3 specific channels hit in either module, others don't matter

16'h0311: MC <= ( ( multi00 >= multi_req1 ) | ( multi01 >= multi_req1 ) | ( multi02 >= multi_req1 ) ); // 1 or more in either
module
16'h0312: MC <= ( ( multi00 >= multi_req1 ) | ( multi01 >= multi_req1 ) | ( multi02 >= multi_req1 ) ); // 2 or more in either
module
16'h0313: MC <= ( ( multi00 >= multi_req1 ) | ( multi01 >= multi_req1 ) | ( multi02 >= multi_req1 ) ); // 3 or more in either
module
16'h0314: MC <= ( ( multi00 == multi_req1 ) | ( multi01 == multi_req1 ) | ( multi02 >= multi_req1 ) ); // 1 each in either module
16'h0315: MC <= ( ( multi00 == multi_req1 ) | ( multi01 == multi_req1 ) | ( multi02 >= multi_req1 ) ); // 2 each in either module
16'h0316: MC <= ( ( multi00 == multi_req1 ) | ( multi01 == multi_req1 ) | ( multi02 >= multi_req1 ) ); // 3 each in either module

16'h0330: MC <= ( ( pa00 | pa01 ) | ( pa02 | pa03 ) | ( pa04 | pa05 ) ); // one or more out of 6 pairs ; a pair is two ch.hit
16'h0331: MC <= ( ^{(pa00 | pa01) , (pa02 | pa03) , (pa04 | pa05)} ); // one or more pair in one module, not more
16'h0332: MC <= ( ( pa00 ^ pa01 ) | ( pa02 ^ pa03 ) | ( pa04 ^ pa05 ) ); // exactly one pair in one or more modules
16'h0333: MC <= ( ^{(pa00 ^ pa01) , (pa02 ^ pa03) , (pa04 ^ pa05)} ); // exactly one pair in one module

16'h0334: MC <= ( ( po00 | po01 ) | ( po02 | po03 ) | ( po04 | po05 ) ); // one or more out of 6 pairs ; a pair is one of two ch. hit
16'h0335: MC <= ( ^{(po00 | po01) , (po02 | po03) , (po04 | po05)} ); // one or more pair in one module, not more
16'h0336: MC <= ( ( po00 ^ po01 ) | ( po02 ^ po03 ) | ( po04 ^ po05 ) ); // exactly one pair in one or more modules

```

```

16'h0337: MC <= ( ^{(po00 ^ po01) , (po02 ^ po03) , (po04 ^ po05)} ); // exactly one pair in one module

// 4 modules
16'h0400: MC <= (&hit0s[3:0]); // 1 specific channel hit in (each of) 4 specific modules, others don't matter
16'h0401: MC <= (&hit0s[3:0] & &hit1s[3:0]); // 2 specific channels hit in (each of) 4 specific module, others don't matter

// 5 modules
16'h0500: MC <= (&hit0s[4:0]); // 1 specific channel hit in (each of) 5 specific modules, others don't matter
16'h0501: MC <= (&hit0s[4:0] & &hit1s[4:0]); // 2 specific channels hit in (each of) 5 specific module, others don't matter

// 6 modules
16'h0600: MC <= (&hit0s[5:0]); // 1 specific channel hit in (each of) 6 specific modules, others don't matter
16'h0601: MC <= (&hit0s[5:0] & &hit1s[5:0]); // 2 specific channels hit in (each of) 6 specific module, others don't matter

// 7 modules
16'h0700: MC <= (&hit0s[6:0]); // 1 specific channel hit in (each of) 7 specific modules, others don't matter
16'h0701: MC <= (&hit0s[6:0] & &hit1s[6:0]); // 2 specific channels hit in (each of) 7 specific module, others don't matter

// 8 modules
16'h0800: MC <= (&hit0s[7:0]); // 1 specific channel hit in (each of) 8 specific modules, others don't matter
16'h0801: MC <= (&hit0s[7:0] & &hit1s[7:0]); // 2 specific channels hit in (each of) 8 specific module, others don't matter

// 9 modules
16'h0900: MC <= (&hit0s[8:0]); // 1 specific channel hit in (each of) 9 specific modules, others don't matter
16'h0901: MC <= (&hit0s[8:0] & &hit1s[8:0]); // 2 specific channels hit in (each of) 9 specific module, others don't matter

// 10 modules
16'h0A00: MC <= (&hit0s[9:0]); // 1 specific channel hit in (each of) 10 specific modules, others don't matter
16'h0A01: MC <= (&hit0s[9:0] & &hit1s[9:0]); // 2 specific channels hit in (each of) 10 specific module, others don't matter

// 11 modules
16'h0B00: MC <= (&hit0s[10:0]); // 1 specific channel hit in (each of) 11 specific modules, others don't matter
16'h0B01: MC <= (&hit0s[10:0] & &hit1s[10:0]); // 2 specific channels hit in (each of) 11 specific module, others don't matter

//12 modules
16'h0C00: MC <= (&hit0s[11:0]); // 1 specific channel hit in (each of) 12 specific modules, others don't matter
16'h0C01: MC <= (&hit0s[11:0] & &hit1s[11:0]); // 2 specific channels hit in (each of) 12 specific module, others don't matter

// 13 modules
16'h0D00: MC <= (&hit0s[12:0]); // 1 specific channel hit in (each of) 13 specific modules, others don't matter
16'h0D01: MC <= (&hit0s[12:0] & &hit1s[12:0]); // 2 specific channels hit in (each of) 13 specific module, others don't matter

// group tests
16'h1020: MC <= (MSUM_g1 >= 2); // 2 or more modules in first group, rest don't care
16'h1021: MC <= ( (MSUM_g1 >= 2) | (MSUM_all >=3) ); // 2 or more modules in first group, OR at least 3 modules
overall

// ARSA tests
//M0: gamma pairs, M1, M2: beta pairs
16'h1100: MC <= ( (pa00 ^ pa01) & ( ^{po02,po03,po04,po05} ) );
// only one pair (=2 hits) from M0 AND exactly one pair (>0 hits) from M1,M2
16'h1101: MC <= ( (pa00 | pa01) | ( |{po02,po03,po04,po05} ) );
// at least one pair (=2 hits) from M0 OR at least one pair (>0 hits) from M1,M2
16'h1102: MC <= ( (pa00 ^ pa01) | ( |{po02,po03,po04,po05} ) );
// only one pair (=2 hits) from M0 OR at least one pair (>0 hits) from M1,M2

```

```

16'h1103: MC <= ( (pa00 ^ pa01) );
           // only one pair (=2 hits) from M0, rest don't care
16'h1104: MC <= ( (pa00 | pa01) );
           // at least one pair (=2 hits) from M0, rest don't care
16'h1105: MC <= ( ((pa00 & !po01) | (!po00 & pa01)) & ( ^{po02,po03,po04,po05} ) );
           // exactly one pair (2/0 or 0/2 hits) from M0 AND exactly one pair (>0 hits) from M1,M2
16'h1106: MC <= ( ((pa00 & !po01) | (!po00 & pa01)) | ( {po02,po03,po04,po05} ) );
           // exactly one pair (2/0 or 0/2 hits) from M0 OR at least one pair (>0 hits) from M1,M2

default: MC <= 1;

```

Module Coincidence Setup

Each module always tests the *Hit Pattern* of contributing channels against the user defined *Coincidence Pattern* (set in the [Coincidence](#) tab of the *Parameter Setup* panel) before recording and processing an event. This is called the local coincidence test.

A global coincidence test can be applied using a PXI PDM in slot 2: Each module can send its hit pattern to the PXI PDM. The module in slot 2 defines which test the PXI PDM should apply (see above), and the PXI PDM outputs the result of the test on a shared backplane line.

In addition, each module can add the negative result of its local test to the global test, essentially vetoing those events that do not pass its local test for *all* modules. This allows one or more master trigger modules to influence acquisition for all modules. Limited module coincidence requirements can thus be applied even without a PXI PDM

With the checkboxes in the [[Module Coincidence Setup](#)], each module can thus be set to accept events if

- a) only the local coincidence test is passed (check "local")
- b) only the global coincidence test is passed (check "global")
- c) either the local OR the global coincidence test is passed (check "global" and "local")
- d) the global test AND all relevant local tests pass (check "global" and "local adds to global")

The [[Coincidence Pattern](#)] and [[Coincidence Window](#)] for each module are shown for reference only.

Examples:

1. To require a local coincidence of channels 0-1, 2-3, or both, set the coincidence pattern to 0x9008 in the [ModuleRegisterPanel](#) and check only the "local test" box in the Chassis Register Panel

2. To require coincidence of channels 0 and 1 in Module 0, and no other channel/module matters, in the [ModuleRegisterPanel](#) set the coincidence pattern in Module 0 to 0x8888 and in all other modules to 0xFFFF. In the Chassis Register Panel, check the "global test" box for all modules and the "local adds to global" box for module 0. No PXI PDM is required.

3. To require at least 3 channels to be active in all modules, use a PDM module in slot 2 and set the [[PDM control pattern](#)] to 0x0013 for the module in slot 3. Make sure the [[Module writes control pattern ...](#)] box is checked for this module and the [[Send local hit pattern to PDM](#)] box is checked for all modules. Then check the "global test" box for all modules