CXP-12 Interface Card 4C

Applet Feature Reference Manual for

Acq_DualCXP12Area

Functional Description
For pylon or GenTL Usage

Document Number: AW001883
Part Number: 000 (English)
Document Version: 01
Release Date: 5 September 2024
Applet Version 2.4.7.0

Contacting Basler Support Worldwide

Europe, Middle East, Africa
Tel. +49 4102 463 515
support.europe@baslerweb.com

The Americas
Tel. +1 610 280 0171
support.usa@baslerweb.com

Asia-Pacific
Tel. +65 6367 1355
support.asia@baslerweb.com

Singapore
Tel. +65 6367 1355
support.asia@baslerweb.com

Taiwan
Tel. +886 3 558 3955
support.asia@baslerweb.com

China
Tel. +86 10 6295 2828
support.asia@baslerweb.com

Korea
Tel. +82 31 714 3114
support.asia@baslerweb.com

Japan
Tel. +81 3 6672 2333
support.asia@baslerweb.com

https://www.baslerweb.com/en/sales-support/support-contact

Supplemental Information

Acquisition Card Documentation:
https://docs.baslerweb.com/acquisition-cards
Frame Grabber Documentation:
https://docs.baslerweb.com/frame-grabbers
Framegrabber SDK Documentation:
https://docs.baslerweb.com/frame-grabbers/framegrabber-sdk-overview.html

All material in this publication is subject to change without notice and is copyright Basler AG.

Table of Contents

1. Introduction ... 1
    1.1. Features of Applet Acq_DualCXP12Area ... 1
        1.1.1. Parameterization Order ... 2
    1.2. Bandwidth ... 3
    1.3. Requirements ... 3
        1.3.1. Software Requirements ... 3
        1.3.2. Hardware Requirements ... 4
        1.3.3. License ... 4
    1.4. Camera Interface ... 4
    1.5. Frame ID ... 4
    1.6. Image Transfer to PC Memory ... 4
2. CoaXPress ... 6
    2.1. SystemmonitorStreamPacketSize ... 6
    2.2. SystemmonitorCxpStandard ... 6
    2.3. CxpStreamPacketCount ... 7
    2.4. PixelFormat ... 7
    2.5. SystemmonitorUsedCxpConnections ... 8
    2.6. SystemmonitorCxpImageLineMode ... 9
    2.7. TapGeometry ... 9
3. Camera ... 11
    3.1. CameraEvents ... 11
        3.1.1. CameraStreamStatus ... 11
        3.1.2. FrameTransferStart ... 13
        3.1.3. FrameTransferEnd ... 13
4. SensorGeometry ... 14
    4.1. VantagePoint ... 14
    4.2. SensorWidth ... 14
    4.3. SensorHeight ... 15
5. ROI ... 16
    5.1. Width ... 17
    5.2. Height ... 18
    5.3. OffsetX ... 18
    5.4. OffsetY ... 19
6. Trigger ... 20
    6.1. Features and Functional Blocks of Area Trigger ... 20
    6.2. Digital Input/Output Mapping ... 24
    6.3. Event Overview ... 24
    6.4. Trigger Scenarios ... 25
        6.4.1. Internal Frequency Generator / interface card Controlled ... 25
        6.4.2. External Trigger Signals / IO Triggered ... 26
        6.4.3. Control of Two Flash Lights ... 29
        6.4.4. Software Trigger ... 32
        6.4.5. Software Trigger with Trigger Queue ... 34
        6.4.6. External Trigger with Trigger Queue ... 36
        6.4.7. Bypass External Trigger Signals ... 37
        6.4.8. Multi Camera Applications / Synchronized Cameras ... 37
        6.4.9. Hardware System Analysis and Error Detection / Trigger Debugging ... 37
    6.5. Parameters ... 38
        6.5.1. AreaTriggerMode ... 38
        6.5.2. TriggerState ... 40
        6.5.3. TriggerOutputFrequency ... 40
        6.5.4. Trigger Input ... 41
            6.5.4.1. External ... 41
                6.5.4.1.1. TriggerInDebounce ... 41
                6.5.4.1.2. FrontGPI ... 42
                6.5.4.1.3. TriggerInSource ... 43

6.5.4.1.4. TriggerInPolarity ... 43
6.5.4.1.5. TriggerInDownscale ... 44
6.5.4.1.6. TriggerInDownscalePhase ... 44
6.5.4.2. Software Trigger ... 45
6.5.4.2.1. SendSoftwareTrigger ... 45
6.5.4.2.2. SoftwareTriggerIsBusy ... 46
6.5.4.2.3. SoftwareTriggerQueueFillLevel ... 46
6.5.4.3. InStatistics ... 46
6.5.4.3.1. TriggerInStatisticsSource ... 47
6.5.4.3.2. TriggerInStatisticsPolarity ... 47
6.5.4.3.3. TriggerInStatisticsPulseCount ... 47
6.5.4.3.4. TriggerInStatisticsPulseCountClear ... 48
6.5.4.3.5. TriggerInStatisticsFrequency ... 48
6.5.4.3.6. TriggerInStatisticsMinimumFrequency ... 49
6.5.4.3.7. TriggerInStatisticsMaximumFrequency ... 49
6.5.4.3.8. TriggerInStatisticsMinMaxFrequencyClear ... 49
6.5.4.3.9. LineFront0RisingEdge ... 50
6.5.4.3.10. LineFront0FallingEdge ... 50
6.5.5. Sequencer ... 50
6.5.5.1. TriggerMultiplyPulses ... 50
6.5.6. Queue ... 51
6.5.6.1. TriggerQueueMode ... 51
6.5.6.2. TriggerQueueFillLevel ... 52
6.5.6.3. TriggerQueueFillLevelEventOnThreshold ... 52
6.5.6.4. TriggerQueueFillLevelEventOffThreshold ... 53
6.5.6.5. TriggerQueueFilllevelThresholdOn ... 53
6.5.6.6. TriggerQueueFilllevelThresholdOff ... 53
6.5.7. Pulse Form Generator 0 ... 54
6.5.7.1. TriggerPulseFormGenerator0Downscale et al. ... 54
6.5.7.2. TriggerPulseFormGenerator0DownscalePhase et al. ... 55
6.5.7.3. TriggerPulseFormGenerator0Delay et al. ... 56
6.5.7.4. TriggerPulseFormGenerator0Width et al. ... 56
6.5.8. Pulse Form Generator 1 ... 57
6.5.9. Pulse Form Generator 2 ... 57
6.5.10. Pulse Form Generator 3 ... 57
6.5.11. CameraOutSignalMapping ... 57
6.5.11.1. CxpLinkTrigger0Source ... 57
6.5.11.2. CxpLinkTrigger1Source ... 58
6.5.11.3. CxpLinkTrigger2Source ... 59
6.5.11.4. CxpLinkTrigger3Source ... 60
6.5.12. DigitalOutput ... 61
6.5.12.1. TriggerOutSelectFrontGPO0 ... 62
6.5.12.2. TriggerOutSelectFrontGPO1 ... 63
6.5.12.3. OutStatistics ... 64
6.5.12.3.1. TriggerExceededPeriodLimits ... 65
6.5.12.3.2. TriggerExceededPeriodLimitsClear ... 65
6.5.12.3.3. TriggerOutStatisticsSource ... 65
6.5.12.3.4. TriggerOutStatisticsPulseCount ... 66
6.5.12.3.5. TriggerOutStatisticsPulseCountClear ... 66
6.5.12.3.6. MissingCameraFrameResponse ... 67
6.5.12.3.6.1. ... 67
6.5.12.3.7. MissingCameraFrameResponseClear ... 68
6.5.12.3.8. TriggerExceededPeriodLimits ... 68
6.5.12.3.9. FrameTriggerMissed ... 68
6.5.13. OutputEvents ... 68
6.5.13.1. TriggerOutputEventSelect ... 69
6.5.13.2. AcquisitionTrigger ... 69
7. BufferStatus ... 70

7.1. FillLevel ... 70
7.2. Overflow ... 71
7.3. OverflowOffThreshold ... 71
7.4. OverflowOnThreshold ... 72
7.5. OverflowSyncOnThreshold ... 72
7.6. OverflowEventSelect ... 72
7.7. OverflowEvents ... 73
7.7.1. Overflow ... 74
8. ImageSelector ... 75
8.1. ImageSelectPeriod ... 75
8.2. ImageSelect ... 76
9. WhiteBalance ... 77
9.1. ScalingFactorGreen ... 77
9.2. ScalingFactorRed ... 77
9.3. ScalingFactorBlue ... 77
10. ColorConverter ... 79
11. OutputFormat ... 80
11.1. Format ... 80
11.2. BitAlignment ... 83
11.3. PixelDepth ... 84
11.4. CustomBitShiftRight ... 84
12. Miscellaneous ... 86
12.1. Version ... 86
12.1.1. AppletVersion ... 86
12.1.2. AppletRevision ... 86
12.1.3. VisualAppletsBuildVersion ... 87
13. BoardStatus ... 88
13.1. SystemmonitorMappedToFgPort ... 88
13.2. SystemmonitorCurrentLinkSpeed ... 88
13.3. SystemmonitorPcieTrainedPayloadSize ... 89
13.4. SystemmonitorPcieTrainedRequestSize ... 89
13.5. CxpInputMappedToFWPortPort ... 89
14. Errors ... 91
14.1. SystemmonitorDecoder8b10bError ... 91
14.2. SystemmonitorByteAlignment8b10bLocked ... 91
14.3. SystemmonitorRxStreamIncompleteCount ... 92
14.4. SystemmonitorRxUnknownDataReceivedCount ... 92
14.5. CxpOvertriggerRequestPulseCount ... 92
14.6. CxpTriggerAckMissingCount ... 93
14.7. CxpControlAckLostCount ... 93
14.8. CxpControlTagErrorCount ... 94
14.9. CxpControlAckIncompleteCount ... 94
14.10. CxpHeartbeatIncompleteCount ... 95
14.11. CxpHeartbeatMaxPeriodViolationCount ... 95
14.12. PacketTagErrorCount ... 96
14.13. SystemmonitorPacketbufferOverflowCount ... 96
14.14. SystemmonitorPacketbufferOverflowSource ... 97
14.15. CxpImageTagErrorCount ... 97
14.16. CxpStreamIDErrorCount ... 97
14.17. CxpCameraMarkerErrorCount ... 98
14.18. CxpCameraUnexpectedStartupDataStatus ... 98
14.19. CxpCameraFrameLostCount ... 99
14.20. CxpCameraFrameCorruptCount ... 99
14.21. CrcErrors ... 100
14.21.1. SystemmonitorRxPacketCrcErrorCount ... 100
14.21.2. CxpStreamPacketCrcError ... 100
14.21.3. CxpControlAckPacketCrcError ... 101
14.22. LengthErrors ... 101

14.22.1. SystemmonitorRxLengthErrorCount 101
14.22.2. CxpStreamPacketLengthError 102
14.23. ReceivedPacketsCorrected 102
14.23.1. CxpErrorCorrected 102
14.23.2. CxpErrorCorrectedTrigger 103
14.23.3. CxpErrorCorrectedTriggerAck 103
14.23.4. CxpErrorCorrectedStream 104
14.23.5. CxpErrorCorrectedControlAck 104
14.23.6. CxpErrorCorrectedLinkTest 104
14.23.7. CxpErrorCorrectedHeartbeat 105
14.23.8. CameraCorrectedErrorCount 105
14.24. ReceivedPacketsUncorrected 106
14.24.1. CxpErrorUncorrected 106
14.24.2. CxpErrorUncorrectedTrigger 106
14.24.3. CxpErrorUncorrectedTriggerAck 107
14.24.4. CxpErrorUncorrectedStream 107
14.24.5. CxpErrorUncorrectedControlAck 107
14.24.6. CxpErrorUncorrectedLinkTest 108
14.24.7. CxpErrorUncorrectedHeartbeat 108
14.24.8. CameraUncorrectedErrorCount 109
14.25. UnsupportedPackets 109
14.25.1. SystemmonitorRxUnsupportedPacketUnit 109
14.25.2. CxpUnsupportedGpioReceived 110
14.25.3. CxpUnsupportedEventReceived 110
14.25.4. CxpUnsupportedHeartbeatReceived 110
14.25.5. CxpUnsupportedGpioAckReceived 111
14.25.6. CxpUnsupportedGpioRequestReceived 111
15. Revision History 113
15.1. Fixed Issues 113
15.1.1. Fixed in Version 2.4.7.0 113
15.1.2. Fixed in Version 2.2.7.0 114
15.1.3. Fixed in Version 2.0.5.0 114
15.2. Known Issues 114
Glossary 115
Index 118

Chapter 1. Introduction

This document provides you with detailed information on applet "Acq_DualCXP12Area" for CXP-12 Interface Card 4C .

In the following, you will find a full description of the applet's functionality and features.

For information on the hardware or for a general introduction on how to configure the CXP-12 Interface Card using the pylon API, the pylon Viewer, or the gpioTool check the document which can be found in https://docs.baslerweb.com/pc-cards.

All applet-specific parameters described in this document are as represented in the GenTL interface.

For a general explanation of the GenTL interface, check the Basler GenTL interface documentation (https://www.baslerweb.com/en/sales-support/downloads/document-downloads/cxp-gentl-producer-feature-documentation/).

For information on camera features, check the respective camera documentation.

For information on Basler pylon features and for API documentation, check the pylon documentation.

1.1. Features of Applet Acq_DualCXP12Area

"Acq_DualCXP12Area" is a dual-camera applet. Up to two individual cameras can be used in parallel. All features of this applet are available for both camera ports. You can configure the CoaXPress camera interface for CoaXPress cameras version 1.1.1 and 2.0, transferring grayscale (monochrome), Bayer pattern , or color pixels. Allowed pixel formats are Gray (Mono8, Mono10, Mono12, Mono14, Mono16), Bayer (BayerGR8, BayerGR10, BayerGR12, BayerGR14, BayerRG8, BayerRG10, BayerRG12, BayerRG14, BayerGB8, BayerGB10, BayerGB12, BayerGB14, BayerBG8, BayerBG10, BayerBG12, BayerBG14), Color (RGB8, RGB10, RGB12, RGB14, RGB16), and YCbCr422_8. You can use cameras with CoaXPress link aggregation of 2 and 1 with this applet. The maximum link speed is CXP-12. A multi-functional area trigger is included in the applet. This allows you to control the camera or external devices using interface card generated, external, or software generated trigger pulses. Area scan cameras transferring images with a resolution of up to 32768 by 65536 pixels are supported. The applet is processing data at a bit depth of 16 bits. An image selector at the camera port facilitates the selection of one image out of a parameterizable sequence of images. This enables the distribution of the images to multiple interface card and PCs. The applet supports tap geometry sorting 1X-1Y as well as 1X-2YE. For reverse operation, you can mirror the image in x-direction and y-direction before cutting the ROI. Acquired images are buffered in interface card memory. You can select a region of interest (ROI) for further processing. The stepsize of the ROI width is 16 pixel. The ROI stepsize for the image height is 1 line. The high quality Bayer pattern de-mosaicing is based on a 5x5 kernel size. A color converter automatically converts the input pixel formats to the output formats. In this applet conversions from monochrome, RGB or Bayer to monochrome and RGB can be performed.

Processed image data are output by the applet via high speed DMA channels. You can select the pixel format of the output. The pixel format can either be 8 bit, 10 bit packed, 12 bit packed, 14 bit packed, or 16 bits per pixel (or per pixel component if you work with a color format).

Table 1.1. Feature Summary of Acq_DualCXP12Area

Feature	Applet Property
Applet Name	 Acq_DualCXP12Area
Type of Applet	AcquisitionApplets
Board	CXP-12 Interface Card 4C
No. of Cameras	2 , asynchronous or synchronous
Camera Type	CoaXPress, link aggregation max. 2, maximum speed CXP-12, Version 1.1.1 and 2.0
Sensor Type	Area Scan
Camera Format	Monochrome, Bayer pattern or RGB
Pixel Format	Gray (Mono8, Mono10, Mono12, Mono14, Mono16), Bayer (BayerGR8, BayerGR10, BayerGR12, BayerGR14, BayerRG8, BayerRG10, BayerRG12, BayerRG14, BayerGB8, BayerGB10, BayerGB12, BayerGB14, BayerBG8, BayerBG10, BayerBG12, BayerBG14), Color (RGB8, RGB10, RGB12, RGB14, RGB16), and YCbCr422_8.
Processing Bit Depth	16 Bit per color component
Maximum Images Dimensions	32768 * 65536
ROI Stepsize	x: 16, y: 1
Tap Geometry Sorting	1X-1Y, 1X-2YE
Mirroring	Yes, horizontal and vertical (set the parameter *VantagePoint*)
Image Selector	Yes
Noise Filter	No
Shading Correction	No
Dead Pixel Interpolation	No
Bayer Filter	Yes, High Quality Extended (HQe)
Color White Balancing	Yes
Color Converter	yes, Mono, RGB or Bayer to Mono or RGB
Lookup Table	No
DMA	Full Speed
DMA Image Output Format	All grayscale and color formats. See description above.
Event Generation	yes
Overflow Control	yes

1.1.1. Parameterization Order

We recommend to configure the functional blocks which are responsible for sensor setup/correction first. This will be the camera settings, shading correction, and dead pixel interpolation (if available). Afterwards, you can

configure other image enhancement functional blocks such as white balancing, noise filter, and lookup table. By default, all presets are configured for receiving images directly.

1.2. Bandwidth

The maximum bandwidths of applet Acq_DualCXP12Area are listed in the following table.

Table 1.2. Bandwidth of Acq_DualCXP12Area

Description	Bandwidth
Max. CXP Speed	CXP-12
Peak Bandwidth per Camera	2400 MPixel/s
Mean Bandwidth per Camera	2400 MPixel/s
DMA Bandwidth	7200 MByte/s (depends on PC mainboard)

The peak bandwidth defines the maximum allowed bandwidth for each camera at the camera interface. If the camera's peak bandwidth is higher than the mean bandwidth, the interface card on-board buffer will fill up as the data can be buffered, but not be processed at that speed.

The mean bandwidth per camera describes the maximally allowed mean bandwidth for each camera at the camera interface. It is the product of the framerate and the image pixels. For example, with 1-megapixel images at a framerate of 100 frames per second, the mean bandwidth will be 100 MPixel/s. In case of 8bit per pixel as output format, this would be equal to 100 MB per second.

The required output bandwidth of an applet can differ from the input bandwidth. A region of interest (ROI) and the output format can change the required output bandwidth and the maximum mean bandwidth. Moreover, this applet is a Bayer applet. The required output bandwidth will be three times higher than the input bandwidth. (This applies only when debayering is switched to ON.) Mind that the DMA bandwidth is the total bandwidth. The sum of all camera channel bandwidths has to be less than the maximum DMA bandwidth to avoid overflows.

Regard the relation between MPixel/s and MByte/s: The MByte/s depend on the applet and its parameterization concerning the pixel format. It is possible to acquire more than 8 bit per pixel or to convert from one bit depth to another. 1 MByte is 1,000,000 Byte.

Bandwidth Varies

The exact maximum DMA bandwidth depends on the used PC system and its chipset. The camera bandwidth depends on the image size and the selected frame rate. The given values of 7200 MByte/s for the possible DMA bandwidth might be lower due to the chipset and its configuration. Additionally, some PCIe slots do not support the required number of lanes to transfer the requested or expected bandwidth. In these cases, have a look at the mainboard specification. A behaviour like multiplexing between several PCIe slots can be seen in rare cases. Some mainboard manufacturers provide a BIOS feature where you can select the PCIe payload size: Always try to set this to its maximum value or simply to automatic. This can help in specific cases.

1.3. Requirements

In the following, the requirements on software, hardware and interface card license are listed.

1.3.1. Software Requirements

To run this applet, a supporting runtime environment is required. This can be either Basler pylon, or the Basler Framegrabber SDK providing the GenTL interface.

1.3.2. Hardware Requirements

To run applet "Acq_DualCXP12Area", a Basler CXP-12 Interface Card 4C is required.

For PC system requirements, check the interface card hardware documentation. The applet itself does not require any additional PC system requirements.

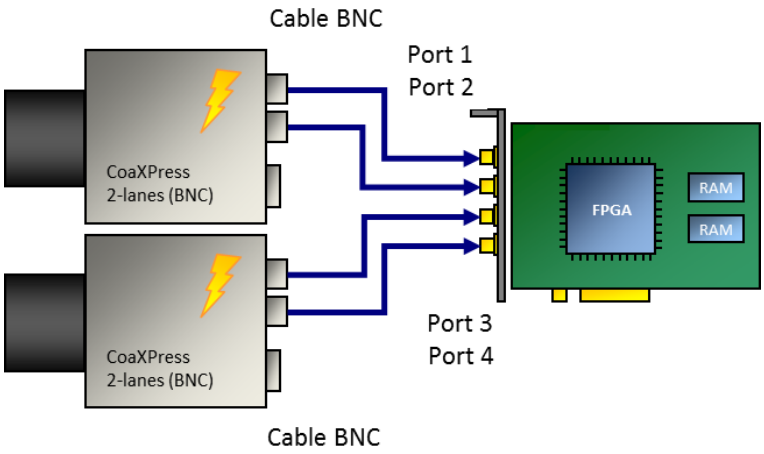
1.3.3. License

This applet is of type AcquisitionApplets. For applets of this type, no license is required. All compatible interface cards can run the applet using the Basler Framegrabber SDK.

1.4. Camera Interface

Applet "Acq_DualCXP12Area" supports 2 CXP cameras. The interface card has 4 connectors. Use two or one CoaXPress cables to connect each of the cameras with the interface card. The maximum link aggregation of this applet is two. The mapping of the ports between the camera and the interface card is not important. You can chose any order.

Figure 1.1. Camera Interface and Camera Cable Setup



1.5. Frame ID

For CoaXPress cameras, each frame includes a source tag also called frame ID. This applet will output each frame to the host PC attached with this frame ID. Moreover, overflow events will also include this frame ID. By this, the exact mapping of a given frame in the host PC to a camera frame is possible.

Check chapter Chapter 7, '*BufferStatus*' for more information about overflow conditions and the overflow event data structure including the frame ID.

The frame ID is processed together with the images in the host PC. Check the Basler GenTL documentation to learn on how to extract the frame ID from the buffer (https://www.baslerweb.com/en/sales-support/downloads/document-downloads/cxp-gentl-producer-feature-documentation/).

1.6. Image Transfer to PC Memory

The image transfer between interface card and PC is performed via DMA transfers. In this applet, 2 DMA channels exist for transferring image data. One channel for each camera. The DMA channels have the same

indices as the cameras, starting with 0. The applet output format can be set via the parameters of the output format module. See Chapter 11, '*OutputFormat*'. All outputs are little-endian coded.

Chapter 2. CoaXPress

This applet can be used with up to 2 area scan cameras. To receive correct image data from your camera, it is crucial that the camera output format matches the selected interface card input format. The following parameters configure the interface card's camera interface to match with the individual camera pixel format. Most cameras support different operation modes. Consult the manual of your camera to obtain the necessary information how to configure the camera to the desired pixel format.

Ensure that the images transferred by the camera do not exceed the maximum allowed image dimensions for this applet (32768 x 65536).

With the following parameters you can define the way trigger packets are sent from the interface card to the camera on the CXP link.

This applet allows the usage of 1x-2ye and 1x-1y sorting of the image data according to the CoaXPress standard.

2.1. SystemmonitorStreamPacketSize

Returns the stream packet size in bytes. Range: between 4 and 65535 bytes in steps of 4 bytes.

Table 2.1. Parameter properties of SystemmonitorStreamPacketSize

Property	Value
Name	**SystemmonitorStreamPacketSize**
Display Name	**Systemmonitor Stream Packet Size**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 2.1. Usage of SystemmonitorStreamPacketSize

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorStreamPacketSizeSelector = i;
    value_ = SystemmonitorStreamPacketSize;
}
```

2.2. SystemmonitorCxpStandard

Returns the version of the used CXP standard.

Table 2.2. CXP Standard Version

CXP Standard Version		
CXP_1_0		
CXP_1_1_1		
CXP_2_0		
Unknown		

Table 2.3. Parameter properties of SystemmonitorCxpStandard

Property	Value
Name	**SystemmonitorCxpStandard**
Display Name	**Systemmonitor CXP Standard**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Beginner**

Example 2.2. Usage of SystemmonitorCxpStandard

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorCxpStandardSelector = i;
    value_ = SystemmonitorCxpStandard;
}
```

2.3. CxpStreamPacketCount

This parameter counts the amount of received stream packets. Bits [29:0] count the number of packets. Bit [30] is set when a counter overflow occurs. Range: 0 to 4294967295 (32 bit).

Table 2.4. Parameter properties of CxpStreamPacketCount

Property	Value
Name	**CxpStreamPacketCount**
Display Name	**CXP Stream Packet Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 2.3. Usage of CxpStreamPacketCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpStreamPacketCountSelector = i;
    value_ = CxpStreamPacketCount;
}
```

2.4. PixelFormat

This parameter specifies the data format of the connected camera.

The formats defined in the following list can be selected. Choose the pixel format which best matches with your camera.

In this applet, the processing data bit depth is 16 bit. The camera interface automatically performs a conversion to the 16 bit format using bit shifting independently from the selected camera format. If the camera bit depth is greater than the processing bit depth, bits will be right shifted to meet the internal bit depth. If the camera bit depth is less than the processing bit depth, bits will be left shifted to meet the internal bit depth. In this case, the lower bits are fixed to zero.

This applet performs a Bayer de-mosaicing. The Bayer pattern is derived from the pixel format.

GenTL Controls the Pixel Format

The GenTL interface has a built in automatic adaptation of the pixel format to the camera settings. Changing the applet pixel format might be overwritten by the GenTL on acquisition start. You can only set the pixel format if the automatic setting is disabled. See the GenTL documentation parameter **`AutomaticFormatControl`** for more details.

Table 2.5. Parameter properties of PixelFormat

Property	Value
Name	**`PixelFormat`**
Display Name	**`Pixel Format`**
Interface	**`IEnumeration`**
Access policy	**`Read/Write`**
Visibility	**`Beginner`**
Allowed values	**`BayerGR8`** Bayer GR 8 **`BayerGR10p`** Bayer GR 10p **`BayerGR12p`** Bayer GR 12p **`BayerGR14p`** Bayer GR 14p **`BayerRG8`** Bayer RG 8 **`BayerRG10p`** Bayer RG 10p **`BayerRG12p`** Bayer RG 12p **`BayerRG14p`** Bayer RG 14p **`BayerGB8`** Bayer GB 8 **`BayerGB10p`** Bayer GB 10p **`BayerGB12p`** Bayer GB 12p **`BayerGB14p`** Bayer GB 14p **`BayerBG8`** Bayer BG 8 **`BayerBG10p`** Bayer BG 10p **`BayerBG12p`** Bayer BG 12p **`BayerBG14p`** Bayer BG 14p **`Mono8`** Mono 8 **`Mono10p`** Mono 10p **`Mono12p`** Mono 12p **`Mono14p`** Mono 14p **`Mono16`** Mono 16p **`RGB8`** RGB 8 **`RGB10p`** RGB 10p **`RGB12p`** RGB 12p **`RGB14p`** RGB 14p **`RGB16`** RGB 16 **`YCbCr422_8`** YCbCr422_8
Default value	**`Mono8`**

Example 2.4. Usage of PixelFormat

```
/* Set */ PixelFormat = Mono8;
/* Get */ value_ = PixelFormat;
```

2.5. SystemmonitorUsedCxpConnections

The currently used number of CXP ports used in this process.

Table 2.6. Parameter properties of SystemmonitorUsedCxpConnections

Property	Value
Name	**SystemmonitorUsedCxpConnections**
Display Name	**System Monitor Used Cxp Connections**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 1** **Maximum 4** **Stepsize 1**

Example 2.5. Usage of SystemmonitorUsedCxpConnections

```
/* Get */ value_ = SystemmonitorUsedCxpConnections;
```

2.6. SystemmonitorCxpImageLineMode

This parameter informs on the current transfer mode, used by the camera. The transfer can be an areascan (= 0) or linescan (= 1) image.

Table 2.7. Parameter properties of SystemmonitorCxpImageLineMode

Property	Value
Name	**SystemmonitorCxpImageLineMode**
Display Name	**System Monitor Cxp Image Line Mode**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 1** **Stepsize 1**

Example 2.6. Usage of SystemmonitorCxpImageLineMode

```
/* Get */ value_ = SystemmonitorCxpImageLineMode;
```

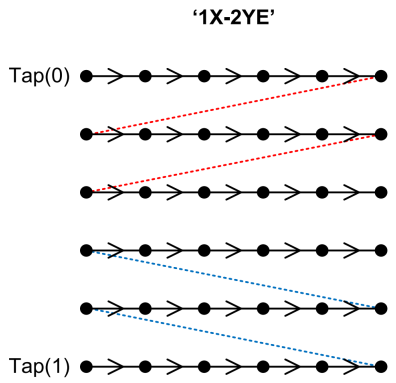
2.7. TapGeometry

Defines the tap geometry sorting of the applet.

The applet supports **Geometry_1X_1Y** or **Geometry_1X_2YE**. With **Geometry_1X_1Y** mode you define that the whole image is transferred as one data stream. This mode is the default mode for most cameras.

As sensors grow bigger, a faster readout is needed. To achieve this, use the **Geometry_1X_2YE** mode. In **Geometry_1X_2YE** mode, the sensor is read out in two subframes at once. Both subframes are transferred simultaneously using two streams.

The following image shows how the frames are composed:



If tap sorting is used, the applet combines all subframes into one frame. In order to get correct frames, the tap sorting of the applet needs to match the camera tap sorting.

If *TapGeometry* is set to **Geometry_1X_2YE**, the parameter *SensorHeight* needs to be set to the camera ROI size and must be an even integer. The parameter dependency will then be: *OffsetY* + *Height* <= *SensorHeight*.

Table 2.8. Parameter properties of TapGeometry

Property	Value
Name	**`TapGeometry`**
Display Name	**`Tap Geometry`**
Interface	**`IEnumeration`**
Access policy	**`Read/Write`**
Visibility	**`Beginner`**
Allowed values	**`Geometry_1X_1Y`** 1X-1Y **`Geometry_1X_2YE`** 1X-2YE
Default value	**`Geometry_1X_1Y`**

Example 2.7. Usage of TapGeometry

```
/* Set */ TapGeometry = Geometry_1X_1Y;
/* Get */ value_ = TapGeometry;
```

Chapter 3. Camera

This applet Acq_DualCXP12Area for the CXP-12 Interface Card 4C acquires the sensor data of an area scan camera. When this is performed some sensor dimension depending information can be used to register an event based callback function.

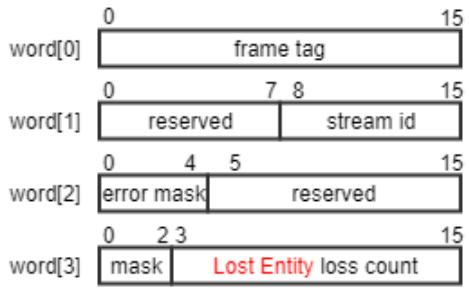
3.1. CameraEvents

In programming or runtime environments, a callback function is a piece of executable code that is passed as an argument, which is expected to call back (execute) exactly that time an event is triggered. This applet can generate some software callback events based on applet-events as explained in the following section. These events are not related to a special camera functionality. Other event sources are described in additional sections of this document.

The Basler Framegrabber SDK enables an application to get these event notifications about certain state changes at the data flow from camera to RAM and the image and trigger processing as well. Please consult the Basler Framegrabber SDK documentation for more details concerning the implementation of this functionality.

3.1.1. CameraStreamStatus

When the operator detects that the received reconstructed frame is larger or smaller than what was promoted by the camera in the CXP image header, a safety circuit gets activated. The operator then cuts off exceeding pixels and lines, so that the subsequent processing pipeline always sees the frame size which was defined in the image header. If the received frame is smaller in its dimensions than what was specified in the image header, the operator fills up the received frame with undefined data to achieve the specified frame dimensions which were defined in the image header. Filling up a smaller frame can cause the follow-up frames to get lost. The loss is then reported per event to the runtime software (Framegrabber SDK)(see the following paragraph). The size mismatch causes an event, too.



The event payload is provided as four 16-bit data words. The event format is defined as follows:

- word [0]
  - bits [0:15]: CXP image tag in which the event occurred.
- word [1]
  - bits [8:15]: Stream ID in which the event occurred.
  - bits [0:7]: Reserved, treat as don't care.
- word [2]
  - bit [0]: CRC error occurred.
  - bit [1]: Stream marker error detected in the image header.

- bit [2]: An error in the image header was detected which could not be corrected.
- bit [3]: A frame size error was detected, i.e. the image size defined in the CXP image header isn't matching the reconstructed frame size from the transmitted packets. This happens when the camera puts one info into the image header but transmits different amount of data as promoted in the header.
- bits [4:15]: Reserved, treat as don't care.

- word [3]
  - bit [0]: Event type, 0 = Corrupted Entity , 1 = Lost Entity.
    - **Corrupted Entity** means that the error happens within a frame and that this frame is already sourced into the VisualApplets pipeline.
    - **Lost Entity** means that the error occurred before the frame was forwarded to the following operators and the frame was discarded by the camera operator.
    - When a corrupted entity is observed, the operator will fill up the frame according to the CXP image header definition so that the following operators will not cause undefined behavior. During this fill-up, a new frame may arrive and will then get lost. The lost entity event will also be raised when the camera sends data with a gap according to the frame tag.
  - bit [1]: An event loss for type **Corrupted Entity** occurred. This means that preceding events of type **Corrupted Entity** got lost. This happens when the runtime software is not reacting to events and the internal event queues ran full.
  - bit [2]: An event loss for type **Lost Entity** occurred. This means that preceding events of type **Lost Entity** got lost. This happens when the runtime software (Framegrabber SDK) is not reacting to events and the internal event queues ran full.
  - bits [3:15]: amount of lost **Lost Entity** events.

There are two types of events: events for corrupted entities and events for lost entities. Bit 0 of word 3 describes which kind of event occurred. If the event buffers are full, it might happen that events get lost. When an event gets lost that marks a corrupted entity, bit 1 of word 3 will be set. When an event gets lost that marks a lost entity, bit 2 of word 3 will be set and bit 3 to 15 will provide the number of lost events indicating a lost frame. If bit 2 is set but the counter is 0, it means that a counter overflow happened.

Every event causes a software interrupt. To reduce the number of events, several events with the same frame tag might be merged together. In that case some error flags are combined. If an event was lost, the event before the lost event contains the information about the lost event and cannot be merged with further events with the same frame tag.

The events caused due to CRC errors report a frame tag, which may not be exactly related to the frame in which the CRC errors happen. The frame tag can be that of the preceding or following frame. This can only happen, when a camera sends a CXP packet, which contains a transition between 2 or more frames. The CRC computation is finished at the end of the packet, but the stream data is reconstructed on-the-fly. This means that a situation can happen, in which a CRC error is detected only after the preceding frame was already sent by the operator. In normal situations, in which the camera packets don't contain data both of the end of the ongoing frame and the beginning of the next frame, the frame tag during CRC error will always be correct. For all other cases as long as the complete frame stream data is less than the maximal packet size of 8k, there might be only 1 frame overlap within 1 packet. In that case, the software application should consider the preceding frame with the frame tag - 1 and the following frame with the frame tag + 1 as potentially corrupted as well.

Differentiating Error Events Between Taps

The error handling and event system are common to both CXP tap streams. Use the stream ID field to relate the received event to the appropriate tap. Normally, tap 0 will get a lower stream ID, typically 0. Tap 1 will get a stream ID, which is larger than the one of tap 0.

Table 3.1. Event parameters of CameraStreamStatus

Name	Interface	Description
EventCameraStreamStatusFrameId	IInteger	Frame ID in which the event occurred.
EventCameraStrear	IInteger	Stream ID in which the event occurred.
EventCameraStreamStatusEventType	IEnumeration	Event type (Corrupted or Lost)
EventCameraStrear	IBoolean	Event loss for type Corrupted occurred.
EventCameraStreamStatusLost	IBoolean	Event loss for type Lost occurred.
EventCameraStrear	IInteger	Amount of lost Lost events.

3.1.2. FrameTransferStart

This event is generated when the first pixel of one camera frame arrives at the applet. Keep in mind that a high framerate can cause high interrupt rates which might slow down the overall PC system. This event can only occur if the acquisition is running.

The applet generates frames from linescan cameras unsing the image trigger module. The event is generated with the first pixel of the generated frame which is simultaneously to the arrival of a camera line. Keep in mind that a high framerate can cause high interrupt rates which might slow down the overall PC system. This event can only occur if the acquisition is running.

3.1.3. FrameTransferEnd

This event is generated right after the last pixel of one camera frame arrives at the applet. Keep in mind that a high framerate can cause high interrupt rates which might slow down the overall PC system. This event can only occur if the acquisition is running.

The applet generates frames from linescan cameras unsing the image trigger module. The event is generated when the last pixel passes through the image trigger module. Note that this might not be at the same time as the pixel arrives from the camera at the framegrabber as the image trigger module needs to delay the data to wait for closing gates. Keep in mind that a high framerate can cause high interrupt rates which might slow down the overall PC system. This event can only occur if the acquisition is running.

Chapter 4. SensorGeometry

Some operations, for example mirroring or tap sorting, require knowledge on the sensor dimension and orientation of the camera. The following parameters supply this kind of information.

4.1. VantagePoint

This parameter defines the vantage point. Use this parameter to mirror the image. Note that when using this parameter for mirroring, the received sensor image is mirrored and not the selected ROI in the frame grabber. Therefore, to mirror the ROI in the frame grabber, ensure to set the correct offsets in the frame grabber.

If a horizontal mirroring is active, the parameter *SensorWidth* limits the maximum width. The parameter dependency will then be *OffsetX* + *Width* <= *SensorWidth*.

If a vertical mirroring is active or *TapGeometry* is set to **Geometry_1X_2YE** , the parameter *SensorHeight* limits the maximum height. The parameter dependency will then be *OffsetY* + *Height* <= *SensorHeight*.

Table 4.1. Parameter properties of VantagePoint

Property	Value
Name	**`VantagePoint`**
Display Name	**`Vantage Point`**
Interface	**`IEnumeration`**
Access policy	**`Read/Write`**
Visibility	**`Beginner`**
Allowed values	**`TopLeft`** Top Left **`TopRight`** Top Right **`BottomLeft`** Bottom Left **`BottomRight`** Bottom Right
Default value	**`TopLeft`**

Example 4.1. Usage of VantagePoint

```
/* Set */ VantagePoint = TopLeft;
/* Get */ value_ = VantagePoint;
```

4.2. SensorWidth

To mirror the incoming data correctly, the parameter *SensorWidth* is required. The value of *SensorWidth* is ignored, if *VantagePoint* = Top-**Left** or Bottom-**Left**. If also a vertical mirroring is used, the available DRAM and sensor height limit the maximum sensor width. This is so, because the sensor image needs to fit twice into the DRAM, because double buffering is used.

If No Mirroring Is Active, the Value of *SensorWidth* Is Not Used

If no mirroring is active, the value of the parameter *SensorWidth* is not used. Instead, the sum of *OffsetX* and *Width* is used. This makes the use of the module easier as an extra configuration is avoided, if defaults are used.

Table 4.2. Parameter properties of SensorWidth

Property	Value
Name	**SensorWidth**
Display Name	**Sensor Width**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Beginner**
Allowed values	**Minimum 48** **Maximum 32768** **Stepsize 16**
Default value	**1024**
Unit of measure	**pixel**

Example 4.2. Usage of SensorWidth

```
/* Set */ SensorWidth = 1024;
/* Get */ value_ = SensorWidth;
```

4.3. SensorHeight

For vertical mirroring or tap geometry sorting in vertical direction, the applet needs to be parameterized with the exact height transferred from the camera to the frame grabber. If you have set a region of interest in the camera, the parameter *SensorHeight* needs to be set to the ROI size, otherwise use the sensor height. If *TapGeometry* is set to **Geometry_1X_2YE**, the parameter *SensorHeight* needs to be set to the ROI size and must be an even integer. The parameter dependency will then be: *OffsetY* + *Height* <= *SensorHeight*.

If Only One Y-Zone Is Used and No Vertical Mirroring Is Active, the Value of *SensorHeight* Is Not Used

If no vertical mirroring is configured and if *TapGeometry* is set to **Geometry_1X_1Y**, the value of the parameter *SensorHeight* is not used. Instead, the sum of *OffsetY* and *Height* is used. This makes the use of the module easier as an extra configuration is avoided, if defaults are used.

Table 4.3. Parameter properties of SensorHeight

Property	Value
Name	**SensorHeight**
Display Name	**Sensor Height**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Beginner**
Allowed values	**Minimum 2** **Maximum 65536** **Stepsize 1**
Default value	**1024**
Unit of measure	**pixel**

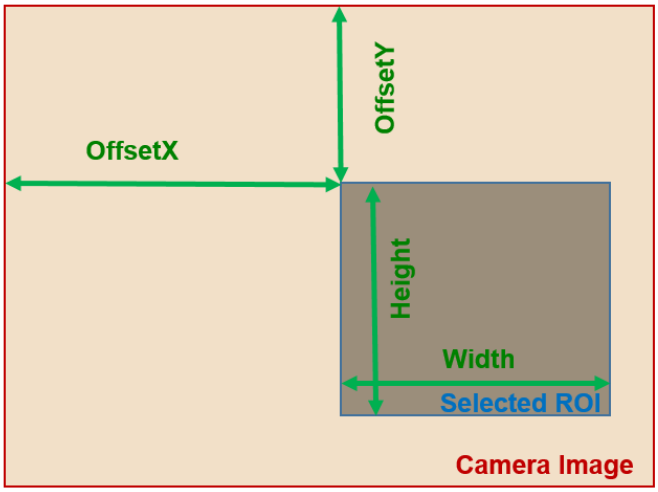
Example 4.3. Usage of SensorHeight

```
/* Set */ SensorHeight = 1024;
/* Get */ value_ = SensorHeight;
```

Chapter 5. ROI

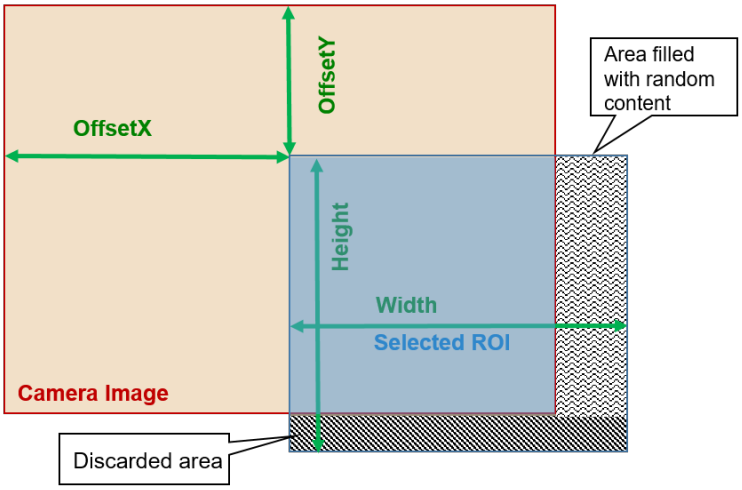
This module allows the definition of a region of interest (ROI), also called area of interest (AOI). A ROI allows the selection of a smaller subset pixel area from the input image. It is defined by using parameters *OffsetX*, *Width*, *OffsetY* and *Height*. The following figure illustrates the parameters.

Figure 5.1. Region of Interest



As can be seen, the region of interest lies within the input image dimensions. Thus, if the image dimension provided by the camera is greater or equal to the specified ROI parameters, the applet will fully cut-out the ROI subset pixel area. However, if the image provided by the camera is smaller than the specified ROI, lines will be filled with random pixel content and the image height might be cut or filled with random image lines as illustrated in the following.

Figure 5.2. Region of Interest Selection Outside the Input Image Dimensions



Furthermore, mind that the image sent by the camera must not exceed the maximum allowed image dimensions. This applet allows a maximum image width of 32768 pixels and a maximum image height of 65536 lines. The chosen ROI settings can have a direct influence on the maximum bandwidth of the applet as they define the image size and thus, define the amount of data.

The parameters have dynamic value ranges. For example an x-offset cannot be set if the sum of the offset and the image width will exceed the maximum image width. To set a high x-offset, the image width has to be reduced, first. Hence, the order of setting the parameters for this module is important. The return values of the function calls in the SDK should always be evaluated to check if changes were accepted.

Mind the minimum step size of the parameters. This applet has a minimum step size of 16 pixel for the width and the x-offset, while the step size for the height and the y-offset is 1.

The settings made in this module will define the display size and buffer size if the applet is used in microDisplay. If you use the applet in your own programs, ensure to define a sufficient buffer size for the DMA transfers in your PC memory.

All ROI parameters can only be changed if the acquisition is not started i.e. stopped.

Automatic Adaptation to Camera Width and Height with the GenTL Adaptor

The GenTL adaptor can automatically copy the image width and height from the camera to the applet settings so that the user does not have to set these values. Changing the *Width* and *Height* of the applet might get overwritten by the Gen TL on acquisition start. You can only set the width and height if this automatic adaptation is disabled. See the GenTL documentation parameter **`AutomaticROIControl`** for more details.

ROI Setting Defines GenTL Buffer Info

The parameters define the DMA output size and therefore the GenTL buffer info values to inform the consumer about the used output image width and height of the interface. See the GenTL documentation parameter **`AutomaticROIControl`** for more details.

Influence on Bandwidth

A ROI might cause a strong reduction of the required bandwidth. If possible, the camera frame dimension should be reduced directly in the camera to the desired size instead of reducing the size in the applet. This will reduce the required bandwidth between the camera and the interface card.

5.1. Width

The parameter specifies the width of the ROI. The values of parameters *Width* + *OffsetX* must not exceed the maximum image width of 32768 pixels. If a horizontal mirroring is active the sensor width limits the maximum width (Width + XOffset). If furthermore vertical mirroring is active the maximum width is limited by the DRAM and sensor height (the sensor dimension needs to fit into the DRAM).

Maximum image width is reduced for horizontal mirrored images

Limitations of the available BRAM in the FPGA allow only to store smaller lines and there for the images that can be mirrored needs to be smaller. A mirrored image can only have width of 21840, the not mirrored image can have the full width of 32768.

Table 5.1. Parameter properties of Width

Property	Value
Name	**`Width`**
Display Name	**`Width`**
Interface	**`IInteger`**
Access policy	**`Read/Write`**
Visibility	**`Expert`**
Allowed values	**`Minimum 48`** **`Maximum 32768`** **`Stepsize 16`**
Default value	**`1024`**
Unit of measure	**`pixel`**

Example 5.1. Usage of Width

```
/* Set */ Width = 1024;
/* Get */ value_ = Width;
```

5.2. Height

The parameter specifies the height of the ROI. The values of parameters *Height* + *OffsetY* must not exceed the maximum image height of 65536 pixels. If a vertical mirroring is active the sensor height limits the maximum height (Height + YOffset). Furthermore he maximum height is limited by the DRAM and the sensor width (the sensor dimension needs to fit into the DRAM).

Table 5.2. Parameter properties of Height

Property	Value
Name	**Height**
Display Name	**Height**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Expert**
Allowed values	**Minimum 2** **Maximum 65536** **Stepsize 1**
Default value	**1024**
Unit of measure	**pixel**

Example 5.2. Usage of Height

```
/* Set */ Height = 1024;
/* Get */ value_ = Height;
```

5.3. OffsetX

The x-offset is defined by this parameter. If a horizontal mirroring is active the sensor width limits the maximum width (Width + XOffset). If furthermore vertical mirroring is active the maximum width is limited by the DRAM and the sensor height (the sensor dimension needs to fit into the DRAM).

Table 5.3. Parameter properties of OffsetX

Property	Value
Name	**OffsetX**
Display Name	**Offset X**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 32720** **Stepsize 16**
Default value	**0**
Unit of measure	**pixel**

Example 5.3. Usage of OffsetX

```
/* Set */ OffsetX = 0;
/* Get */ value_ = OffsetX;
```

5.4. OffsetY

The y-offset is defined by this parameter. If a vertical mirroring is active the sensor height limits the maximum height (Height + YOffset). Furthermore the maximum height is limited by the DRAM and the sensor width (the sensor dimension needs to fit into the DRAM).

Table 5.4. Parameter properties of OffsetY

Property	Value
Name	**OffsetY**
Display Name	**Offset Y**
Interface	**IInteger**
Access policy	**Read/Write**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 65535** **Stepsize 1**
Default value	**0**
Unit of measure	**pixel**

Example 5.4. Usage of OffsetY

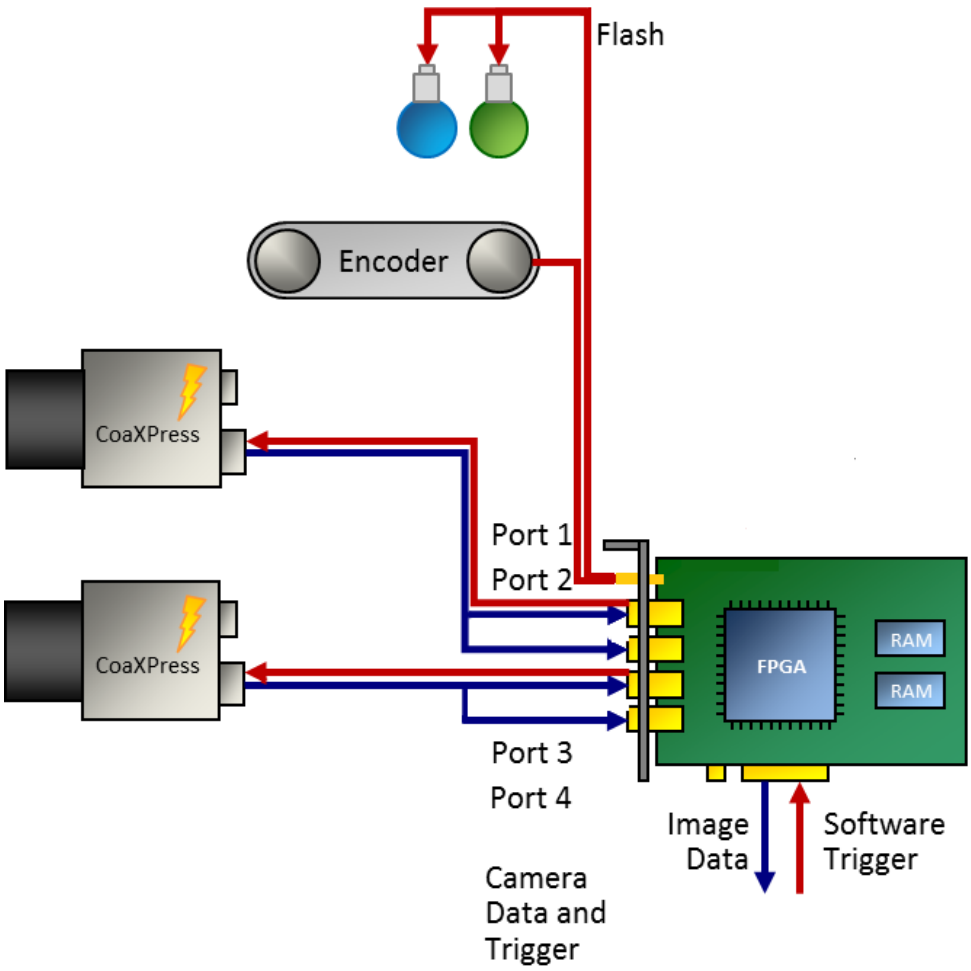
```
/* Set */ OffsetY = 0;
/* Get */ value_ = OffsetY;
```

Chapter 6. Trigger

The area trigger system enables the control of the image acquisition process of the interface card and the connected cameras. In detail it controls the exact exposure time of the camera and controls external devices. The trigger source can be external devices, internal frequency generators or the user's software application.

The CXP-12 Interface Card 4C interface card has 4 inputs on the front IO connector. Check the hardware documentation for more information. The CXP-12 Interface Card 4C generates the desired trigger outputs and control signals from the input events according to the trigger system's parameterization. The trigger system outputs can be routed to the camera via the CoaXPress link. Additionally, outputs can be routed to the digital outputs for control of external devices such as flash lights, for synchronizing or for debugging.

Figure 6.1. CXP-12 Interface Card 4C Trigger System



The following is an introduction into the Basler CXP-12 Interface Card 4C trigger system. Several trigger scenarios show the possibilities and functionalities and help to understand the trigger system. The documentation includes the parameter reference where all parameters of the trigger system are listed and their functionality is explained in detail.

6.1. Features and Functional Blocks of Area Trigger

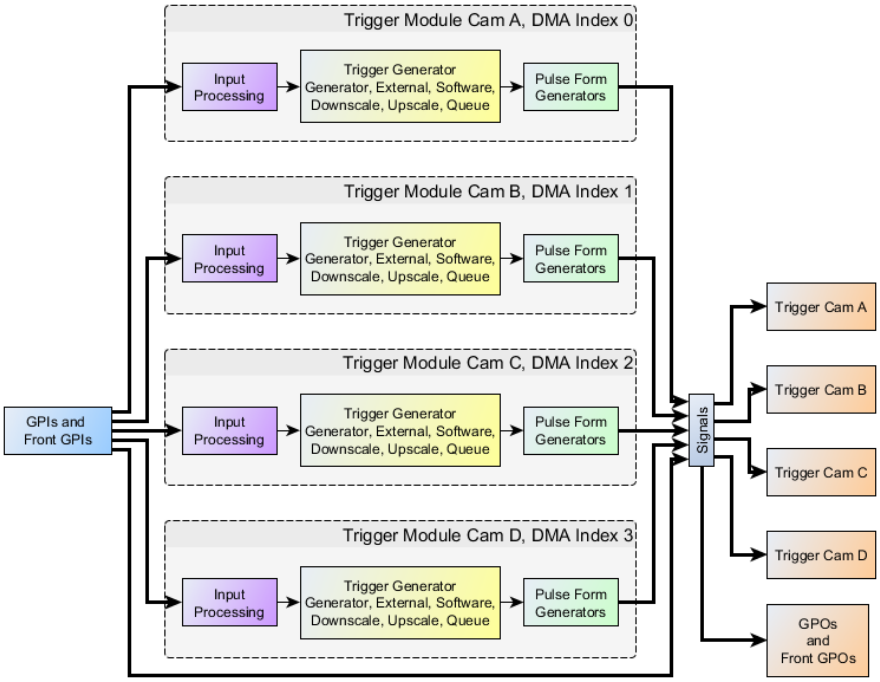
The Basler trigger system was designed to fulfill the requirements of various applications. Powerful features for trigger generation, controlling and monitoring were included in the implementation. This includes:

- Trigger signal generation for cameras and external devices.
- **External devices** such as encoders and light barriers can be used to source the trigger system and control the trigger signal generation.
- In alternative an internal **frequency generator** can be used to generate trigger pulses.
- The trigger signal generation can be fully controlled by **software** . Single pulses or sequences of pulses can be generated. The trigger system will automatically control and limit the output frequency.
- Input **signal monitoring** .
- Input signal **frequency analysis** and **pulse counting** .
- Input signal **debouncing**
- Input signal **downscaling**
- **Pulse multiplication** using a sequencer and controllable maximum output frequency. Make up to 65,000 output pulses out of a single input pulse.
- **Trigger pulse queue** for buffering up to 2000 pulses and control the output using a maximum frequency valve.
- Four **pulse form generators** for individual controlling of pulse widths, delays and output downscaling.
- **Up to 10 outputs depending on the interface card type** plus the CoaXPress trigger outputs.
- A **bypass** option to keep the pulse forms of the input signals and forward them to outputs and cameras.
- **Event generation** for input and output monitoring by application software.
- Trigger state events for fill level monitoring, trigger busy states and lost trigger signals give full control of the system.
- Camera **frame loss notification** .
- Full **trigger signal reliability** and easy error detections.

The trigger system is controlled and configured using parameters. Several read only parameters return status information on the current trigger state. Moreover, the trigger system is capable of generating events for efficient monitoring and controlling of the trigger system, the software, the interface card and external hardware.

The complex trigger system can be easily used and parameterized. The following block diagram figure shows an overview of the trigger system. As can be seen, the trigger system consists of four different main functional blocks.

Figure 6.2. Trigger System

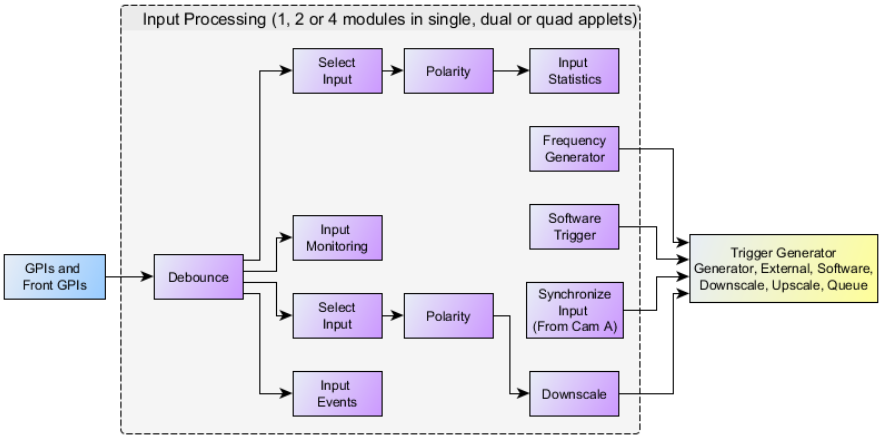


1. Trigger Input:

   Trigger inputs can be external signals, as well as software generated inputs and the frequency generator. An input monitoring and input statistics module allows analysis if the input signals.

   External input signals are debounced and split into several paths for monitoring, and further processing.

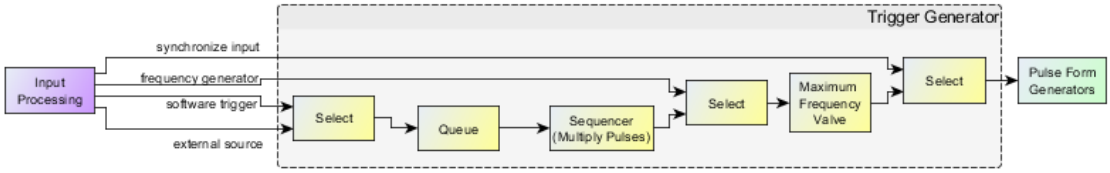
   Figure 6.3. Trigger Input Block Diagram



2. Input Pulse Processing:

   The second main block of the trigger system is the Input Pulse Processing. External inputs as well as software trigger generated pulses can be queued and multiplied in a sequencer if desired. All external trigger pulses are processed in a maximum frequency valve. Pulses are only processed by this valve if their frequency is higher than the previously parameterized limit. If a higher frequency is present at the input, pulses will be rejected or the trigger pulse queue is filled if activated. The maximum frequency valve ensures that the output-pulses will not exceed the maximum possible frequency which can be processed by the camera.

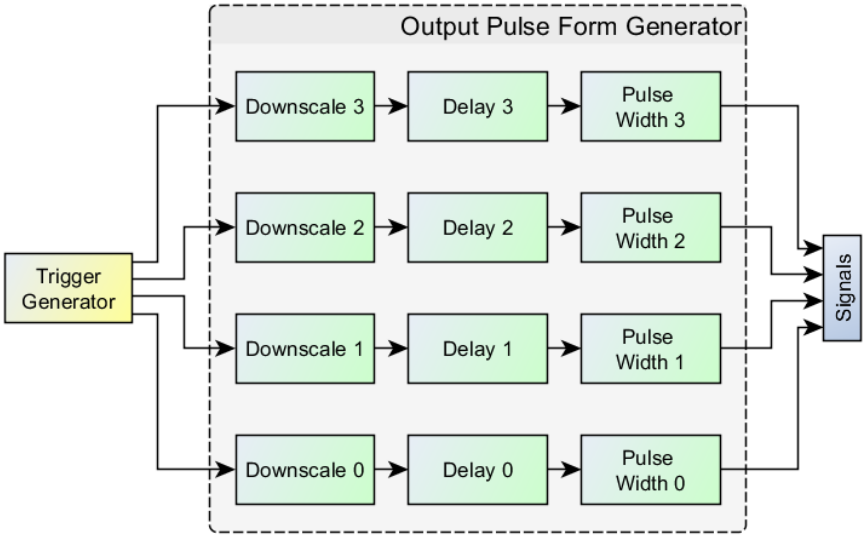
Figure 6.4. Trigger Pulse Processing Block Diagram



3. Output Pulse Form Generators:

   After the input pulses have been processed, they are feed into four optional pulse form generators. These pulse form generators define the signal width, a delay and a possible downscale. The four pulse form generators can arbitrarily allocated to the outputs which makes the trigger system capable for numerous applications such as muliple flash light control, varying camera exposure times etc.

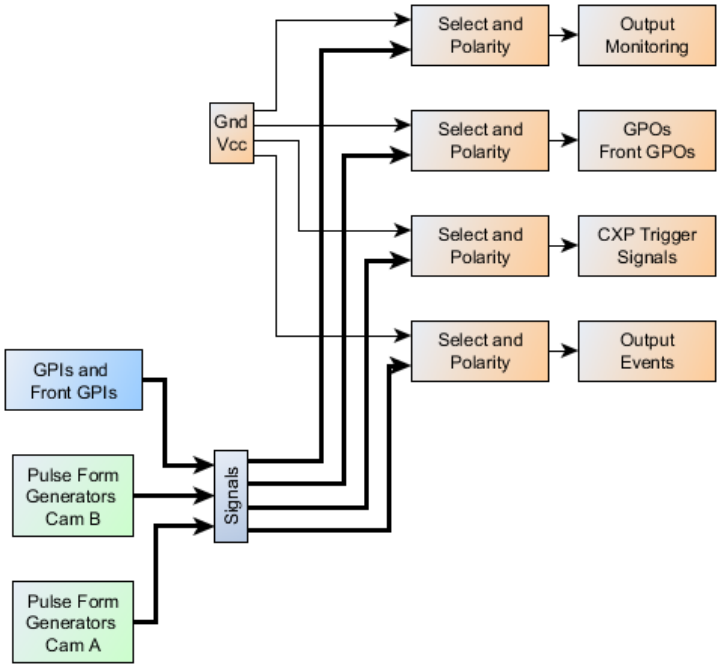
Figure 6.5. Trigger Pulse Processing Block Diagram



4. Trigger Output:

   The last block is related to the trigger outputs. The pulse form generator signals can be output at the digital outputs and directly to the camera. Moreover, they can be monitored using registers and events.

Figure 6.6. Trigger Output Block Diagram



6.2. Digital Input/Output Mapping

The CXP-12 Interface Card 4C supports four digital front inputs. It has two front trigger outputs.

The four front inputs have the indices 0 to 3. In the documentation of the trigger IO boards and CXP-12 Interface Card 4C the allocation of these inputs to pins is described.

The available outputs can arbitrarily allocated to a trigger module or directly to a GPI.. See Section 6.5.12, 'DigitalOutput' for explanation.

6.3. Event Overview

In programming or runtime environments, a callback function is a piece of executable code that is passed as an argument, which is expected to call back (execute) exactly that time an event is triggered.

For a general explanation on events see Event.

In the following, a list of all events of the trigger system is presented. Detailed explanations can be found in the respective module descriptions. The events are available for all cameras. Replace CAM0 by the respective camera index if necessary.

- *LineFront0RisingEdge*, *LineFront0FallingEdge* to *LineFront3RisingEdge*, *LineFront3FallingEdge*

  Trigger input events on the Front GPIs. Events can be generated for all digital trigger inputs. The events are triggered by either rising or falling signal edges.

- *TriggerExceededPeriodLimits*

  The event is generated for each lost input trigger pulse.

- *TriggerQueueFilllevelThresholdOn* and *TriggerQueueFilllevelThresholdOff*

This event is generated when the trigger queue exceeds the upper ON-threshold or falls below the OFF-threshold.

- *AcquisitionTrigger*

  Events for trigger output.

- *FrameTriggerMissed*

  The event is generated for a missing camera frame response.

6.4. Trigger Scenarios

In the following, trigger sample scenarios are presented. These scenarios will help you to use the trigger system and facilitate easy adaptation to own requirements.

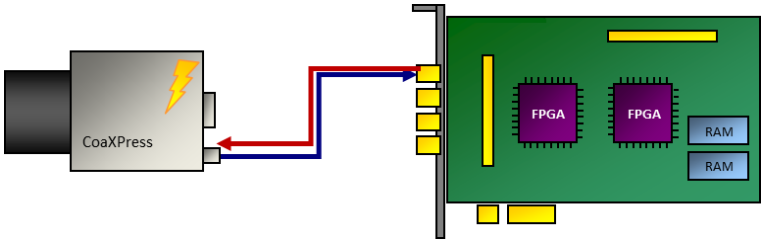
The scenarios show real life configurations. They explain the requirements, illustrate the inputs and outputs and list the required parameters and their values.

6.4.1. Internal Frequency Generator / interface card Controlled

Let's start the trigger system examples with a simple scenario. In this case we simply want to control the frequency of the camera's image output and the exposure time with the interface card. Assume that there is no additional external source for trigger events and we do not need to control any flash lights. Thus the interface card's trigger system has to control the frequency of the trigger pulses and the exposure time.

Figure 6.7 shows the hardware setup. Only the camera connected to the interface card is required.

Figure 6.7. Generator Controlled Trigger Scenario

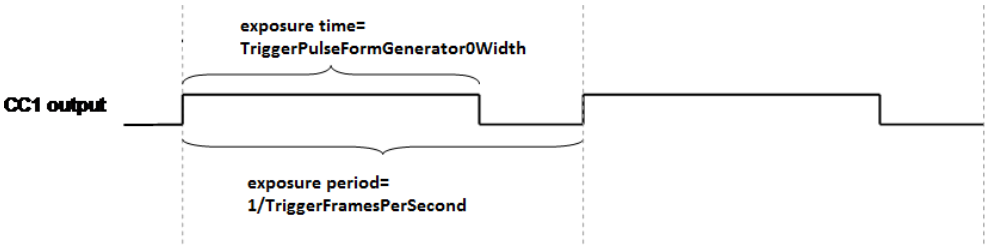


To put this scenario into practice, you will need to set your camera into an external trigger mode. Consult the vendor's user manual for more information.

After the camera is set to an external trigger mode, the exposure period and the exposure time can be controlled by one of the camera control inputs. Use the CXP cable as trigger source. The names of the camera trigger modes vary. You will need to use an external trigger mode, where the exposure period is programmable. If you also want to define the exposure time using the interface card, the respective trigger mode needs to support this, too.

In the following, a waveform is shown which illustrates the interface card trigger output. Most cameras will start the acquisition on the rising or falling edge of the signal. The exposure time is defined by the length of the signal. Note that some cameras use inverted inputs. In this case, the signal has to be 'low active' instead of being 'high active'. Thus the interface card output has to be inverted which is explained later on.

Figure 6.8. Waveform of Generator Controlled Trigger Scenario



After hardware setup and camera configuration we can start parameterizing the interface card's trigger system.

In the following, all required parameters and their values are listed.

- *AreaTriggerMode* = **Generator**

  First, we will need to configure the trigger system to use the internal frequency generator.

- *TriggerOutputFrequency* = 10

  Next, the output frequency is defined. In this example, we use a frequency of 10Hz.

- *TriggerPulseFormGenerator0Width* = 200

  So far, we have set the trigger system to generate trigger pulses at a rate of 10Hz. However, we have not set the pulse form of these pulses i.e. the signal length or signal width. The interface card's trigger system includes four pulse form generators which allow to set the signal width, a delay and a downscaling. In our example, we only have one output and therefore, we will need only one pulse form generator, respectively pulse form generator 0. Moreover, only the signal length has to be defined, a delay and a downscaling is not required.

  Suppose, that we require an exposure time of 200µs. Thus, we will set the parameter to value 200 since the unit is µs.

- *CxpLinkTrigger0Source* = **PulseGeneratorRisingEdge** and *CxpLinkTrigger1Source* = **PulseGenerator0FallingEdge**

  The only thing left to do is to allocate the output of pulse form generator 0 to the camera trigger output.

Now, the trigger is fully configured. However the trigger signal generation is not started yet. Set parameter *TriggerState* to **Active** to start the system. Of course, you will also need to start your image acquisition. It is up to you if you like to start the trigger generation prior or after the acquisition has been started. If the trigger system is started first, the camera will already send images to the interface card. These images are discarded as no acquisition is started.

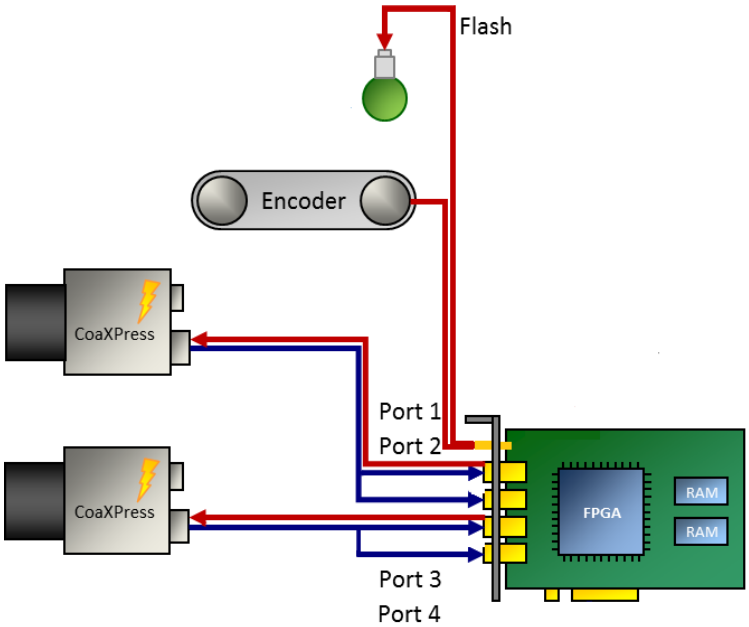
You will now receive images from your camera. Change the frequency and the signal width to see the influence of these parameters. A higher frequency will give you a higher frame rater. A shorter exposure time will make the images 'darker'. You will realize, that it is not possible to set an exposure time which is longer than the exposure period. In this case, writing to the parameter will result in an error. Therefore, the order of changing parameter values might be of importance. Also be careful to not select a frequency or exposure time which exceeds the camera's specifications. In this cases you will loose trigger pulses, as the camera cannot progress them. Get the maximum ranges from the camera's specification sheets.

To stop the trigger pulse generation, set parameter *TriggerState* to **SyncStop**. The trigger system will then finalize the current pulse and stop any further output until the system is activated again. The asynchronous stop mode is not required in this scenario.

6.4.2. External Trigger Signals / IO Triggered

In the previous example we used an internal frequency generator to control the camera's exposure. In this scenario, an external source will define the exact moment of exposure. This can be, for example, a light barrier as illustrated in the following figure. Objects move in front of the camera, a light barrier will define the moment, when an object is located directly under the camera. In practice, it might not be possible to locate the light barrier and the camera at the exact position. Therefore, a delay is required which delays the pulses from the light barrier before using them to trigger the camera. Moreover, in our scenario, we assume that a flash light has to be controlled by the trigger system, too.

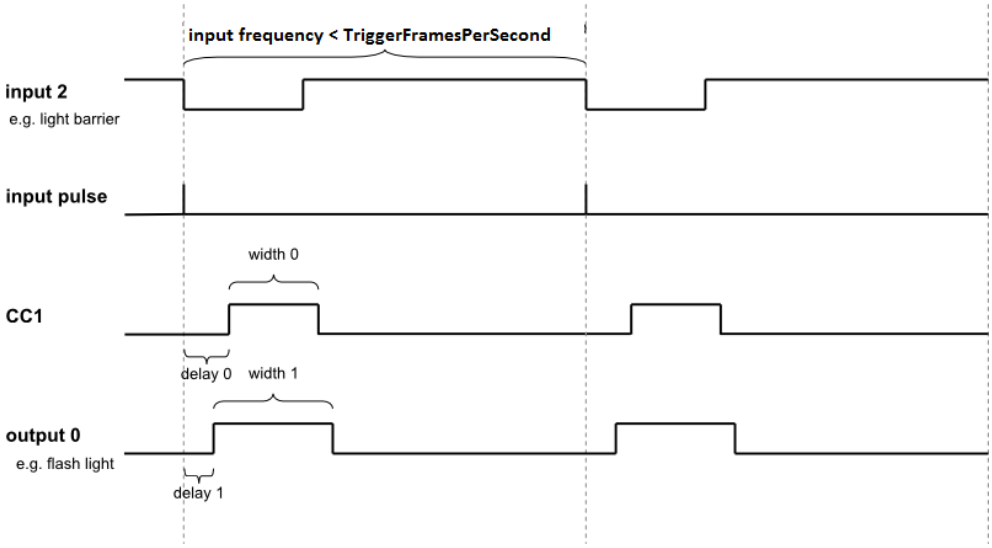
Figure 6.9. External Controlled Trigger Scenario



An exemplary waveform (Figure 6.10) provides information on the input signal and shows the desired output signals. The input is shown on top. As you can see, the falling edge of the signal defines the moment which is used for trigger generation. Thus, the signal is 'low active'. Mind that the pulse length of any external input is ignored (second row), only falling edges are considered.

The output to the camera is shown in the third row. Here we can see an inserted delay. This delay will compensate the positions of the light barrier and the camera. The signal width at the trigger camera output defines the exposure time, if the camera is configured to the respective trigger mode. Control of the flash light is done using trigger output 0. Again, a delay is added. Depending on the requirements of the flash light, this delay has to be shorter or longer than the trigger camera output delay. Similarly, the required pulse length varies for different hardware.

Figure 6.10. Waveform of External Trigger Scenario



Before parameterizing the applet, ensure that your camera has been set to an external trigger mode. Check the previous trigger scenario for more explanations.

In this example, we have to parameterize the trigger mode, the input source and we have to configure two trigger outputs.

- *AreaTriggerMode* = **External**

  In external trigger mode, the trigger system will not use the internal frequency generator. External pulses control the output of trigger signals. This requires the selection of an input source and the configuration of the input polarity.

- *TriggerInSource* = **TriggerInSourceFrontGPI2**

  Select the trigger input by use of this parameter. You can choose any of the inputs. If you use a multi-camera applet, cameras can share same sources.

- *TriggerInPolarity* = **LowActive**

  For the given scenario, we assume that a trigger is required on a falling edge of the input signal.

- *TriggerOutputFrequency* = 500

  Do not forget to set this parameter. For any use of the trigger system, the correct parameterization of this parameter is required. If you do not use the internal frequency generator, this parameter defines the maximum allowed trigger pulse frequency. In other words, you can set a limit with this parameter. The limiting frequency could be the maximum exposure frequency of the camera.

  The advantage of setting this limit is the information on lost trigger signals. Let's suppose the frequency of the external trigger signals will get to high for the camera or the applet. In this case, you will loose images or obtain corrupted images. If you have set a correct frequency limit in the trigger system, the trigger system will provide you with information of these exceeding line periods. This information can be obtained by register polling or you can use the event system. Thus you always have the possibility to prevent your application of getting into a bad, probably undefined state and you will always get the information of when and how many pulses got lost. Check the explanations of parameters *TriggerOutputFrequency* and *TriggerExceededPeriodLimits* as well as the event *TriggerExceededPeriodLimits* for more information.

  More information on error detection and analysis can be found in scenario Section 6.4.9, 'Hardware System Analysis and Error Detection / Trigger Debugging'

  The trigger system also allows the queuing of trigger pulses if you have a short period of excess pulses. We will have a look at this in a later scenario.

In our example, we set the maximum frequency to 500 frames per second. If you do not want to use this feature, set *TriggerOutputFrequency* to a high value, such as 1MHz.

- *TriggerPulseFormGenerator0Width* = 200

  So far, we have set the trigger system to accept external signals and generate the trigger pulses out of these signals. Next, we need to output these pulses. For realization, we need to define the pulse form of the output signals. Just as shown in the previous scenario, we use pulse form generator 0 for generating the pulse form of the trigger signals. We set a pulse width of 200µs.

- *TriggerPulseFormGenerator0Delay* = 50

  In addition to the signal width, a delay will give us the possibility to delay the output as the light barrier might not be positioned at the exact location. For this fictitious scenario we use a delay of 50µs.

- *TriggerPulseFormGenerator1Width* = 250

  In addition to the trigger output we want to control a flash light. We use pulse form generator 1 for this purpose and set the signal width to 250µs.

- *TriggerPulseFormGenerator1Delay* = 25

  A delay for the flash output is set, too.

- *CxpLinkTrigger0Source* = **PulseGeneratorRisingEdge** and *CxpLinkTrigger1Source* = **PulseGenerator0FallingEdge**

  Finally, we have to allocate the camera trigger output with the pulse form generator 0.

- *TriggerOutSelectFrontGPO0* = **PulseGenerator1**

  The flash light, connected to output 0 has to be allocated to pulse form generator 1.

- *TriggerOutSelectFrontGPO1* = **PulseGenerator0**

  Let's assume that it is necessary to measure the camera trigger output using a logic analyzer. Hence, we allocate output 1 to pulse form generator 0 as well.

The trigger is now fully configured. Just as described in the previous scenario, you can now start the acquisition and activate the trigger system using parameter *TriggerState*.

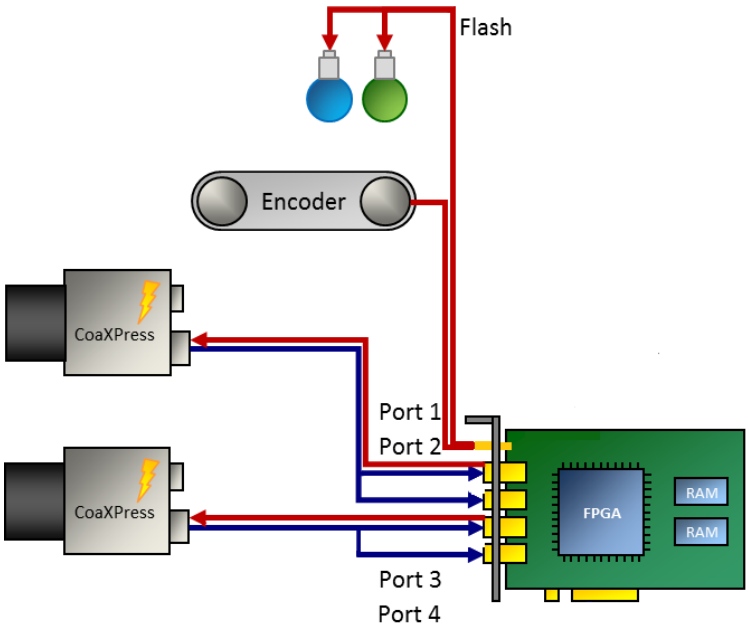
You will now receive images from the camera for each external trigger pulse. Compare the number of external pulses with the generated trigger signals and the received images for verification. Use parameter *TriggerInStatisticsPulseCount* of category Trigger Input -> Input Statistics and parameter *TriggerOutStatisticsPulseCount* of the output statistics parameters to get the number of input pulses and generated pulses. You can compare these values with the received image numbers.

6.4.3. Control of Two Flash Lights

This scenario is similar to the previous one. We use an external trigger to control the camera and a flash light. But in difference, we want to get three images from one external trigger pulse. Images one and three out of the sequence of three images have to use the first light source and image two has to use the second light source. Thus, in this scenario we will learn on how to use a trigger pulse multiplication and on how to control two lights connected to the interface card.

The application idea behind this scenario is that an object is acquired using different light sources. This could result in a HDR image or switching between normal and infrared illumination. The following figure illustrates the hardware setup. As you can see, we have two light sources this time. The objects move in front of the camera. The light barrier will provide the information on when to trigger the camera. Let's suppose that the objects stop in front of the camera or the movement is slow enough to generate two images with the different illuminations.

Figure 6.11. External Controlled Trigger Scenario



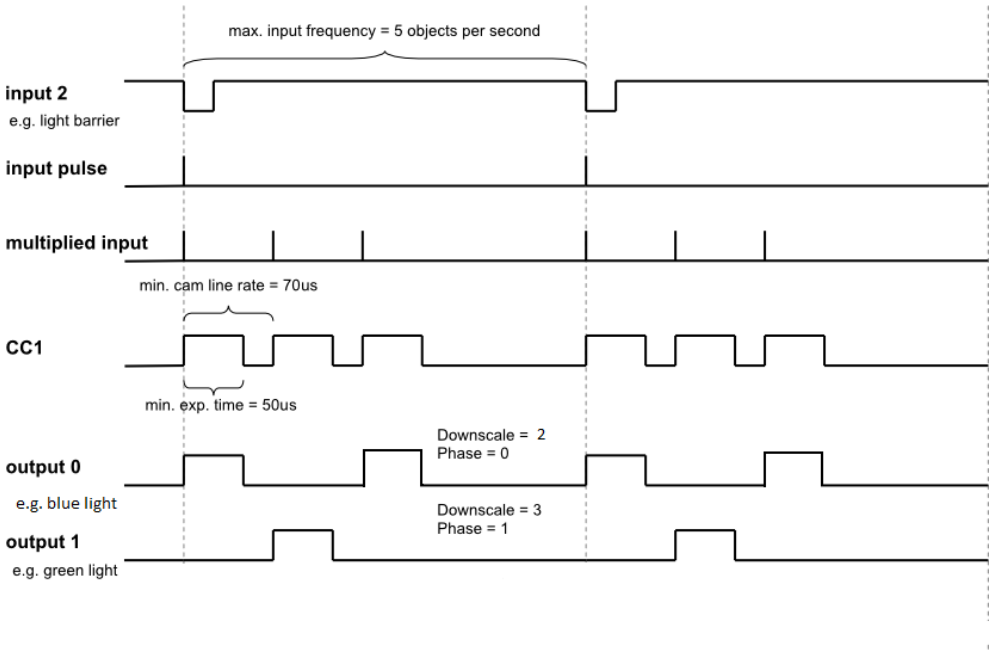
Before looking at the waveform, let's have a look at our fictitious hardware specifications.

Table 6.1. Fictitious Hardware Specifications of Trigger Scenario Three Light Sources

Element	Limit
Object Speed	Max. 100 Objects per Second
Minimum Camera Exposure Time	50µs
Minimum Camera Frame Period	70µs

The object speed is 100 objects per second. The minimum camera exposure time is 50µs at a minimum camera frame period of 70µs. Thus we only need 210µs to acquire the three images. The following waveform shows the input and output signals, as well as the multiplied input signals. The first row shows the input. Each falling edge represents the light barrier event as marked in the second row. The third row shows the multiplied input pulses with a gap of 70µs between the pulses. The trigger signal is generated for each of these pulses, however the trigger flash outputs 0 and 1 are downscaled by two and three and a delay is added.

Figure 6.12. Waveform of External Trigger Scenario Controlling Two Flash Lights



Parameterization is similar to the previous example. In contrast, this time, we have to set the trigger pulse sequencer using a multiplication factor and we have to use the pulse form generators.

- *AreaTriggerMode* = **External**
- *TriggerInSource* = 2
- *TriggerInPolarity* = **LowActive**
- *TriggerMultiplyPulses* = 3

  The parameter specifies the multiplication factor of the sequencer. For each input pulse, we have to generate three internal pulses. The period time of this multiplication is defined by parameter *TriggerOutputFrequency*

- *TriggerOutputFrequency* = 14285

  This time, the maximum frames per second correspond to the gap between the multiplied trigger pulses. We need a gap of 70µs which results in a frequency of 14285Hz.

- *TriggerPulseFormGenerator0Width* = 50

  Again, we use pulse form generator 0 for trigger signal generation. The pulse width is 50µs. A delay or downscaling is not required.

- *TriggerPulseFormGenerator1Width* = 50

  The pulse width for the flash lights depends on the hardware used. We assume a width of 50µs in this example.

- *TriggerPulseFormGenerator2Width* = 50
- *TriggerPulseFormGenerator1Downscale* = 2
- *TriggerPulseFormGenerator2Downscale* = 3
- *TriggerPulseFormGenerator1DownscalePhase* = 0

  We use the phase shift for delaying the downscaled signals of the outputs. You could use the delay instead, but any frequency change will require a change of the delay as well. The phase shift of pulse form generator 1 i.e. the first flash light is 0.

- *TriggerPulseFormGenerator2DownscalePhase* = 1

  The phase shift of pulse form generator 2 i.e. the second flash light is 1.

- *CxpLinkTrigger0Source* = **PulseGeneratorRisingEdge** and *CxpLinkTrigger1Source* = **PulseGenerator0FallingEdge**

  The output allocation is as usual.

- *TriggerOutSelectFrontGPO0* = **PulseGenerator1**

- *TriggerOutSelectFrontGPO1* = **PulseGenerator2**

Start the trigger system using parameter *TriggerState* as usual. You will notice that you get thrice the number of images from the interface card than external trigger pulses have been generated by the light barrier. Equally to the previous example, check for exceeding line periods at the input when you run your application or ensure that your external hardware will not generate the input pulses with an exceeding frequency.

Keep in mind to start the acquisition before activating the trigger system. This is because you will receive three images for one external trigger pulse. If you start the acquisition after the trigger system, you cannot ensure that the first transfered image is the first image out of a sequence.

6.4.4. Software Trigger

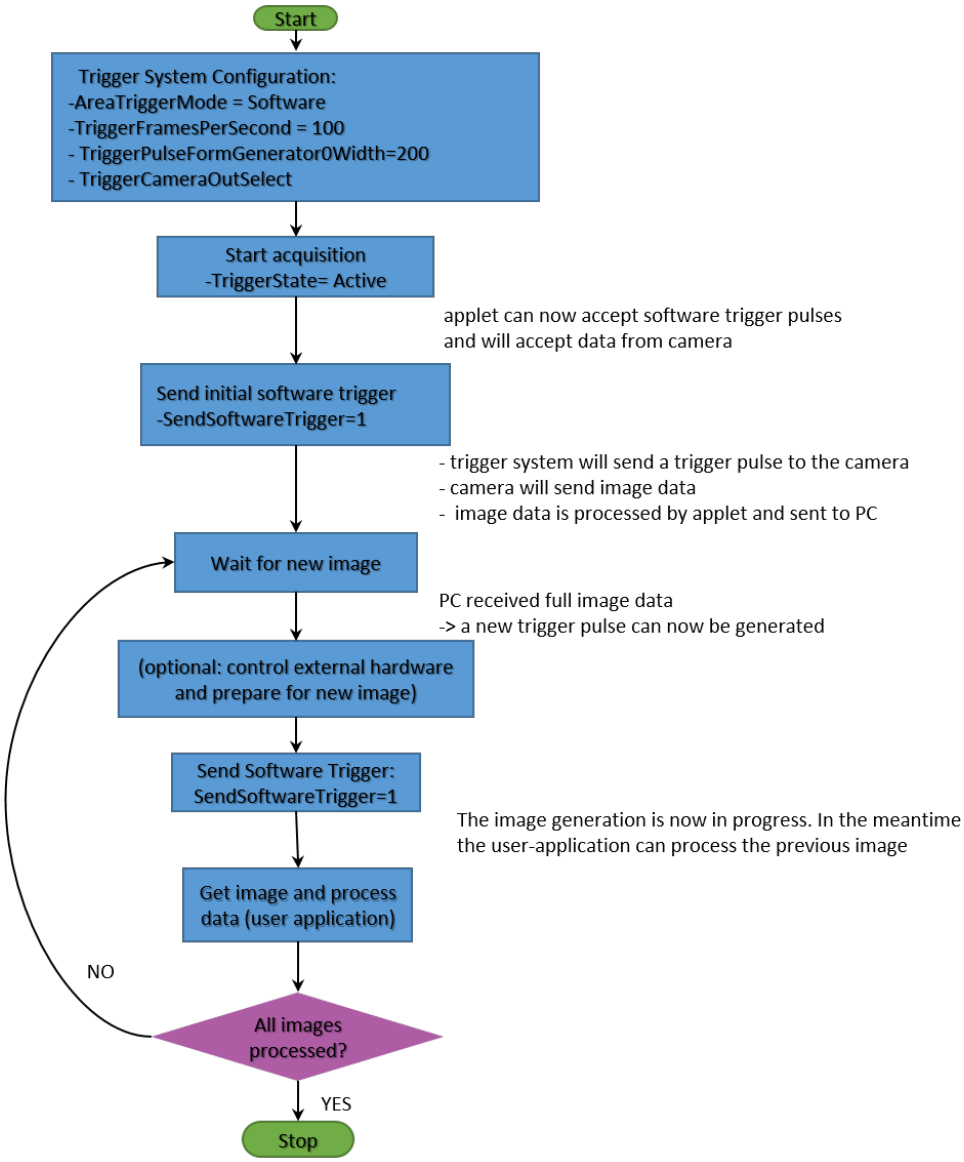
The previous examples showed the use of the internal frequency generator and the use of external trigger pulses to trigger your camera and generate digital output signals. Another trigger mode is the software trigger. In this mode, you can control the generation of each trigger pulse using your software application. To use the software triggered mode, set parameter *AreaTriggerMode* to **Software**. Next, configure the pulse form generators and the outputs as usual and start the trigger system (set *TriggerState* to **Active**) and the acquisition. Now, you can generate a trigger pulse by writing value '1' to parameter *SendSoftwareTrigger* i.e. each time you write to this parameter, a trigger pulse is generated. The relevant blocks of the trigger system are illustrated in the following figure.

Keep in mind that the time between two pulses has to be larger than 1 / *TriggerOutputFrequency* as this will limit the maximum trigger frequency. The trigger system offers the possibility to check if a new software trigger pulse can be accepted i.e. the trigger system is not busy anymore. Read parameter *SoftwareTriggerIsBusy* to check it's state. While the parameter has value **Busy**, writing to parameter *SendSoftwareTrigger* is not allowed and will be ignored. You should always check if the system is not busy before writing a pulse. To check if you lost a pulse, read parameter *TriggerExceededPeriodLimits*.

In some cases, you might want to generate a sequence of pulses for each software trigger. To do this, simply set parameter *TriggerMultiplyPulses* to the desired sequence length. Now, for every software trigger pulse written to the trigger system, a sequence of the define length with a frequency defined by parameter *TriggerOutputFrequency* is generated. Again, the system cannot accept further inputs while a sequence is being processed.

Let's have a look at some flow chart examples on how to use the trigger system in software triggered mode. The flow charts visualize the steps of a fictitious user software implementation. In the first example, we simply generate single software trigger pulses using parameter *SendSoftwareTrigger*. When the applet receives this pulse, it will trigger the camera. The camera will send an image to the interface card which will be processed there and will be output to the PC via DMA transfer. In the meantime, the users software application will wait for any DMA transfers. After the application got the notification that a new image has been fully tranferred to the PC it will send a new software trigger pulse and the interface card and camera will start again generating an image. Our software application will now have the time to process the previously received image until it is waiting for a new transfer. Thus, the software can process images while image generation is in progress. Of course, you can first process your images and afterwards generate a new trigger pulse, as well. So the steps for a repeating sequence are: Generate a SW trigger pulse, wait for image, generate a SW trigger pulse, wait for image. The flowchart of this example can be found in the following figure.

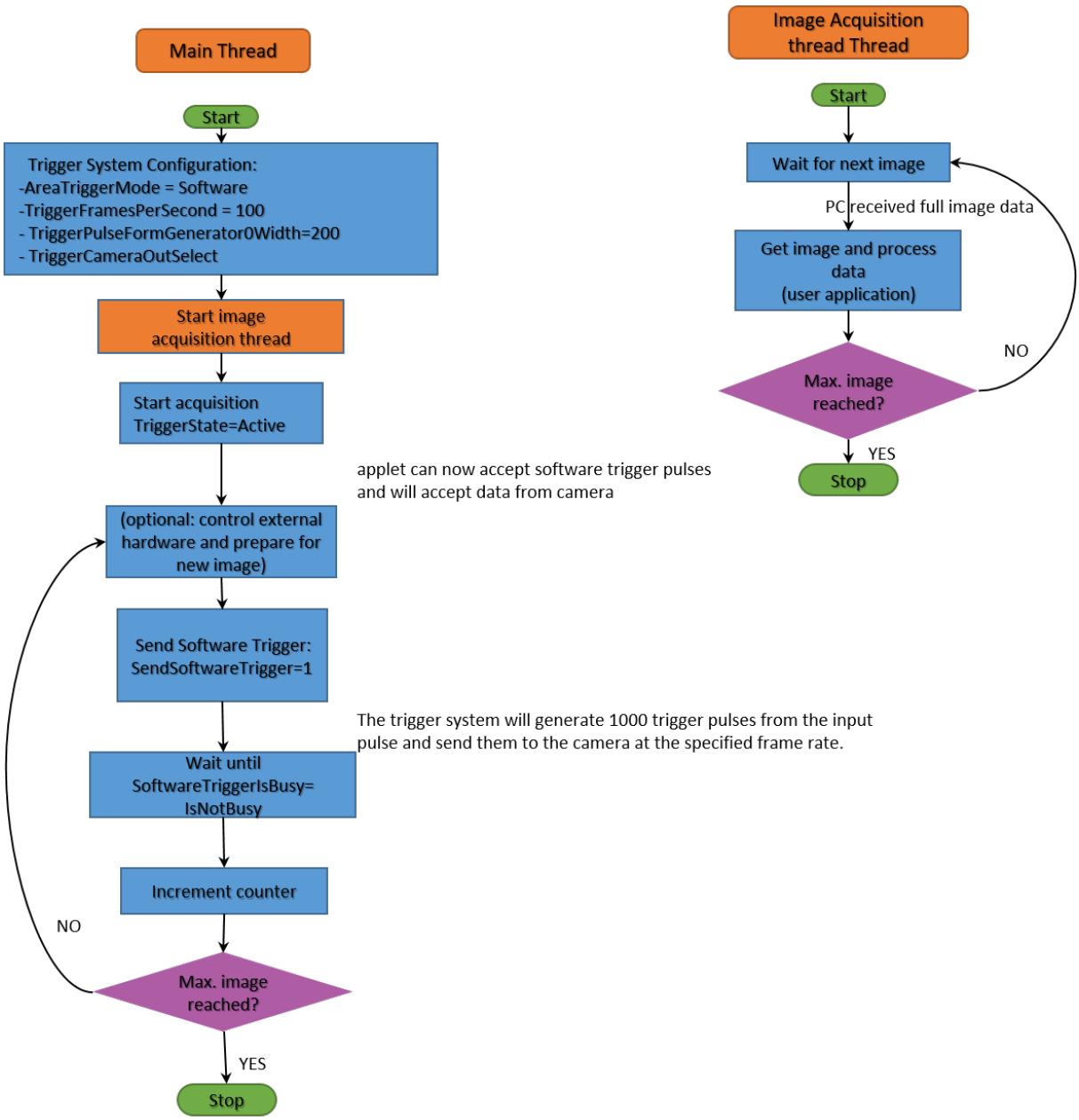
Figure 6.13. Flowchart of Software Application Using the Software Trigger



In the sample application shown above, it is ensured that the trigger system is not busy after you received the image. Therefore, we do not need to check for the software trigger busy flag in this example. One drawback of the example is that we might not acquire the frames at the maximum speed. This is because we have to wait for the full transfer of images before generating a new trigger pulse. Cameras can accept new trigger pulses while they transfer image data. The next example will therefore use the trigger sequencer.

The next example uses two threads. One thread for trigger generation and one thread for image acquisition and processing. In comparison to the previous example, we use the trigger sequencer for pulse multiplication and we will have to use the busy flag. This will allow an acquisition at a higher frame rate.

Figure 6.14. Flowchart of Software Application Using the Software Trigger with a Sequencer



The main thread will configure and start the trigger system and the acquisition. For each software trigger pulse we send to the interface card, 1000 pulses are generated and send to the camera at the framerate specified by *TriggerOutputFrequency*. After sending a software trigger pulse to the interface card we wait until the software is not busy anymore by polling on register *SoftwareTriggerIsBusy*. To control the number of generated trigger pulses we count each successful sequence generation. If more images are required we can send another software trigger pulse to the interface card to start a new sequence.

The second thread is used for image acquisition and image data processing. Here, the software will wait for new incoming images (Use function Fg_getLastPicNumberBlockingEx() for example) and process the received images. The thread can exit if the desired number of images have been acquired and processed.

6.4.5. Software Trigger with Trigger Queue

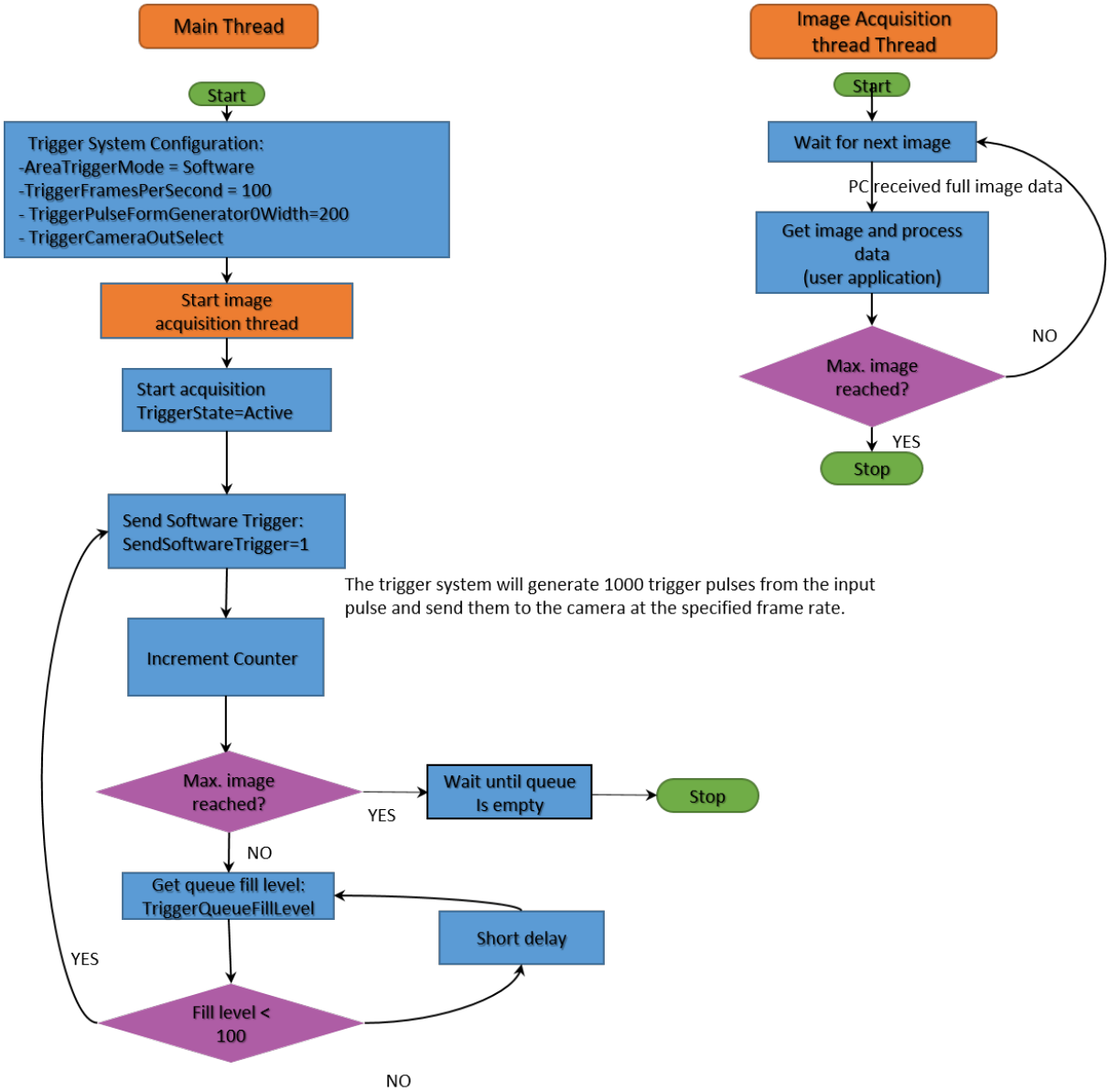
To understand the following scenario you should have read the previous scenario first. In the following we will have a look at the software trigger once again. This time, we use the trigger queue. The trigger queue enables the buffering of trigger pulses from external sources or from the software trigger and will output these

pulses at the maximum allowed frequency specified by *TriggerOutputFrequency*. Therefore, we can write to *SendSoftwareTrigger* multiple times even if the trigger system is still busy. Parameter *SoftwareTriggerIsBusy* will only have value **Busy** if the queue is full. Instead of writing multiple times to *SendSoftwareTrigger* you can directly write the number of required pulses to the parameter.

The trigger queue can buffer 2040 sequence pulses. Thus if you have a certain sequence length of N pulses and currently 200 pulses in the queue, the trigger system can store additional 1840 remaining pulses. You can check the fill level by reading parameter *TriggerQueueFillLevel*.

In the following flow chart you can see a queue fill level minimum limit of 10 pulses. In our supposed application we will check the queue fill level and compare it with our limit. If less pulses are in the queue, we generate a new software trigger pulse. Thus, on startup, the queue will fill-up until it contains 10 pulses. We count the software trigger pulses send to the trigger system. Multiplied with our sequence length, we can obtain the number of pulses which will be send to the camera. If enough pulses have been generated, we can stop the trigger pulse generation.

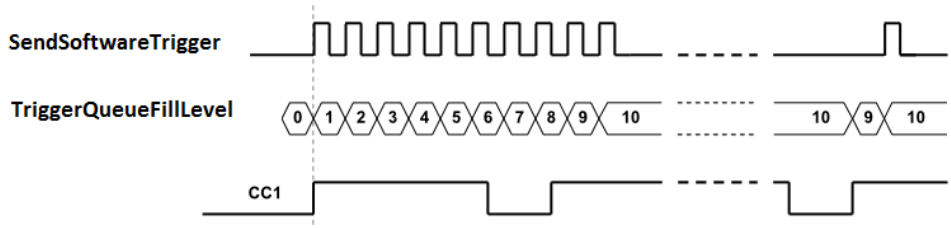
Figure 6.15. Flowchart of Software Application Using the Software Trigger with Trigger Queue



When having a look at the waveform (Figure 6.16) we can see the initialization phase where the queue is filled. After fill level value 10 has been reached, no more software trigger pulses are written to the applet. The system will now continue the output of trigger pulses. As our sequence length is 1000 pulses we have to wait for 1000 pulses to be generated until a change in the fill level will occur. After the 1000th pulse has been completely

generated, the fill level will change to 9. This will cause the generation of another software trigger pulse by our sample application which will cause a fill level of 10 again.

Figure 6.16. Waveform Illustrating Software Trigger with Queue Example"



When using the trigger queue, the stopping of the trigger system is of interest. If you set parameter *TriggerState* to **SyncStop**, the trigger system will stop accepting inputs such as software trigger pulses, but it will complete the trigger pulse generation until the queue is empty and all pulses are fully output. You can immediately cancel the pulse generation by setting the *TriggerState* to **AsyncStop**.

6.4.6. External Trigger with Trigger Queue

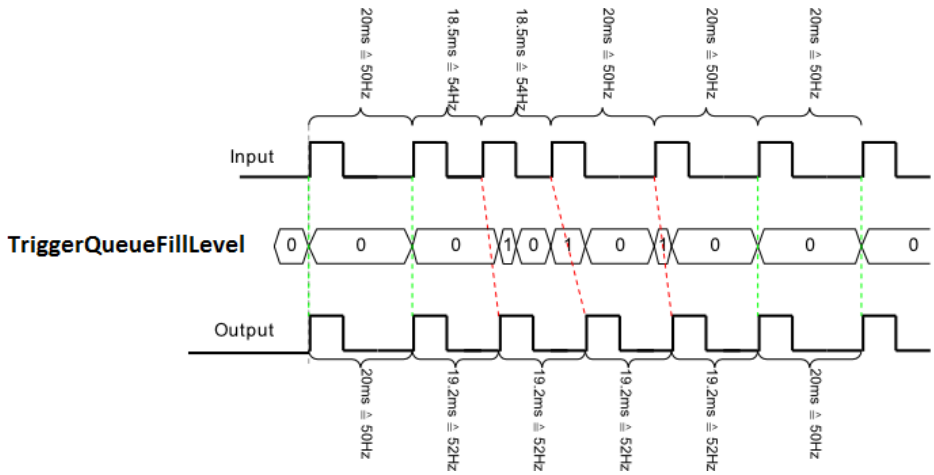
Of course, we can use the trigger queue with external triggers, too. This will give us a possibility to buffer 'jumpy' external encoders or any other external trigger signal generators. Let's suppose an external encoder which is configured to generate trigger pulses with a frequency of 50Hz and a camera which can be run at a maximum frequency of 52Hz. Thus, we set parameter *TriggerOutputFrequency* to 52Hz. Now assume that the external hardware is a little 'jumpy' and the 50Hz is just an average. So if we have inputs with a frequency higher than 52Hz we will loose at least one pulse. You can check this using the trigger lost events or by reading parameter *TriggerExceededPeriodLimits*.

Now let's have a look at the same scenario if the queue is enabled. If it is enabled, we can buffer trigger pulses. Thus, we can buffer the exceeding input frequency and output the pulses at the maximum camera trigger frequency which is 52Hz in our example. After the input frequency is reduced, the queue will get empty and the pulse output is synchronous to the input again. Note that the delay might result in images with wrong content such as 'shifted' object positions.

To enable the queue, just write value **On** to *TriggerQueueMode*.

The following waveform illustrates the input signal, the queue fill level and the output signal. At the beginning, the gap between the first two input signals is 20ms i.e. the frequency is less than 52Hz. Thus, the queue will not fill with pulses and the trigger system will directly output the second pulse. Now, the gap between the second and the third as well as the fourth pulse is less than 19.2ms and therefore, the trigger system will delay the output of these pulses to have a minimum gap of 19.2ms. During this period, the queue fill level will increment to value 1 for short periods. The gap between the fourth and the following input pulses is sufficiently long enough, however, the system will have to delay these pulses, too.

Figure 6.17. Using External Trigger Signal Sources together with the Trigger Queue



Note that the trigger lost event and *TriggerExceededPeriodLimits* will only be set if the queue is full i.e. in overflow condition.

6.4.7. Bypass External Trigger Signals

When external trigger signals are used, the duty cycle i.e. signal width or signal length will always be ignored. Only the rising or falling edge depending on the polarity settings is considered. However, you can bypass an external source directly to an output. For example, you can bypass an external source to the camera which allows you to control the exposure time with the external source. Mind that you will bypass the trigger core system and therefore, no frequency checks or downscales can be performed.

Use the output select parameters for camera control or digital outputs to select a bypass source. These are for example:

- *CxpLinkTrigger0Source* = BYPASS_FRONT_GPI_0_RISING and *CxpLinkTrigger1Source* = BYPASS_FRONT_GPI_0_FALLING
- *TriggerOutSelectFrontGPO0* = BYPASS_FRONT_GPI_1

6.4.8. Multi Camera Applications / Synchronized Cameras

A basic application is that multiple cameras at one or more interface cards are connected to the same trigger source. If all cameras have to acquire images for every trigger pulse. Simply connect the trigger source to all interface cards and set the same trigger configuration for all cameras. This applet supports up to two cameras. Set the same parameters for both cameras. Multiple trigger systems are allowed to share the same trigger input, so you do not have to connect your trigger source to two inputs.

If you do not have an external trigger source, but use the generator or the software trigger you can synchronize the triggers to ensure camera exposures at the same moment. Simply output the camera control signal on a digital trigger output and connect this output to a digital input of other interface cards which have to be synchronized with the master. In the slave applets bypass the input to the camera control outputs. In addition to that, this applet includes a special trigger mode called **Synchronized**. This mode can be chosen for the second camera. The second camera uses the trigger pulses at the output of the first camera as input source. However, users will still have to configure the pulse form generators in the trigger system of the second camera and will still have to allocate them to the trigger outputs.

Arbitrary Output Allocation

In multiple camera applets you can also select another camera trigger module source. For example, CXP trigger source for camera 1 can use **CamAPulseGenerator0**.

6.4.9. Hardware System Analysis and Error Detection / Trigger Debugging

The Basler trigger system includes powerful monitoring possibilities. They allow a convenient and efficient system analysis and will help you to detect errors in your hardware setup and wrong parameterizations.

Let's have a look at the simple external trigger example once again. Assume that you have set up all devices and have fully configured the applet. You start the system and receive images. Unfortunately, the number of acquired images or the framerate is not as expected. This means, at some point trigger signals or frames got lost. To analyze the error, let's have a look at the monitoring applet registers.

- Trigger Input Statistics

  The parameters of the trigger input statistics category allow an analysis of the external trigger pulses. Parameter *TriggerInStatisticsFrequency* performs a continuous frequency measurement of the input signals. Compare this value with the expected trigger input frequency. If the measured frequency is much higher or

lower than the expected frequency, check your external hardware. Also check if the correct trigger input has been chosen by parameter *TriggerInSource* and if the pulse width of the input is long enough to be detected by the hardware.

To validate a constant input frequency, the trigger system will also show the maximum and minimum detected frequencies using parameters *TriggerInStatisticsMaximumFrequency* and *TriggerInStatisticsMinimumFrequency*. On startup, you will have a very low frequency as no external pulses might have been detected so far. Therefore, you have to clear the measurement using parameter *TriggerInStatisticsMinMaxFrequencyClear* first. If you detect an unwanted deviation from the expected values or the difference between the minimum and maximum frequency is comparably high, your external trigger generating hardware might be 'jumpy', skips pulses or is 'bouncing' which causes pulse multiplication. In this case, you might be able to compensate the problem using a higher debouncing value, set a lower maximum allowed frequency (see Section 6.4.2, 'External Trigger Signals / IO Triggered') or use the trigger queue (see Section 6.4.6, 'External Trigger with Trigger Queue').

Another feature of the input statistics module is the pulse counting. This feature can be used to compare the number of input pulses with the output pulses and acquired images. Read the pulse count value from parameter *TriggerInStatisticsPulseCount*. To ensure a synchronized counting of the input and any output pulses and images you should clear the pulse counter before generating external trigger inputs.

- Trigger Output Statistics

  A pulse counter is connected to the trigger output, too. Here you can select one of the pulse form generators using parameter *TriggerOutStatisticsSource* and read the value with parameter *TriggerOutStatisticsPulseCount*. Reset the pulse counter using *TriggerOutStatisticsPulseCountClear*.

  Use the pulse count value to compare it with the input pulse counter. If the values vary, pulses in the interface card have been discarded. This can happen if the input frequency is higher than the maximum allowed frequency specified by parameter *TriggerOutputFrequency*. If this happens, flag *TriggerExceededPeriodLimits* will be set. Moreover, if the pulse counter values dramatically differ, ensure that no trigger multiplication and/or downscaling has been set. Check parameters *TriggerInDownscale*, *TriggerMultiplyPulses* and the downscale parameters of the pulse form generators.

  It is also possible to count the input and output pulses with the input events and the output event *AcquisitionTrigger*.

- Camera Response Check

  Trigger pulses might get lost in the link to the camera or the trigger frequency is to high to be processed by the camera. In this case, the number of frames received by the interface card differs from the trigger pulses sent. For this error, the trigger system includes the missing camera frame response detection module. The module can detect missing frames and generate an event for each lost frame or set a register. Check Section 6.5.12.3, 'OutStatistics' for more information and usage.

- Acquired Image Compare

  Of course, it is also possible to count the number of acquired images i.e. the number of DMA transfers and compare them with the generated trigger pulses. If the values differ, you might have lost trigger pulses in the camera. In this case, check that the trigger frequency is not to high for the camera. Ensure that you do not run the applet in overflow state, where images can get lost in the applet. If the applet is run in overflow, check the maximum bandwidths of the applet. A smaller region of interest might solve the problems.

For every monitoring values, check the maximum and minimum ranges of the parameters. If pulse counters reached their maximum value, they will reset and start from zero.

6.5. Parameters

6.5.1. AreaTriggerMode

The area trigger system of this applet can be run in three different operation modes.

- Generator

  An internal frequency generator at a specified frequency will be used as trigger source. All digital trigger inputs and software trigger pulses will be ignored.

- External

  In this mode, one of the digital inputs is used as trigger source i.e. you can use an external source for trigger generation.

- Software

  In software triggered mode, you will need to manually generate the trigger input signals. This has to be done by writing to an applet parameter.

- Synchronized

  The synchronized mode is not available for the first camera. If the area trigger mode of a process (Process) is set to synchronized mode, the trigger source of the process will be the output of the previous process. For example, if the area trigger mode of process 1 is set to synchronized mode, the trigger system is sourced by the output of process 0.

  In Section 6.4.8, 'Multi Camera Applications / Synchronized Cameras' an example of the usage is presented. The block diagram in Figure 6.2, 'Trigger System' illustrates the sources of the synchronize outputs and inputs.

Free-Run Mode

If you like to use your camera in free run mode you can use any of the modes described above. The camera will ignore all trigger pulses or, if required, you can disable the output or deactivate the trigger using parameter *TriggerState*.

Allowed Frequencies

Mind the influence of parameter *TriggerOutputFrequency* in external and software triggered mode. Always set this parameter for these modes.

Table 6.2. Parameter properties of AreaTriggerMode

Property	Value	
Name	**`AreaTriggerMode`**	
Display Name	**`Area Trigger Mode`**	
Interface	**`IEnumeration`**	
Access policy	**`Read/Write/Change`**	
Visibility	**`Beginner`**	
Allowed values	**`Generator`**	Generator
	`External`	External
	`Software`	Software
	`Synchronized`	Synchronized
Default value	**`Generator`**	

Example 6.1. Usage of AreaTriggerMode

```
/* Set */ AreaTriggerMode = Generator;
/* Get */ value_ = AreaTriggerMode;
```

6.5.2. TriggerState

The area trigger system is operating in three trigger states. In the 'Active' state, the module is fully enabled. Trigger sources are used, pulses are queued, downscaled, multiplied and the output signals get their parameterized pulse forms. If the trigger is set into the 'Sync Stop' mode, the module will ignore further input pulses or stop the generation of pulses. However, the module will still process the pulses in the system. This means, a filled queue and the sequencer will continue processing the pulses and furthermore, the pulse form generators will output the signals according to the parameterized parameters. Finally, the 'Async Stop' mode asynchronously and immediately stops the full trigger system for the respective camera process. Note that this stop might result in output signals of undefined signal length as a current signal generation could be interrupted. Also note that a restart of a previously stopped trigger i.e. switching to the 'Active' state will clear the queue and the sequencer.

Table 6.3. Parameter properties of TriggerState

Property	Value
Name	**`TriggerState`**
Display Name	**`Trigger State`**
Interface	**`IEnumeration`**
Access policy	**`Read/Write/Change`**
Visibility	**`Beginner`**
Allowed values	**`Active`** Active **`AsyncStop`** Async Stop **`SyncStop`** Sync Stop
Default value	**`SyncStop`**

Example 6.2. Usage of TriggerState

```
/* Set */ TriggerState = SyncStop;
/* Get */ value_ = TriggerState;
```

6.5.3. TriggerOutputFrequency

This is a very important parameter of the trigger system. It is used for multiple functionalities.

If you run the trigger system in 'Generator' mode, this parameter will define the frequency of the generator. If you run the trigger system in 'External' or 'Software Trigger' operation mode, this parameter will specify the maximum allowed input frequency. Input frequencies which exceed this limit will cause the loss of the input pulse. To notify the user of this error, a read register contains an error flag or an event is generated. However, if the trigger queue is enabled, the exceeding pulses will be buffered and output at the maximum frequency which is defined by *TriggerOutputFrequency*. Thus, the parameter also defines the maximum queue output frequency. Moreover, it defines the maximum sequencer frequency. The maximum valid value of *TriggerOutputFrequency* is limited by *CamerasimulatorFramerate* in camera simulator mode.

Note that the range of this parameter depends on the settings in the pulse form generators. If you want to increase the frequency you might need to decrease the width or delay of one of the pulse form generators.

Equation 6.1. Dependency of Frequency and Pulse Form Generators

$$\frac{1}{\text{fps}} > Max\left\{\begin{array}{l}\frac{Max\{\text{WIDTH0},\text{DELAY0}\}}{\text{DOWNSCALE0}},\\ \frac{Max\{\text{WIDTH1},\text{DELAY1}\}}{\text{DOWNSCALE1}},\\ \frac{Max\{\text{WIDTH2},\text{DELAY2}\}}{\text{DOWNSCALE2}},\\ \frac{Max\{\text{WIDTH3},\text{DELAY3}\}}{\text{DOWNSCALE3}}\end{array}\right\}$$

- fps = *TriggerOutputFrequency*
- WIDTH[0..3] = *TriggerPulseFormGenerator[0..3]Width*
- DELAY[0..3] = *TriggerPulseFormGenerator[0..3]Delay*
- DOWNSCALE[0..3] = *TriggerPulseFormGenerator[0..3]Downscale*

Read the general trigger system explanations and the respective parameter explanations for more information.

Table 6.4. Parameter properties of TriggerOutputFrequency

Property	Value
Name	**TriggerOutputFrequency**
Display Name	**Trigger Output Frequency**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.01455191523** **Maximum 3.1249999999999996E7** **Stepsize 2.220446049250313E-16**
Default value	**8.0**
Unit of measure	**Hz**

Example 6.3. Usage of TriggerOutputFrequency

```
/* Set */ TriggerOutputFrequency = 8.0;
/* Get */ value_ = TriggerOutputFrequency;
```

6.5.4. Trigger Input

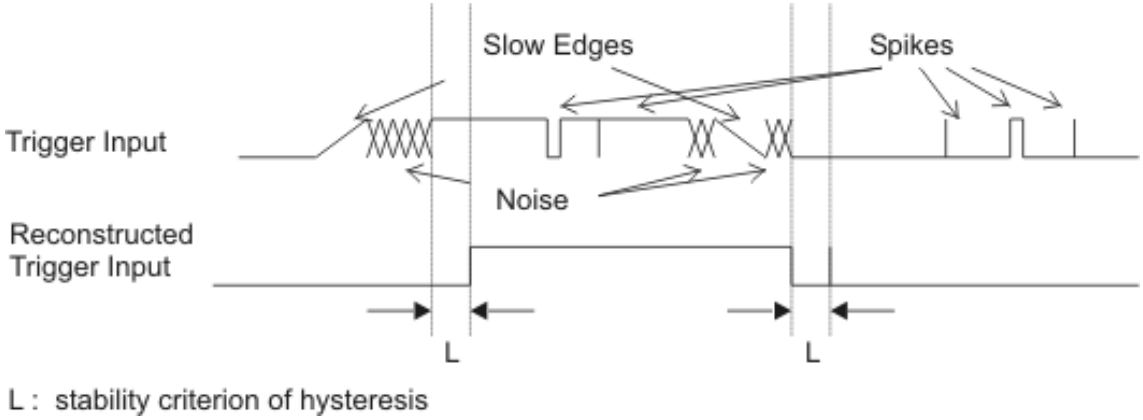
The parameters of category Trigger Input are used to configure the input source of the trigger system. The category is divided into sub categories. All external sources are configured in category external. Category software trigger allows the configuration, monitoring and controlling of software trigger pulses. In category statistics the parameters for input statistics are present.

6.5.4.1. External

6.5.4.1.1. TriggerInDebounce

In general, a perfect and steady trigger input signal can not be guaranteed in practice. A transfer using long cable connections and the operation in bad shielded environments might have a distinct influence on the signal quality. Typical problems are strong flattening of the digital's signal edges, occurring interferences during toggling and inducing of short jamming pulses (spikes). In the following figure, some of the influences are illustrated.

Figure 6.18. Faulty Signal and it's Reconstruction



The trigger system has been designed to work highly reliable even under problematic signal conditions. An internal debouncing of the inputs will eliminate unwanted trigger pulses. It is comparable to a hysteresis. Only signal changes which are constant for a specified time (marked 'L' in the figure) are accepted which makes the input insensitive to jamming pulses. Also multiple triggering will be effectively disabled, which occurs by slow signal transfers and bouncing. Set the debounce time according to your requirements in µs. Note that the debounce time will also be the delay time before the trigger signal can be processed. The settings made for this parameter affect all digital inputs. The parameter is camera process independent i.e. the latest settings will apply for all camera inputs.

Table 6.5. Parameter properties of TriggerInDebounce

Property	Value
Name	**`TriggerInDebounce`**
Display Name	**`Input Debounce`**
Interface	**`IFloat`**
Access policy	**`Read/Write/Change`**
Visibility	**`Beginner`**
Allowed values	**`Minimum 0.0`** **`Maximum 209.71200000000002`** **`Stepsize 0.0032`**
Default value	**`1.0`**
Unit of measure	**`µs`**

Example 6.4. Usage of TriggerInDebounce

```
/* Set */ TriggerInDebounce = 1.0;
/* Get */ value_ = TriggerInDebounce;
```

6.5.4.1.2. FrontGPI

Parameter *FrontGPI* is used to monitor the digital inputs of the interface card.

You can read the current state of these inputs using parameter *FrontGPI*. Bit 0 of the read value represents input 0, bit 1 represents input 1 and so on. For example, if you obtain the value 10 or hexadecimal 0xA the interface card will have high level on it's digital inputs 1 and 3.

Table 6.6. Parameter properties of FrontGPI

Property	Value
Name	**FrontGPI**
Display Name	**Front GPI**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 15** **Stepsize 1**

Example 6.5. Usage of FrontGPI

```
/* Get */ value_ = FrontGPI;
```

6.5.4.1.3. TriggerInSource

To use the external trigger you have to select the input carrying the image trigger signal. Select one of the eight inputs. four Front GPI inputs. If *AreaTriggerMode* is not set to external, this parameter will select the input for the input statistics only.

Table 6.7. Parameter properties of TriggerInSource

Property	Value
Name	**TriggerInSource**
Display Name	**Trigger In Source**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**TriggerInSourceFrontGPI0** Trigger In Source Front GPI 0 **TriggerInSourceFrontGPI1** Trigger In Source Front GPI 1 **TriggerInSourceFrontGPI2** Trigger In Source Front GPI 2 **TriggerInSourceFrontGPI3** Trigger In Source Front GPI 3
Default value	**TriggerInSourceFrontGPI0**

Example 6.6. Usage of TriggerInSource

```
/* Set */ TriggerInSource = TriggerInSourceFrontGPI0;
/* Get */ value_ = TriggerInSource;
```

6.5.4.1.4. TriggerInPolarity

For the selected input using parameter *TriggerInSource* the polarity is set with this parameter.

Table 6.8. Parameter properties of TriggerInPolarity

Property	Value
Name	**TriggerInPolarity**
Display Name	**Trigger In Polarity**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LowActive** Low Active **HighActive** High Active
Default value	**HighActive**

Example 6.7. Usage of TriggerInPolarity

```
/* Set */ TriggerInPolarity = HighActive;
/* Get */ value_ = TriggerInPolarity;
```

6.5.4.1.5. TriggerInDownscale

If you use the trigger system in external trigger mode, you can downscale the trigger inputs selected by *TriggerInSource*. See *TriggerInDownscalePhase* for more information.

Table 6.9. Parameter properties of TriggerInDownscale

Property	Value
Name	**TriggerInDownscale**
Display Name	**Trigger In Downscale**
Interface	**IInteger**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum** 1 **Maximum** 2147483647 **Stepsize** 1
Default value	**1**

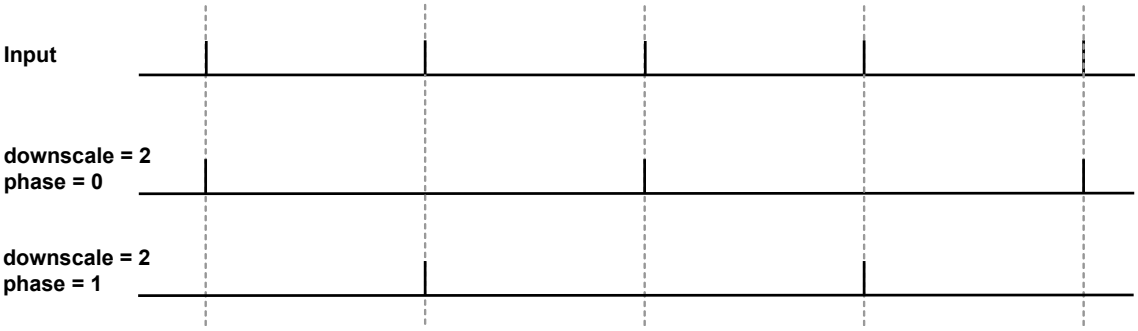
Example 6.8. Usage of TriggerInDownscale

```
/* Set */ TriggerInDownscale = 1;
/* Get */ value_ = TriggerInDownscale;
```

6.5.4.1.6. TriggerInDownscalePhase

Parameters *TriggerInDownscale* and *TriggerInDownscalePhase* are used to downscale external trigger inputs. The downscale value represents the factor. For example value three will remove two out of three successive trigger pulses. The phase is used to make the selection of the pulse in the sequence. For the given example, a phase set to value zero will forward the first pulse and will remove pulses two and three of a sequence of three pulses. See the following figure for more explanations.

Figure 6.19. Triggerin Dowscale



Mind the dependency between the downscale factor and the phase. The value of the downscale factor has to be greater than the phase!

Table 6.10. Parameter properties of TriggerInDownscalePhase

Property	Value
Name	**`TriggerInDownscalePhase`**
Display Name	**`Trigger In Downscale Phase`**
Interface	**`IInteger`**
Access policy	**`Read/Write/Change`**
Visibility	**`Beginner`**
Allowed values	**`Minimum 0`** **`Maximum 4294967294`** **`Stepsize 1`**
Default value	**`0`**

Example 6.9. Usage of TriggerInDownscalePhase

```
/* Set */ TriggerInDownscalePhase = 0;
/* Get */ value_ = TriggerInDownscalePhase;
```

6.5.4.2. Software Trigger

6.5.4.2.1. SendSoftwareTrigger

If the trigger system is run in software triggered mode (see parameter *AreaTriggerMode*), this parameter is activated. Write value '1' to this parameter to input a software trigger. If the trigger queue is activated multiple software trigger pulses can be written to the interface card. They will fill the queue and being processed with the maximum allowed frequency parameterized by *TriggerOutputFrequency*.

Note that software trigger pulses can only be written if the trigger system has been activated using parameter *TriggerState*. Moreover, if the queue has not been activated, new software trigger pulses can only be written if the trigger system is not busy. Therefore, writing to the parameter can cause an Software Trigger Busy error.

Table 6.11. Parameter properties of SendSoftwareTrigger

Property	Value
Name	**`SendSoftwareTrigger`**
Display Name	**`Send Software Trigger`**
Interface	**`ICommand`**
Access policy	**`Write/Change`**
Visibility	**`Beginner`**

Example 6.10. Usage of SendSoftwareTrigger

```
/* Set */ SendSoftwareTrigger();
```

6.5.4.2.2. SoftwareTriggerIsBusy

After writing one or multiple pulses to the trigger system using the software trigger, the system might be busy for a while. To check if there are no pulses left for processing use this parameter.

Table 6.12. Parameter properties of SoftwareTriggerIsBusy

Property	Value
Name	**SoftwareTriggerIsBusy**
Display Name	**Software Trigger Is Busy**
Interface	**IBoolean**
Access policy	**Read-Only**
Visibility	**Beginner**

Example 6.11. Usage of SoftwareTriggerIsBusy

```
/* Get */ value_ = SoftwareTriggerIsBusy;
```

6.5.4.2.3. SoftwareTriggerQueueFillLevel

The value of this parameter represents the number of pulses in the software trigger queue which have to be processed. The fill level depends on the number of pulses written to *SendSoftwareTrigger*, the trigger pulse multiplication factor *TriggerMultiplyPulses* and the maximum output frequency defined by *TriggerOutputFrequency*. The value decrement is given in steps of *TriggerMultiplyPulses*.

Table 6.13. Parameter properties of SoftwareTriggerQueueFillLevel

Property	Value
Name	**SoftwareTriggerQueueFillLevel**
Display Name	**Software Trigger Queue Fill Level**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 2040** **Stepsize 1**
Unit of measure	**pulses**

Example 6.12. Usage of SoftwareTriggerQueueFillLevel

```
/* Get */ value_ = SoftwareTriggerQueueFillLevel;
```

6.5.4.3. InStatistics

The trigger input statistics module will offer you frequency analysis and pulse counting of the selected input. The digital input for the statistics is selected by *TriggerInPolarity*. Measurements are performed after debouncing and polarity selection but before downscaling.

The statistics section also includes a list of digital input events.

6.5.4.3.1. TriggerInStatisticsSource

The trigger statistics module allows you to individually select one of the inputs as source. Select one of the eight inputs.

Table 6.14. Parameter properties of TriggerInStatisticsSource

Property	Value
Name	**TriggerInStatisticsSource**
Display Name	**Trigger In Statistics Source**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**TriggerInSourceFrontGPI0** Trigger In Source Front GPI 0 **TriggerInSourceFrontGPI1** Trigger In Source Front GPI 1 **TriggerInSourceFrontGPI2** Trigger In Source Front GPI 2 **TriggerInSourceFrontGPI3** Trigger In Source Front GPI 3
Default value	**TriggerInSourceFrontGPI0**

Example 6.13. Usage of TriggerInStatisticsSource

```
/* Set */ TriggerInStatisticsSource = TriggerInSourceFrontGPI0;
/* Get */ value_ = TriggerInStatisticsSource;
```

6.5.4.3.2. TriggerInStatisticsPolarity

For the selected input using parameter *TriggerInStatisticsSource* the polarity is set using this parameter.

Table 6.15. Parameter properties of TriggerInStatisticsPolarity

Property	Value
Name	**TriggerInStatisticsPolarity**
Display Name	**Trigger In Statistics Polarity**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LowActive** Low Active **HighActive** High Active
Default value	**HighActive**

Example 6.14. Usage of TriggerInStatisticsPolarity

```
/* Set */ TriggerInStatisticsPolarity = HighActive;
/* Get */ value_ = TriggerInStatisticsPolarity;
```

6.5.4.3.3. TriggerInStatisticsPulseCount

The input pulses are count and the current value can be read with this parameter. Use the counter for verification of your system. For example, compare the counter value with the received number of images to check for exceeding periods.

Table 6.16. Parameter properties of TriggerInStatisticsPulseCount

Property	Value
Name	**TriggerInStatisticsPulseCount**
Display Name	**Trigger In Statistics Pulse Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 65535** **Stepsize 1**
Unit of measure	**pulses**

Example 6.15. Usage of TriggerInStatisticsPulseCount

```
/* Get */ value_ = TriggerInStatisticsPulseCount;
```

6.5.4.3.4. TriggerInStatisticsPulseCountClear

Clear the input pulse counter by writing to this register.

Table 6.17. Parameter properties of TriggerInStatisticsPulseCountClear

Property	Value
Name	**TriggerInStatisticsPulseCountClear**
Display Name	**Trigger In Statistics Pulse Count Clear**
Interface	**ICommand**
Access policy	**Write/Change**
Visibility	**Beginner**

Example 6.16. Usage of TriggerInStatisticsPulseCountClear

```
/* Set */ TriggerInStatisticsPulseCountClear();
```

6.5.4.3.5. TriggerInStatisticsFrequency

The current frequency can be read using this parameter. It shows the frequency of the last two received pulses at the interface card.

Table 6.18. Parameter properties of TriggerInStatisticsFrequency

Property	Value
Name	**TriggerInStatisticsFrequency**
Display Name	**Trigger In Statistics Frequency**
Interface	**IFloat**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 3.125E8** **Stepsize 2.220446049250313E-16**
Unit of measure	**Hz**

Example 6.17. Usage of TriggerInStatisticsFrequency

```
/* Get */ value_ = TriggerInStatisticsFrequency;
```

6.5.4.3.6. TriggerInStatisticsMinimumFrequency

The trigger system will memorize the minimum detected input frequency. This will give you information about frequency peaks.

Table 6.19. Parameter properties of TriggerInStatisticsMinimumFrequency

Property	Value
Name	**TriggerInStatisticsMinimumFrequency**
Display Name	**Trigger In Statistics Minimum Frequency**
Interface	**IFloat**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 3.125E8** **Stepsize 2.220446049250313E-16**
Unit of measure	**Hz**

Example 6.18. Usage of TriggerInStatisticsMinimumFrequency

```
/* Get */ value_ = TriggerInStatisticsMinimumFrequency;
```

6.5.4.3.7. TriggerInStatisticsMaximumFrequency

The trigger system will memorize the maximum detected input frequency. This will give you information about frequency peaks.

Table 6.20. Parameter properties of TriggerInStatisticsMaximumFrequency

Property	Value
Name	**TriggerInStatisticsMaximumFrequency**
Display Name	**Trigger In Statistics Maximum Frequency**
Interface	**IFloat**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 3.125E8** **Stepsize 2.220446049250313E-16**
Unit of measure	**Hz**

Example 6.19. Usage of TriggerInStatisticsMaximumFrequency

```
/* Get */ value_ = TriggerInStatisticsMaximumFrequency;
```

6.5.4.3.8. TriggerInStatisticsMinMaxFrequencyClear

To clear the minimum and maximum frequency measurements, write to this register. The minimum and maxumum frequency will then be the current input frequency.

Table 6.21. Parameter properties of TriggerInStatisticsMinMaxFrequencyClear

Property	Value
Name	**`TriggerInStatisticsMinMaxFrequencyClear`**
Display Name	**`Trigger In Statistics Min Max Frequency Clear`**
Interface	**`ICommand`**
Access policy	**`Write/Change`**
Visibility	**`Beginner`**

Example 6.20. Usage of TriggerInStatisticsMinMaxFrequencyClear

```
/* Set */ TriggerInStatisticsMinMaxFrequencyClear();
```

6.5.4.3.9. LineFront0RisingEdge

This event is generated for each rising signal edge at trigger input 0. Except for the timestamp, the event has no additional data included. Keep in mind that fast changes of the input signal can cause high interrupt rates which might slow down the system. This event can occur independent of the acquisition status.

For a general explanation on events see Event.

6.5.4.3.10. LineFront0FallingEdge

This event is generated for each falling signal edge at trigger input 0. Except for the timestamp, the event has no additional data included. Keep in mind that fast changes of the input signal can cause high interrupt rates which might slow down the system. This event can occur independent of the acquisition status.

For a general explanation on events see Event.

6.5.5. Sequencer

The sequencer is a powerful feature to generate multiple pulses out of one input pulse. It is available in external and software trigger mode, but not in generator mode. The sequencer multiplies an input pulse using the factor set by *TriggerMultiplyPulses*. The inserted pulses will have a time delay to the original signal according to the setting made for parameter *TriggerOutputFrequency*. Thus, the inserted pulses are not evenly distributed between the input pulses, they will be inserted with a delay specified by *TriggerOutputFrequency*. Hence, it is very important, that the multiplicate pulses with a parameterized delay will not cause a loss of input signals.

Let's have a look at an example. Suppose you have an external trigger source generating a pulse once every second. Your input frequency will then be 1Hz. Assume that the sequencer is set to a multiplication factor of 2 and the maximum frequency defined by *TriggerOutputFrequency* is set to 2.1Hz.

The trigger system will forward each external pulse into the trigger system and will also generate a second pulse 0.48 seconds later. As you can see, the multiplication frequency is chosen to be slightly higher than the doubled input frequency. This will allow the compensation of varying input frequencies. If the time between two pulses at the input will be less than 0.96 seconds, you will loose the second pulse. Basler recommends the multiplication frequency to be fast enough to not loose pulses or recommends the activation of the trigger queue for compensation. You can check for lost pulses with parameter *TriggerExceededPeriodLimits*.

6.5.5.1. TriggerMultiplyPulses

Set the trigger input multiplication factor.

Table 6.22. Parameter properties of TriggerMultiplyPulses

Property	Value
Name	**TriggerMultiplyPulses**
Display Name	**Upscale Trigger Pulses**
Interface	**IInteger**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 1** **Maximum 65535** **Stepsize 1**
Default value	**1**

Example 6.21. Usage of TriggerMultiplyPulses

```
/* Set */ TriggerMultiplyPulses = 1;
/* Get */ value_ = TriggerMultiplyPulses;
```

6.5.6. Queue

The maximum trigger output frequency is limited to the the setting of parameter *TriggerOutputFrequency*. This can avoid the loss of trigger pulses in the camera which is hard to detect. In some cases it is possible, that the frequency of your external trigger source varies. To prevent the loose of trigger pulses, you can activate the trigger queue to buffer these pulses. Furthermore, the queue can be used to buffer trigger input pulses if you use the sequencer and the software trigger.

Activate the trigger queue using parameter *TriggerQueueMode*.

The queue fill level can be monitored by parameter *TriggerQueueFillLevel*. Moreover, two events allow monitoring of the fill level. Using parameters *TriggerQueueFillLevelEventOnThreshold* and *TriggerQueueFillLevelEventOffThreshold* it is possible to set two thresholds. If the fill level exceeds the ON-threshold, the respective event *TriggerQueueFilllevelThresholdOn* is generated. If the fill level gets less or equal than the OFF-threshold, the event *TriggerQueueFilllevelThresholdOff* is generated.

Note that a fill level value *n* indicates that between *n* and *n* + 1 trigger pulses have to be processed by the system. Therefore, a fill level value zero means that no more values are in the queue, but there might be still a pulse (or multiple pulses if the sequencer is used) to be processed. There exists one exception for value zero obtained with *TriggerQueueFillLevel* i.e. the parameter and not the events. This value at this parameter truly indicates that no more pulses are in the queue and all pulses have been full processed.

6.5.6.1. TriggerQueueMode

Activate the queue using this parameter. Note that a queue de-activation will erase all remaining values in the queue.

Table 6.23. Parameter properties of TriggerQueueMode

Property	Value
Name	**TriggerQueueMode**
Display Name	**Trigger Queue Mode**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**On** On **Off** Off
Default value	**Off**

Example 6.22. Usage of TriggerQueueMode

```
/* Set */ TriggerQueueMode = Off;
/* Get */ value_ = TriggerQueueMode;
```

6.5.6.2. TriggerQueueFillLevel

Obtain the currently queued pulses with this parameter. At maximum 2040 pulses can be queued. The queue fill level includes the input pulses, i.e. the external trigger pulses in the queue or the software trigger pulses in the queue. The fill level does not include the pulses generated by the sequencer. The fill level is zero, if the trigger system is not busy anymore i.e. no more pulses are left to be processed.

Table 6.24. Parameter properties of TriggerQueueFillLevel

Property	Value
Name	**TriggerQueueFillLevel**
Display Name	**Trigger Queue Fill Level**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum** 0 **Maximum** 2040 **Stepsize** 1
Unit of measure	**pulses**

Example 6.23. Usage of TriggerQueueFillLevel

```
/* Get */ value_ = TriggerQueueFillLevel;
```

6.5.6.3. TriggerQueueFillLevelEventOnThreshold

Set the ON-threshold for fill level event generation with this parameter.

Table 6.25. Parameter properties of TriggerQueueFillLevelEventOnThreshold

Property	Value
Name	**TriggerQueueFillLevelEventOnThreshold**
Display Name	**Trigger Queue Fill Level Event On Threshold**
Interface	**IInteger**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 2** **Maximum 2047** **Stepsize 1**
Default value	**2047**
Unit of measure	**pulses**

Example 6.24. Usage of TriggerQueueFillLevelEventOnThreshold

```
/* Set */ TriggerQueueFillLevelEventOnThreshold = 2047;
/* Get */ value_ = TriggerQueueFillLevelEventOnThreshold;
```

6.5.6.4. TriggerQueueFillLevelEventOffThreshold

Set the OFF-threshold for fill level event generation with this parameter.

Table 6.26. Parameter properties of TriggerQueueFillLevelEventOffThreshold

Property	Value
Name	**TriggerQueueFillLevelEventOffThreshold**
Display Name	**Trigger Queue Fill Level Event Off Threshold**
Interface	**IInteger**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 2** **Maximum 2047** **Stepsize 1**
Default value	**2**
Unit of measure	**pulses**

Example 6.25. Usage of TriggerQueueFillLevelEventOffThreshold

```
/* Set */ TriggerQueueFillLevelEventOffThreshold = 2;
/* Get */ value_ = TriggerQueueFillLevelEventOffThreshold;
```

6.5.6.5. TriggerQueueFilllevelThresholdOn

The event is generated if the queue fill level exceeds the ON-threshold set by parameter *TriggerQueueFillLevelEventOnThreshold*. Except for the timestamp, the event has no additional data included.

For a general explanation on events see Event.

6.5.6.6. TriggerQueueFilllevelThresholdOff

The event is generated if the queue fill level gets less or equal than the OFF-threshold set by parameter *TriggerQueueFillLevelEventOffThreshold*. Except for the timestamp, the event has no additional data included.

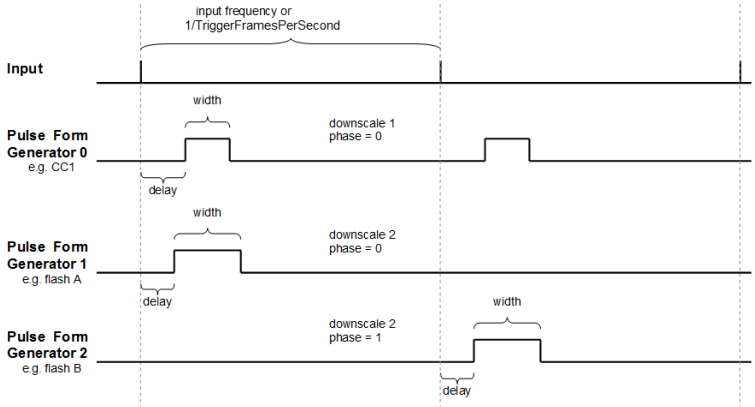
For a general explanation on events see Event.

6.5.7. Pulse Form Generator 0

The parameters explained previously were used to generate the trigger pulses. Next, we will need to prepare the pulses for the outputs. The area trigger system includes four individual pulse form generators. These generators define the width and delay of the output signals and also support downscaling of pulses which can be useful if different light sources are used successively. After parameterizing the pulse form generators you can arbitrarily allocate the pulse form generators to the outputs.

The following figure illustrates the output of the pulse form generators and the parameters.

Figure 6.20. Pulse Form Generators



Once again, note that the ranges of the parameters depend on the other settings in the pulse form generators and on paramerter *TriggerOutputFrequency*. If you want to increase the frequency you might need to decrease the width or delay of one of the pulse form generators.

Equation 6.2. Dependency of Frequency and Pulse Form Generators

$$\frac{1}{\text{fps}} > Max\left\{\begin{array}{l}\frac{Max\{\text{WIDTH0},\text{DELAY0}\}}{\text{DOWNSCALE0}},\\ \frac{Max\{\text{WIDTH1},\text{DELAY1}\}}{\text{DOWNSCALE1}},\\ \frac{Max\{\text{WIDTH2},\text{DELAY2}\}}{\text{DOWNSCALE2}},\\ \frac{Max\{\text{WIDTH3},\text{DELAY3}\}}{\text{DOWNSCALE3}}\end{array}\right\}$$

- fps = *TriggerOutputFrequency*
- WIDTH[0..3] = *TriggerPulseFormGenerator[0..3]Width*
- DELAY[0..3] = *TriggerPulseFormGenerator[0..3]Delay*
- DOWNSCALE[0..3] = *TriggerPulseFormGenerator[0..3]Downscale*

6.5.7.1. TriggerPulseFormGenerator0Downscale et al.

Note

This description applies also to the following parameters: TriggerPulseFormGenerator1Downscale, TriggerPulseFormGenerator2Downscale, TriggerPulseFormGenerator3Downscale

The trigger pulses can be downscaled. Set the downscale factor by use of this parameter. Note the dependency between this parameter and the phase. See *TriggerPulseFormGenerator[0..3]DownscalePhase* for more information.

Table 6.27. Parameter properties of TriggerPulseFormGenerator0Downscale

Property	Value
Name	**TriggerPulseFormGenerator0Downscale**
Display Name	**Trigger Pulse Form Generator0 Downscale**
Interface	**IInteger**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 1** **Maximum 7** **Stepsize 1**
Default value	**1**

Example 6.26. Usage of TriggerPulseFormGenerator0Downscale

```
/* Set */ TriggerPulseFormGenerator0Downscale = 1;
/* Get */ value_ = TriggerPulseFormGenerator0Downscale;
```

6.5.7.2. TriggerPulseFormGenerator0DownscalePhase et al.

Note

This description applies also to the following parameters: TriggerPulseFormGenerator1DownscalePhase, TriggerPulseFormGenerator2DownscalePhase, TriggerPulseFormGenerator3DownscalePhase

The parameter *TriggerPulseFormGenerator[0..3]Downscale* defines the number of phases and parameter *TriggerPulseFormGenerator[0..3]DownscalePhase* selects the one being used. The downscale value represents the factor. For example value three will remove two out of three successive trigger pulses. The phase is used to make the selection of the pulse in the sequence. For the given example, a phase set to value zero will forward the first pulse and will remove pulses two and three of a sequence of three pulses. Check Section 6.5.7, 'Pulse Form Generator 0' for more information.

Take care of the dependency between the downscale factor and the phase. The factor has to be greater than the phase.

Table 6.28. Parameter properties of TriggerPulseFormGenerator0DownscalePhase

Property	Value
Name	**TriggerPulseFormGenerator0DownscalePhase**
Display Name	**Trigger Pulse Form Generator0 Downscale Phase**
Interface	**IInteger**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 6** **Stepsize 1**
Default value	**0**

Example 6.27. Usage of TriggerPulseFormGenerator0DownscalePhase

```
/* Set */ TriggerPulseFormGenerator0DownscalePhase = 0;
```

```
/* Get */ value_ = TriggerPulseFormGenerator0DownscalePhase;
```

6.5.7.3. TriggerPulseFormGenerator0Delay et al.

Note

This description applies also to the following parameters: TriggerPulseFormGenerator1Delay, TriggerPulseFormGenerator2Delay, TriggerPulseFormGenerator3Delay

Set a signal delay with this parameter. The unit of this parameter is µs.

Table 6.29. Parameter properties of TriggerPulseFormGenerator0Delay

Property	Value
Name	**TriggerPulseFormGenerator0Delay**
Display Name	**Trigger Pulse Form Generator0 Delay**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 3.4E7** **Stepsize 0.0032**
Default value	**0.0**
Unit of measure	**µs**

Example 6.28. Usage of TriggerPulseFormGenerator0Delay

```
/* Set */ TriggerPulseFormGenerator0Delay = 0.0;
/* Get */ value_ = TriggerPulseFormGenerator0Delay;
```

6.5.7.4. TriggerPulseFormGenerator0Width et al.

Note

This description applies also to the following parameters: TriggerPulseFormGenerator1Width, TriggerPulseFormGenerator2Width, TriggerPulseFormGenerator3Width

Set the signal width, i.e. the active time of the output signal. The unit of this parameter is µs.

Table 6.30. Parameter properties of TriggerPulseFormGenerator0Width

Property	Value
Name	**TriggerPulseFormGenerator0Width**
Display Name	**Trigger Pulse Form Generator0 Width**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0032** **Maximum 6.8E7** **Stepsize 0.0032**
Default value	**4.0**
Unit of measure	**µs**

Example 6.29. Usage of TriggerPulseFormGenerator0Width

```
/* Set */ TriggerPulseFormGenerator0Width = 4.0;
/* Get */ value_ = TriggerPulseFormGenerator0Width;
```

6.5.8. Pulse Form Generator 1

The settings for pulse form generator 1 are equal to those of pulse form generator 0. Please read Section 6.5.7, 'Pulse Form Generator 0' for a detailed description.

6.5.9. Pulse Form Generator 2

The settings for pulse form generator 2 are equal to those of pulse form generator 0. Please read Section 6.5.7, 'Pulse Form Generator 0' for a detailed description.

6.5.10. Pulse Form Generator 3

The settings for pulse form generator 3 are equal to those of pulse form generator 0. Please read Section 6.5.7, 'Pulse Form Generator 0' for a detailed description.

6.5.11. CameraOutSignalMapping

The camera interface of the CXP-12 Interface Card 4C is equipped with a trigger output channel to trigger the camera.

Moreover, two front general purpose outputs (GPOs) to the camera exist.
To identify the required signals and their mapping, consult the vendor's manual of your camera.

The trigger system of this applet provides several possibilities of mapping pulse sources to the camera channels:

- Pulse form generators 0 to 3

  The pulse form generators are the main output sources of the trigger system. You can map either the start or the end of the signals to the CXP camera port. If your camera runs in **`Timed`** mode with external CXP trigger, you only need to send the start signal, usually on CXP LinkTrigger0. If your camera runs in **`TriggerControlled`** mode, you need to send the start of exposure signal on CXP LinkTrigger0 and the end of exposure signal on CXP LinkTrigger1.

- If you don't need to send any pulses on CXP LinkTrigger, set *CxpLinkTrigger0Source* to **`GND`**.

- The input bypass

  The frame grabber trigger system ignores the signal length of the input signals. If you want to directly bypass one of the input signals to a CXP LinkTrigger, you can select the input start or end signal of pulse, i.e. rising or falling edge.

6.5.11.1. CxpLinkTrigger0Source

Table 6.31. Parameter properties of CxpLinkTrigger0Source

Property	Value
Name	**CxpLinkTrigger0Source**
Display Name	**CXP Link Trigger 0 Source**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**GND** GND **CamAPulseGenerator0RisingEdge** Cam A Pulse Generator 0 Rising **CamAPulseGenerator1RisingEdge** Cam A Pulse Generator 1 Rising **CamAPulseGenerator2RisingEdge** Cam A Pulse Generator 2 Rising **CamAPulseGenerator3RisingEdge** Cam A Pulse Generator 3 Rising **CamAPulseGenerator0FallingEdge** Cam A Pulse Generator 0 Falling **CamAPulseGenerator1FallingEdge** Cam A Pulse Generator 1 Falling **CamAPulseGenerator2FallingEdge** Cam A Pulse Generator 2 Falling **CamAPulseGenerator3FallingEdge** Cam A Pulse Generator 3 Falling **CamBPulseGenerator0RisingEdge** Cam B Pulse Generator 0 Rising **CamBPulseGenerator1RisingEdge** Cam B Pulse Generator 1 Rising **CamBPulseGenerator2RisingEdge** Cam B Pulse Generator 2 Rising **CamBPulseGenerator3RisingEdge** Cam B Pulse Generator 3 Rising **CamBPulseGenerator0FallingEdge** Cam B Pulse Generator 0 Falling **CamBPulseGenerator1FallingEdge** Cam B Pulse Generator 1 Falling **CamBPulseGenerator2FallingEdge** Cam B Pulse Generator 2 Falling **CamBPulseGenerator3FallingEdge** Cam B Pulse Generator 3 Falling **BypassFrontGPI0RisingEdge** Bypass Front GPI 0 Rising **BypassFrontGPI0FallingEdge** Bypass Front GPI 0 Falling **BypassFrontGPI1RisingEdge** Bypass Front GPI 1 Rising **BypassFrontGPI1FallingEdge** Bypass Front GPI 1 Falling **BypassFrontGPI2RisingEdge** Bypass Front GPI 2 Rising **BypassFrontGPI2FallingEdge** Bypass Front GPI 2 Falling **BypassFrontGPI3RisingEdge** Bypass Front GPI 3 Rising **BypassFrontGPI3FallingEdge** Bypass Front GPI 3 Falling **PulseGeneratorRisingEdge** Pulse Generator 0 Rising **PulseGenerator1RisingEdge** Pulse Generator 1 Rising **PulseGenerator2RisingEdge** Pulse Generator 2 Rising **PulseGenerator3RisingEdge** Pulse Generator 3 Rising **PulseGenerator0FallingEdge** Pulse Generator 0 Falling **PulseGenerator1FallingEdge** Pulse Generator 1 Falling **PulseGenerator2FallingEdge** Pulse Generator 2 Falling **PulseGenerator3FallingEdge** Pulse Generator 3 Falling
Default value	**PulseGeneratorRisingEdge**

Example 6.30. Usage of CxpLinkTrigger0Source

```
/* Set */ CxpLinkTrigger0Source = PulseGeneratorRisingEdge;
/* Get */ value_ = CxpLinkTrigger0Source;
```

6.5.11.2. CxpLinkTrigger1Source

Table 6.32. Parameter properties of CxpLinkTrigger1Source

Property	Value
Name	**CxpLinkTrigger1Source**
Display Name	**CXP Link Trigger 1 Source**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**GND** GND **CamAPulseGenerator0RisingEdge** Cam A Pulse Generator 0 Rising **CamAPulseGenerator1RisingEdge** Cam A Pulse Generator 1 Rising **CamAPulseGenerator2RisingEdge** Cam A Pulse Generator 2 Rising **CamAPulseGenerator3RisingEdge** Cam A Pulse Generator 3 Rising **CamAPulseGenerator0FallingEdge** Cam A Pulse Generator 0 Falling **CamAPulseGenerator1FallingEdge** Cam A Pulse Generator 1 Falling **CamAPulseGenerator2FallingEdge** Cam A Pulse Generator 2 Falling **CamAPulseGenerator3FallingEdge** Cam A Pulse Generator 3 Falling **CamBPulseGenerator0RisingEdge** Cam B Pulse Generator 0 Rising **CamBPulseGenerator1RisingEdge** Cam B Pulse Generator 1 Rising **CamBPulseGenerator2RisingEdge** Cam B Pulse Generator 2 Rising **CamBPulseGenerator3RisingEdge** Cam B Pulse Generator 3 Rising **CamBPulseGenerator0FallingEdge** Cam B Pulse Generator 0 Falling **CamBPulseGenerator1FallingEdge** Cam B Pulse Generator 1 Falling **CamBPulseGenerator2FallingEdge** Cam B Pulse Generator 2 Falling **CamBPulseGenerator3FallingEdge** Cam B Pulse Generator 3 Falling **BypassFrontGPI0RisingEdge** Bypass Front GPI 0 Rising **BypassFrontGPI0FallingEdge** Bypass Front GPI 0 Falling **BypassFrontGPI1RisingEdge** Bypass Front GPI 1 Rising **BypassFrontGPI1FallingEdge** Bypass Front GPI 1 Falling **BypassFrontGPI2RisingEdge** Bypass Front GPI 2 Rising **BypassFrontGPI2FallingEdge** Bypass Front GPI 2 Falling **BypassFrontGPI3RisingEdge** Bypass Front GPI 3 Rising **BypassFrontGPI3FallingEdge** Bypass Front GPI 3 Falling **PulseGeneratorRisingEdge** Pulse Generator 0 Rising **PulseGenerator1RisingEdge** Pulse Generator 1 Rising **PulseGenerator2RisingEdge** Pulse Generator 2 Rising **PulseGenerator3RisingEdge** Pulse Generator 3 Rising **PulseGenerator0FallingEdge** Pulse Generator 0 Falling **PulseGenerator1FallingEdge** Pulse Generator 1 Falling **PulseGenerator2FallingEdge** Pulse Generator 2 Falling **PulseGenerator3FallingEdge** Pulse Generator 3 Falling
Default value	**PulseGenerator0FallingEdge**

Example 6.31. Usage of CxpLinkTrigger1Source

```
/* Set */ CxpLinkTrigger1Source = PulseGenerator0FallingEdge;
/* Get */ value_ = CxpLinkTrigger1Source;
```

6.5.11.3. CxpLinkTrigger2Source

Table 6.33. Parameter properties of CxpLinkTrigger2Source

Property	Value
Name	**CxpLinkTrigger2Source**
Display Name	**CXP Link Trigger 2 Source**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**GND** GND **CamAPulseGenerator0RisingEdge** Cam A Pulse Generator 0 Rising **CamAPulseGenerator1RisingEdge** Cam A Pulse Generator 1 Rising **CamAPulseGenerator2RisingEdge** Cam A Pulse Generator 2 Rising **CamAPulseGenerator3RisingEdge** Cam A Pulse Generator 3 Rising **CamAPulseGenerator0FallingEdge** Cam A Pulse Generator 0 Falling **CamAPulseGenerator1FallingEdge** Cam A Pulse Generator 1 Falling **CamAPulseGenerator2FallingEdge** Cam A Pulse Generator 2 Falling **CamAPulseGenerator3FallingEdge** Cam A Pulse Generator 3 Falling **CamBPulseGenerator0RisingEdge** Cam B Pulse Generator 0 Rising **CamBPulseGenerator1RisingEdge** Cam B Pulse Generator 1 Rising **CamBPulseGenerator2RisingEdge** Cam B Pulse Generator 2 Rising **CamBPulseGenerator3RisingEdge** Cam B Pulse Generator 3 Rising **CamBPulseGenerator0FallingEdge** Cam B Pulse Generator 0 Falling **CamBPulseGenerator1FallingEdge** Cam B Pulse Generator 1 Falling **CamBPulseGenerator2FallingEdge** Cam B Pulse Generator 2 Falling **CamBPulseGenerator3FallingEdge** Cam B Pulse Generator 3 Falling **BypassFrontGPI0RisingEdge** Bypass Front GPI 0 Rising **BypassFrontGPI0FallingEdge** Bypass Front GPI 0 Falling **BypassFrontGPI1RisingEdge** Bypass Front GPI 1 Rising **BypassFrontGPI1FallingEdge** Bypass Front GPI 1 Falling **BypassFrontGPI2RisingEdge** Bypass Front GPI 2 Rising **BypassFrontGPI2FallingEdge** Bypass Front GPI 2 Falling **BypassFrontGPI3RisingEdge** Bypass Front GPI 3 Rising **BypassFrontGPI3FallingEdge** Bypass Front GPI 3 Falling **PulseGeneratorRisingEdge** Pulse Generator 0 Rising **PulseGenerator1RisingEdge** Pulse Generator 1 Rising **PulseGenerator2RisingEdge** Pulse Generator 2 Rising **PulseGenerator3RisingEdge** Pulse Generator 3 Rising **PulseGenerator0FallingEdge** Pulse Generator 0 Falling **PulseGenerator1FallingEdge** Pulse Generator 1 Falling **PulseGenerator2FallingEdge** Pulse Generator 2 Falling **PulseGenerator3FallingEdge** Pulse Generator 3 Falling
Default value	**GND**

Example 6.32. Usage of CxpLinkTrigger2Source

```
/* Set */ CxpLinkTrigger2Source = GND;
/* Get */ value_ = CxpLinkTrigger2Source;
```

6.5.11.4. CxpLinkTrigger3Source

Table 6.34. Parameter properties of CxpLinkTrigger3Source

Property	Value
Name	**CxpLinkTrigger3Source**
Display Name	**CXP Link Trigger 3 Source**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**GND** GND **CamAPulseGenerator0RisingEdge** Cam A Pulse Generator 0 Rising **CamAPulseGenerator1RisingEdge** Cam A Pulse Generator 1 Rising **CamAPulseGenerator2RisingEdge** Cam A Pulse Generator 2 Rising **CamAPulseGenerator3RisingEdge** Cam A Pulse Generator 3 Rising **CamAPulseGenerator0FallingEdge** Cam A Pulse Generator 0 Falling **CamAPulseGenerator1FallingEdge** Cam A Pulse Generator 1 Falling **CamAPulseGenerator2FallingEdge** Cam A Pulse Generator 2 Falling **CamAPulseGenerator3FallingEdge** Cam A Pulse Generator 3 Falling **CamBPulseGenerator0RisingEdge** Cam B Pulse Generator 0 Rising **CamBPulseGenerator1RisingEdge** Cam B Pulse Generator 1 Rising **CamBPulseGenerator2RisingEdge** Cam B Pulse Generator 2 Rising **CamBPulseGenerator3RisingEdge** Cam B Pulse Generator 3 Rising **CamBPulseGenerator0FallingEdge** Cam B Pulse Generator 0 Falling **CamBPulseGenerator1FallingEdge** Cam B Pulse Generator 1 Falling **CamBPulseGenerator2FallingEdge** Cam B Pulse Generator 2 Falling **CamBPulseGenerator3FallingEdge** Cam B Pulse Generator 3 Falling **BypassFrontGPI0RisingEdge** Bypass Front GPI 0 Rising **BypassFrontGPI0FallingEdge** Bypass Front GPI 0 Falling **BypassFrontGPI1RisingEdge** Bypass Front GPI 1 Rising **BypassFrontGPI1FallingEdge** Bypass Front GPI 1 Falling **BypassFrontGPI2RisingEdge** Bypass Front GPI 2 Rising **BypassFrontGPI2FallingEdge** Bypass Front GPI 2 Falling **BypassFrontGPI3RisingEdge** Bypass Front GPI 3 Rising **BypassFrontGPI3FallingEdge** Bypass Front GPI 3 Falling **PulseGeneratorRisingEdge** Pulse Generator 0 Rising **PulseGenerator1RisingEdge** Pulse Generator 1 Rising **PulseGenerator2RisingEdge** Pulse Generator 2 Rising **PulseGenerator3RisingEdge** Pulse Generator 3 Rising **PulseGenerator0FallingEdge** Pulse Generator 0 Falling **PulseGenerator1FallingEdge** Pulse Generator 1 Falling **PulseGenerator2FallingEdge** Pulse Generator 2 Falling **PulseGenerator3FallingEdge** Pulse Generator 3 Falling
Default value	**GND**

Example 6.33. Usage of CxpLinkTrigger3Source

```
/* Set */ CxpLinkTrigger3Source = GND;
/* Get */ value_ = CxpLinkTrigger3Source;
```

6.5.12. DigitalOutput

The CXP-12 Interface Card 4C has two front general purpose outputs (GPOs).

The trigger system of this applet provides several possibilities of mapping sources to the digital output signals:

- Pulse form generators

The pulse form generators are the main output sources of the trigger system. You can either directly bypass one of the four sources to a digital output or invert its signal. As this is a multi camera applet you can choose any of the trigger modules for each output. For example you can select Cam A pulse form generator 0 for all outputs.

- Ground or Vcc if a digital output is not used or you want to manually set the signal level.
- The input bypass

  The trigger system will ignore the signal length of the input signals. If you want to bypass an input directly to the output you can select the specific input or its inverted version.

6.5.12.1. TriggerOutSelectFrontGPO0

Select the source for the output on the repsective Front GPO.

Table 6.35. Parameter properties of TriggerOutSelectFrontGPO0

Property	Value	
Name	**TriggerOutSelectFrontGPO0**	
Display Name	**Trigger Out Select Front GPO 0**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**VCC**	VCC
	GND	GND
	CamAPulseGenerator0	Cam A Pulse Generator 0
	CamAPulseGenerator1	Cam A Pulse Generator 1
	CamAPulseGenerator2	Cam A Pulse Generator 2
	CamAPulseGenerator3	Cam A Pulse Generator 3
	NotCamAPulseGenerator0	Not Cam A Pulse Generator 0
	NotCamAPulseGenerator1	Not Cam A Pulse Generator 1
	NotCamAPulseGenerator2	Not Cam A Pulse Generator 2
	NotCamAPulseGenerator3	Not Cam A Pulse Generator 3
	CamBPulseGenerator0	Cam B Pulse Generator 0
	CamBPulseGenerator1	Cam B Pulse Generator 1
	CamBPulseGenerator2	Cam B Pulse Generator 2
	CamBPulseGenerator3	Cam B Pulse Generator 3
	NotCamBPulseGenerator0	Not Cam B Pulse Generator 0
	NotCamBPulseGenerator1	Not Cam B Pulse Generator 1
	NotCamBPulseGenerator2	Not Cam B Pulse Generator 2
	NotCamBPulseGenerator3	Not Cam B Pulse Generator 3
	BypassFrontGPI0	Bypass Front GPI 0
	NotBypassFrontGPI0	Not Bypass Front GPI 0
	BypassFrontGPI1	Bypass Front GPI 1
	NotBypassFrontGPI1	Not Bypass Front GPI 1
	BypassFrontGPI2	Bypass Front GPI 2
	NotBypassFrontGPI2	Not Bypass Front GPI 2
	BypassFrontGPI3	Bypass Front GPI 3
	NotBypassFrontGPI3	Not Bypass Front GPI 3
	PulseGenerator0	Pulse Generator 0
	PulseGenerator1	Pulse Generator 1
	PulseGenerator2	Pulse Generator 2
	PulseGenerator3	Pulse Generator 3
	NotPulseGenerator0	Not Pulse Generator 0
	NotPulseGenerator1	Not Pulse Generator 1
	NotPulseGenerator2	Not Pulse Generator 2
	NotPulseGenerator3	Not Pulse Generator 3
Default value	**NotCamAPulseGenerator0**	

Example 6.34. Usage of TriggerOutSelectFrontGPO0

```
/* Set */ TriggerOutSelectFrontGPO0 = NotCamAPulseGenerator0;
/* Get */ value_ = TriggerOutSelectFrontGPO0;
```

6.5.12.2. TriggerOutSelectFrontGPO1

Select the source for the output on the repsective Front GPO.

Table 6.36. Parameter properties of TriggerOutSelectFrontGPO1

Property	Value	
Name	**TriggerOutSelectFrontGPO1**	
Display Name	**Trigger Out Select Front GPO 1**	
Interface	**IEnumeration**	
Access policy	**Read/Write/Change**	
Visibility	**Beginner**	
Allowed values	**VCC**	VCC
	GND	GND
	CamAPulseGenerator0	Cam A Pulse Generator 0
	CamAPulseGenerator1	Cam A Pulse Generator 1
	CamAPulseGenerator2	Cam A Pulse Generator 2
	CamAPulseGenerator3	Cam A Pulse Generator 3
	NotCamAPulseGenerator0	Not Cam A Pulse Generator 0
	NotCamAPulseGenerator1	Not Cam A Pulse Generator 1
	NotCamAPulseGenerator2	Not Cam A Pulse Generator 2
	NotCamAPulseGenerator3	Not Cam A Pulse Generator 3
	CamBPulseGenerator0	Cam B Pulse Generator 0
	CamBPulseGenerator1	Cam B Pulse Generator 1
	CamBPulseGenerator2	Cam B Pulse Generator 2
	CamBPulseGenerator3	Cam B Pulse Generator 3
	NotCamBPulseGenerator0	Not Cam B Pulse Generator 0
	NotCamBPulseGenerator1	Not Cam B Pulse Generator 1
	NotCamBPulseGenerator2	Not Cam B Pulse Generator 2
	NotCamBPulseGenerator3	Not Cam B Pulse Generator 3
	BypassFrontGPI0	Bypass Front GPI 0
	NotBypassFrontGPI0	Not Bypass Front GPI 0
	BypassFrontGPI1	Bypass Front GPI 1
	NotBypassFrontGPI1	Not Bypass Front GPI 1
	BypassFrontGPI2	Bypass Front GPI 2
	NotBypassFrontGPI2	Not Bypass Front GPI 2
	BypassFrontGPI3	Bypass Front GPI 3
	NotBypassFrontGPI3	Not Bypass Front GPI 3
	PulseGenerator0	Pulse Generator 0
	PulseGenerator1	Pulse Generator 1
	PulseGenerator2	Pulse Generator 2
	PulseGenerator3	Pulse Generator 3
	NotPulseGenerator0	Not Pulse Generator 0
	NotPulseGenerator1	Not Pulse Generator 1
	NotPulseGenerator2	Not Pulse Generator 2
	NotPulseGenerator3	Not Pulse Generator 3
Default value	**NotCamBPulseGenerator0**	

Example 6.35. Usage of TriggerOutSelectFrontGPO1

```
/* Set */ TriggerOutSelectFrontGPO1 = NotCamBPulseGenerator0;
/* Get */ value_ = TriggerOutSelectFrontGPO1;
```

6.5.12.3. OutStatistics

The output statistics module counts the number of output pulses. The source can be selected by parameter *TriggerOutStatisticsSource*. The count value can be read from parameter *TriggerOutStatisticsPulseCount*. Parameter *TriggerOutStatisticsSource* also selects the source for the missing frame detection functionality.

6.5.12.3.1. TriggerExceededPeriodLimits

This read-only register has value **Yes** if the input signal frequency exceeded the maximum allowed frequency defined by parameter *TriggerOutputFrequency*. If the queue is enabled, the register is only set if the queue is full and cannot store a new input pulse. Reading the register will not reset it. It is required to reset the register by writing to *TriggerExceededPeriodLimitsClear*.

Table 6.37. Parameter properties of TriggerExceededPeriodLimits

Property	Value
Name	**`TriggerExceededPeriodLimits`**
Display Name	**`Trigger Exceeded Period Limits`**
Interface	**`IEnumeration`**
Access policy	**`Read-Only`**
Visibility	**`Beginner`**
Allowed values	**`Yes`** Yes **`No`** No

Example 6.36. Usage of TriggerExceededPeriodLimits

```
/* Get */ value_ = TriggerExceededPeriodLimits;
```

6.5.12.3.2. TriggerExceededPeriodLimitsClear

Reset *TriggerExceededPeriodLimits* with this parameter.

Table 6.38. Parameter properties of TriggerExceededPeriodLimitsClear

Property	Value
Name	**`TriggerExceededPeriodLimitsClear`**
Display Name	**`Clear Exceeded Period Limits Register`**
Interface	**`ICommand`**
Access policy	**`Write/Change`**
Visibility	**`Beginner`**

Example 6.37. Usage of TriggerExceededPeriodLimitsClear

```
/* Set */ TriggerExceededPeriodLimitsClear();
```

6.5.12.3.3. TriggerOutStatisticsSource

Table 6.39. Parameter properties of TriggerOutStatisticsSource

Property	Value
Name	**TriggerOutStatisticsSource**
Display Name	**Trigger Out Statistics Source**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**PulseGenerator0** Pulse Generator 0 **PulseGenerator1** Pulse Generator 1 **PulseGenerator2** Pulse Generator 2 **PulseGenerator3** Pulse Generator 3
Default value	**PulseGenerator0**

Example 6.38. Usage of TriggerOutStatisticsSource

```
/* Set */ TriggerOutStatisticsSource = PulseGenerator0;
/* Get */ value_ = TriggerOutStatisticsSource;
```

6.5.12.3.4. TriggerOutStatisticsPulseCount

Output pulse count read register. Select the source for the pulse counter by parameter *TriggerOutStatisticsSource*.

Table 6.40. Parameter properties of TriggerOutStatisticsPulseCount

Property	Value
Name	**TriggerOutStatisticsPulseCount**
Display Name	**Trigger Out Statistics Pulse Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum** 0 **Maximum** 65535 **Stepsize** 1
Unit of measure	**pulses**

Example 6.39. Usage of TriggerOutStatisticsPulseCount

```
/* Get */ value_ = TriggerOutStatisticsPulseCount;
```

6.5.12.3.5. TriggerOutStatisticsPulseCountClear

Output pulse count register clear.

Table 6.41. Parameter properties of TriggerOutStatisticsPulseCountClear

Property	Value
Name	**TriggerOutStatisticsPulseCountClear**
Display Name	**Trigger Out Statistics Pulse Count Clear**
Interface	**ICommand**
Access policy	**Write/Change**
Visibility	**Beginner**

Example 6.40. Usage of TriggerOutStatisticsPulseCountClear

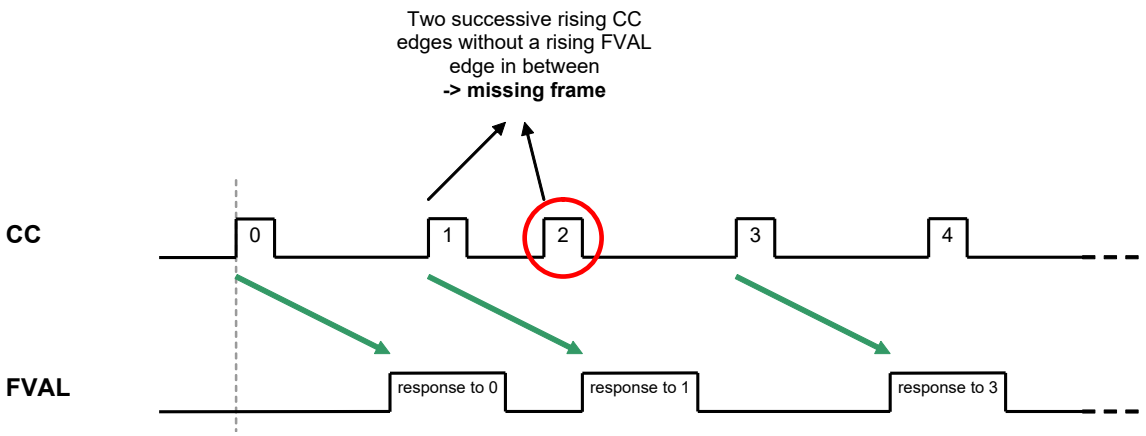
```
/* Set */ TriggerOutStatisticsPulseCountClear();
```

6.5.12.3.6. MissingCameraFrameResponse

This applet is equipped with a detection of missing camera frame responses to trigger pulses. If the camera does not send a frame for each output trigger pulse, the register is set to **Yes** until cleared by writing to parameter *MissingCameraFrameResponseClear*.

The idea of the frame loss detection is that for every trigger pulse generated by the trigger system, the camera sends a frame to the interface card. If a trigger pulse gets lost, or the camera cannot send a frame, this register is set to **Yes**. Technically, between two output signal edges, an incoming image has to exist. Or in other words: There must not be two or more successive trigger start edges without a valid frame in between. The behavior depends on the camera timing and interface. For very high frame rates the latency of sending an image can get bigger so that the detection mechanism of missing responses is not working. In this case you cannot use this parameter to detect missing triggers. Use the trigger and frame counters instead. You can also use the CXP source tag and trigger acknowledges to detect lost frames or trigger. The following figure illustrates the behavior.

Figure 6.21. Missing Camera Frame Response



The pulse form generator allocated to the camera trigger signal line carrying the image trigger pulses has to be selected by *TriggerOutStatisticsSource*. The missing frame response system might not work correctly for all camera models due to different timings.

Select Camera Control/Trigger Signal Line

Ensure you select the pulse form generator feeding the camera trigger signal line which carries the image trigger pulses by setting parameter *TriggerOutStatisticsSource* to the respective source.

Acquisition Start Before Trigger Activation

You must start the acquisition before activating the trigger. Otherwise, the trigger pulses sent will get lost. Any changes of the camera configuration might result in invalid data transfers.

Events for Missing Frame Response

If you want to monitor the exact moment of a missing frame response and the exact number of missing frames, use event *FrameTriggerMissed*. This approach exists for all 2 cameras connected to the applet addressed by its index.

Table 6.42. Parameter properties of MissingCameraFrameResponse

Property	Value
Name	**MissingCameraFrameResponse**
Display Name	**Missing Camera Frame Response**
Interface	**IEnumeration**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Yes** Yes **No** No

Example 6.41. Usage of MissingCameraFrameResponse

```
/* Get */ value_ = MissingCameraFrameResponse;
```

6.5.12.3.7. MissingCameraFrameResponseClear

Clear the *MissingCameraFrameResponse* flag by writing to this parameter.

Table 6.43. Parameter properties of MissingCameraFrameResponseClear

Property	Value
Name	**MissingCameraFrameResponseClear**
Display Name	**Clear Missing Camera Frame Response Register**
Interface	**ICommand**
Access policy	**Write/Change**
Visibility	**Beginner**

Example 6.42. Usage of MissingCameraFrameResponseClear

```
/* Set */ MissingCameraFrameResponseClear();
```

6.5.12.3.8. TriggerExceededPeriodLimits

The event is generated for each lost input trigger pulse. A trigger loss can occur if the input frequency is higher than the maximum allowed frequency set by parameter *TriggerOutputFrequency*. If the trigger queue is enabled, the events will only be generated if the queue is full i.e. for overflows. In generator and synchronized trigger modes, the events will not be generated. Except for the timestamp, the event has no additional data included.

For a general explanation on events see Event.

6.5.12.3.9. FrameTriggerMissed

The missing camera frame response event is generated for each camera output trigger pulse with no frame response. Please read the description of the detection and the documentation on how to configure the mechanism in Chapter 6.5.12.3.6.1 carefully. If the mechanism is compatible with the used camera and set up correct, the number of events is equal to the number of lost frames. Except for the timestamp, the event has no additional data included.

For a general explanation on events see Event.

6.5.13. OutputEvents

You can select one of the trigger outputs to generate a software event. i.e. a software callback function. Use parameter *TriggerOutputEventSelect* to specified the source pulse form generator for the event. The events name itself is *AcquisitionTrigger*.

For a general explanation on events check Event.

6.5.13.1. TriggerOutputEventSelect

Select the source for the output event *AcquisitionTrigger* respectively *AcquisitionTrigger* with this register. One of the pulse form generators can be selected.

Table 6.44. Parameter properties of TriggerOutputEventSelect

Property	Value
Name	**`TriggerOutputEventSelect`**
Display Name	**`Output Event Select`**
Interface	**`IEnumeration`**
Access policy	**`Read/Write/Change`**
Visibility	**`Beginner`**
Allowed values	**`PulseGenerator0`** Pulse Generator 0 **`PulseGenerator1`** Pulse Generator 1 **`PulseGenerator2`** Pulse Generator 2 **`PulseGenerator3`** Pulse Generator 3
Default value	**`PulseGenerator0`**

Example 6.43. Usage of TriggerOutputEventSelect

```
/* Set */ TriggerOutputEventSelect = PulseGenerator0;
/* Get */ value_ = TriggerOutputEventSelect;
```

6.5.13.2. AcquisitionTrigger

This event is generated for each start of an output trigger pulse. The respective pulse form generator has to be selected by parameter *TriggerOutputEventSelect*. Except for the timestamp, the event has no additional data included. Keep in mind that a high output frequency can cause high interrupt rates which might slow down the system.

Chapter 7. BufferStatus

The applet processes image data as fast as possible. Any image data sent by the camera is immediately processed and sent to the PC. The latency is minimal. In general, only one concurrent image line is stored and processed in the interface card. However, the transfer bandwidth to the PC via DMA channel can vary caused by interrupts, other hardware and the current CPU load. Furthermore, if operated in **selective mode**, it is possible to queue buffer slower than the camera offers new images and therefore generate an overflow condition on the frame grabber. Also, the camera frame rate can vary due to an fluctuating trigger. For these cases, the applet is equipped with a memory to buffer the input frames. The fill level of the buffer can be obtained by reading from parameter *FillLevel*.

In normal operation conditions the buffer will always remain almost empty. For fluctuating camera bandwidths or for short and fast acquisitions, the buffer can easily fill up quickly. Of course, the input bandwidth must not exceed the maximum bandwidth of the applet. Check Section 1.2, 'Bandwidth' for more information.

If the buffer's fill level reaches 100%, the applet is in overflow condition, as no more data can be buffered and camera data will be discarded. This can result in two different behaviors:

- Corrupted Frames:

  The transfer of a current frame is interrupted by an overflow. This means, the first pixels or lines of the frame were transfered into the buffer, but not the full frame. The output of the applet i.e. the DMA transfer will be shorter. The output image will not have it's full height. These images will be marked incomplete. Check the Basler GenTL documentation to learn on how to identify incompleted buffers (https://www.baslerweb.com/en/sales-support/downloads/document-downloads/cxp-gentl-producer-feature-documentation/).

- Lost Frames:

  A full camera frame was discarded due to a full buffer memory. No DMA transfer will exist for the discarded frame. This means the number of applet output images can differ from the number of applet input images.

The buffer overflow threshold *OverflowOnThreshold* and *OverflowSyncOnThreshold* default ensures that under normal conditions frames can be completed or will be fully dropped so that corrupted frames are avoide

A way to detect the overflows is to read parameter *Overflow* or check for event *Overflow*. Reading from the parameter will provide information about an overflow condition. As soon as the parameter is read, it will reset. Using the parameter an overflow condition can be detect, but it is not possible to obtain the exact image number and the moment. For this, the overflow event can be used.

7.1. FillLevel

The fill-level of the interface card buffers used in this applet can be read-out by use of this parameter. The value allows to check if the mean input bandwidth of the camera is to high to be processed with the applet.

Table 7.1. Parameter properties of FillLevel

Property	Value
Name	**`FillLevel`**
Display Name	**`Fill Level`**
Interface	**`IInteger`**
Access policy	**`Read-Only`**
Visibility	**`Beginner`**
Allowed values	**`Minimum 0`** **`Maximum 100`** **`Stepsize 1`**
Unit of measure	**`%`**

Example 7.1. Usage of FillLevel

```
/* Get */ value_ = FillLevel;
```

7.2. Overflow

If the applet runs into overflow, a value "1" can be read by the use of this parameter. Note that an overflow results in loss of images. To avoid overflows reduce the mean input bandwidth.

The parameter is reset at each readout cycle. The program microDisplayX will continuously poll the value, thus the occurrence of an overflow might not be visible in microDisplayX.

A more effective and robust way is to detect overflows is the use of the event system.

Table 7.2. Parameter properties of Overflow

Property	Value
Name	**Overflow**
Display Name	**Buffer overflow**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 1** **Stepsize 1**

Example 7.2. Usage of Overflow

```
/* Get */ value_ = Overflow;
```

7.3. OverflowOffThreshold

The Overflow state will be deactivated once the buffer Fillevel (*FillLevel*) will fall below this value. As long as the applet remains in overflow state all images arriving will be discarded. This will result in Overflow events with a set "lost" flag.

Table 7.3. Parameter properties of OverflowOffThreshold

Property	Value
Name	**OverflowOffThreshold**
Display Name	**Overflow Off Threshold**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 100.0** **Stepsize 0.5**
Default value	**50.0**

Example 7.3. Usage of OverflowOffThreshold

```
/* Set */ OverflowOffThreshold = 50.0;
/* Get */ value_ = OverflowOffThreshold;
```

7.4. OverflowOnThreshold

The applet will enter Overflow state once the buffer Fillevel exceeds this filllevel (*FillLevel*). If the overflow state is active images will be stopped imidiately. This may lead to an incomplete frame. Incomplete frames are marked incomplete in the image Tag and an overflow event can be generated.

Table 7.4. Parameter properties of OverflowOnThreshold

Property	Value
Name	**OverflowOnThreshold**
Display Name	**Overflow On Threshold**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 100.0** **Stepsize 0.5**
Default value	**99.5**

Example 7.4. Usage of OverflowOnThreshold

```
/* Set */ OverflowOnThreshold = 99.5;
/* Get */ value_ = OverflowOnThreshold;
```

7.5. OverflowSyncOnThreshold

The applet will enter Overflow state once the buffer fillevel (*FillLevel*) exceeds this filllevel and the currently arriving frame is stored to the buffer. If the applet remains in overflow state frames might be droped. If the buffer falls below this fillevel frames are accepted again. There is no hysteresis for this threshold.

Table 7.5. Parameter properties of OverflowSyncOnThreshold

Property	Value
Name	**OverflowSyncOnThreshold**
Display Name	**Overflow Sync On Threshold**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 100.0** **Stepsize 0.5**
Default value	**80.0**

Example 7.5. Usage of OverflowSyncOnThreshold

```
/* Set */ OverflowSyncOnThreshold = 80.0;
/* Get */ value_ = OverflowSyncOnThreshold;
```

7.6. OverflowEventSelect

The *Overflow* Event. Allows to generate events if one of the following conditions is meet.

Table 7.6. Event select for *Overflow*

Value	Description
Incomplete	Each incomplete frame will generate an Event containing the information that the frame is incomplete and the frameID
Lost	Each lost frame will generate an Event containing the information that the frame is lost and the frameID
IncompleteLost	Each lost or incomplete frame will generate an Event containing the information that the frame is lost/ incomplete and the frameID
OK	Each correct frame will generate an Event containing the information that the frame is transfered correct and the frameID of the frame
IncompleteOK	Each incomplete or correct frame will generate an Event containing the information that the frame is correct or incomplete and the frameID
LostOK	Each lost or correct frame will generate an Event containing the information that the frame is correct or lost and the frameID
All	Each frame will generate an Event containing the status(lost, incomplete or correct) of the frame and the frameID

Table 7.7. Parameter properties of OverflowEventSelect

Property	Value	
Name	**`OverflowEventSelect`**	
Display Name	**`Overflow Event Select`**	
Interface	**`IEnumeration`**	
Access policy	**`Read/Write/Change`**	
Visibility	**`Beginner`**	
Allowed values	**`Incomplete`**	Incomplete
	`Lost`	Lost
	`IncompleteLost`	Incomplete Lost
	`OK`	OK
	`IncompleteOK`	Incomplete OK
	`LostOK`	Lost OK
	`All`	All
Default value	**`IncompleteLost`**	

Example 7.6. Usage of OverflowEventSelect

```
/* Set */ OverflowEventSelect = IncompleteLost;
/* Get */ value_ = OverflowEventSelect;
```

7.7. OverflowEvents

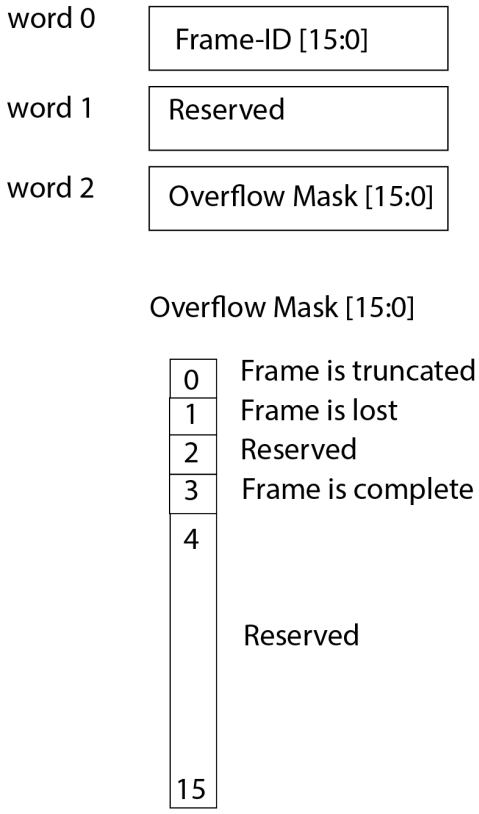
In programming or runtime environments, a callback function is a piece of executable code that is passed as an argument, which is expected to call back (execute) exactly that time an event is triggered. This applet can generate some software callback events based on the memory overflow condition as explained in the following section. These events are not related to a special camera functionality. Other event sources are described in additional sections of this document.

The Basler Framegrabber SDK and pylon SDK via GenTL enables an application to get these event notifications about certain state changes at the data flow from camera to RAM and the image and trigger processing as well. Please consult the Basler Framegrabber SDK, pylon SDK or GenTL documentation for more details concerning the implementation of this functionality.

7.7.1. Overflow

Overflow events are generated for each truncated, lost or complete frame. The selection can be done using *OverflowEventSelect*. The overflow event contains data, namely the type of overflow, the image number and the timestamp. The following figure illustrates the event data. Data is contained in a 64-bit data packet. The first 16 bits contain the frame-ID from the camera. Bits 32 to 47 provide an overflow mask.

Figure 7.1. Illustration of Overflow Data Packet



Note that the frame-ID is taken from the camera stream. See Section 1.5, 'Frame ID' for more information. The frame-ID is a 16-bit value. If its maximum is reached, the frame-ID starts at zero again. If the **frame truncated** flag is set, the frame with the frame-ID in the event is truncated i.e. it doesn't have its full length but is still transferred via DMA channel. If the **frame lost** flag is set, the frame with the frame-ID in the event was fully discarded. No DMA transfer exists for this frame. The **truncated frame** flag and the **frame lost** flag never occur for the same event.

Table 7.8. Event parameters of Overflow

Name	Interface	Description
EventOverflowFrameID	IInteger	Camera frame-ID for area scan applets or grabber frame-ID for line scan applets.
EventOverflowIsTru	IBoolean	Frame is truncated.
EventOverflowIsLost	IBoolean	Frame is lost.
EventOverflowIsCor	IBoolean	Frame is complete.

Chapter 8. ImageSelector

The Image Selector allows the user to cut out a period of *p* images from the image stream and select a particular image *n* from it.

The following example will explain the settings of *p* and *n* which represent the frame grabber parameters *ImageSelectPeriod* and *ImageSelect*. Suppose two frame grabbers being connected to a camera signal multiplexer, providing all camera images to both devices. Grabber 0 is required to process all even frames, while grabber 1 is required to process all odd frames. The settings will then be:

1. Grabber 0:
   - *ImageSelectPeriod* = 2

     ImageSelect = 0
2. Grabber 1:
   - *ImageSelectPeriod* = 2

     ImageSelect = 1

Ensure that both grabbers are used synchronously. This is possible with a triggered camera. To do so, initialize and configure both frame grabbers. Configure the camera for external trigger and the trigger system of master grabber which is directly connected to the camera. Next, the acquisitions of both grabbers have to be started and finally, the trigger generation has to be enabled, generally by setting *TriggerState* to active. Now the camera will start sending image data and the grabbers acquire those synchronously. More information can be found in the trigger chapter Chapter 6, '*Trigger*'.

8.1. ImageSelectPeriod

This parameter specifies the period length *p*. The parameter can be changed at any time. However, changing during acquisition can result in an asynchronous switching which will result in the loss of a synchronous grabbing. It is recommended to change the parameter only when the acquisition is stopped.

The parameter's value has to be greater than *ImageSelect*.

Table 8.1. Parameter properties of ImageSelectPeriod

Property	Value
Name	**ImageSelectPeriod**
Display Name	**Image Select Period**
Interface	**IInteger**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 1** **Maximum 256** **Stepsize 1**
Default value	**1**
Unit of measure	**image**

Example 8.1. Usage of ImageSelectPeriod

```
/* Set */ ImageSelectPeriod = 1;
/* Get */ value_ = ImageSelectPeriod;
```

8.2. ImageSelect

The parameter *ImageSelect* specifies a particular image from the image set defined by *ImageSelectPeriod*. This parameter can be changed at any time. However, changing during acquisition can result in an asynchronous switching which will result in the loss of a synchronous grabbing. It is recommended to change the parameter only when the acquisition is stopped.

The parameter's value has to be less than *ImageSelectPeriod*.

Table 8.2. Parameter properties of ImageSelect

Property	Value
Name	**ImageSelect**
Display Name	**Image Select**
Interface	**IInteger**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 255** **Stepsize 1**
Default value	**0**
Unit of measure	**image**

Example 8.2. Usage of ImageSelect

```
/* Set */ ImageSelect = 0;
/* Get */ value_ = ImageSelect;
```

Chapter 9. WhiteBalance

The applet enables a spectral adaptation of the image to the lighting situation of the application. The color values for the red, green and blue components can be individually enhanced or reduced by a scaling factor to adjust the spectral sensibility of the camera sensor.

The applet Acq_DualCXP12Area performs a Bayer de-mosaicing. The white balancing is performed prior to the Bayer de-mosaicing, to ensure the correction of the raw data and avoid subsequent faults during processing.

9.1. ScalingFactorGreen

Table 9.1. Parameter properties of ScalingFactorGreen

Property	Value
Name	**ScalingFactorGreen**
Display Name	**Scaling Factor Green**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 3.9990234375** **Stepsize 9.765625E-4**
Default value	**1.0**

Example 9.1. Usage of ScalingFactorGreen

```
/* Set */ ScalingFactorGreen = 1.0;
/* Get */ value_ = ScalingFactorGreen;
```

9.2. ScalingFactorRed

Table 9.2. Parameter properties of ScalingFactorRed

Property	Value
Name	**ScalingFactorRed**
Display Name	**Scaling Factor Red**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 3.9990234375** **Stepsize 9.765625E-4**
Default value	**1.0**

Example 9.2. Usage of ScalingFactorRed

```
/* Set */ ScalingFactorRed = 1.0;
/* Get */ value_ = ScalingFactorRed;
```

9.3. ScalingFactorBlue

Table 9.3. Parameter properties of ScalingFactorBlue

Property	Value
Name	**ScalingFactorBlue**
Display Name	**Scaling Factor Blue**
Interface	**IFloat**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**Minimum 0.0** **Maximum 3.9990234375** **Stepsize 9.765625E-4**
Default value	**1.0**

Example 9.3. Usage of ScalingFactorBlue

```
/* Set */ ScalingFactorBlue = 1.0;
/* Get */ value_ = ScalingFactorBlue;
```

Chapter 10. ColorConverter

The color converter module is used to convert the input pixel format to an output pixel format. The conversion is performed post to the Bayer de-mosicing and just before the lookup table.

This applet can perform the following conversions.

Table 10.1. Color Conversion

Input Format	Mono	RGB	Bayer	YCbCr
Output Format				
Mono	yes	yes	yes	N/A
RGB	yes	yes	yes	N/A
Bayer	N/A	N/A	yes	N/A
YCbCr	N/A	N/A	N/A	yes

By setting the input and output format the conversion is automatically applied if a conversion is possible. Otherwise the applet will output unchanged values. See *PixelFormat* and *Format*.

Chapter 11. OutputFormat

The following parameter can be used to configure the applet's image output format i.e. the format and bit alignment.

Automatic Adaptation of the Output Format by the GenTL Adaptor

The GenTL adaptor can automatically set the output format based on the camera settings and a given mapping table. Changing the output format of the applet might get overwritten by the GenTL adaptor on acquisition start. You can only set the output format if this automatic adaptation is disabled. See the GenTL documentation parameter **`AutomaticFormatControl`** for more details.

The automatic adaptation applies for parameters *PixelFormat*, *Format*, *BitAlignment* and *CustomBitShiftRight*.

Depending on the setting of GenTL interface parameter **`OutputPackedFormats`** the automatic adaptation will either use the same pixel format as coming from the camera or an unpacked PC output format. Changing the output format of the applet might get overwritten by the GenTL on acquisition start. You can only set the output format if this automatic adaptation is disabled. See the GenTL documentation parameter **`AutomaticFormatControl`** for more details.

Output Format Setting Defines GenTL Buffer Info

The parameters define the DMA output format and therefore the GenTL buffer info values to inform the consumer about the used output pixel format of the interface.

11.1. Format

Parameter *Format* is used to set and determine the output formats of the DMA channels. An output format value specifies the number of bits and the color format of the output.

This applet has an internal processing bit width of 16 bits. Any selected camera pixel format is mapped to this internal bit width. Check the camera parameter section to learn about the mapping of the camera bits to the internal bit width. For a definition on how to map the internal bits to the output bits, check parameter *BitAlignment*.

Moreover, the color converter of this applet can convert between different color formats of the input and output. Check Chapter 10, '*ColorConverter*' for more information.

This applet supports the following output formats:

- **BGR8** and **RGB8**: 24 bit BGR/RGB color format with 8 bit/component.
- **BGRa8** and **RGBa8**: Color format with 8 bit/component. Component "a" has value zero.
- **BGR10p** and **RGB10p**: 30 bit BGR/RGB color format with 10 bit/component.

30 Bit Output Format

Note that in the 30 bit output format 1 pixel and its 3 color components are distributed over multiple bytes. Also, two successive pixel might share one byte. The pixel are directly aligned in memory. Thus 8 successive color components are stored in 10 byte. The DMA transfer might be filled with random content for the last bytes.

- **BGR12p** and **RGB12p**: 36 bit BGR/RGB color format with 12 bit/component.

36 Bit Output Format

Note that in the 36 bit output format 1 pixel and its 3 color components are distributed over multiple bytes. Also, two successive pixel might share one byte. The pixel are directly aligned in memory. Thus 2 successive color components are stored in 3 byte or two pixel in 9 Byte. The DMA transfer might be filled with random content for the last bytes.

- **BGR14p** and **RGB14p**: 42 bit BGR/RGB color format with 14 bit/component.

42 Bit Output Format

Note that in the 42 bit output format 1 pixel and its 3 color components are distributed over multiple bytes. Also, two successive pixel might share one byte. The pixel are directly aligned in memory. Thus 4 successive color components are stored in 7 byte or four pixel in 21 Byte. The DMA transfer might be filled with random content for the last bytes.

- **BGR16** and **RGB16**: 48 bit BGR/RGB color format with 16 bit/component.

BGR vs. RGB Memory Alignement

Note that the color components are either written to the PC buffer in the common blue, green, red (BGR) or red, green, blue order. So either the blue or red color component is at the lower memory address.

- **Mono8**: 8 bit grayscale format
- **Mono10p**: 10 bit grayscale format

10 Bit Output Format

Note that in the 10 bit output format 1 pixel is distributed over more than one byte. Also, two successive pixel share one byte. The pixel are directly aligned in memory. Thus 8 successive pixel are stored in 10 byte. The DMA transfer might be filled with random content for the last bytes.

- **Mono12p**: 12 bit grayscale format

12 Bit Output Format

Note that in the 12 bit output format 1 pixel is distributed over more than one byte. Also, two successive pixel share the same byte. The pixel are directly aligned in memory. Thus 2 successive pixel are stored in 3 byte. The DMA transfer might be filled with random content for the last bytes.

- **Mono14p**: 14 bit grayscale format

14 Bit Output Format

Note that in the 14 bit output format 1 pixel is distributed over more than one byte. Also, two successive pixel share the same byte. The pixel are directly aligned in memory. Thus 12 successive pixel are stored in 21 byte. The DMA transfer might be filled with random content for the last bytes.

- **Mono16**: 16 bit grayscale format

DMA Bandwidth

Keep in mind that for the 16 bit output mode, the DMA bandwidth might not be sufficient to process the camera input data. Check Section 1.2, 'Bandwidth' for more information.

- **BayerGR8**, **BayerRG8**, **BayerGB8** and **BayerBG8**: 8 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.

- **BayerGR10p**, **BayerRG10p**, **BayerGB10p** and **BayerBG10p**: 10 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.

10 Bit Output Format

Note that in the 10 bit output format 1 pixel is distributed over more than one byte. Also, two successive pixel share one byte. The pixel are directly aligned in memory. Thus 8 successive pixel are stored in 10 byte. The DMA transfer might be filled with random content for the last bytes.

- **BayerGR12p**, **BayerRG12p**, **BayerGB12p** and **BayerBG12p**: 12 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.

12 Bit Output Format

Note that in the 12 bit output format 1 pixel is distributed over more than one byte. Also, two successive pixel share the same byte. The pixel are directly aligned in memory. Thus 2 successive pixel are stored in 3 byte. The DMA transfer might be filled with random content for the last bytes.

- **BayerGR14p**, **BayerRG14p**, **BayerGB14p** and **BayerBG14p**: 14 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.

14 Bit Output Format

Note that in the 14 bit output format 1 pixel is distributed over more than one byte. Also, two successive pixel share the same byte. The pixel are directly aligned in memory. Thus 12 successive pixel are stored in 21 byte. The DMA transfer might be filled with random content for the last bytes.

- **BayerGR16**, **BayerRG16**, **BayerGB16** and **BayerBG16**: 16 bit Bayer format Green-followed-by-Red, Red-followed-by-Green, Green-followed-by-Blue and Blue-followed-by-Green.

DMA Bandwidth

Keep in mind that for the 16 bit output mode, the DMA bandwidth might not be sufficient to process the camera input data. Check Section 1.2, 'Bandwidth' for more information.

- **YCbCr422_8**: YUV 422 output in 8 bit per component.

Table 11.1. Parameter properties of Format

Property	Value
Name	**Format**
Display Name	**Output Format**
Interface	**IEnumeration**
Access policy	**Read/Write**
Visibility	**Beginner**
Allowed values	**Mono8** Mono 8 **Mono10p** Mono 10p **Mono12p** Mono 12p **Mono14p** Mono 14p **Mono16** Mono 16 **BGR8** BGR 8bit **BGR10p** BGR 10bit **BGR12p** BGR 12bit **BGR14p** BGR 14p **BGR16** BGR 16bit **RGB8** RGB 8 **RGB10p** RGB 10p **RGB12p** RGB 12p **RGB14p** RGB 14p **RGB16** RGB 16 **BGRa8** BGRA 8 **RGBa8** RGBA 8 **BayerGR8** Bayer GR 8 **BayerGR10p** Bayer GR 10p **BayerGR12p** Bayer GR 12p **BayerGR14p** Bayer GR 14p **BayerGR16** Bayer GR 16 **BayerRG8** Bayer RG 8 **BayerRG10p** Bayer RG 10p **BayerRG12p** Bayer RG 12p **BayerRG14p** Bayer RG 14p **BayerRG16** Bayer RG 16 **BayerGB8** Bayer GB 8 **BayerGB10p** Bayer GB 10p **BayerGB12p** Bayer GB 12p **BayerGB14p** Bayer GB 14p **BayerGB16** Bayer GB 16 **BayerBG8** Bayer BG 8 **BayerBG10p** Bayer BG 10p **BayerBG12p** Bayer BG 12p **BayerBG14p** Bayer BG 14p **BayerBG16** Bayer BG 16 **YCbCr422_8** YCbCr422_8
Default value	**Mono8**

Example 11.1. Usage of Format

```
/* Set */ Format = Mono8;
/* Get */ value_ = Format;
```

11.2. BitAlignment

The bit alignment is used to map the pixel bits of the internal processing with a depth of 16 bit to the configured DMA output bit depth defined by parameter *Format*.

You can select three different modes: Left aligned, right aligned and a custom shift mode. If you select left aligned, the applet will map the upper bits of the internal processing bit width to the available output bits. If you select right aligned, the applet will map the lower bits of the internal processing bit width to the available output bits. If you want to define a custom bit shift, you'll need to set the parameter to CustomBitShift and use parameter *CustomBitShiftRight* to define the bit shift.

Keep in mind that the internal processing bit width has nothing to do with the camera pixel format. Check the camera parameter section to learn about the mapping of the camera bits to the internal bit width.

Table 11.2. Parameter properties of BitAlignment

Property	Value
Name	**BitAlignment**
Display Name	**Bit Alignment**
Interface	**IEnumeration**
Access policy	**Read/Write/Change**
Visibility	**Beginner**
Allowed values	**LeftAligned** Left Aligned **RightAligned** Right Aligned **CustomBitShift** Custom Bit Shift
Default value	**LeftAligned**

Example 11.2. Usage of BitAlignment

```
/* Set */ BitAlignment = LeftAligned;
/* Get */ value_ = BitAlignment;
```

11.3. PixelDepth

The pixel depth read-only parameter is used to determine the number of bits used to process a pixel in the applet. It represents the internal bit width.

Table 11.3. Parameter properties of PixelDepth

Property	Value
Name	**PixelDepth**
Display Name	**Pixel Depth**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum** 0 **Maximum** 128 **Stepsize** 1
Unit of measure	**bit**

Example 11.3. Usage of PixelDepth

```
/* Get */ value_ = PixelDepth;
```

11.4. CustomBitShiftRight

This parameter can only be used if parameter *BitAlignment* is set to **CustomBitShift**. If it is enabled, you can define a custom right bit shift value for the DMA output of the interface card. A shift of 0 means that the most significant bits (MSB) of the internal processing bit width are mapped to the output MSB. For example, if the applet has an internal processing bit width of 12 bit and you select a 10 bit output, the upper 10 bits are mapped to the output. If you select however a bit width of two, the lower 10 bits are mapped to the output. Note that this applet has an internal bit width of 16 bits.

Table 11.4. Parameter properties of CustomBitShiftRight

Property	Value
Name	**`CustomBitShiftRight`**
Display Name	**`Bit Shift Right`**
Interface	**`IInteger`**
Access policy	**`Read/Write/Change`**
Visibility	**`Beginner`**
Allowed values	**`Minimum 0`** **`Maximum 15`** **`Stepsize 1`**
Default value	**`0`**
Unit of measure	**`bit`**

Example 11.4. Usage of CustomBitShiftRight

```
/* Set */ CustomBitShiftRight = 0;
/* Get */ value_ = CustomBitShiftRight;
```

Chapter 12. Miscellaneous

This category summarizes other read and write parameters such as the camera status, buffer fill levels, DMA transfer lengths, and time stamps.

12.1. Version

The category provides version information.

12.1.1. AppletVersion

This parameter indicates the version number of the applet. Report this value when contacting the Basler support.

Table 12.1. Parameter properties of AppletVersion

Property	Value
Name	**AppletVersion**
Display Name	**Applet Version**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 256** **Stepsize 1**

Example 12.1. Usage of AppletVersion

```
/* Get */ value_ = AppletVersion;
```

12.1.2. AppletRevision

This parameter indicates the revision number of the applet. Report this value when contacting the Basler support.

Table 12.2. Parameter properties of AppletRevision

Property	Value
Name	**AppletRevision**
Display Name	**Applet Revision**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Beginner**
Allowed values	**Minimum 0** **Maximum 256** **Stepsize 1**

Example 12.2. Usage of AppletRevision

```
/* Get */ value_ = AppletRevision;
```

12.1.3. VisualAppletsBuildVersion

Returns the VisualApplets version used to build the applets.

Table 12.3. Parameter properties of VisualAppletsBuildVersion

Property	Value
Name	**`VisualAppletsBuildVersion`**
Display Name	**`Visual Applets Build Version`**
Interface	**`IString`**
Access policy	**`Read-Only`**
Visibility	**`Beginner`**

Example 12.3. Usage of VisualAppletsBuildVersion

```
/* Get */ value_ = VisualAppletsBuildVersion;
```

Chapter 13. BoardStatus

This category gives information about the current framegrabber board status. For example, the number of used PCIe lanes, or the mapping of the physical and logical CXP ports. For imaWorx and imaFLex, it also shows if a trigger board is connected.

13.1. SystemmonitorMappedToFgPort

Indicates the frame grabber port mapping. Range: between 0 and 3.

Table 13.1. Parameter properties of SystemmonitorMappedToFgPort

Property	Value
Name	**SystemmonitorMappedToFgPort**
Display Name	**Systemmonitor Mapped to Fg Port**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 13.1. Usage of SystemmonitorMappedToFgPort

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorMappedToFgPortSelector = i;
    value_ = SystemmonitorMappedToFgPort;
}
```

13.2. SystemmonitorCurrentLinkSpeed

Returns the current link width of the frame grabber representing the number of PCIe lanes that are used for data transfer. This is a value that should correspond to the number of hardware lanes the frame grabber is requiring, otherwise the possible maximum of DMA bandwidth can be reduced drastically.

Table 13.2. Parameter properties of SystemmonitorCurrentLinkSpeed

Property	Value
Name	**SystemmonitorCurrentLinkSpeed**
Display Name	**Systemmonitor Current Link Speed**
Interface	**IFloat**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0.0** **Maximum 1000.0** **Stepsize 0.0**
Unit of measure	**Gb/s**

Example 13.2. Usage of SystemmonitorCurrentLinkSpeed

```
/* Get */ value_ = SystemmonitorCurrentLinkSpeed;
```

13.3. SystemmonitorPcieTrainedPayloadSize

Returns the PCIe packet size that was evaluated during the training period at boot-time.

Table 13.3. Parameter properties of SystemmonitorPcieTrainedPayloadSize

Property	Value
Name	**SystemmonitorPcieTrainedPayloadSize**
Display Name	**Systemmonitor PCIe Trained Payload Size**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 1024** **Stepsize 0**
Unit of measure	**byte**

Example 13.3. Usage of SystemmonitorPcieTrainedPayloadSize

```
/* Get */ value_ = SystemmonitorPcieTrainedPayloadSize;
```

13.4. SystemmonitorPcieTrainedRequestSize

Returns the size (in bytes) of the PCIe packets payload that are used for the data transmission between the frame grabber and the PCIe bridge.

Table 13.4. Parameter properties of SystemmonitorPcieTrainedRequestSize

Property	Value
Name	**SystemmonitorPcieTrainedRequestSize**
Display Name	**Systemmonitor PCIe Trained Request Size**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 4096** **Stepsize 0**
Unit of measure	**byte**

Example 13.4. Usage of SystemmonitorPcieTrainedRequestSize

```
/* Get */ value_ = SystemmonitorPcieTrainedRequestSize;
```

13.5. CxpInputMappedToFWPortPort

This parameter returns the firmware CXP channel, which is currently monitored by the module. There is not necessarily a one-by-one mapping between firmware port (i.e. the camera port resource) and frame grabber port (i.e. the physical connector). Instead, the mapping can be any permutation. The software discovery process reorders the channels and ports to achieve correct virtual interconnect. Range: 0 to 3 (2 bit).

Table 13.5. Parameter properties of CxpInputMappedToFWPortPort

Property	Value
Name	**CxpInputMappedToFWPortPort**
Display Name	**CXP Input Mapped to Firmware Port Port**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 13.5. Usage of CxpInputMappedToFWPortPort

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpInputMappedToFWPortPortSelector = i;
    value_ = CxpInputMappedToFWPortPort;
}
```

Chapter 14. Errors

This category gives information about the current error status. It shows error counters for different error types, such as packet errors, missing connection, undefined data or overtriggering. Additionally, it reports warning type errors, like the number of both corrected and uncorrected packets.

14.1. SystemmonitorDecoder8b10bError

Link stability counter. It is incremented when the number of measured symbols received by the channel transceiver are not in 8b10b encoding or/and have wrong disparity. Range: 0 to (2^48 - 1) (48 bit).

Table 14.1. Parameter properties of SystemmonitorDecoder8b10bError

Property	Value
Name	**SystemmonitorDecoder8b10bError**
Display Name	**Systemmonitor Decoder 8b10b Error**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.1. Usage of SystemmonitorDecoder8b10bError

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorDecoder8b10bErrorSelector = i;
    value_ = SystemmonitorDecoder8b10bError;
}
```

14.2. SystemmonitorByteAlignment8b10bLocked

Monitors whether the clock recovery has worked and valid 8b/10b signals are recognized.

Table 14.2. Parameter properties of SystemmonitorByteAlignment8b10bLocked

Property	Value
Name	**SystemmonitorByteAlignment8b10bLocked**
Display Name	**Systemmonitor Byte Alignment 8B 10 B Locked**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.2. Usage of SystemmonitorByteAlignment8b10bLocked

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorByteAlignment8b10bLockedSelector = i;
    value_ = SystemmonitorByteAlignment8b10bLocked;
}
```

14.3. SystemmonitorRxStreamIncompleteCount

Returns the number of received incomplete stream counts. Range: between 0 and 8191 in steps of 1.

Table 14.3. Parameter properties of SystemmonitorRxStreamIncompleteCount

Property	Value
Name	**SystemmonitorRxStreamIncompleteCount**
Display Name	**Systemmonitor Rx Stream Incomplete Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.3. Usage of SystemmonitorRxStreamIncompleteCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorRxStreamIncompleteCountSelector = i;
    value_ = SystemmonitorRxStreamIncompleteCount;
}
```

14.4. SystemmonitorRxUnknownDataReceivedCount

Returns the number of received unknown data, i.e. packets received that aren't defined in the CXP standard. Range: between 0 and 8191 in steps of 1.

Table 14.4. Parameter properties of SystemmonitorRxUnknownDataReceivedCount

Property	Value
Name	**SystemmonitorRxUnknownDataReceivedCount**
Display Name	**Systemmonitor Rx Unknown Data Received Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.4. Usage of SystemmonitorRxUnknownDataReceivedCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorRxUnknownDataReceivedCountSelector = i;
    value_ = SystemmonitorRxUnknownDataReceivedCount;
}
```

14.5. CxpOvertriggerRequestPulseCount

This parameter counts the trigger requests that were skipped, because the transmitter was still busy by sending the previous trigger packet. See CXP 2.0 standard, chapter 9.3.2. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range: 0 to 4095 (12 bit). Bit 12 indicates an overflow.

Table 14.5. Parameter properties of CxpOvertriggerRequestPulseCount

Property	Value
Name	**CxpOvertriggerRequestPulseCount**
Display Name	**CXP Overtrigger Request Pulse Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.5. Usage of CxpOvertriggerRequestPulseCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpOvertriggerRequestPulseCountSelector = i;
    value_ = CxpOvertriggerRequestPulseCount;
}
```

14.6. CxpTriggerAckMissingCount

This parameter counts the situations in which a trigger packet was sent, but no acknowledgment packet was received for it yet, which then led to a timeout (480ns for 1-6Gb/s, 240ns for 10-12.5Gb/s). See CXP 2.0 standard, chapter 9.3.2. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range: 0 to 8191 (13 bit).

Table 14.6. Parameter properties of CxpTriggerAckMissingCount

Property	Value
Name	**CxpTriggerAckMissingCount**
Display Name	**CXP Lost Trigger ACK Missing Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.6. Usage of CxpTriggerAckMissingCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpTriggerAckMissingCountSelector = i;
    value_ = CxpTriggerAckMissingCount;
}
```

14.7. CxpControlAckLostCount

This parameter counts situations in which a control packet was sent but no acknowledgment packet was received for it yet and the timeout of 200 ms is reached. See CXP 2.0 standard, chapter 9.6.1.1. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 14.7. Parameter properties of CxpControlAckLostCount

Property	Value
Name	**CxpControlAckLostCount**
Display Name	**CXP Control ACK Lost Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.7. Usage of CxpControlAckLostCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpControlAckLostCountSelector = i;
    value_ = CxpControlAckLostCount;
}
```

14.8. CxpControlTagErrorCount

This parameter counts situations in which an acknowledgment for a control packet was received with a tag that doesn't match the expected tag sent in the corresponding request control packet. See CXP 2.0 standard, chapter 9.6.1.2. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 14.8. Parameter properties of CxpControlTagErrorCount

Property	Value
Name	**CxpControlTagErrorCount**
Display Name	**CXP Control Tag Error Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.8. Usage of CxpControlTagErrorCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpControlTagErrorCountSelector = i;
    value_ = CxpControlTagErrorCount;
}
```

14.9. CxpControlAckIncompleteCount

This parameter counts situations in which an incorrectly formatted acknowledgment for a control packet was received. Incorrectly formatted means that e.g. the end of packet indicator is missing etc. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 14.9. Parameter properties of CxpControlAckIncompleteCount

Property	Value
Name	**CxpControlAckIncompleteCount**
Display Name	**CXP Control ACK Incomplete Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.9. Usage of CxpControlAckIncompleteCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpControlAckIncompleteCountSelector = i;
    value_ = CxpControlAckIncompleteCount;
}
```

14.10. CxpHeartbeatIncompleteCount

This parameter counts situations in which the received heart beat packet is incomplete, e.g. it misses the end of the packet indicator. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 14.10. Parameter properties of CxpHeartbeatIncompleteCount

Property	Value
Name	**CxpHeartbeatIncompleteCount**
Display Name	**CXP Heartbeat Incomplete Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.10. Usage of CxpHeartbeatIncompleteCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpHeartbeatIncompleteCountSelector = i;
    value_ = CxpHeartbeatIncompleteCount;
}
```

14.11. CxpHeartbeatMaxPeriodViolationCount

The heartbeat period is defined in CXP 2.0 standard as 100ms maximum, i.e. within that time at least 1 heartbeat packet must be sent by the camera. This parameter counts the situations in which heartbeat packets exceeded this timeout (100ms). Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 14.11. Parameter properties of CxpHeartbeatMaxPeriodViolationCount

Property	Value
Name	**CxpHeartbeatMaxPeriodViolationCount**
Display Name	**CXP Hearbeat Max Period Violation Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.11. Usage of CxpHeartbeatMaxPeriodViolationCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpHeartbeatMaxPeriodViolationCountSelector = i;
    value_ = CxpHeartbeatMaxPeriodViolationCount;
}
```

14.12. PacketTagErrorCount

The parameter counts the number of lost CXP stream packets.

Table 14.12. Parameter properties of PacketTagErrorCount

Property	Value
Name	**PacketTagErrorCount**
Display Name	**Packet Tag Error Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 4095** **Stepsize 1**

Example 14.12. Usage of PacketTagErrorCount

```
/* Get */ value_ = PacketTagErrorCount;
```

14.13. SystemmonitorPacketbufferOverflowCount

This parameter counts the number of overflows that occur due to not correctly aligned package orders.

Table 14.13. Parameter properties of SystemmonitorPacketbufferOverflowCount

Property	Value
Name	**SystemmonitorPacketbufferOverflowCount**
Display Name	**Systemmonitor Packetbuffer Overflow Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 4095** **Stepsize 1**

Example 14.13. Usage of SystemmonitorPacketbufferOverflowCount

```
/* Get */ value_ = SystemmonitorPacketbufferOverflowCount;
```

14.14. SystemmonitorPacketbufferOverflowSource

This parameter returns the port that has overflows due to not correctly aligned package order.

Table 14.14. Parameter properties of SystemmonitorPacketbufferOverflowSource

Property	Value
Name	**SystemmonitorPacketbufferOverflowSource**
Display Name	**Systemmonitor Packetbuffer Overflow Source**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 15** **Stepsize 1**

Example 14.14. Usage of SystemmonitorPacketbufferOverflowSource

```
/* Get */ value_ = SystemmonitorPacketbufferOverflowSource;
```

14.15. CxpImageTagErrorCount

This parameter returns the number of image tag errors (jumps) in the CXP headers.

Table 14.15. Parameter properties of CxpImageTagErrorCount

Property	Value
Name	**CxpImageTagErrorCount**
Display Name	**CXP Image Tag Error Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 8191** **Stepsize 1**

Example 14.15. Usage of CxpImageTagErrorCount

```
/* Get */ value_ = CxpImageTagErrorCount;
```

14.16. CxpStreamIDErrorCount

The parameter counts how often the received stream ID value in the stream packets mismatches the stream ID value specified in the image header. The parameter is 13 bit wide, where the bits [11:0] represent the actual counter value and the bit [12] stands for the counter overflow. When the overflow bit is set, the counter value shall be treated as don't care. Range: 0 to 8191 (13 bit).

Table 14.16. Parameter properties of CxpStreamIDErrorCount

Property	Value
Name	**CxpStreamIDErrorCount**
Display Name	**CXP Stream ID Error Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 8191** **Stepsize 1**

Example 14.16. Usage of CxpStreamIDErrorCount

```
/* Get */ value_ = CxpStreamIDErrorCount;
```

14.17. CxpCameraMarkerErrorCount

This parameter counts how often the sequence of the CXP stream marker and the header or the line markers were incorrect. The parameter is 13 bit wide, where the bits [11:0] represent the actual counter value and the bit [12] stands for the counter overflow. When the overflow bit is set, the counter value shall be treated as don't care. Range: 0 to 8192 (13 bit).

Table 14.17. Parameter properties of CxpCameraMarkerErrorCount

Property	Value
Name	**CxpCameraMarkerErrorCount**
Display Name	**CXP Camera Marker Error Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 8191** **Stepsize 1**

Example 14.17. Usage of CxpCameraMarkerErrorCount

```
/* Get */ value_ = CxpCameraMarkerErrorCount;
```

14.18. CxpCameraUnexpectedStartupDataStatus

This parameter detects the error situation in which the first data value after the operator reset was unexpected, i.e. no image header has been received. This situation can happen due to a buggy implementation of the camera, frame grabber firmware or wrong software control of the discovery procedure. Also, a hardware defect of the camera could theoretically cause such a situation. Range: NO or YES.

Table 14.18. Parameter properties of CxpCameraUnexpectedStartupDataStatus

Property	Value
Name	**CxpCameraUnexpectedStartupDataStatus**
Display Name	**CXP Camera Unexpected Startup Data Status**
Interface	**IEnumeration**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Yes** Yes **No** No

Example 14.18. Usage of CxpCameraUnexpectedStartupDataStatus

```
/* Get */ value_ = CxpCameraUnexpectedStartupDataStatus;
```

14.19. CxpCameraFrameLostCount

This parameter counts the frames that were lost during acquisition and aren't sent into the applet image pipeline. Frames are lost when an error in the image header is detected or when a frame overlaps with another frame. The parameter is 25 bit wide where the bits [23:0] represent the actual counter value and bit [24] stands for the counter overflow. When the overflow bit is set, the counter value shall be treated as don't care. Range 0 to 33554431 (25 bit).

Table 14.19. Parameter properties of CxpCameraFrameLostCount

Property	Value
Name	**CxpCameraFrameLostCount**
Display Name	**CXP Camera Frame Lost Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum** 0 **Maximum** 33554431 **Stepsize** 1

Example 14.19. Usage of CxpCameraFrameLostCount

```
/* Get */ value_ = CxpCameraFrameLostCount;
```

14.20. CxpCameraFrameCorruptCount

This parameter counts the corrupted frames during acquisition. Corrupted frames are frames with error pixels which are sent to the applet image pipeline. The parameter is 25 bit wide where the bits [23:0] represent the actual counter value and bit [24] stands for the counter overflow. When the overflow bit is set, the counter value shall be treated as don't care. Range 0 to 33554431 (25 bit).

Table 14.20. Parameter properties of CxpCameraFrameCorruptCount

Property	Value
Name	**CxpCameraFrameCorruptCount**
Display Name	**CXP Camera Frame Corrupt Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 33554431** **Stepsize 1**

Example 14.20. Usage of CxpCameraFrameCorruptCount

```
/* Get */ value_ = CxpCameraFrameCorruptCount;
```

14.21. CrcErrors

This category gives information about packet CRC errors detected for stream packets and control packets.

14.21.1. SystemmonitorRxPacketCrcErrorCount

Returns the number of received packet CRC errors. Range: between 0 and 8191 in steps of 1.

Table 14.21. Parameter properties of SystemmonitorRxPacketCrcErrorCount

Property	Value
Name	**SystemmonitorRxPacketCrcErrorCount**
Display Name	**Systemmonitor Rx Packet CRC Error Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.21. Usage of SystemmonitorRxPacketCrcErrorCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorRxPacketCrcErrorCountSelector = i;
    value_ = SystemmonitorRxPacketCrcErrorCount;
}
```

14.21.2. CxpStreamPacketCrcError

This parameter returns information whether there were CRC errors in received stream packets. Range 0 (NO) to 1 (YES).

Table 14.22. Parameter properties of CxpStreamPacketCrcError

Property	Value
Name	**CxpStreamPacketCrcError**
Display Name	**CXP Stream Packet CRC Error**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.22. Usage of CxpStreamPacketCrcError

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpStreamPacketCrcErrorSelector = i;
    value_ = CxpStreamPacketCrcError;
}
```

14.21.3. CxpControlAckPacketCrcError

This parameter returns information whether there were CRC errors in received control acknowledgement packets. Range 0 (NO) to 1 (YES).

Table 14.23. Parameter properties of CxpControlAckPacketCrcError

Property	Value
Name	**CxpControlAckPacketCrcError**
Display Name	**CXP Control ACK Packet CRC Error**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.23. Usage of CxpControlAckPacketCrcError

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpControlAckPacketCrcErrorSelector = i;
    value_ = CxpControlAckPacketCrcError;
}
```

14.22. LengthErrors

This category gives information about packet length mismatches for different types of packets.

14.22.1. SystemmonitorRxLengthErrorCount

This parmeter counts how often the length of a CXP packet doesn't correspond to what is specified in the header and returns the number of length errors. Range: between 0 and 8191 in steps of 1.

Table 14.24. Parameter properties of SystemmonitorRxLengthErrorCount

Property	Value
Name	**SystemmonitorRxLengthErrorCount**
Display Name	**Systemmonitor Rx Length Error Count**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.24. Usage of SystemmonitorRxLengthErrorCount

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorRxLengthErrorCountSelector = i;
    value_ = SystemmonitorRxLengthErrorCount;
}
```

14.22.2. CxpStreamPacketLengthError

This parameter returns information whether a length error in the stream packets was detected. Range: 0 (NO) to 1 (YES).

Table 14.25. Parameter properties of CxpStreamPacketLengthError

Property	Value
Name	**CxpStreamPacketLengthError**
Display Name	**CXP Stream Packet Length Error**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.25. Usage of CxpStreamPacketLengthError

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpStreamPacketLengthErrorSelector = i;
    value_ = CxpStreamPacketLengthError;
}
```

14.23. ReceivedPacketsCorrected

This category gives information about errors which occurred in received packets which have been corrected.

14.23.1. CxpErrorCorrected

This parameter counts errors received in packet headers and trailers that were corrected. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 14.26. Parameter properties of CxpErrorCorrected

Property	Value
Name	**CxpErrorCorrected**
Display Name	**CXP Error Corrected**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.26. Usage of CxpErrorCorrected

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedSelector = i;
    value_ = CxpErrorCorrected;
}
```

14.23.2. CxpErrorCorrectedTrigger

This parameter returns the information whether errors were corrected in received trigger packets. Range 0 (NO) to 1 (YES).

Table 14.27. Parameter properties of CxpErrorCorrectedTrigger

Property	Value
Name	**CxpErrorCorrectedTrigger**
Display Name	**CXP Error Corrected Trigger**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.27. Usage of CxpErrorCorrectedTrigger

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedTriggerSelector = i;
    value_ = CxpErrorCorrectedTrigger;
}
```

14.23.3. CxpErrorCorrectedTriggerAck

This parameter returns the information whether errors were corrected in received trigger acknowledge packets. Range 0 (NO) to 1 (YES).

Table 14.28. Parameter properties of CxpErrorCorrectedTriggerAck

Property	Value
Name	**CxpErrorCorrectedTriggerAck**
Display Name	**CXP Error Corrected Trigger ACK**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.28. Usage of CxpErrorCorrectedTriggerAck

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedTriggerAckSelector = i;
    value_ = CxpErrorCorrectedTriggerAck;
}
```

14.23.4. CxpErrorCorrectedStream

This parameter returns the information whether errors were corrected in received stream packets. Range 0 (NO) to 1 (YES).

Table 14.29. Parameter properties of CxpErrorCorrectedStream

Property	Value
Name	**CxpErrorCorrectedStream**
Display Name	**CXP Error Corrected Stream**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.29. Usage of CxpErrorCorrectedStream

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedStreamSelector = i;
    value_ = CxpErrorCorrectedStream;
}
```

14.23.5. CxpErrorCorrectedControlAck

This parameter returns the information whether errors were corrected in received stream acknowledge packets. Range 0 (NO) to 1 (YES).

Table 14.30. Parameter properties of CxpErrorCorrectedControlAck

Property	Value
Name	**CxpErrorCorrectedControlAck**
Display Name	**CXP Error Corrected Control ACK**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.30. Usage of CxpErrorCorrectedControlAck

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedControlAckSelector = i;
    value_ = CxpErrorCorrectedControlAck;
}
```

14.23.6. CxpErrorCorrectedLinkTest

This parameter returns the information whether errors were corrected in received link test packets. Range 0 (NO) to 1 (YES).

Table 14.31. Parameter properties of CxpErrorCorrectedLinkTest

Property	Value
Name	**CxpErrorCorrectedLinkTest**
Display Name	**CXP Error Corrected Link Test**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.31. Usage of CxpErrorCorrectedLinkTest

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedLinkTestSelector = i;
    value_ = CxpErrorCorrectedLinkTest;
}
```

14.23.7. CxpErrorCorrectedHeartbeat

This parameter returns the information whether errors were corrected in received heartbeat packets. Range 0 (NO) to 1 (YES).

Table 14.32. Parameter properties of CxpErrorCorrectedHeartbeat

Property	Value
Name	**CxpErrorCorrectedHeartbeat**
Display Name	**CXP Error Corrected Heartbeat**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.32. Usage of CxpErrorCorrectedHeartbeat

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorCorrectedHeartbeatSelector = i;
    value_ = CxpErrorCorrectedHeartbeat;
}
```

14.23.8. CameraCorrectedErrorCount

The parameter counts the number of single-byte error corrections in CXP stream packets.

Table 14.33. Parameter properties of CameraCorrectedErrorCount

Property	Value
Name	**CameraCorrectedErrorCount**
Display Name	**Corrected Error Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 4095** **Stepsize 1**

Example 14.33. Usage of CameraCorrectedErrorCount

```
/* Get */ value_ = CameraCorrectedErrorCount;
```

14.24. ReceivedPacketsUncorrected

This category gives information about errors which occurred in received packets and which could not be corrected.

14.24.1. CxpErrorUncorrected

This parameter counts errors received in packet headers and trailers that haven't been corrected. Bits [11:0] count the amount of violations. Bit [12] is set when a counter overflow occurs. Range 0 to 8191 (13 bit).

Table 14.34. Parameter properties of CxpErrorUncorrected

Property	Value
Name	**CxpErrorUncorrected**
Display Name	**CXP Error Uncorrected**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.34. Usage of CxpErrorUncorrected

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedSelector = i;
    value_ = CxpErrorUncorrected;
}
```

14.24.2. CxpErrorUncorrectedTrigger

This parameter returns the information whether there were errors in received trigger packets that haven't been corrected. Range 0 (NO) to 1 (YES).

Table 14.35. Parameter properties of CxpErrorUncorrectedTrigger

Property	Value
Name	**CxpErrorUncorrectedTrigger**
Display Name	**CXP Error Uncorrected Trigger**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.35. Usage of CxpErrorUncorrectedTrigger

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedTriggerSelector = i;
```

```
    value_ = CxpErrorUncorrectedTrigger;
}
```

14.24.3. CxpErrorUncorrectedTriggerAck

This parameter returns the information whether there were errors in received trigger acknowledgement packets that haven't been corrected. Range 0 (NO) to 1 (YES).

Table 14.36. Parameter properties of CxpErrorUncorrectedTriggerAck

Property	Value
Name	**CxpErrorUncorrectedTriggerAck**
Display Name	**CXP Error Uncorrected Trigger ACK**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.36. Usage of CxpErrorUncorrectedTriggerAck

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedTriggerAckSelector = i;
    value_ = CxpErrorUncorrectedTriggerAck;
}
```

14.24.4. CxpErrorUncorrectedStream

This parameter returns the information whether there were errors in received stream packets that haven't been corrected. Range 0 (NO) to 1 (YES).

Table 14.37. Parameter properties of CxpErrorUncorrectedStream

Property	Value
Name	**CxpErrorUncorrectedStream**
Display Name	**CXP Error Uncorrected Stream**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.37. Usage of CxpErrorUncorrectedStream

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedStreamSelector = i;
    value_ = CxpErrorUncorrectedStream;
}
```

14.24.5. CxpErrorUncorrectedControlAck

This parameter returns information whether there were errors in received control acknowledgement packets that haven't been corrected. Range 0 (NO) to 1 (YES).

Table 14.38. Parameter properties of CxpErrorUncorrectedControlAck

Property	Value
Name	**CxpErrorUncorrectedControlAck**
Display Name	**CXP Error Uncorrected Control ACK**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.38. Usage of CxpErrorUncorrectedControlAck

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedControlAckSelector = i;
    value_ = CxpErrorUncorrectedControlAck;
}
```

14.24.6. CxpErrorUncorrectedLinkTest

This parameter returns information whether there were errors in received link test packets that haven't been corrected. Range 0 (NO) to 1 (YES).

Table 14.39. Parameter properties of CxpErrorUncorrectedLinkTest

Property	Value
Name	**CxpErrorUncorrectedLinkTest**
Display Name	**CXP Error Uncorrected Link Test**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.39. Usage of CxpErrorUncorrectedLinkTest

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedLinkTestSelector = i;
    value_ = CxpErrorUncorrectedLinkTest;
}
```

14.24.7. CxpErrorUncorrectedHeartbeat

This parameter returns information whether there were errors in received heartbeat packets that haven't been corrected. Range 0 (NO) to 1 (YES).

Table 14.40. Parameter properties of CxpErrorUncorrectedHeartbeat

Property	Value
Name	**CxpErrorUncorrectedHeartbeat**
Display Name	**CXP Error Uncorrected Heartbeat**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.40. Usage of CxpErrorUncorrectedHeartbeat

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpErrorUncorrectedHeartbeatSelector = i;
    value_ = CxpErrorUncorrectedHeartbeat;
}
```

14.24.8. CameraUncorrectedErrorCount

This parameter counts the number of uncorrected errors. Bit[2] indicates multiple byte errors in CXP stream packets.

Table 14.41. Parameter properties of CameraUncorrectedErrorCount

Property	Value
Name	**CameraUncorrectedErrorCount**
Display Name	**Uncorrected Error Count**
Interface	**IInteger**
Access policy	**Read-Only**
Visibility	**Expert**
Allowed values	**Minimum 0** **Maximum 4095** **Stepsize 1**

Example 14.41. Usage of CameraUncorrectedErrorCount

```
/* Get */ value_ = CameraUncorrectedErrorCount;
```

14.25. UnsupportedPackets

This category gives information about unsupported packets that have been received.

14.25.1. SystemmonitorRxUnsupportedPacketUnit

This parameter returns the number of received unsupported packets. Range: between 0 and 8191 in steps of 1.

Table 14.42. Parameter properties of SystemmonitorRxUnsupportedPacketUnit

Property	Value
Name	**SystemmonitorRxUnsupportedPacketUnit**
Display Name	**Systemmonitor Rx Unsupported Packet Unit**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.42. Usage of SystemmonitorRxUnsupportedPacketUnit

```
/* Get */ for (i = 0; i < 4; ++i)
{
    SystemmonitorRxUnsupportedPacketUnitSelector = i;
```

```
    value_ = SystemmonitorRxUnsupportedPacketUnit;
}
```

14.25.2. CxpUnsupportedGpioReceived

This parameter returns information whether a GPIO packet was received while using a CXP standard higher than 1.0. Range: 0 (NO) to 1 (YES).

Table 14.43. Parameter properties of CxpUnsupportedGpioReceived

Property	Value
Name	**CxpUnsupportedGpioReceived**
Display Name	**CXP Unsupported GPIO Received**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.43. Usage of CxpUnsupportedGpioReceived

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpUnsupportedGpioReceivedSelector = i;
    value_ = CxpUnsupportedGpioReceived;
}
```

14.25.3. CxpUnsupportedEventReceived

This parameter returns information whether an event packet was received while using a CXP standard less than 2.0. Range: 0 (NO) to 1 (YES).

Table 14.44. Parameter properties of CxpUnsupportedEventReceived

Property	Value
Name	**CxpUnsupportedEventReceived**
Display Name	**CXP Unsupported Event Received**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.44. Usage of CxpUnsupportedEventReceived

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpUnsupportedEventReceivedSelector = i;
    value_ = CxpUnsupportedEventReceived;
}
```

14.25.4. CxpUnsupportedHeartbeatReceived

This parameter returns information whether a heartbeat packet was received while using a CXP standard less than 2.0. Range: 0 (NO) to 1 (YES).

Table 14.45. Parameter properties of CxpUnsupportedHeartbeatReceived

Property	Value
Name	**CxpUnsupportedHeartbeatReceived**
Display Name	**CXP Unsupported Hearbeat Received**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.45. Usage of CxpUnsupportedHeartbeatReceived

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpUnsupportedHeartbeatReceivedSelector = i;
    value_ = CxpUnsupportedHeartbeatReceived;
}
```

14.25.5. CxpUnsupportedGpioAckReceived

This parameter returns information whether a GPIO acknowledgment was received while using a CXP standard higher than 1.0. Range: 0 (NO) to 1 (YES).

Table 14.46. Parameter properties of CxpUnsupportedGpioAckReceived

Property	Value
Name	**CxpUnsupportedGpioAckReceived**
Display Name	**CXP Unsupported GPIO ACK Received**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.46. Usage of CxpUnsupportedGpioAckReceived

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpUnsupportedGpioAckReceivedSelector = i;
    value_ = CxpUnsupportedGpioAckReceived;
}
```

14.25.6. CxpUnsupportedGpioRequestReceived

This parameter returns information whether a GPIO request from VisualApplets was received while using a CXP standard higher than 1.0. Range: 0 (NO) to 1 (YES).

Table 14.47. Parameter properties of CxpUnsupportedGpioRequestReceived

Property	Value
Name	**CxpUnsupportedGpioRequestReceived**
Display Name	**CXP Unsupported GPIO Request Received**
Interface	**IInteger (Field)**
Field Size	**4**
Access policy	**Read-Only**
Visibility	**Expert**

Example 14.47. Usage of CxpUnsupportedGpioRequestReceived

```
/* Get */ for (i = 0; i < 4; ++i)
{
    CxpUnsupportedGpioRequestReceivedSelector = i;
    value_ = CxpUnsupportedGpioRequestReceived;
}
```

Chapter 15. Revision History

Revision history of AcquisitionApplets releases.

Applet Version	Release Date	Change Log	Delivered with
1.1.4.0	06 Sep 2021	Initial version of this applet.	pylon 6.3.0
2.0.5.0	1 Mar 2022	Added an extended overflow capability, which allows to monitor frame-IDs from the device.	pylon 7.2.0 & Framegrabber SDK 5.10
2.1.6.0	24 Jan 2023	No functional changes.	pylon 7.3.0
2.2.7.0	5 Apr 2023	• The applet now supports mirroring, **ImageSelector**, white balancing, and color converter. • The applet now supports tap geometry sorting. • The CXP-12 frame grabber now supports stream packet sizes of maximum 16 KB. • Bugfixes.	pylon 7.4.0 & Framegrabber SDK 5.11
2.3.7.0	22 Dec 2023	• CoaXPress trigger packets updated to CoaXPress 2.1: The applet now supports sending the CXP **`LinkTrigger0`**, **`LinkTrigger1`**, **`LinkTrigger2`** and **`LinkTrigger3`**. For each link trigger, an individual source can be selected. By default, **`LinkTrigger0`** is the rising edge of the trigger source and **`LinkTrigger1`** is the falling edge of the trigger, which usually represents the exposure time. The default setting is equal to previous versions. Thus, the applet is fully compatible to CXP 1.1 and 2.0. **`LinkTrigger2`** and **`LinkTrigger3`** are not used by default. The parameters *CxpTriggerPacketMode* (CxpTriggerPacketMode) and *TriggerCameraOutSelect* (TriggerCameraOutSelect) became legacy.	pylon 7.5.0 & Framegrabber SDK 5.11.2
2.4.7.0	31 Jul 2024	• This applet has been extended with additional parameters and events for problem analysis, e.g. to find errors in the CXP stream. The new parameters and events have been added to the parameter tree. • Bug fixes. See Section 15.1, 'Fixed Issues' for a detailed list of fixed issues.	pylon 8.0.0 & Framegrabber SDK 5.11.3

15.1. Fixed Issues

15.1.1. Fixed in Version 2.4.7.0

- Solved communication issues with Opto-Coupled Trigger 5 board. (Ticket ID: 306774)
- Before fixing this issue, in some cases CXP status parameters showed wrong and misleading values. This has been fixed. (Ticket ID: 282288)

15.1.2. Fixed in Version 2.2.7.0

- The following CXP status parameters did not work correctly and have been fixed:
  - *SystemmonitorUsedCxpConnections*
  - *PacketTagErrorCount*
  - *CameraCorrectedErrorCount*
  - *CameraUncorrectedErrorCount*
  - *SystemmonitorPacketbufferOverflowCount*
  - *SystemmonitorPacketbufferOverflowSource*
  - *SystemmonitorCxpImageLineMode*
  - **FG_TRIGGER_EVENT_COUNT**
  - **FG_TRIGGER_ACKNOWLEDGEMENT_COUNT**
  - **FG_TRIGGER_WAVE_VIOLATION**

15.1.3. Fixed in Version 2.0.5.0

Bitrate switching has been fixed. Prior to fixing this issue, switching the bitrate of a CXP link could put your frame grabber into a failure state. This has been fixed.

15.2. Known Issues

- Setting the tap geometry might fail. If you set tap geometry to **Geometry_1X_2YE**, you have to ensure that *SensorWidth* is greater or equal to *Width* and *SensorHeight* is greater or equal to *Height*. The order of setting the parameter is important. Otherwise, the tap geometry sorting will fail. (Ticket-ID: 270154)
- In rare cases, the log warning “Failed to read packet. Error code: -1997” can occur with boost v cameras. The camera communication and image acquisition are not affected. This warning can be ignored. (Ticket-ID: 262668)
- In rare cases, loading an applet can fail with the error message “Fg_init(...) : -2050 (Design is invalid)”. In this case, re-load the applet and contact the Basler Support [https://www.baslerweb.com/en/sales-support/support-contact/]. (Ticket-ID: 259458)

Glossary

Area of Interest (AOI)	See Region of Interest.
Board	A Basler hardware. Usually, a board is represented by a interface card. Boards might comprise multiple devices.
Board ID Number	An identification number of a Basler board in a PC system. The number is not fixed to a specific hardware but has to be unique in a PC system.
Camera Index	The index of a camera connected to a interface card. The first camera will have index zero. Mind the difference between the camera index and the interface card camera port. See also Camera Port.
Camera Port	The Basler interface card connectors for cameras are called camera ports. They are numbered {0, 1, 2, ...} or enumerated {A, B, C, ... }. Depending on the interface one camera could be connected to multiple camera ports. Also, multiple cameras could be connected to one camera port.
Camera Tap	See Tap.
Device	A board can consist of multiple devices. Devices are numbered. The first device usually has number one.
Direct Memory Access (DMA)	A DMA transfer allows hardware subsystems within the computer to access the system memory independently of the central processing unit (CPU). Basler uses DMAs for data transfer such as image data between a board e.g. a interface card and a PC. Data transfers can be established in multiple directions i.e. from a interface card to the PC (download) and from the PC to a interface card (upload). Multiple DMA channels may exist for one board. Control and configuration data usually do not use DMA channels.
DMA Channel	See DMA Index.
DMA Index	The index of a DMA transfer channel. See also Direct Memory Access.
Event	In programming or runtime environments, a callback function is a piece of executable code that is passed as an argument, which is expected to call back (execute) exactly that time an event is triggered. These events are not related to a special camera functionality and based on interface card internal functionality. Basler uses hardware interrupts for the event transfer and processing is absolutely optimized for low latency. These interrupts are only produced by the interface card if an event is registered and activated by software. If an event is fired at a very high frequency this may influence the system performance. For example these events can be used to check the reliability between a frame trigger input and the resulting and expected camera frame. The Basler Framegrabber SDK enables an application to get these event notifications about certain state changes at the data flow from camera to RAM and the image and trigger processing as well. Please consult the Basler Framegrabber SDK documentation for more details concerning the implementation of this functionality. Some events are enabled to produce additional data, which is described for the event itself.

Frame Grabber	Usually a PC hardware using PCI express to interface the camera and grab camera images. The frame grabber will grab, buffer, pre-process and forward the images to the PC memory. Moreover, the frame grabber performs the trigger signal processing to trigger the camera, external lights and controllers. On V-series frame grabber custom processing can be implemented using VisualApplets. See also Direct Memory Access, Interface Card, VisualApplets.
GenICam	Generic Interface for Cameras is a generic programming interface for machine vision (industrial) cameras.
GenTL	GenICam Transport Layer. This is the transport layer interface for enumerating cameras, grabbing images from the camera, and moving them to the user application.
Interface Card	Usually a PC hardware using PCI express to interface the camera and grab camera images. The interface card will grab, buffer and forward the images to the PC memory. Moreover, the interface card performs the trigger signal processing to trigger the camera, external lights and controllers. See also Direct Memory Access, Frame Grabber.
Port	See Camera Port.
Process	An image or signal data processing block. A process can include one or more cameras, one or more DMA channels and modules.
Region of Interest (ROI)	Represents a part of a frame. Mostly rectangular and within the original image boundaries. Defined by source coordinates and its dimension. The interface card cuts the region of interest from the camera image. A region of interest might reduce or increase the required bandwidth and the corresponding image dimension.
Sensor Tap	See Tap.
Software Callback	See Event.
Tap	Some cameras have multiple taps. This means, they can acquire or transfer more than one pixel at a time which increses the camera's acquisition speed. The camera sensor tap readout order varies. Some cameras read the pixels interlaced using multiple taps, while some cameras read the pixel simultaneously from different locations on the sensor. The reconstruction of the frame is called sensor readout correction. The Camera Link interface is also using multiple taps for image transfer to increase the bandwidth. These taps are independent from the sensor taps.
Trigger	In machine vision and image processing, a trigger is an event which causes an action. This can be for example the initiation of a new line or frame acquisition, the control of external hardware such as flash lights or actions by a software applications. Trigger events can be initiated by external sources, an internal frequency generator (timer) or software applications. The event itself is mostly based on a rising or falling edge of a electrical signal.
Trigger Input	A logic input of a trigger IO. The first input has index 0. Check mapping of input pins to logic inputs in the hardware documentation.
Trigger Output	A logic output of a trigger IO. The first output has index 1. Please check the mapping of output pins to logic outputs in the hardware documentation. The electrical characteristics and specification can be found related to the selected or used trigger board/connector.
Trigger Reliability	See Event.

User Interrupt	See Event.
VisualApplets	Simple programming of FPGA-based image processing devices. VisualApplets enables access to the FPGA processors in the image processing hardware, such as frame grabbers, industrial cameras and image processing devices, to implement individual image processing applications.

Index

A

AcquisitionTrigger, 69
AppletRevision, 86
AppletVersion, 86
Area of Interest, 16
AreaTriggerMode, 38

B

Bandwidth, 3
BitAlignment, 83
Boardstatus, 88

C

Camera, 11
  Events, 11
  Format, 6
  Interface, 4, 11
Camera::Events, 11
CameraCorrectedErrorCount, 105
CameraStreamStatus, 11
CameraUncorrectedErrorCount, 109
CoaXPress, 6
Color Converter, 79
CustomBitShiftRight, 84
CXP Source Tag, 4
CxpCameraFrameCorruptCount, 99
CxpCameraFrameLostCount, 99
CxpCameraMarkerErrorCount, 98
CxpCameraUnexpectedStartupDataStatus, 98
CxpControlAckIncompleteCount, 94
CxpControlAckLostCount, 93
CxpControlAckPacketCrcError, 101
CxpControlTagErrorCount, 94
CxpErrorCorrected, 102
CxpErrorCorrectedControlAck, 104
CxpErrorCorrectedHeartbeat, 105
CxpErrorCorrectedLinkTest, 104
CxpErrorCorrectedStream, 104
CxpErrorCorrectedTrigger, 103
CxpErrorCorrectedTriggerAck, 103
CxpErrorUncorrected, 106
CxpErrorUncorrectedControlAck, 107
CxpErrorUncorrectedHeartbeat, 108
CxpErrorUncorrectedLinkTest, 108
CxpErrorUncorrectedStream, 107
CxpErrorUncorrectedTrigger, 106
CxpErrorUncorrectedTriggerAck, 107
CxpHeartbeatIncompleteCount, 95
CxpHeartbeatMaxPeriodViolationCount, 95
CxpImageTagErrorCount, 97
CxpInputMappedToFWPortPort, 89
CxpLinkTrigger0Source, 57
CxpLinkTrigger1Source, 58
CxpLinkTrigger2Source, 59
CxpLinkTrigger3Source, 60

CxpOvertriggerRequestPulseCount, 92
CxpStreamIDErrorCount, 97
CxpStreamPacketCount, 7
CxpStreamPacketCrcError, 100
CxpStreamPacketLengthError, 102
CxpTriggerAckMissingCount, 93
CxpUnsupportedEventReceived, 110
CxpUnsupportedGpioAckReceived, 111
CxpUnsupportedGpioReceived, 110
CxpUnsupportedGpioRequestReceived, 111
CxpUnsupportedHeartbeatReceived, 110

E

Errors, 91
Errors::CRC, 100
Errors::LengthErrors, 101
Errors::ReceivedPacketsCorrected, 102
Errors::ReceivedPacketsUncorrected, 106
Errors::UnsupportedPackets, 109
Events
  Camera, 11
  Digital Inputs, 24, 46
  Overflow, 73
  Trigger, 24
  Trigger Lost Detection, 64
  Trigger Output, 69
  Trigger Queue, 51

F

Features, 1
FillLevel, 70
Format, 80, 80
Frame ID, 4
FrameTransferEnd, 13
FrameTransferStart, 13
FrameTriggerMissed, 68
FrontGPI, 42

G

GPI, 42

H

Height, 18

I

Image Select, 75
Image Selector, 75
Image Transfer, 4
ImageSelect, 76
ImageSelectPeriod, 75

L

LineFront0FallingEdge, 50
LineFront0RisingEdge, 50

M

Miscellaneous, 86
Miscellaneous::Version, 86

MissingCameraFrameResponse, 67
MissingCameraFrameResponseClear, 68

O

OffsetX, 18
OffsetY, 19
Output Format, 80
Overflow, 70, 70, 71, 74
  Events, 73
Overflow::Events, 73
OverflowEventSelect, 72
OverflowOffThreshold, 71
OverflowOnThreshold, 72
OverflowSyncOnThreshold, 72

P

PacketTagErrorCount, 96
PC Interface, 4
Pixel Format, 6
PixelDepth, 84
PixelFormat, 7

R

Region of Interest, 16
ROI, 16

S

ScalingFactorBlue, 77
ScalingFactorGreen, 77
ScalingFactorRed, 77
SendSoftwareTrigger, 45
Sensor Geometry, 14, 14
SensorHeight, 15
SensorWidth, 14
SoftwareTriggerIsBusy, 46
SoftwareTriggerQueueFillLevel, 46
Source Tag, 4
Specifications, 1
SystemmonitorByteAlignment8b10bLocked, 91
SystemmonitorCurrentLinkSpeed, 88
SystemmonitorCxpImageLineMode, 9
SystemmonitorCxpStandard, 6
SystemmonitorDecoder8b10bError, 91
SystemmonitorMappedToFgPort, 88
SystemmonitorPacketbufferOverflowCount, 96
SystemmonitorPacketbufferOverflowSource, 97
SystemmonitorPcieTrainedPayloadSize, 89
SystemmonitorPcieTrainedRequestSize, 89
SystemmonitorRxLengthErrorCount, 101
SystemmonitorRxPacketCrcErrorCount, 100
SystemmonitorRxStreamIncompleteCount, 92
SystemmonitorRxUnknownDataReceivedCount, 92
SystemmonitorRxUnsupportedPacketUnit, 109
SystemmonitorStreamPacketSize, 6
SystemmonitorUsedCxpConnections, 8

T

TapGeometry, 9

Trigger, 20, 20
Activate, 40
Busy, 46
Bypass, 37
Camera Signal Mapping, 57
Debounce, 41
Debugging, 37
Digital Input, 24, 42
Digital Input Output Mapping, 24
Digital Output, 24, 26, 61
Downscale Input, 44
Encoder, 26
Error Detection, 37
Events, 24
Exceeded Period Limits, 65
Exsync, 26
External, 26, 39, 41
Flash, 26, 29
Frame Rate, 25
Framerate, 40
Generator, 25, 39
GPI, 42
Grabber Controlled, 25
Image Trigger, 26
Input, 41, 42
Input Statistics, 46
IO Triggered, 26
Length, 26
Lost Trigger, 37, 65
Missing Frame Response, 67
Mode, 39
Multi Camera, 37
Multiply Pulses, 50
Output, 61
Output Event, 69
Output Statistics, 64
Period, 40
Pin Allocation, 24
Polarity Input, 43, 47
Pulse Form Generator, 54
Pulse Multiplication, 29
Queue, 34, 36, 51
Sequencer, 29, 50
Signal Length, 26
Signal Width, 26
Software Controlled, 45
Software Trigger, 32, 39, 45
Start, 40
Stop, 40
Synchronized, 37, 39
Synchronized Cameras, 37
System Analysis, 37
Trigger IO, 24
Width, 26
Trigger::Camera Out Signal Mapping, 57
Trigger::Digital Output, 61
Trigger::Digital Output::Statistics, 64
Trigger::Output Event, 68

Trigger::Pulse Form Generator 0, 54
Trigger::Pulse Form Generator 1, 57
Trigger::Pulse Form Generator 2, 57
Trigger::Pulse Form Generator 3, 57
Trigger::Queue, 51
Trigger::Sequencer, 50
Trigger::Trigger Input, 41
Trigger::Trigger Input::External, 41
Trigger::Trigger Input::Software Trigger, 45
Trigger::Trigger Input::Statistics, 46
TriggerExceededPeriodLimits, 65, 68
TriggerExceededPeriodLimitsClear, 65
TriggerInDebounce, 41
TriggerInDownscale, 44
TriggerInDownscalePhase, 44
TriggerInPolarity, 43
TriggerInSource, 43
TriggerInStatisticsFrequency, 48
TriggerInStatisticsMaximumFrequency, 49
TriggerInStatisticsMinimumFrequency, 49
TriggerInStatisticsMinMaxFrequencyClear, 49
TriggerInStatisticsPolarity, 47
TriggerInStatisticsPulseCount, 47
TriggerInStatisticsPulseCountClear, 48
TriggerInStatisticsSource, 47
TriggerMultiplyPulses, 31, 50
TriggerOutputEventSelect, 69
TriggerOutputFrequency, 28, 40
TriggerOutSelectFrontGPO0, 62
TriggerOutSelectFrontGPO1, 63
TriggerOutStatisticsPulseCount, 66
TriggerOutStatisticsPulseCountClear, 66
TriggerOutStatisticsSource, 65
TriggerPulseFormGenerator0Delay, 56
TriggerPulseFormGenerator0Downscale, 54
TriggerPulseFormGenerator0DownscalePhase, 55
TriggerPulseFormGenerator0Width, 56
TriggerPulseFormGenerator1Delay, 56
TriggerPulseFormGenerator1Downscale, 54
TriggerPulseFormGenerator1DownscalePhase, 55
TriggerPulseFormGenerator1Width, 56
TriggerPulseFormGenerator2Delay, 56
TriggerPulseFormGenerator2Downscale, 54
TriggerPulseFormGenerator2DownscalePhase, 55
TriggerPulseFormGenerator2Width, 56
TriggerPulseFormGenerator3Delay, 56
TriggerPulseFormGenerator3Downscale, 54
TriggerPulseFormGenerator3DownscalePhase, 55
TriggerPulseFormGenerator3Width, 56
TriggerQueueFillLevel, 52
TriggerQueueFillLevelEventOffThreshold, 53
TriggerQueueFillLevelEventOnThreshold, 52
TriggerQueueFilllevelThresholdOff, 53
TriggerQueueFilllevelThresholdOn, 53
TriggerQueueMode, 51
TriggerState, 40

V

VantagePoint, 14
VisualAppletsBuildVersion, 87

W

White Balance, 77, 77
Width, 17