

# CXP-12 Interface Card 1C

Feature reference manual for  
**Acq\_SingleCXP12x1Area**

Functional description and  
GenTL parameters

Document Number: AW001605  
Document Version: 02 Language: 000 (English)  
Release Date: 16 April 2020  
Applet Version 3.2.3.0

# Contacting Basler Support Worldwide

## Europe, Middle East, Africa

Basler AG  
An der Strusbek 60–62  
22926 Ahrensburg  
Germany

Tel. +49 4102 463 515  
Fax +49 4102 463 599

[support.europe@baslerweb.com](mailto:support.europe@baslerweb.com)

## The Americas

Basler, Inc.  
855 Springdale Drive, Suite 203  
Exton, PA 19341  
USA

Tel. +1 610 280 0171  
Fax +1 610 280 7608

[support.usa@baslerweb.com](mailto:support.usa@baslerweb.com)

## Asia-Pacific

Basler Asia Pte. Ltd.  
35 Marsiling Industrial Estate Road 3  
#05–06  
Singapore 739257

Tel. +65 6367 1355  
Fax +65 6367 1255

[support.asia@baslerweb.com](mailto:support.asia@baslerweb.com)

**[www.baslerweb.com](http://www.baslerweb.com)**

## Supplemental Information

Configuring the CXP-12 Interface Card 1C :

<https://docs.baslerweb.com/configuring-the-cxp-12-interface-card-1c.html>

Hardware Documentation:

<https://docs.baslerweb.com/cxp-12-interface-card-1c.html>

CXP GenTL Producer Feature Documentation:

<https://www.baslerweb.com/en/sales-support/downloads/document-downloads/cxp-gentl-producer-feature-documentation/>

**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_SingleCXP12x1Area .....	1
1.1.1. Parameterization Order .....	2
1.2. Bandwidth .....	2
1.3. Requirements .....	3
1.3.1. Software Requirements .....	3
1.3.2. Hardware Requirements .....	3
1.3.3. License .....	4
1.4. Camera Interface .....	4
1.5. Image Transfer to PC Memory .....	4
2. CoaXPRESS .....	5
2.1. PixelFormat .....	5
2.2. CxpTriggerPacketMode .....	6
2.3. CxpStatus .....	7
2.4. TriggerEventCount .....	7
2.5. TriggerAcknowledgementCount .....	8
2.6. TriggerWaveViolation .....	8
3. ROI .....	10
3.1. Width .....	11
3.2. Height .....	12
3.3. OffsetX .....	12
3.4. OffsetY .....	13
4. Trigger .....	14
4.1. Features and Functional Blocks of Area Trigger .....	14
4.2. Digital Input/Output Mapping .....	17
4.3. Trigger Scenarios .....	18
4.3.1. Internal Frequency Generator / interface card Controlled .....	18
4.3.2. External Trigger Signals / IO Triggered .....	19
4.3.3. Control of Two Flash Lights .....	22
4.3.4. Software Trigger .....	25
4.3.5. Software Trigger with Trigger Queue .....	27
4.3.6. External Trigger with Trigger Queue .....	29
4.3.7. Bypass External Trigger Signals .....	30
4.3.8. Multi Camera Applications / Synchronized Cameras .....	30
4.3.9. Hardware System Analysis and Error Detection / Trigger Debugging .....	30
4.4. Parameters .....	31
4.4.1. AreaTriggerMode .....	31
4.4.2. TriggerState .....	32
4.4.3. TriggerFramesPerSecond .....	33
4.4.4. Trigger Input .....	34
4.4.4.1. External .....	34
4.4.4.1.1. TriggerInDebounce .....	34
4.4.4.1.2. FrontGPI .....	35
4.4.4.1.3. TriggerInSource .....	35
4.4.4.1.4. TriggerInPolarity .....	36
4.4.4.1.5. TriggerInDownscale .....	36
4.4.4.1.6. TriggerInDownscalePhase .....	37
4.4.4.2. Software Trigger .....	38
4.4.4.2.1. SendSoftwareTrigger .....	38
4.4.4.2.2. SoftwareTriggerIsBusy .....	38
4.4.4.2.3. SoftwareTriggerQueueFillLevel .....	39
4.4.4.3. InStatistics .....	39
4.4.4.3.1. TriggerInStatisticsSource .....	39
4.4.4.3.2. TriggerInStatisticsPolarity .....	40
4.4.4.3.3. TriggerInStatisticsPulseCount .....	40
4.4.4.3.4. TriggerInStatisticsPulseCountClear .....	40

4.4.4.3.5. TriggerInStatisticsFrequency .....	41
4.4.4.3.6. TriggerInStatisticsMinimumFrequency .....	41
4.4.4.3.7. TriggerInStatisticsMaximumFrequency .....	42
4.4.4.3.8. TriggerInStatisticsMinMaxFrequencyClear .....	42
4.4.5. Sequencer .....	42
4.4.5.1. TriggerMultiplyPulses .....	43
4.4.6. Queue .....	43
4.4.6.1. TriggerQueueMode .....	43
4.4.6.2. TriggerQueueFillLevel .....	44
4.4.7. Pulse Form Generator 0 .....	44
4.4.7.1. TriggerPulseFormGenerator0Downscale et al. ....	45
4.4.7.2. TriggerPulseFormGenerator0DownscalePhase et al. ....	46
4.4.7.3. TriggerPulseFormGenerator0Delay et al. ....	47
4.4.7.4. TriggerPulseFormGenerator0Width et al. ....	47
4.4.8. Pulse Form Generator 1 .....	48
4.4.9. Pulse Form Generator 2 .....	48
4.4.10. Pulse Form Generator 3 .....	48
4.4.11. Camera Out Signal Mapping .....	48
4.4.11.1. TriggerCameraOutSelect .....	48
4.4.12. Digital Output .....	49
4.4.12.1. TriggerOutSelectFrontGPO0 et al. ....	50
4.4.12.2. OutStatistics .....	50
4.4.12.2.1. TriggerExceededPeriodLimits .....	51
4.4.12.2.2. TriggerExceededPeriodLimitsClear .....	51
4.4.12.2.3. TriggerOutStatisticsSource .....	51
4.4.12.2.4. TriggerOutStatisticsPulseCount .....	52
4.4.12.2.5. TriggerOutStatisticsPulseCountClear .....	52
4.4.12.2.6. MissingCameraFrameResponse .....	53
4.4.12.2.7. MissingCameraFrameResponseClear .....	54
5. BufferStatus .....	55
5.1. FillLevel .....	55
5.2. Overflow .....	55
6. Output Format .....	57
6.1. Format .....	57
6.2. BitAlignment .....	59
6.3. PixelDepth .....	60
6.4. CustomBitShiftRight .....	60
7. Camera Simulator .....	62
7.1. CamerasimulatorEnable .....	62
7.2. CamerasimulatorWidth .....	63
7.3. CamerasimulatorLineGap .....	63
7.4. CamerasimulatorHeight .....	64
7.5. CamerasimulatorFrameGap .....	64
7.6. CamerasimulatorPattern .....	65
7.7. CamerasimulatorPatternOffset .....	66
7.8. CamerasimulatorRoll .....	66
7.9. CamerasimulatorSelectMode .....	66
7.10. CamerasimulatorPixelFrequency .....	67
7.11. CamerasimulatorLinerate .....	67
7.12. CamerasimulatorFramerate .....	68
7.13. CamerasimulatorTriggerMode .....	69
7.14. CamerasimulatorActive .....	69
7.15. CamerasimulatorPassive .....	70
8. Miscellaneous .....	71
8.1. Timeout .....	71
8.2. AppletVersion .....	71
8.3. AppletRevision .....	71
8.4. AppletId .....	72

## Table of Contents

---

8.5. AppletBuildTime .....	72
8.6. HapFile .....	72
8.7. DMAStatus .....	73
8.8. SystemmonitorFpgaDnaLow .....	73
8.9. SystemmonitorFpgaDnaHigh .....	74
9. Revision History .....	75
Glossary .....	76
Index .....	79

---

# Chapter 1. Introduction

This document provides you with detailed information on applet "Acq\_SingleCXP12x1Area" for CXP-12 Interface Card 1C .



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

For a general introduction on how to configure the CXP-12 Interface Card 1C using the pylon API, the pylon Viewer, or the gpioTool check the document which can be found in <https://docs.baslerweb.com/configuring-the-cxp-12-interface-card-1c.html>.

For information on the hardware, check the hardware reference manual for CXP-12 Interface Card 1C (<https://docs.baslerweb.com/cxp-12-interface-card-1c.html>).

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\_SingleCXP12x1Area

"Acq\_SingleCXP12x1Area" is an applet for one camera (single-camera applet). 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), Bayer (BayerGR8, BayerGR10p, BayerGR12p, BayerRG8, BayerRG10p, BayerRG12p, BayerGB8, BayerGB10p, BayerGB12p, BayerBG8, BayerBG10p, BayerBG12p), Color (RGB8, RGB10p, RGB12p), and YCbCr422\_8. You can use a camera with a single CoaXPress link 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 49152 by 65535 pixels are supported. The applet is processing data at a bit depth of 14 bits. 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 8 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.

Processed image data are output by the applet via a high speed DMA channel. You can select the pixel format of the output. The pixel format can either be 8 bit, 10 bit packed, 12 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\_SingleCXP12x1Area

Feature	Applet Property
Applet Name	 Acq_SingleCXP12x1Area
Type of Applet	AcquisitionApplets
Board	CXP-12 Interface Card 1C
No. of Cameras	1
Camera Type	CoaXPress, link aggregation max. 1, maximum speed CXP-12, Version 1.1.1 and 2.0
Sensor Type	Area Scan
Camera Format	Grayscale, Bayer Pattern or RGB
Pixel Format	Gray (Mono8, Mono10, Mono12), Bayer (BayerGR8 BayerGR10p BayerGR12p BayerRG8 BayerRG10p BayerRG12p BayerGB8 BayerGB10p BayerGB12p BayerBG8 BayerBG10p BayerBG12p), Color (RGB8, RGB10p, RGB12p) and YCbCr422_8.
Processing Bit Depth	14 Bit per color component
Maximum Images Dimensions	49152 * 65535
ROI Stepsize	x: 8, y: 1
Mirroring	none
Noise Filter	No
Shading Correction	No
Dead Pixel Interpolation	No
Bayer Filter	Yes, High Quality Extended (HQe)
Color White Balancing	No
Lookup Table	No
DMA	Full Speed
DMA Image Output Format	All grayscale and color formats. See description above.
Event Generation	no
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\_SingleCXP12x1Area are listed in the following table.

Table 1.2. Bandwidth of Acq\_SingleCXP12x1Area

Description	Bandwidth
Max. CXP Speed	CXP-12
Peak Bandwidth per Camera	1200 MPixel/s
Mean Bandwidth per Camera	1200 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.)

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 a Silicon Software runtime installation providing the GenTL interface.

### 1.3.2. Hardware Requirements

To run applet "Acq\_SingleCXP12x1Area", a Basler CXP-12 Interface Card 1C is required.

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



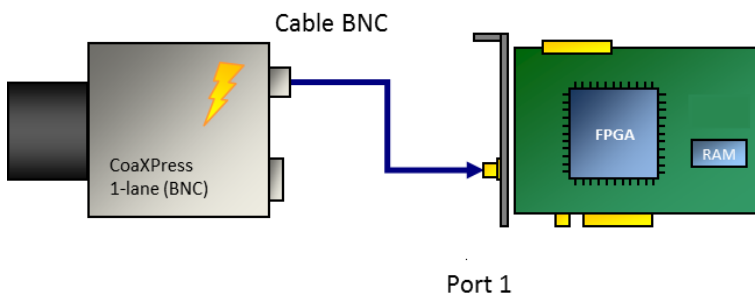
### 1.3.3. License

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 runtime software.

## 1.4. Camera Interface

Applet "Acq\_SingleCXP12x1Area" supports 1 CXP camera. The interface card has 1 connector. Use a single CoaXPress cable to connect the camera with the interface card. The maximum link aggregation of this applet is one.

Figure 1.1. Camera Interface and Camera Cable Setup



## 1.5. Image Transfer to PC Memory

The image transfer between interface card and PC is performed via DMA transfers. In this applet, only one DMA channel exists for transferring image data. The DMA channel has index 0. The applet output format can be set via the parameters of the output format module. See Chapter 6, '*Output Format*'. All outputs are little-endian coded.

---

# Chapter 2. CoaXPress

This applet can be used with one area scan camera. 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 parameter configure the interface card's camera interface to match with the individual camera pixel format. Most cameras support different operation modes. Please, 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 (49152 x 65535).

A second parameter defines the way trigger packets are sent from the interface card to the camera on the CXP link.

## 2.1. 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 14 bit. The camera interface automatically performs a conversion to the 14 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 2.1. Parameter properties of PixelFormat

Property	Value
Name	<b>PixelFormat</b>
Display Name	<b>Pixel Format</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read/Write</b>
Visibility	<b>Beginner</b>
Allowed values	<b>BayerGR8</b> BayerGR 8bit <b>BayerGR10p</b> BayerGR 10bit <b>BayerGR12p</b> BayerGR 12bit <b>BayerRG8</b> BayerRG 8bit <b>BayerRG10p</b> BayerRG 10bit <b>BayerRG12p</b> BayerRG 12bit <b>BayerGB8</b> BayerGB 8bit <b>BayerGB10p</b> BayerGB 10bit <b>BayerGB12p</b> BayerGB 12bit <b>BayerBG8</b> BayerBG 8bit <b>BayerBG10p</b> BayerBG 10bit <b>BayerBG12p</b> BayerBG 12bit <b>Mono8</b> Mono 8bit <b>Mono10</b> Mono 10bit <b>Mono12</b> Mono 12bit <b>RGB8</b> RGB 8bit <b>RGB10p</b> RGB 10bit <b>RGB12p</b> RGB 12bit <b>YCbCr422_8</b> YUV422 8bit
Default value	<b>Mono8</b>

Example 2.1. Usage of PixelFormat

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

## 2.2. CxpTriggerPacketMode

Defines the trigger packet mode. For CXP, a packet for the trigger start i.e. rising edge and the trigger end i.e. falling edge is sent on the CXP links. The CXP standard defines a maximum trigger frequency based on the available uplink data rate. Since this can be limited to a certain value the maximum can be reduced. In order to overcome this limitation you can try to set this parameter to **CXPTriggerRising** so that only half the number of packets need to be transferred. However, this setting will violate the CXP standard and will only work with supported cameras.

Table 2.2. Parameter properties of CxpTriggerPacketMode

Property	Value
Name	<b>CxpTriggerPacketMode</b>
Display Name	<b>CXP Trigger Packet Mode</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>CXPTriggerStandard</b> Standard <b>CXPTriggerRising</b> Rising Edge Only
Default value	<b>CXPTriggerStandard</b>

Example 2.2. Usage of CxpTriggerPacketMode

```
/* Set */ CxpTriggerPacketMode = CXPTriggerStandard;
/* Get */ value_ = CxpTriggerPacketMode;
```

## 2.3. CxpStatus

The parameter reflects the current status of the camera operator. Bit[0] signalizes CXP stream packet loss detection. Bit[1] signalizes single byte error correction in CXP stream packets. Bit[2] signalizes multiple byte error detection in CXP stream packets. Bit[3..6] are reserved. This parameter might change in future versions.

Table 2.3. Parameter properties of CxpStatus

Property	Value
Name	<b>CxpStatus</b>
Display Name	<b>CXP Status</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 255 <b>Stepsize</b> 1

Example 2.3. Usage of CxpStatus

```
/* Get */ value_ = CxpStatus;
```

## 2.4. TriggerEventCount

The parameter indicates how many trigger edge events have been sent to the camera.

Table 2.4. Parameter properties of TriggerEventCount

Property	Value
Name	<b>TriggerEventCount</b>
Display Name	<b>Trigger Event Count</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 1048575 <b>Stepsize</b> 1

Example 2.4. Usage of TriggerEventCount

```
/* Get */ value_ = TriggerEventCount;
```

## 2.5. TriggerAcknowledgementCount

The parameter indicates how many trigger acknowledgement packets sent by the camera (in answer to the trigger edge packets sent before) have been received by the interface card.

Table 2.5. Parameter properties of TriggerAcknowledgementCount

Property	Value
Name	<b>TriggerAcknowledgementCount</b>
Display Name	<b>Trigger Acknowledgement Count</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 1048575 <b>Stepsize</b> 1

Example 2.5. Usage of TriggerAcknowledgementCount

```
/* Get */ value_ = TriggerAcknowledgementCount;
```

## 2.6. TriggerWaveViolation

The parameter is set to 1 if the applet detects a distance between two trigger edges which violates the minimal edge frequency. The parameter holds its value until it has been read. After being read, the parameter updates the value. Frequency control is running permanently and is not influenced by the read status of the parameter.

Table 2.6. Parameter properties of TriggerWaveViolation

Property	Value
Name	<b>TriggerWaveViolation</b>
Display Name	<b>Trigger Wave Violation</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 1048575</b> <b>Stepsize 1</b>

Example 2.6. Usage of TriggerWaveViolation

---

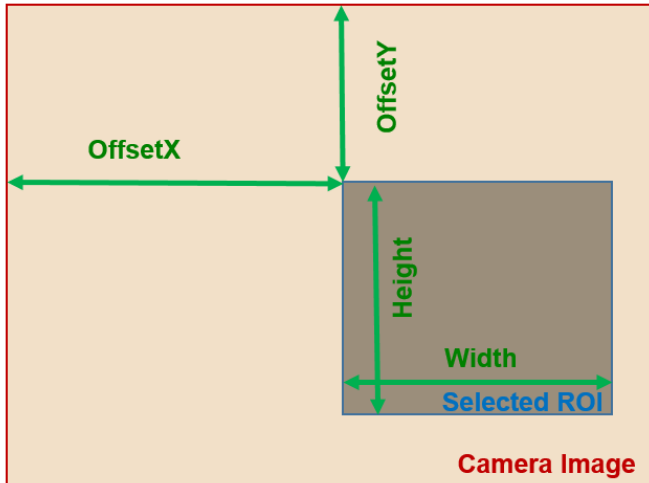
```
/* Get */ value_ = TriggerWaveViolation;
```

---

# Chapter 3. ROI

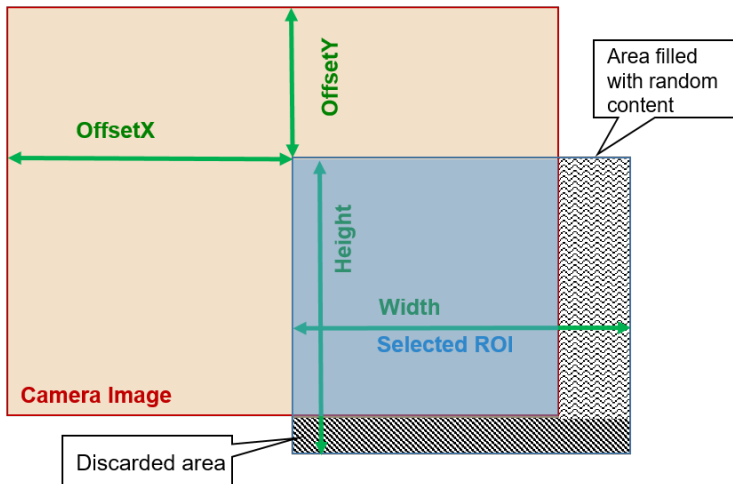
This module allows the definition of a region of interest (ROI), also called area of interest (AOI). A ROI allows the selection of a smaller subset pixel area from the input image. It is defined by using parameters *OffsetX*, *Width*, *OffsetY* and *Height*. The following figure illustrates the parameters.

Figure 3.1. 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 3.2. 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 49152 pixels and a maximum image height of 65535 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 8 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.

### 3.1. Width

The parameter specifies the width of the ROI. The values of parameters *Width* + *OffsetX* must not exceed the maximum image width of 49152 pixels. If a horizontal mirroring is active SensorWidth limits the maximum Width (*Width* + *XOffset*). If furthermore vertical Mirroring is active the maximum Width is limited by the DRAM and SensorHeight (Sensor dimension need to fit into the DRAM).

Table 3.1. Parameter properties of Width

Property	Value
Name	<b>Width</b>
Display Name	<b>Width</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 8 <b>Maximum</b> 49152 <b>Stepsize</b> 8
Default value	<b>1024</b>
Unit of measure	<b>pixel</b>

#### Example 3.1. Usage of Width

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



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

## 3.2. Height

The parameter specifies the height of the ROI. The values of parameters *Height* + *OffsetY* must not exceed the maximum image height of 65535 pixels. If a vertical mirroring is active SensorHeight limits the maximum Height (Height + YOffset). Furthermore the maximum Height is limited by the DRAM and SensorWidth (Sensor dimension need to fit into the DRAM).

Table 3.2. Parameter properties of Height

Property	Value
Name	<b>Height</b>
Display Name	<b>Height</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> <b>1</b> <b>Maximum</b> <b>65535</b> <b>Stepsize</b> <b>1</b>
Default value	<b>1024</b>
Unit of measure	<b>pixel</b>

Example 3.2. Usage of Height

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

## 3.3. OffsetX

The x-offset is defined by this parameter. If a horizontal mirroring is active SensorWidth limits the maximum Width (Width + XOffset). If furthermore vertical Mirroring is active the maximum Width is limited by the DRAM and SensorHeight (Sensor dimension need to fit into the DRAM).

Table 3.3. Parameter properties of OffsetX

Property	Value
Name	<b>OffsetX</b>
Display Name	<b>XOffset</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> <b>0</b> <b>Maximum</b> <b>49144</b> <b>Stepsize</b> <b>8</b>
Default value	<b>0</b>
Unit of measure	<b>pixel</b>

Example 3.3. Usage of OffsetX

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

---

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

---

### 3.4. OffsetY

The y-offset is defined by this parameter. If a vertical mirroring is active SensorHeight limits the maximum Height (Height + YOffset). Furthermore the maximum Height is limited by the DRAM and SensorWidth (Sensor dimension need to fit into the DRAM).

Table 3.4. Parameter properties of OffsetY

Property	Value
Name	<b>OffsetY</b>
Display Name	<b>YOffset</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 65534 <b>Stepsize</b> 1
Default value	<b>0</b>
Unit of measure	<b>pixel</b>

Example 3.4. Usage of OffsetY

---

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

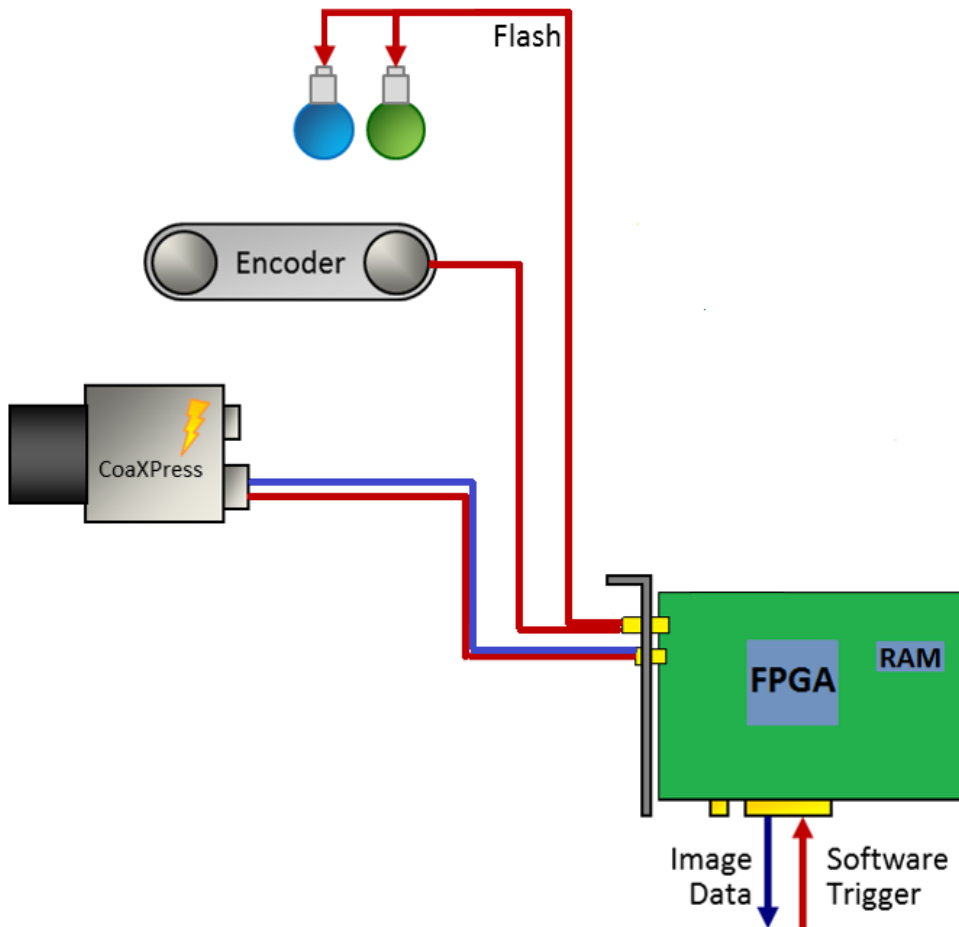
---

# Chapter 4. Trigger

The area trigger system enables the control of the image acquisition process of the interface card and the connected camera. 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 1C interface card has 4 inputs on the front IO connector. Check the hardware documentation for more information. The CXP-12 Interface Card 1C 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 4.1. CXP-12 Interface Card 1C Trigger System



In the following an introduction into the Basler CXP-12 Interface Card 1C trigger system is presented. Several trigger scenarios will show the possibilities and functionalities and will 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.

## 4.1. 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:

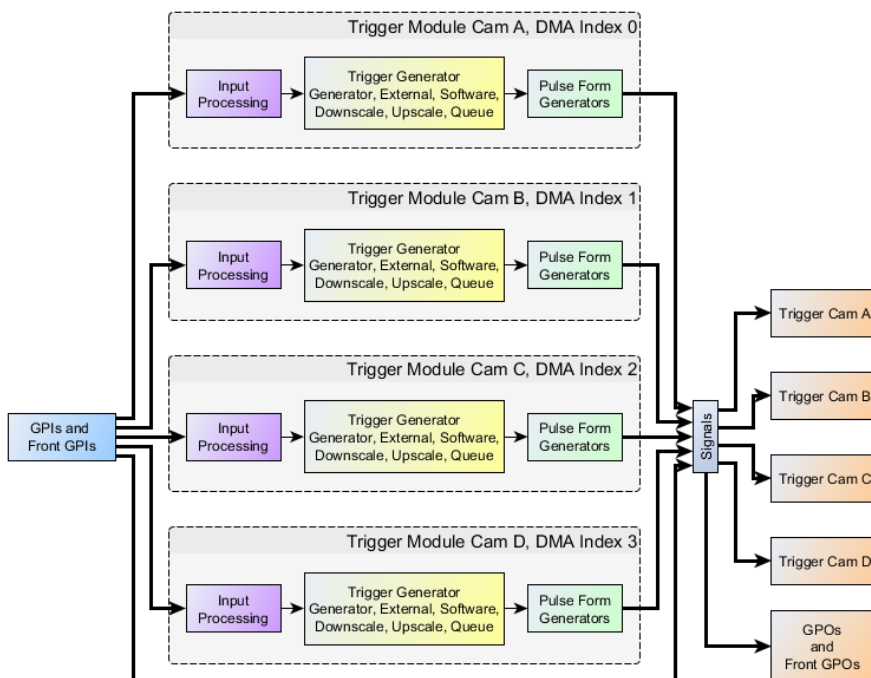
- Trigger signal generation for cameras and external devices.

- **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.
- 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.

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 4.2. Trigger System

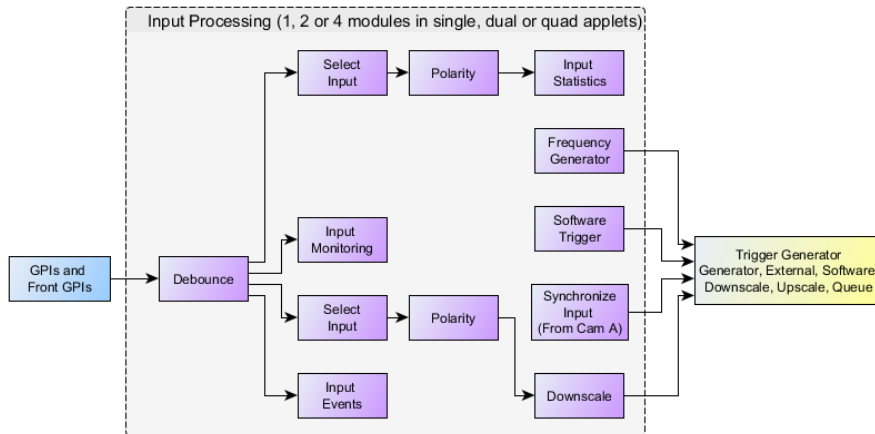


### 1. Trigger Input:

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 of the input signals.

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

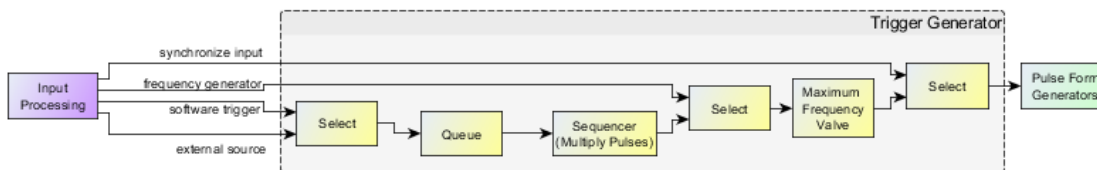
**Figure 4.3. Trigger Input Block Diagram**



### 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.

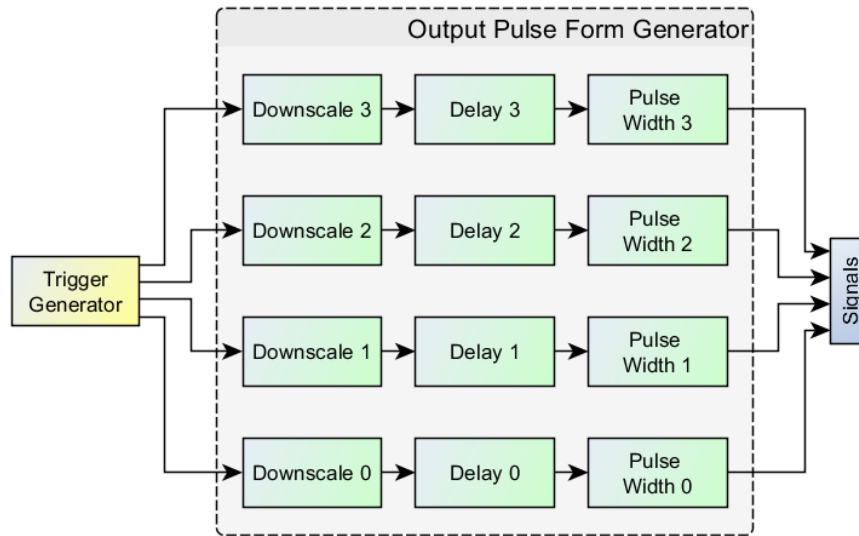
**Figure 4.4. Trigger Pulse Processing Block Diagram**



### 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.

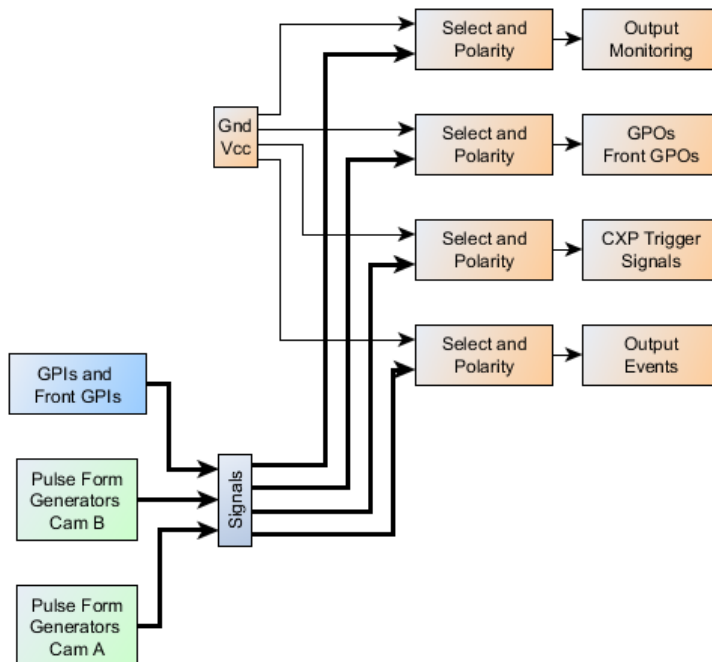
Figure 4.5. Trigger Pulse Processing Block Diagram



#### 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 .

Figure 4.6. Trigger Output Block Diagram



## 4.2. Digital Input/Output Mapping

The CXP-12 Interface Card 1C 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 1C 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 4.4.12, 'Digital Output' for explanation.

## 4.3. 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.

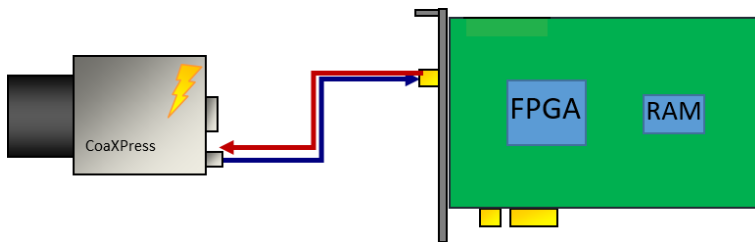
The scenarios show real life configurations. They explain the requirements, illustrate the inputs and outputs and list the required parameters and their values.

### 4.3.1. 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 4.7 shows the hardware setup. Only the camera connected to the interface card is required.

Figure 4.7. Generator Controlled Trigger Scenario

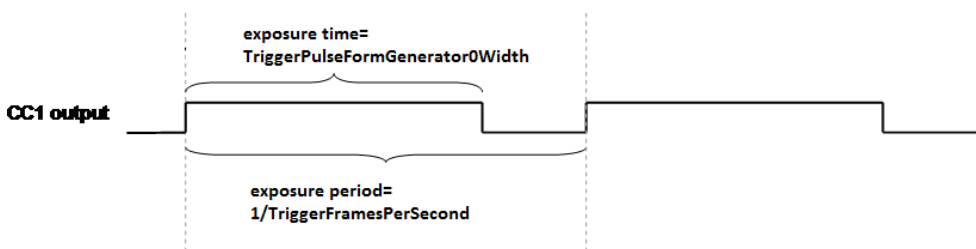


To put this scenario into practice, you will need to set your camera into an external trigger mode. Consult the vendor's user manual for more information.

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 4.8. 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.

- *TriggerFramesPerSecond* = 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 200 $\mu$ s. Thus, we will set the parameter to value 200 since the unit is  $\mu$ s.

- *TriggerCameraOutSelect* = **PulseGenerator0**

The only thing left to do is to allocate the output of pulse form generator 0 to the camera trigger output. If your camera requires low active signals, choose **NotPulseGenerator0** instead.

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 rate. 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 lose trigger pulses, as the camera cannot progress them. Get the maximum ranges from the camera's specification sheets.

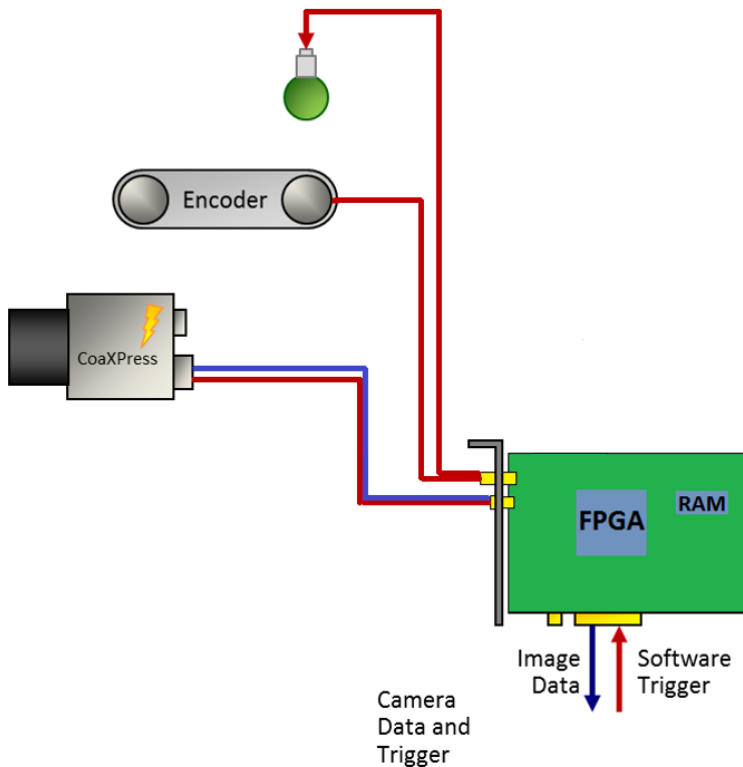
To stop the trigger pulse generation, set parameter *TriggerState* to **SyncStop**. The trigger system will then finalize the current pulse and stop any further output until the system is activated again. The asynchronous stop mode is not required in this scenario.

#### 4.3.2. 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.



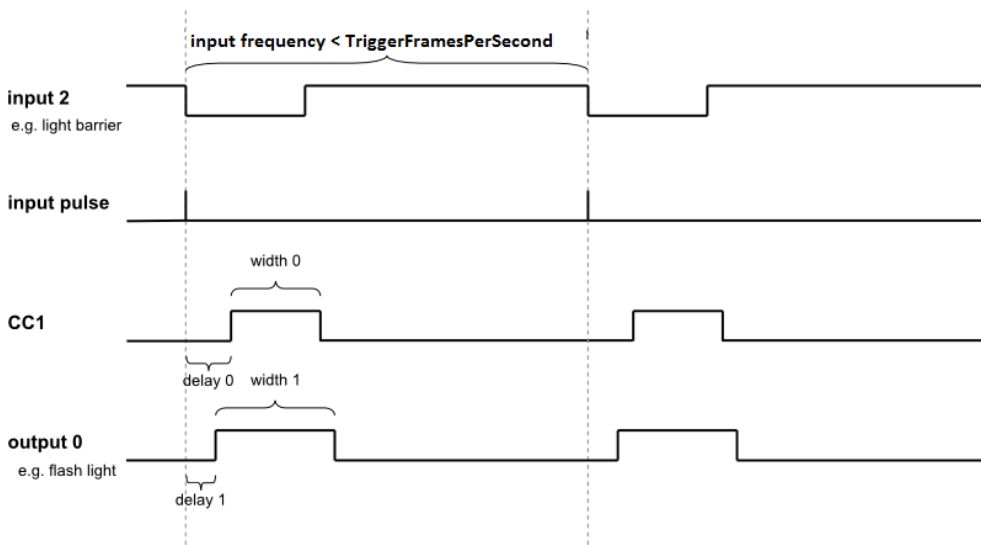
Figure 4.9. External Controlled Trigger Scenario



An exemplary waveform (Figure 4.10) 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 4.10. 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.

- *TriggerFramesPerSecond* = 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 lose 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. 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 *TriggerFramesPerSecond* and *TriggerExceededPeriodLimits* for more information.

More information on error detection and analysis can be found in scenario Section 4.3.9, '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 *TriggerFramesPerSecond* 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 CC1 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 CC 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.

- *TriggerCameraOutSelect* = **PulseGenerator0**

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*.

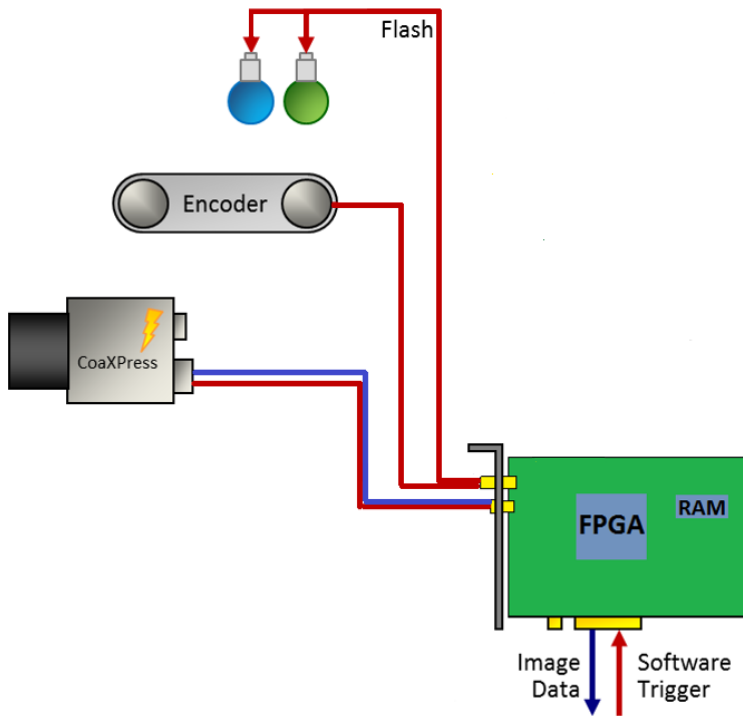
You will now receive images from the camera for each external trigger pulse. Compare the number of external pulses with the generated trigger signals and the received images for verification. Use parameter *TriggerInStatisticsPulseCount* of category Trigger Input -> Input Statistics and parameter *TriggerOutStatisticsPulseCount* 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.

### 4.3.3. 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 4.11. External Controlled Trigger Scenario



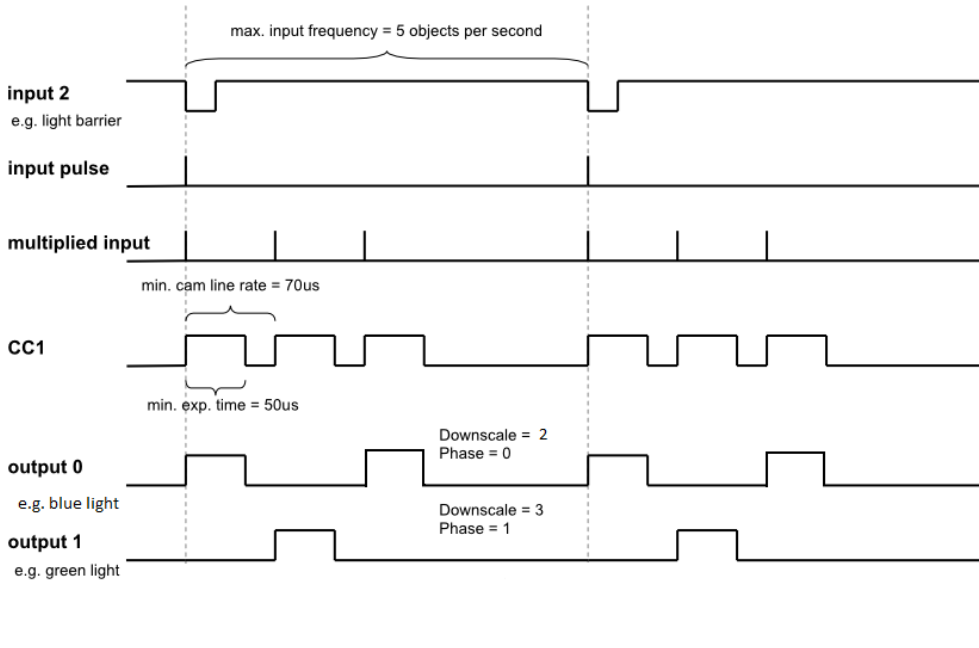
Before looking at the waveform, let's have a look at our fictitious hardware specifications.

Table 4.1. Fictitious Hardware Specifications of Trigger Scenario Three Light Sources

Element	Limit
Object Speed	Max. 100 Objects per Second
Minimum Camera Exposure Time	50 $\mu$ s
Minimum Camera Frame Period	70 $\mu$ s

The object speed is 100 objects per second. The minimum camera exposure time is 50 $\mu$ s at a minimum camera frame period of 70 $\mu$ s. Thus we only need 210 $\mu$ s 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 70 $\mu$ s 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 4.12. 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 *TriggerFramesPerSecond*

- *TriggerFramesPerSecond* = 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 CC 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 downsampled 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.

- *TriggerCameraOutSelect* = **PulseGenerator0**

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 transferred image is the first image out of a sequence.

#### 4.3.4. Software Trigger

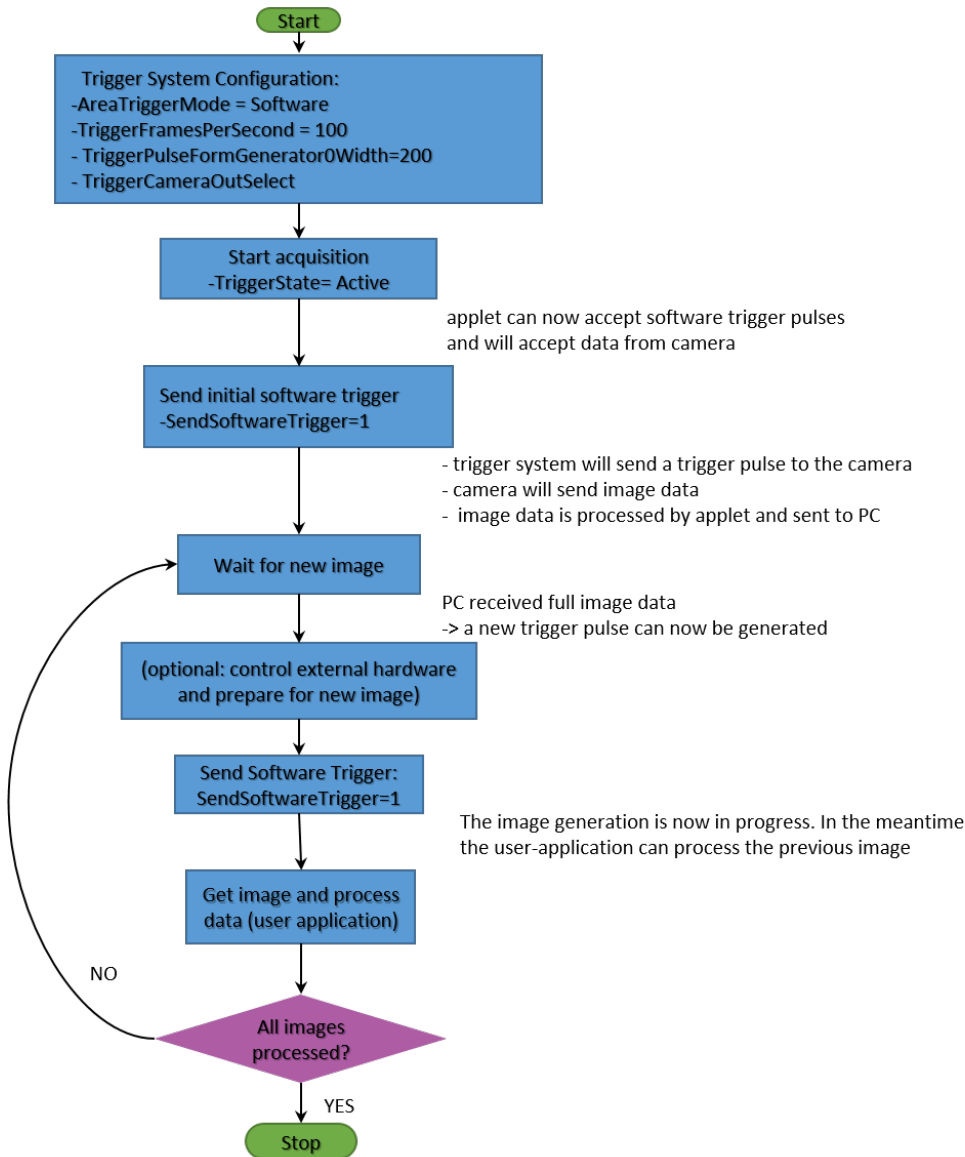
The previous examples showed the use of the internal frequency generator and the use of external trigger pulses to trigger your camera and generate digital output signals. Another trigger mode is the software trigger. In this mode, you can control the generation of each trigger pulse using your software 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  $1 / \text{TriggerFramesPerSecond}$  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 *TriggerFramesPerSecond* 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 transferred 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.

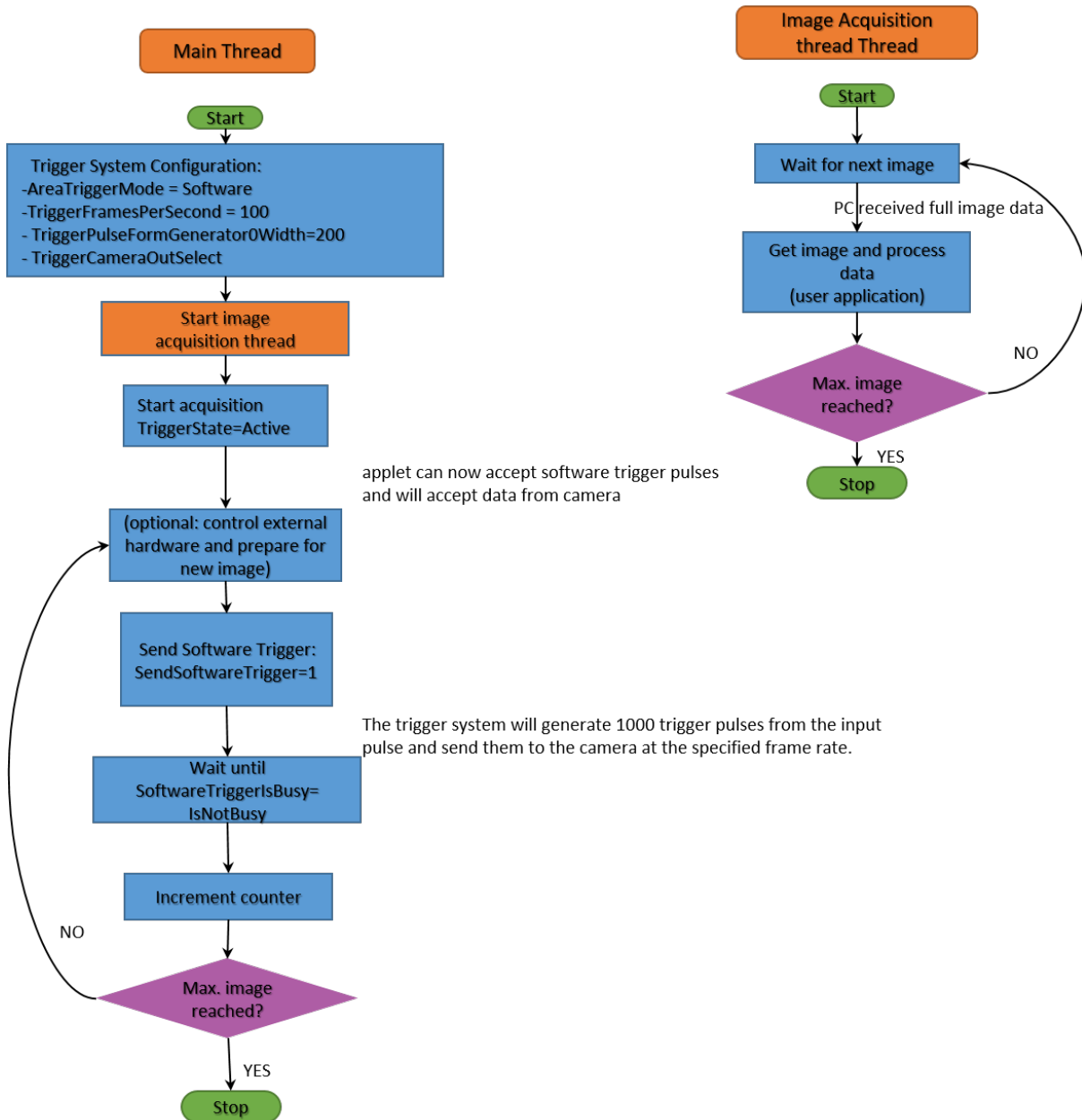
Figure 4.13. Flowchart of Software Application Using the Software Trigger



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 4.14. 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 *TriggerFramesPerSecond*. 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.

#### 4.3.5. 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

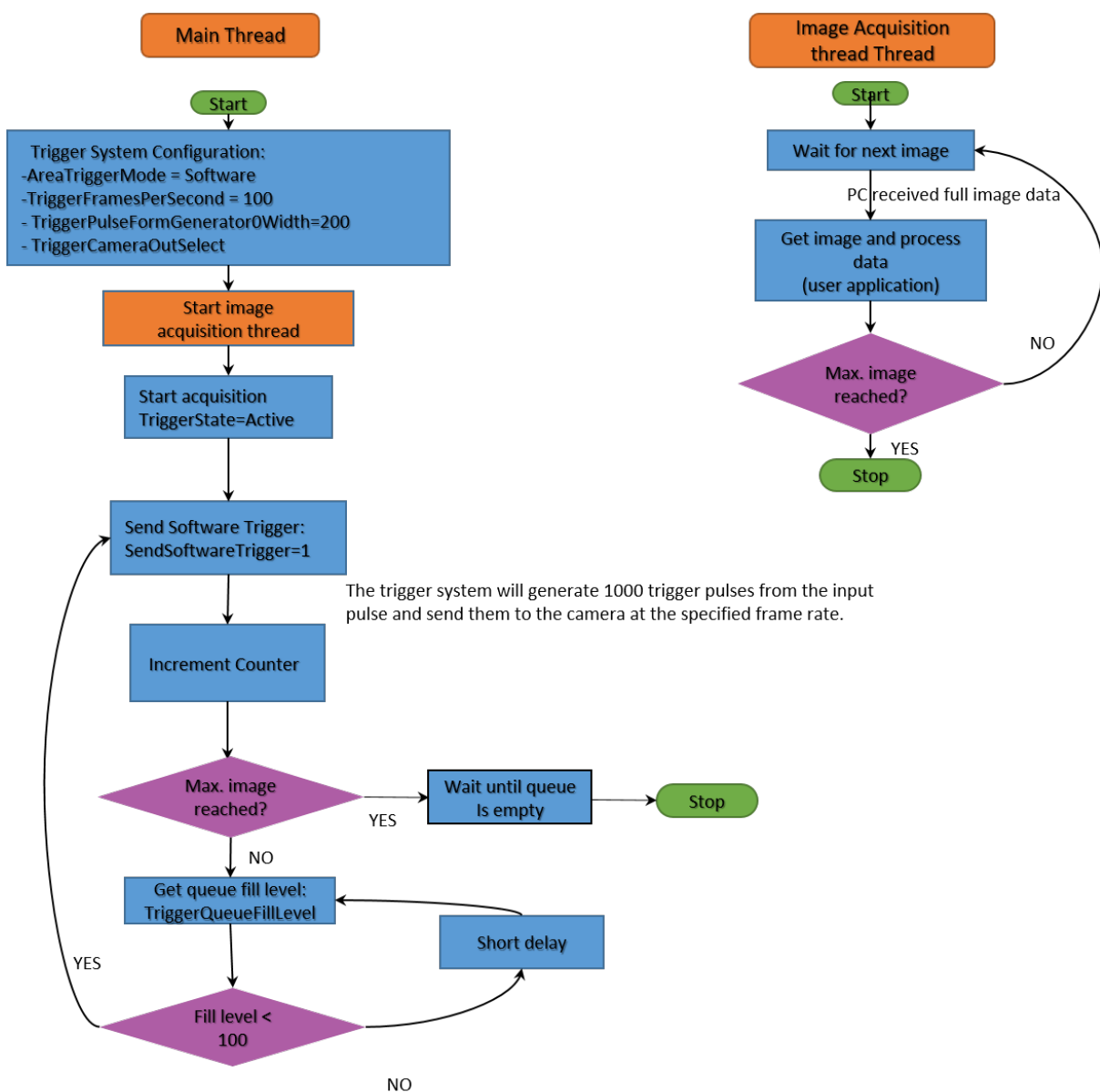


pulses at the maximum allowed frequency specified by *TriggerFramesPerSecond*. Therefore, we can write to *SendSoftwareTrigger* multiple times even if the trigger system is still busy. Parameter *SoftwareTriggerIsBusy* will only have value **Busy** if the queue is full. Instead of writing multiple times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

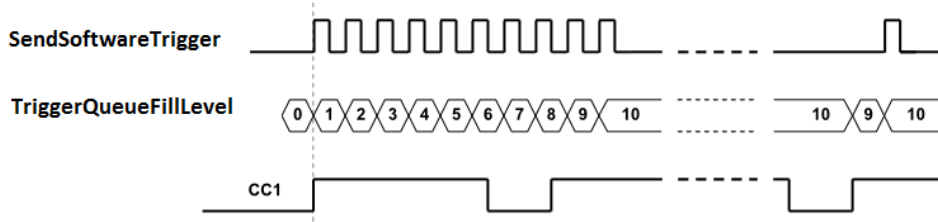
Figure 4.15. Flowchart of Software Application Using the Software Trigger with Trigger Queue



When having a look at the waveform (Figure 4.16) 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 4.16. 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**.

#### 4.3.6. External Trigger with Trigger Queue

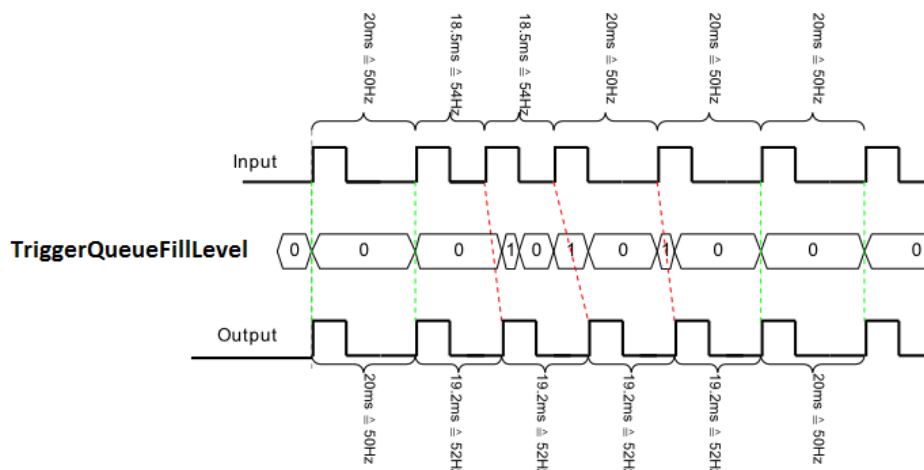
Of course, we can use the trigger queue with external triggers, too. This will give us a possibility to buffer 'jumpy' external encoders or any other external trigger signal generators. Let's suppose an external encoder which is configured to generate trigger pulses with a frequency of 50Hz and a camera which can be run at a maximum frequency of 52Hz. Thus, we set parameter *TriggerFramesPerSecond* 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 lose at least one pulse. You can check this 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 4.17. Using External Trigger Signal Sources together with the Trigger Queue



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

#### 4.3.7. 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:

- *TriggerCameraOutSelect* = *BYPASS\_GPI\_0*
- *TriggerOutSelectFrontGPO0* = *BYPASS\_FRONT\_GPI\_1*

#### 4.3.8. 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. Some applets support more than one camera. In this case, the same parameters for all cameras should be set. They may share the same trigger input.

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 (CC) 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**.

#### 4.3.9. 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 4.3.2, 'External Trigger Signals / IO Triggered') or use the trigger queue (see Section 4.3.6, '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 *TriggerFramesPerSecond*. 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.

- Camera Response Check

Trigger pulses might get lost in the link to the camera or the trigger frequency is too 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 set a register. Check Section 4.4.12.2, '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 too 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.

## 4.4. Parameters

### 4.4.1. 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 in this applet. Multi-camera applets include this option. Check the respective applet documentations in this case.



## 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 *TriggerFramesPerSecond* in external and software triggered mode. Always set this parameter for these modes.

Table 4.2. Parameter properties of AreaTriggerMode

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

Example 4.1. Usage of AreaTriggerMode

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

### 4.4.2. 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. 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 4.3. Parameter properties of TriggerState

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

Example 4.2. Usage of TriggerState

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

#### 4.4.3. TriggerFramesPerSecond

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. However, if the trigger queue is enabled, the exceeding pulses will be buffered and output at the maximum frequency which is defined by *TriggerFramesPerSecond*. Thus, the parameter also defines the maximum queue output frequency. Moreover, it defines the maximum sequencer frequency. The maximum valid value of *TriggerFramesPerSecond* 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 4.1. Dependency of Frequency and Pulse Form Generators

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

- $\text{fps} = \text{TriggerFramesPerSecond}$
- $\text{WIDTH}[0..3] = \text{TriggerPulseFormGenerator}[0..3]\text{Width}$
- $\text{DELAY}[0..3] = \text{TriggerPulseFormGenerator}[0..3]\text{Delay}$
- $\text{DOWNSCALE}[0..3] = \text{TriggerPulseFormGenerator}[0..3]\text{Downscale}$

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

Table 4.4. Parameter properties of TriggerFramesPerSecond

Property	Value
Name	<b>TriggerFramesPerSecond</b>
Display Name	<b>Frames Sec</b>
Interface	<b>IFloat</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 0.07275957614183425 <b>Maximum</b> 3.1249999999999996E7 <b>Stepsize</b> 2.220446049250313E-16
Default value	<b>8.0</b>
Unit of measure	<b>Hz</b>

Example 4.3. Usage of TriggerFramesPerSecond

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

#### 4.4.4. Trigger Input

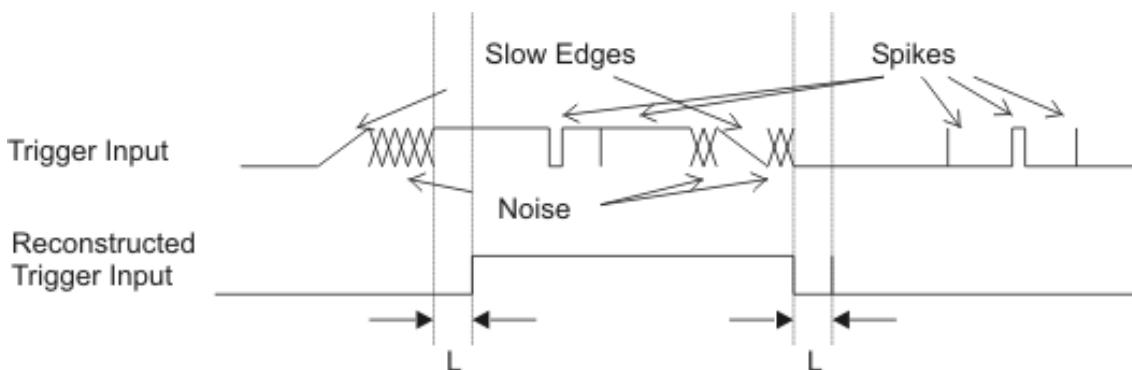
The parameters of category Trigger Input are used to configure the input source of the trigger system. The category is divided into sub categories. All external sources are configured in category external. Category software trigger allows the configuration, monitoring and controlling of software trigger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

##### 4.4.4.1. External

##### 4.4.4.1.1. 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 4.18. Faulty Signal and it's Reconstruction



L : stability criterion of hysteresis

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  $\mu\text{s}$ . 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.

Table 4.5. Parameter properties of TriggerInDebounce

Property	Value
Name	<b>TriggerInDebounce</b>
Display Name	<b>Input Debounce</b>
Interface	<b>IFloat</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 0.009600000000000001 <b>Maximum</b> 0.6048000000000001 <b>Stepsize</b> 0.009600000000000001
Default value	<b>1.0</b>
Unit of measure	<b><math>\mu\text{s}</math></b>

Example 4.4. Usage of TriggerInDebounce

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

#### 4.4.4.1.2. 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 its digital inputs 1 and 3.

Table 4.6. Parameter properties of FrontGPI

Property	Value
Name	<b>FrontGPI</b>
Display Name	<b>Digital Input at Front GPI</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 15 <b>Stepsize</b> 1

Example 4.5. Usage of FrontGPI

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

#### 4.4.4.1.3. TriggerInSource

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



Table 4.7. Parameter properties of TriggerInSource

Property	Value
Name	<b>TriggerInSource</b>
Display Name	<b>Source</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Beginner</b>
Allowed values	<b>TriggerInSourceFrontGPIO0</b> Front GPI Trigger Source 0 <b>TriggerInSourceFrontGPIO1</b> Front GPI Trigger Source 1 <b>TriggerInSourceFrontGPIO2</b> Front GPI Trigger Source 2 <b>TriggerInSourceFrontGPIO3</b> Front GPI Trigger Source 3
Default value	<b>TriggerInSourceFrontGPIO0</b>

Example 4.6. Usage of TriggerInSource

```
/* Set */ TriggerInSource = TriggerInSourceFrontGPIO0;
/* Get */ value_ = TriggerInSource;
```

#### 4.4.4.1.4. TriggerInPolarity

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

Table 4.8. Parameter properties of TriggerInPolarity

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

Example 4.7. Usage of TriggerInPolarity

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

#### 4.4.4.1.5. 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 4.9. Parameter properties of TriggerInDownscale

Property	Value
Name	<b>TriggerInDownscale</b>
Display Name	<b>Input Downscale</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 1 <b>Maximum</b> 2147483647 <b>Stepsize</b> 1
Default value	<b>1</b>

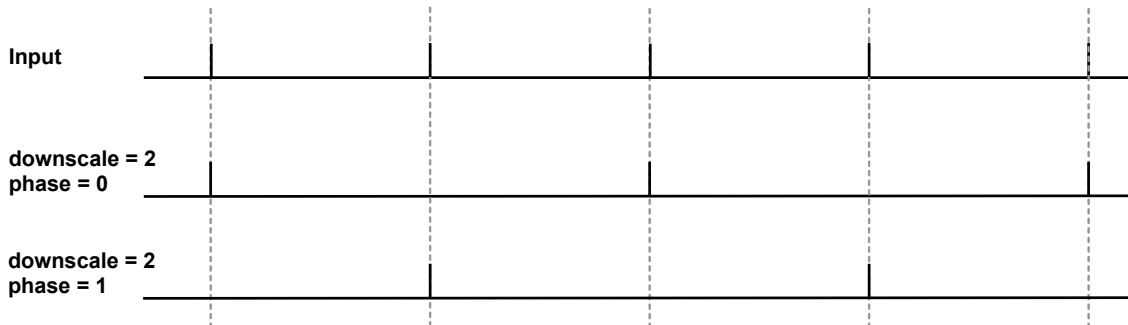
Example 4.8. Usage of TriggerInDownscale

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

#### 4.4.4.1.6. 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 4.19. 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 4.10. Parameter properties of TriggerInDownscalePhase

Property	Value
Name	<b>TriggerInDownscalePhase</b>
Display Name	<b>Downscale Phase</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 4294967294 <b>Stepsize</b> 1
Default value	<b>0</b>

**Example 4.9. Usage of TriggerInDownscalePhase**

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

**4.4.4.2. Software Trigger****4.4.4.2.1. 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 *TriggerFramesPerSecond*.

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 FG\_SOFTWARE\_TRIGGER\_BUSY error.

**Table 4.11. Parameter properties of SendSoftwareTrigger**

Property	Value
Name	<b>SendSoftwareTrigger</b>
Display Name	<b>Send Pulse</b>
Interface	<b>ICommand</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Beginner</b>

**Example 4.10. Usage of SendSoftwareTrigger**

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

**4.4.4.2.2. 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 4.12. Parameter properties of SoftwareTriggerIsBusy**

Property	Value
Name	<b>SoftwareTriggerIsBusy</b>
Display Name	<b>Software Trigger Busy</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Busy</b> Busy Flag is set <b>NotBusy</b> Busy Flag is not set

**Example 4.11. Usage of SoftwareTriggerIsBusy**

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

#### 4.4.4.2.3. 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 *TriggerFramesPerSecond*. The value decrement is given in steps of `FG_TRIGGER_MULTIPLY_PULSES`.

Table 4.13. Parameter properties of SoftwareTriggerQueueFillLevel

Property	Value
Name	<b>SoftwareTriggerQueueFillLevel</b>
Display Name	<b>Software Trigger Queue Filllevel</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 2040 <b>Stepsize</b> 1
Unit of measure	<b>pulses</b>

Example 4.12. Usage of SoftwareTriggerQueueFillLevel

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

#### 4.4.4.3. 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.

##### 4.4.4.3.1. TriggerInStatisticsSource

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

Table 4.14. Parameter properties of TriggerInStatisticsSource

Property	Value
Name	<b>TriggerInStatisticsSource</b>
Display Name	<b>Statistics Source</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>TriggerInSourceFrontGPIO</b> Front GPI Trigger Source 0 <b>TriggerInSourceFrontGPIO1</b> Front GPI Trigger Source 1 <b>TriggerInSourceFrontGPIO2</b> Front GPI Trigger Source 2 <b>TriggerInSourceFrontGPIO3</b> Front GPI Trigger Source 3
Default value	<b>TriggerInSourceFrontGPIO</b>

Example 4.13. Usage of TriggerInStatisticsSource

```
/* Set */ TriggerInStatisticsSource = TriggerInSourceFrontGPIO;
```

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

#### 4.4.4.3.2. TriggerInStatisticsPolarity

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

Table 4.15. Parameter properties of TriggerInStatisticsPolarity

Property	Value
Name	<b>TriggerInStatisticsPolarity</b>
Display Name	<b>Statistics Polarity</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>LowActive</b> Low Active <b>HighActive</b> High Active
Default value	<b>HighActive</b>

Example 4.14. Usage of TriggerInStatisticsPolarity

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

#### 4.4.4.3.3. 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 4.16. Parameter properties of TriggerInStatisticsPulseCount

Property	Value
Name	<b>TriggerInStatisticsPulseCount</b>
Display Name	<b>Input Pulses</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 65535 <b>Stepsize</b> 1
Unit of measure	<b>pulses</b>

Example 4.15. Usage of TriggerInStatisticsPulseCount

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

#### 4.4.4.3.4. TriggerInStatisticsPulseCountClear

Clear the input pulse counter by writing to this register.

Table 4.17. Parameter properties of TriggerInStatisticsPulseCountClear

Property	Value
Name	<b>TriggerInStatisticsPulseCountClear</b>
Display Name	<b>Clear Pulse Count</b>
Interface	<b> ICommand</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>

Example 4.16. Usage of TriggerInStatisticsPulseCountClear

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

#### 4.4.4.3.5. 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 4.18. Parameter properties of TriggerInStatisticsFrequency

Property	Value
Name	<b>TriggerInStatisticsFrequency</b>
Display Name	<b>Current Frequency</b>
Interface	<b>IFloat</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0.0 <b>Maximum</b> 3.125E8 <b>Stepsize</b> 2.220446049250313E-16
Unit of measure	<b>Hz</b>

Example 4.17. Usage of TriggerInStatisticsFrequency

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

#### 4.4.4.3.6. TriggerInStatisticsMinimumFrequency

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

Table 4.19. Parameter properties of TriggerInStatisticsMinimumFrequency

Property	Value
Name	<b>TriggerInStatisticsMinimumFrequency</b>
Display Name	<b>Minimum Frequency</b>
Interface	<b>IFloat</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0.0 <b>Maximum</b> 3.125E8 <b>Stepsize</b> 2.220446049250313E-16
Unit of measure	<b>Hz</b>

**Example 4.18. Usage of TriggerInStatisticsMinimumFrequency**

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

**4.4.4.3.7. TriggerInStatisticsMaximumFrequency**

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

**Table 4.20. Parameter properties of TriggerInStatisticsMaximumFrequency**

Property	Value
Name	<b>TriggerInStatisticsMaximumFrequency</b>
Display Name	<b>Maximum Frequency</b>
Interface	<b>IFloat</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0.0 <b>Maximum</b> 3.125E8 <b>Stepsize</b> 2.220446049250313E-16
Unit of measure	<b>Hz</b>

**Example 4.19. Usage of TriggerInStatisticsMaximumFrequency**

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

**4.4.4.3.8. TriggerInStatisticsMinMaxFrequencyClear**

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

**Table 4.21. Parameter properties of TriggerInStatisticsMinMaxFrequencyClear**

Property	Value
Name	<b>TriggerInStatisticsMinMaxFrequencyClear</b>
Display Name	<b>Clear Min Max Frequency</b>
Interface	<b>ICommand</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>

**Example 4.20. Usage of TriggerInStatisticsMinMaxFrequencyClear**

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

**4.4.5. 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 *TriggerFramesPerSecond*. Thus, the inserted pulses are not evenly distributed

between the input pulses, they will be inserted with a delay specified by *TriggerFramesPerSecond*. 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 *TriggerFramesPerSecond* 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*.

#### 4.4.5.1. TriggerMultiplyPulses

Set the trigger input multiplication factor.

Table 4.22. Parameter properties of TriggerMultiplyPulses

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

Example 4.21. Usage of TriggerMultiplyPulses

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

#### 4.4.6. Queue

The maximum trigger output frequency is limited to the the setting of parameter *TriggerFramesPerSecond*. 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*.

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.

##### 4.4.6.1. TriggerQueueMode



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

Table 4.23. Parameter properties of TriggerQueueMode

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

Example 4.22. Usage of TriggerQueueMode

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

#### 4.4.6.2. 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 4.24. Parameter properties of TriggerQueueFillLevel

Property	Value
Name	<b>TriggerQueueFillLevel</b>
Display Name	<b>Filllevel</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 2040 <b>Stepsize</b> 1
Unit of measure	<b>pulses</b>

Example 4.23. Usage of TriggerQueueFillLevel

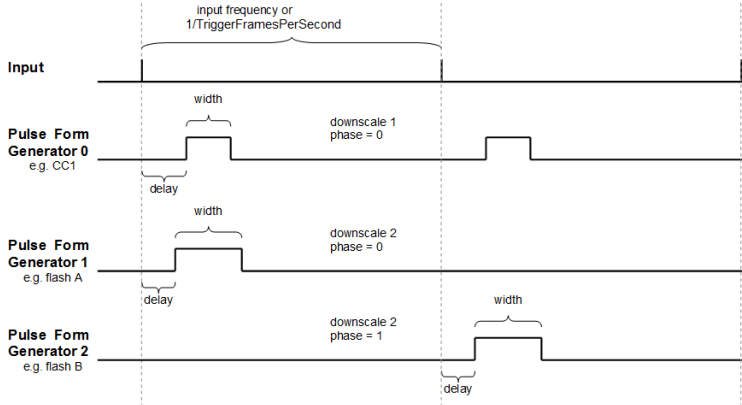
```
/* Get */ value_ = TriggerQueueFillLevel;
```

#### 4.4.7. 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 4.20. Pulse Form Generators



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

Equation 4.2. Dependency of Frequency and Pulse Form Generators

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

- $\text{fps} = \text{TriggerFramesPerSecond}$
- $\text{WIDTH}[0..3] = \text{TriggerPulseFormGenerator}[0..3]\text{Width}$
- $\text{DELAY}[0..3] = \text{TriggerPulseFormGenerator}[0..3]\text{Delay}$
- $\text{DOWNSCALE}[0..3] = \text{TriggerPulseFormGenerator}[0..3]\text{Downscale}$

#### 4.4.7.1. 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 4.25. Parameter properties of TriggerPulseFormGenerator0Downscale

Property	Value
Name	<b>TriggerPulseFormGenerator0Downscale</b>
Display Name	<b>Downscale</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 1 <b>Maximum</b> 7 <b>Stepsize</b> 1
Default value	<b>1</b>

Example 4.24. Usage of TriggerPulseFormGenerator0Downscale

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

#### 4.4.7.2. 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 4.4.7, '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 4.26. Parameter properties of TriggerPulseFormGenerator0DownscalePhase

Property	Value
Name	<b>TriggerPulseFormGenerator0DownscalePhase</b>
Display Name	<b>Phase</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 6 <b>Stepsize</b> 1
Default value	<b>0</b>

Example 4.25. Usage of TriggerPulseFormGenerator0DownscalePhase

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

#### 4.4.7.3. 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  $\mu\text{s}$ .

Table 4.27. Parameter properties of TriggerPulseFormGenerator0Delay

Property	Value
Name	<b>TriggerPulseFormGenerator0Delay</b>
Display Name	<b>Delay</b>
Interface	<b>IFloat</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 0.0 <b>Maximum</b> 3.4E7 <b>Stepsize</b> 0.0032
Default value	<b>0.0</b>
Unit of measure	<b><math>\mu\text{s}</math></b>

Example 4.26. Usage of TriggerPulseFormGenerator0Delay

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

#### 4.4.7.4. 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  $\mu\text{s}$ .

Table 4.28. Parameter properties of TriggerPulseFormGenerator0Width

Property	Value
Name	<b>TriggerPulseFormGenerator0Width</b>
Display Name	<b>Signal Width</b>
Interface	<b>IFloat</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum</b> 1.0 <b>Maximum</b> 6.8E7 <b>Stepsize</b> 0.0032
Default value	<b>4.0</b>
Unit of measure	<b><math>\mu\text{s}</math></b>

**Example 4.27. Usage of TriggerPulseFormGenerator0Width**

---

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

---

### 4.4.8. Pulse Form Generator 1

The settings for pulse form generator 1 are equal to those of pulse form generator 0. Please read Section 4.4.7, 'Pulse Form Generator 0' for a detailed description.

### 4.4.9. Pulse Form Generator 2

The settings for pulse form generator 2 are equal to those of pulse form generator 0. Please read Section 4.4.7, 'Pulse Form Generator 0' for a detailed description.

### 4.4.10. Pulse Form Generator 3

The settings for pulse form generator 3 are equal to those of pulse form generator 0. Please read Section 4.4.7, 'Pulse Form Generator 0' for a detailed description.

### 4.4.11. Camera Out Signal Mapping

The camera interface of the CXP-12 Interface Card 1C is equipped with a trigger output channel to trigger the camera. Moreover, 8 general purpose outputs to the camera exist. Please, consult the vendor's manual of your camera to identify the required signals and their mapping.

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 either directly connect one of the four sources to a camera signal channel or invert the signal if you need low active signals.

- Ground or Vcc if a CC line is not used or you want to temporarily deactivate or activate the line.
- The input bypass

The trigger system will ignore the signal length of the input signals. If you want to directly bypass one of the inputs to a camera signal channel, you can set the respective channel to bypass or the inverted bypass.

#### 4.4.11.1. TriggerCameraOutSelect

Table 4.29. Parameter properties of TriggerCameraOutSelect

Property	Value	
Name	TriggerCameraOutSelect	
Display Name	Camera Trigger Out Mapping	
Interface	IEnumeration	
Access policy	Read/Write/Change	
Visibility	Beginner	
Allowed values	<div> <div>VCC</div> <div>GND</div> <div>CamAPulseGenerator0</div> <div>CamAPulseGenerator1</div> <div>CamAPulseGenerator2</div> <div>CamAPulseGenerator3</div> <div>NotCamAPulseGenerator0</div> <div>NotCamAPulseGenerator1</div> <div>NotCamAPulseGenerator2</div> <div>NotCamAPulseGenerator3</div> <div>BypassFrontGPI0</div> <div>NotBypassFronGPI0</div> <div>BypassFrontGPI1</div> <div>NotBypassFronGPI1</div> <div>BypassFrontGPI2</div> <div>NotBypassFronGPI2</div> <div>BypassFrontGPI3</div> <div>NotBypassFronGPI3</div> <div>PulseGenerator0</div> <div>PulseGenerator1</div> <div>PulseGenerator2</div> <div>PulseGenerator3</div> <div>NotPulseGenerator0</div> <div>NotPulseGenerator1</div> <div>NotPulseGenerator2</div> <div>NotPulseGenerator3</div> </div> <div> <div>Vcc</div> <div>Gnd</div> <div>Cam A Pulse Generator 0</div> <div>Cam A Pulse Generator 1</div> <div>Cam A Pulse Generator 2</div> <div>Cam A Pulse Generator 3</div> <div>Not Cam A Pulse Generator 0</div> <div>Not Cam A Pulse Generator 1</div> <div>Not Cam A Pulse Generator 2</div> <div>Not Cam A Pulse Generator 3</div> <div>Bypass Front-GPI 0</div> <div>Not Bypass Front-GPI 0</div> <div>Bypass Front-GPI 1</div> <div>Not Bypass Front-GPI 1</div> <div>Bypass Front-GPI 2</div> <div>Not Bypass Front-GPI 2</div> <div>Bypass Front-GPI 3</div> <div>Not Bypass Front-GPI 3</div> <div>Pulse Generator 0</div> <div>Pulse Generator 1</div> <div>Pulse Generator 2</div> <div>Pulse Generator 3</div> <div>Not Pulse Generator 0</div> <div>Not Pulse Generator 1</div> <div>Not Pulse Generator 2</div> <div>Not Pulse Generator 3</div> </div>	
Default value	PulseGenerator0	

Example 4.28. Usage of TriggerCameraOutSelect

```

/* Set */ TriggerCameraOutSelect = PulseGenerator0;
/* Get */ value_ = TriggerCameraOutSelect;

```

#### 4.4.12. Digital Output

The CXP-12 Interface Card 1C 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.

- 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.

#### 4.4.12.1. TriggerOutSelectFrontGPO0 et al.



#### Note

This description applies also to the following parameters: TriggerOutSelectFrontGPO1

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

Table 4.30. Parameter properties of TriggerOutSelectFrontGPO0

Property	Value	
Name	<b>TriggerOutSelectFrontGPO0</b>	
Display Name	<b>Output Front GPO 0</b>	
Interface	<b>IEnumeration</b>	
Access policy	<b>Read/Write/Change</b>	
Visibility	<b>Beginner</b>	
Allowed values	<div> <div>VCC</div> <div>GND</div> <div>CamAPulseGenerator0</div> <div>CamAPulseGenerator1</div> <div>CamAPulseGenerator2</div> <div>CamAPulseGenerator3</div> <div>NotCamAPulseGenerator0</div> <div>NotCamAPulseGenerator1</div> <div>NotCamAPulseGenerator2</div> <div>NotCamAPulseGenerator3</div> <div>BypassFrontGPI0</div> <div>NotBypassFronGPI0</div> <div>BypassFrontGPI1</div> <div>NotBypassFronGPI1</div> <div>BypassFrontGPI2</div> <div>NotBypassFronGPI2</div> <div>BypassFrontGPI3</div> <div>NotBypassFronGPI3</div> <div>PulseGenerator0</div> <div>PulseGenerator1</div> <div>PulseGenerator2</div> <div>PulseGenerator3</div> <div>NotPulseGenerator0</div> <div>NotPulseGenerator1</div> <div>NotPulseGenerator2</div> <div>NotPulseGenerator3</div> </div> <div> <div>Vcc</div> <div>Gnd</div> <div>Cam A Pulse Generator 0</div> <div>Cam A Pulse Generator 1</div> <div>Cam A Pulse Generator 2</div> <div>Cam A Pulse Generator 3</div> <div>Not Cam A Pulse Generator 0</div> <div>Not Cam A Pulse Generator 1</div> <div>Not Cam A Pulse Generator 2</div> <div>Not Cam A Pulse Generator 3</div> <div>Bypass Front-GPI 0</div> <div>Not Bypass Front-GPI 0</div> <div>Bypass Front-GPI 1</div> <div>Not Bypass Front-GPI 1</div> <div>Bypass Front-GPI 2</div> <div>Not Bypass Front-GPI 2</div> <div>Bypass Front-GPI 3</div> <div>Not Bypass Front-GPI 3</div> <div>Pulse Generator 0</div> <div>Pulse Generator 1</div> <div>Pulse Generator 2</div> <div>Pulse Generator 3</div> <div>Not Pulse Generator 0</div> <div>Not Pulse Generator 1</div> <div>Not Pulse Generator 2</div> <div>Not Pulse Generator 3</div> </div>	
Default value	<b>NotCamAPulseGenerator0</b>	

Example 4.29. Usage of TriggerOutSelectFrontGPO0

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

#### 4.4.12.2. 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.

#### 4.4.12.2.1. TriggerExceededPeriodLimits

This read-only register has value **Yes** if the input signal frequency exceeded the maximum allowed frequency defined by parameter *TriggerFramesPerSecond*. 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 4.31. Parameter properties of TriggerExceededPeriodLimits

Property	Value
Name	<b>TriggerExceededPeriodLimits</b>
Display Name	<b>Trigger Exceeded Period Limits</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Yes</b> Yes <b>No</b> No

Example 4.30. Usage of TriggerExceededPeriodLimits

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

#### 4.4.12.2.2. TriggerExceededPeriodLimitsClear

Reset *TriggerExceededPeriodLimits* with this parameter.

Table 4.32. Parameter properties of TriggerExceededPeriodLimitsClear

Property	Value
Name	<b>TriggerExceededPeriodLimitsClear</b>
Display Name	<b>Clear Exceeded Period Limits Register</b>
Interface	<b>ICommand</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>

Example 4.31. Usage of TriggerExceededPeriodLimitsClear

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

#### 4.4.12.2.3. TriggerOutStatisticsSource



Table 4.33. Parameter properties of TriggerOutStatisticsSource

Property	Value
Name	<b>TriggerOutStatisticsSource</b>
Display Name	<b>Source</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>PulseGenerator0</b> Pulse Generator 0 <b>PulseGenerator1</b> Pulse Generator 1 <b>PulseGenerator2</b> Pulse Generator 2 <b>PulseGenerator3</b> Pulse Generator 3
Default value	<b>PulseGenerator0</b>

Example 4.32. Usage of TriggerOutStatisticsSource

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

#### 4.4.12.2.4. TriggerOutStatisticsPulseCount

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

Table 4.34. Parameter properties of TriggerOutStatisticsPulseCount

Property	Value
Name	<b>TriggerOutStatisticsPulseCount</b>
Display Name	<b>Pulse Count</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 65535 <b>Stepsize</b> 1
Unit of measure	<b>pulses</b>

Example 4.33. Usage of TriggerOutStatisticsPulseCount

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

#### 4.4.12.2.5. TriggerOutStatisticsPulseCountClear

Output pulse count register clear.

Table 4.35. Parameter properties of TriggerOutStatisticsPulseCountClear

Property	Value
Name	<b>TriggerOutStatisticsPulseCountClear</b>
Display Name	<b>Clear pulse counter</b>
Interface	<b>ICommand</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>

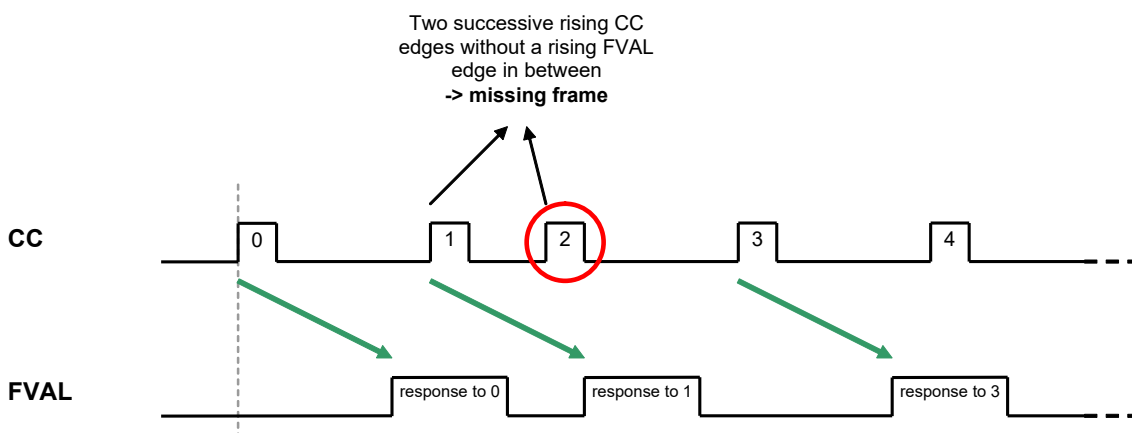
**Example 4.34. Usage of TriggerOutStatisticsPulseCountClear**

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

**4.4.12.2.6. MissingCameraFrameResponse**

This applet is equipped with a detection of missing camera frame responses to trigger pulses. If the camera will 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 will send a frame to the interface card. If a trigger pulse gets lost, or the camera cannot send a frame, this register will be set to **Yes**. Technically, between two output signal edges, a 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 following figure illustrates the behavior.

**Figure 4.21. 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 correct for all camera models due to different timings.

**Select Camera Control/Trigger Signal Line**

Take care to 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**

Keep in mind to start the acquisition before activating the trigger. Otherwise, the trigger pulses sent will get lost. Also keep in mind, that any changes of the camera configuration might result in invalid data transfers.

**Table 4.36. Parameter properties of MissingCameraFrameResponse**

Property	Value
Name	<b>MissingCameraFrameResponse</b>
Display Name	<b>Missing Camera Frame Response</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Yes</b> Yes <b>No</b> No

**Example 4.35. Usage of MissingCameraFrameResponse**

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

**4.4.12.2.7. MissingCameraFrameResponseClear**

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

**Table 4.37. Parameter properties of MissingCameraFrameResponseClear**

Property	Value
Name	<b>MissingCameraFrameResponseClear</b>
Display Name	<b>Clear Frame Response Register</b>
Interface	<b>ICommand</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>

**Example 4.36. Usage of MissingCameraFrameResponseClear**

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

# Chapter 5. 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. 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. In this case, the applet is in an illegal condition and the correct functionality can not be guaranteed. As overflows occur in very short periods, there is no possibility to detect an overflow in this specific applet. Ensure that the buffer fill level always is at a minimum. In other applets, events can be used to detect overflow states.

## 5.1. 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 5.1. Parameter properties of FillLevel

Property	Value
Name	<b>FillLevel</b>
Display Name	<b>Buffer fill level</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Beginner</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 100</b> <b>Stepsize 1</b>
Unit of measure	<b>%</b>

Example 5.1. Usage of FillLevel

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

## 5.2. 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 microDisplay will continuously poll the value, thus the occurrence of an overflow might not be visible in microDisplay.

Table 5.2. Parameter properties of Overflow

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

Example 5.2. Usage of Overflow

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

---

# Chapter 6. Output Format

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.

### 6.1. 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 14 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*.

This applet supports the following output formats:

- **BGR8**: 24 bit RGB color format with 8 bit/component.
- **BGR10p**: 30 bit 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**: 36 bit 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.

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



## BGR Memory Alignment

Note that the color components are written to the PC buffer in the common blue, green, red (BGR) order. This means, that the blue component is at the lower memory address, while red is at the highest memory address of the components triple.

- **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.

- **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.

Table 6.1. Parameter properties of Format

Property	Value	
Name	<b>Format</b>	
Display Name	<b>Output Format</b>	
Interface	<b>IEnumeration</b>	
Access policy	<b>Read/Write</b>	
Visibility	<b>Beginner</b>	
Allowed values	<b>Mono8</b> Gray 8bit <b>Mono10p</b> Gray 10bit <b>Mono12p</b> Gray 12bit <b>Mono16</b> Gray 16bit <b>BGR8</b> Color 24bit <b>BGR10p</b> Color 30bit <b>BGR12p</b> Color 36bit <b>BGR16</b> Color 48bit <b>BayerGR8</b> BayerGR8 <b>BayerGR10p</b> BayerGR10 <b>BayerGR12p</b> BayerGR12 <b>BayerGR16</b> BayerGR16 <b>BayerRG8</b> BayerRG8 <b>BayerRG10p</b> BayerRG10 <b>BayerRG12p</b> BayerRG12 <b>BayerRG16</b> BayerRG16 <b>BayerGB8</b> BayerGB8 <b>BayerGB10p</b> BayerGB10 <b>BayerGB12p</b> BayerGB12 <b>BayerGB16</b> BayerGB16 <b>BayerBG8</b> BayerBG8 <b>BayerBG10p</b> BayerBG10 <b>BayerBG12p</b> BayerBG12 <b>BayerBG16</b> BayerBG16 <b>YCbCr422_8</b> YUV422 8bit	
Default value	<b>Mono8</b>	

Example 6.1. Usage of Format

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

## 6.2. BitAlignment

The bit alignment is used to map the pixel bits of the internal processing with a depth of 14 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 `FG_CUSTOM_SHIFT_MODE` 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 6.2. Parameter properties of BitAlignment

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

Example 6.2. Usage of BitAlignment

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

## 6.3. 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 6.3. Parameter properties of PixelDepth

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

Example 6.3. Usage of PixelDepth

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

## 6.4. 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 14 bits.

Table 6.4. Parameter properties of CustomBitShiftRight

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

Example 6.4. Usage of CustomBitShiftRight

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

---

# Chapter 7. Camera Simulator

The camera simulator is a convenient way to simulate cameras for first time applet tests. If the simulator is enabled it generates pattern frames of specified size and speed. The image data is replaced at the position of the camera i.e. all applet processing functionalities are applied to the generated images. Note that camera specific settings of the applet will not have any functionality.



## Limited Usage with GenTL

In GenTL an interface cannot be used without a device. So it is not possible to use the camera simulator without a camera. However, you can replace the camera data with the simulator data when the simulator is enabled.

The generated images are horizontal, diagonal or vertical grayscale patterns, such as the one shown in the following figure.

Figure 7.1. Generator Pattern



## No Sub-Sensor sorting in Generated Images

The camera simulator will generate a simple grayscale pattern. If the camera or this applet uses sub sensor pixel sorting (sensor correction), the simulator will not generate images which represent the camera sensor.

### 7.1. CamerasimulatorEnable

The camera simulator is enabled with this parameter. When you switch between camera mode and simulator, the applet will finalize the current frame before switching to the other input.



## Only 8bit support

The camera simulator will produce valid 8bit values only for 8bit pixel format. All other pixel formats will consist of packed 8bit data inside the packed format.

This will cause strange images in the simulation for higher bit depth than 8bit. Since this function is not related to productive usage this should be acceptable.

Table 7.1. Parameter properties of CamerasimulatorEnable

Property	Value
Name	<b>CamerasimulatorEnable</b>
Display Name	<b>Image Source</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>SourceCamera</b> Camera <b>SourceSimulator</b> Simulator
Default value	<b>SourceCamera</b>

Example 7.1. Usage of CamerasimulatorEnable

```
/* Set */ CamerasimulatorEnable = SourceCamera;
/* Get */ value_ = CamerasimulatorEnable;
```

## 7.2. CamerasimulatorWidth

The width of the generated frame is set with this parameter. You can enter any value. The applet will automatically round up to the next valid value limited due to internal processing granularity.

The range of the width depends on other parameters and is automatically determined from the applet. Decrease the speed for extending the range of the width value.

Table 7.2. Parameter properties of CamerasimulatorWidth

Property	Value
Name	<b>CamerasimulatorWidth</b>
Display Name	<b>Width</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 1 <b>Maximum</b> 258048 <b>Stepsize</b> 1
Default value	<b>1024</b>
Unit of measure	<b>pixel</b>

Example 7.2. Usage of CamerasimulatorWidth

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

## 7.3. CamerasimulatorLineGap

The simulator will generate a gap between the lines. The length of the gap is defined by this parameter. So the time of the gap depends on the pixel clock and the value.

You can enter any value. The applet will automatically round up to the next valid value.

The range of the line gap depends on other parameters and is automatically determined from the applet. Decrease the speed for extending the range of the line gap value.

The parameter can only be changed if *CamerasimulatorSelectMode* is set to **PixelFrequency**.

Table 7.3. Parameter properties of *CamerasimulatorLineGap*

Property	Value
Name	<b>CamerasimulatorLineGap</b>
Display Name	<b>Line Gap</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 258048</b> <b>Stepsize 1</b>
Default value	<b>0</b>
Unit of measure	<b>pixel</b>

Example 7.3. Usage of *CamerasimulatorLineGap*

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

## 7.4. CamerasimulatorHeight

The height of the generated frame is set with this parameter.

The range of the height depends on other parameters and is automatically determined from the applet. Decrease the speed for extending the range of the height value.

Table 7.4. Parameter properties of *CamerasimulatorHeight*

Property	Value
Name	<b>CamerasimulatorHeight</b>
Display Name	<b>Height</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum 1</b> <b>Maximum 65536</b> <b>Stepsize 1</b>
Default value	<b>1024</b>
Unit of measure	<b>pixel</b>

Example 7.4. Usage of *CamerasimulatorHeight*

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

## 7.5. CamerasimulatorFrameGap

The simulator will generate a gap between the frames. The length of the gap is defined by this parameter. So the time of the gap depends on the line rate and the value.

The range of the frame gap depends on other parameters and is automatically determined from the applet. Decrease the speed for extending the range of the frame gap value.

The parameter can not be changed if parameter *CamerasimulatorSelectMode* is set to **FrameRate**.

Table 7.5. Parameter properties of *CamerasimulatorFrameGap*

Property	Value
Name	<b>CamerasimulatorFrameGap</b>
Display Name	<b>Frame Gap</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 65536 <b>Stepsize</b> 1
Default value	<b>0</b>
Unit of measure	<b>pixel</b>

Example 7.5. Usage of *CamerasimulatorFrameGap*

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

## 7.6. CamerasimulatorPattern

The simulator will generate pixel value ramps from 0 to 255. As this applet is capable of using Gray, bayer or RGB inputs.

The following three types of patterns can be generated and selected by this parameter.

- **PatternHorizontal**

A horizontal pattern. Values are increased by 1 in x-direction.

- **PatternVertical**

A vertical pattern. Values are increased by 1 in y-direction.

- **PatternDiagonal**

A diagonal pattern. Values are increased by 1 in x and y-direction.

Table 7.6. Parameter properties of *CamerasimulatorPattern*

Property	Value
Name	<b>CamerasimulatorPattern</b>
Display Name	<b>Pattern</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>PatternHorizontal</b> Horizontal <b>PatternVertical</b> Vertical <b>PatternDiagonal</b> Diagonal
Default value	<b>PatternDiagonal</b>

**Example 7.6. Usage of CamerasimulatorPattern**

```
/* Set */ CamerasimulatorPattern = PatternDiagonal;
/* Get */ value_ = CamerasimulatorPattern;
```

## 7.7. CamerasimulatorPatternOffset

Using this parameter, an offset value can be added to the generated patterns. After acquisition start, the offset is added. For example, the very first pixel of an image will start with the offset value instead of 0.

**Table 7.7. Parameter properties of CamerasimulatorPatternOffset**

Property	Value
Name	<b>CamerasimulatorPatternOffset</b>
Display Name	<b>Pattern Offset</b>
Interface	<b>IInteger</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 255 <b>Stepsize</b> 1
Default value	<b>0</b>
Unit of measure	<b>pixel value</b>

**Example 7.7. Usage of CamerasimulatorPatternOffset**

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

## 7.8. CamerasimulatorRoll

The generated pattern can be 'rolled'. With every new frame, all pattern pixels are increased by value one. At the wrap-around value 256, the pixel will get value 0. The generated images look like a moving (rolling) image.

**Table 7.8. Parameter properties of CamerasimulatorRoll**

Property	Value
Name	<b>CamerasimulatorRoll</b>
Display Name	<b>Roll</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>On</b> On <b>Off</b> Off
Default value	<b>On</b>

**Example 7.8. Usage of CamerasimulatorRoll**

```
/* Set */ CamerasimulatorRoll = On;
/* Get */ value_ = CamerasimulatorRoll;
```

## 7.9. CamerasimulatorSelectMode

The simulator will generate the images with a certain speed. Users are allowed to select whether they want to set the pixel frequency, line rate or frame rate to control the speed. This parameter selects the mode.

Table 7.9. Parameter properties of *CamerasimulatorSelectMode*

Property	Value
Name	<b>CamerasimulatorSelectMode</b>
Display Name	<b>Speed Mode</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>PixelFrequency</b> Pixel Frequency <b>LineRate</b> Line Rate <b>FrameRate</b> Frame Rate
Default value	<b>FrameRate</b>

Example 7.9. Usage of *CamerasimulatorSelectMode*

```
/* Set */ CamerasimulatorSelectMode = FrameRate;
/* Get */ value_ = CamerasimulatorSelectMode;
```

## 7.10. CamerasimulatorPixelFrequency

This parameter sets the pixel frequency. Note that the generator only simulates cameras. It is made for a first time use of the applet and user SDK verification. The camera simulator cannot reflect the exact timings and frequencies of cameras.

To set the pixel frequency, you will need to set parameter *CamerasimulatorSelectMode* to **PixelFrequency**.

Any floating point value can be inserted to the parameter. However, the applet will round the value to the next valid value. Read the parameter value to find out the new rounded value.

Table 7.10. Parameter properties of *CamerasimulatorPixelFrequency*

Property	Value
Name	<b>CamerasimulatorPixelFrequency</b>
Display Name	<b>Pixel Frequency</b>
Interface	<b>IFloat</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0.0 <b>Maximum</b> 4000.0 <b>Stepsize</b> 2.220446049250313E-16
Default value	<b>40.0</b>
Unit of measure	<b>MHz</b>

Example 7.10. Usage of *CamerasimulatorPixelFrequency*

```
/* Set */ CamerasimulatorPixelFrequency = 40.0;
/* Get */ value_ = CamerasimulatorPixelFrequency;
```

## 7.11. CamerasimulatorLinerate

This parameter sets the line rate of the generated images.



To set the line rate, you will need to set parameter *CamerasimulatorSelectMode* to **LineRate**.

In line rate mode, the pixel frequency is set to the maximum.

Any floating point value can be inserted to the parameter. However, the applet will round the value to the next valid value. Read the parameter value to find out the new rounded value.

Table 7.11. Parameter properties of *CamerasimulatorLinerate*

Property	Value
Name	<b>CamerasimulatorLinerate</b>
Display Name	<b>Line Rate</b>
Interface	<b>IFloat</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0.1 <b>Maximum</b> 1.0E7 <b>Stepsize</b> 2.220446049250313E-16
Default value	<b>10240.0</b>
Unit of measure	<b>Hz</b>

Example 7.11. Usage of *CamerasimulatorLinerate*

```
/* Set */ CamerasimulatorLinerate = 10240.0;
/* Get */ value_ = CamerasimulatorLinerate;
```

## 7.12. CamerasimulatorFramerate

This parameter sets the frame rate of the generated images. For parameter *TriggerFramesPerSecond* only frame values up to the upper value limit of *CamerasimulatorFramerate* are valid.

To set the frame rate, you will need to set parameter *CamerasimulatorSelectMode* to **FrameRate**.

In frame rate mode, the pixel frequency is set to the maximum and the line gap is set to zero.

Any floating point value can be inserted to the parameter. However, the applet will round the value to the next valid value. Read the parameter value to find out the new rounded value.

Table 7.12. Parameter properties of *CamerasimulatorFramerate*

Property	Value
Name	<b>CamerasimulatorFramerate</b>
Display Name	<b>Framerate</b>
Interface	<b>IFloat</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0.1 <b>Maximum</b> 1.0E7 <b>Stepsize</b> 2.220446049250313E-16
Default value	<b>10.0</b>
Unit of measure	<b>Hz</b>

Example 7.12. Usage of *CamerasimulatorFramerate*

```
/* Set */ CamerasimulatorFramerate = 10.0;
```

```
/* Get */ value_ = CamerasimulatorFramerate;
```

## 7.13. CamerasimulatorTriggerMode

You can either use the camera simulator in free run mode or the simulator can be triggered by the output of the trigger module of this applet. As this applet uses a CoaxPress camera interface, the CXP trigger output of the respective camera port is used as camera simulator trigger input. The rising edge of the trigger will be used.

You can choose between line trigger and frame trigger mode. In line trigger mode, a rising edge at the input will output a line from the camera simulator. For frame trigger mode, the input will trigger the output of a frame.



### Trigger frequency must not exceed the speed of the camera simulator

Same as for real cameras, it is very important that the frequency of the trigger pulses do not exceed the maximum speed of the camera simulator. Set the camera simulator to a sufficiently large speed to avoid line or frame lost.

Table 7.13. Parameter properties of CamerasimulatorTriggerMode

Property	Value
Name	<b>CamerasimulatorTriggerMode</b>
Display Name	<b>Trigger Mode</b>
Interface	<b>IEnumeration</b>
Access policy	<b>Read/Write/Change</b>
Visibility	<b>Invisible</b>
Allowed values	<b>FreeRunMode</b> Free Run <b>LineTrgMode</b> Rising Edge Triggers Line <b>FrameTrgMode</b> Rising Edge Triggers Frame
Default value	<b>FreeRunMode</b>

Example 7.13. Usage of CamerasimulatorTriggerMode

```
/* Set */ CamerasimulatorTriggerMode = FreeRunMode;  
/* Get */ value_ = CamerasimulatorTriggerMode;
```

## 7.14. CamerasimulatorActive

Table 7.14. Parameter properties of CamerasimulatorActive

Property	Value
Name	<b>CamerasimulatorActive</b>
Display Name	<b>Active Parts</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 1 <b>Maximum</b> 2000 <b>Stepsize</b> 1
Unit of measure	<b>pixel</b>

Example 7.14. Usage of CamerasimulatorActive

```
/* Get */ value_ = CamerasimulatorActive;
```

## 7.15. CamerasimulatorPassive

Table 7.15. Parameter properties of CamerasimulatorPassive

Property	Value
Name	<b>CamerasimulatorPassive</b>
Display Name	<b>Passive Parts</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum 1</b> <b>Maximum 2000</b> <b>Stepsize 1</b>
Unit of measure	<b>pixel</b>

Example 7.15. Usage of CamerasimulatorPassive

```
/* Get */ value_ = CamerasimulatorPassive;
```

---

# Chapter 8. Miscellaneous

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

## 8.1. Timeout

This parameter is used to set a timeout for DMA transfers. After a timeout the acquisition is stopped. But it is only a internal value that should not be used directly. Please use the timeout value described in the runtime SDK or microDisplay for acquisition in order to handle the functionality correctly.

Table 8.1. Parameter properties of Timeout

Property	Value
Name	Timeout
Display Name	Timeout
Interface	IInteger
Access policy	Read/Write/Change
Visibility	Invisible
Allowed values	Minimum 2 Maximum 2147483646 Stepsize 1
Default value	1000000
Unit of measure	seconds

Example 8.1. Usage of Timeout

```
/* Set */ Timeout = 1000000;  
/* Get */ value_ = Timeout;
```

## 8.2. AppletVersion

This parameter represents the version number of the applet. Please report this value for any support of the applet.

Table 8.2. Parameter properties of AppletVersion

Property	Value
Name	AppletVersion
Display Name	Applet version
Interface	IInteger
Access policy	Read-Only
Visibility	Invisible

Example 8.2. Usage of AppletVersion

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

## 8.3. AppletRevision

This parameter represents the revision number of the applet. Please report this value for any support case with the applet.

Table 8.3. Parameter properties of AppletRevision

Property	Value
Name	<b>AppletRevision</b>
Display Name	<b>Applet revision</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>

Example 8.3. Usage of AppletRevision

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

## 8.4. AppletId

This parameter returns the unique applet id of the applet as a string parameter.

Table 8.4. Parameter properties of AppletId

Property	Value
Name	<b>AppletId</b>
Display Name	<b>Applet Id</b>
Interface	<b>IString</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>

Example 8.4. Usage of AppletId

```
/* Get */ value_ = AppletId;
```

## 8.5. AppletBuildTime

This string parameter returns the hardware applet (HAP) build timestamp. To obtain the build time of the applet, check the DLL / SO file details. Mainly, this parameter is required for internal usage only.

Table 8.5. Parameter properties of AppletBuildTime

Property	Value
Name	<b>AppletBuildTime</b>
Display Name	<b>Build Time</b>
Interface	<b>IString</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>

Example 8.5. Usage of AppletBuildTime

```
/* Get */ value_ = AppletBuildTime;
```

## 8.6. HapFile

The name of the Hardware-Applet (HAP) file on which this applet is based. Please report this read-only string parameter for any support case of the applet.

Table 8.6. Parameter properties of HapFile

Property	Value
Name	<b>HapFile</b>
Display Name	<b>HAP file</b>
Interface	<b>IString</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>

Example 8.6. Usage of HapFile

## 8.7. DMAStatus

Using this parameter the status of a DMA channel can be obtained. Value "1" represents a started DMA i.e. a started acquisition. Value "0" represents a stopped acquisition.

Table 8.7. Parameter properties of DMAStatus

Property	Value
Name	<b>DMAStatus</b>
Display Name	<b>DMA Status</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>

Example 8.7. Usage of DMAStatus

```
/* Get */ value_ = DMAStatus;
```

## 8.8. SystemmonitorFpgaDnaLow

The parameter *SystemmonitorFpgaDnaLow* provides the lower 57 bit unique FPGA DNA.

Table 8.8. Parameter properties of SystemmonitorFpgaDnaLow

Property	Value
Name	<b>SystemmonitorFpgaDnaLow</b>
Display Name	<b>FPGA DNA Low</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum</b> 0 <b>Maximum</b> 144115188075855872 <b>Stepsize</b> 0

Example 8.8. Usage of SystemmonitorFpgaDnaLow

```
/* Get */ value_ = SystemmonitorFpgaDnaLow;
```

## 8.9. SystemmonitorFpgaDnaHigh

The parameter *SystemmonitorFpgaDnaHigh* provides the upper 32s bit unique FPGA DNA.

Table 8.9. Parameter properties of SystemmonitorFpgaDnaHigh

Property	Value
Name	<b>SystemmonitorFpgaDnaHigh</b>
Display Name	<b>FPGA DNA High</b>
Interface	<b>IInteger</b>
Access policy	<b>Read-Only</b>
Visibility	<b>Invisible</b>
Allowed values	<b>Minimum 0</b> <b>Maximum 4294967295</b> <b>Stepsize 0</b>

Example 8.9. Usage of SystemmonitorFpgaDnaHigh

```
/* Get */ value_ = SystemmonitorFpgaDnaHigh;
```

---

## Chapter 9. Revision History

Document Number	Date	Changes
AW00158201000	09 January 2020	Initial version of this document
AW00158202000	16 April 2020	Revision: Corrected figures in Chapter 4; corrected section 4.3.3 for two flash lights



---

# 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	<p>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.</p> <p>See also Camera Port.</p>
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)	<p>A DMA transfer allows hardware subsystems within the computer to access the system memory independently of the central processing unit (CPU).</p> <p>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.</p>
DMA Channel	See DMA Index.
DMA Index	<p>The index of a DMA transfer channel.</p> <p>See also Direct Memory Access.</p>
Event	<p>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.</p> <p>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.</p> <p>For example these events can be used to check the reliability between a frame trigger input and the resulting and expected camera frame.</p> <p>Our Runtime/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 our Runtime/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.</p>

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 Silicon Software 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 increases 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	<p>Simple programming of FPGA-based image processing devices.</p> <p>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.</p>

---

# Index

## A

AppletBuildTime, 72  
AppletId, 72  
AppletRevision, 71  
AppletVersion, 71  
Area of Interest, 10  
AreaTriggerMode, 31

## B

Bandwidth, 2  
BitAlignment, 59

## C

Camera  
    Format, 5  
    Interface, 4  
Camera Simulator, 62, 62  
CamerasimulatorActive, 69  
CamerasimulatorEnable, 62  
CamerasimulatorFrameGap, 64  
CamerasimulatorFramerate, 68  
CamerasimulatorHeight, 64  
CamerasimulatorLineGap, 63  
CamerasimulatorLinerate, 67  
CamerasimulatorPassive, 70  
CamerasimulatorPattern, 65  
CamerasimulatorPatternOffset, 66  
CamerasimulatorPixelFrequency, 67  
CamerasimulatorRoll, 66  
CamerasimulatorSelectMode, 66  
CamerasimulatorTriggerMode, 69  
CamerasimulatorWidth, 63  
CoaXPress, 5  
CustomBitShiftRight, 60  
CxpStatus, 7  
CxpTriggerPacketMode, 6

## D

DMAStatus, 73

## E

Events  
    Digital Inputs, 39  
    Trigger Lost Detection, 51  
    Trigger Queue, 43

## F

Features, 1  
FillLevel, 55  
Format, 57, 57  
FrontGPI, 35

## G

Generator, 62

GPI, 35

## H

HapFile, 72

Height, 12

## I

Image Transfer, 4

## M

Miscellaneous, 71

MissingCameraFrameResponse, 53

MissingCameraFrameResponseClear, 54

## O

OffsetX, 12

OffsetY, 13

Output Format, 57

Overflow, 55, 55, 55

## P

PC Interface, 4

Pixel Format, 5

PixelDepth, 60

PixelFormat, 5

## R

Region of Interest, 10

ROI, 10

## S

SendSoftwareTrigger, 38

SoftwareTriggerIsBusy, 38

SoftwareTriggerQueueFillLevel, 39

Specifications, 1

SystemmonitorFpgaDnaHigh, 74

SystemmonitorFpgaDnaLow, 73

## T

Timeout, 71

Trigger, 14, 14

    Activate, 32

    Busy, 38

    Bypass, 30

    Camera Signal Mapping, 48

    Debounce, 34

    Debugging, 30

    Digital Input, 17, 35

    Digital Input Output Mapping, 17

    Digital Output, 17, 19, 49

    Downscale Input, 36

    Encoder, 19

    Error Detection, 30

    Exceeded Period Limits, 51

    Exsync, 19

    External, 19, 31, 34

    Flash, 19, 22

    Frame Rate, 18

- Framerate, 33
- Generator, 18, 31
- GPI, 35
- Grabber Controlled, 18
- Image Trigger, 19
- Input, 34, 35
- Input Statistics, 39
- IO Triggered, 19
- Length, 19
- Lost Trigger, 30, 51
- Missing Frame Response, 53
- Mode, 31
- Multi Camera, 30
- Multiply Pulses, 42
- Output, 49
- Output Statistics, 50
- Period, 33
- Pin Allocation, 17
- Polarity Input, 36, 40
- Pulse Form Generator, 44
- Pulse Multiplication, 22
- Queue, 27, 29, 43
- Sequencer, 22, 42
- Signal Length, 19
- Signal Width, 19
- Software Controlled, 38
- Software Trigger, 25, 31, 38
- Start, 32
- Stop, 32
- Synchronized, 30
- Synchronized Cameras, 30
- System Analysis, 30
- Trigger IO, 17
- Width, 19
- Trigger::Camera Out Signal Mapping, 48
- Trigger::Digital Output, 49
- Trigger::Digital Output::Statistics, 50
- Trigger::Pulse Form Generator 0, 44
- Trigger::Pulse Form Generator 1, 48
- Trigger::Pulse Form Generator 2, 48
- Trigger::Pulse Form Generator 3, 48
- Trigger::Queue, 43
- Trigger::Sequencer, 42
- Trigger::Trigger Input, 34
- Trigger::Trigger Input::External, 34
- Trigger::Trigger Input::Software Trigger, 38
- Trigger::Trigger Input::Statistics, 39
- TriggerAcknowledgementCount, 8
- TriggerCameraOutSelect, 48
- TriggerEventCount, 7
- TriggerExceededPeriodLimits, 51
- TriggerExceededPeriodLimitsClear, 51
- TriggerFramesPerSecond, 21, 33
- TriggerInDebounce, 34
- TriggerInDownscale, 36
- TriggerInDownscalePhase, 37
- TriggerInPolarity, 36
- TriggerInSource, 35

TriggerInStatisticsFrequency, 41  
TriggerInStatisticsMaximumFrequency, 42  
TriggerInStatisticsMinimumFrequency, 41  
TriggerInStatisticsMinMaxFrequencyClear, 42  
TriggerInStatisticsPolarity, 40  
TriggerInStatisticsPulseCount, 40  
TriggerInStatisticsPulseCountClear, 40  
TriggerInStatisticsSource, 39  
TriggerMultiplyPulses, 24, 43  
TriggerOutSelectFrontGPO0, 50  
TriggerOutSelectFrontGPO1, 50  
TriggerOutStatisticsPulseCount, 52  
TriggerOutStatisticsPulseCountClear, 52  
TriggerOutStatisticsSource, 51  
TriggerPulseFormGenerator0Delay, 47  
TriggerPulseFormGenerator0Downscale, 45  
TriggerPulseFormGenerator0DownscalePhase, 46  
TriggerPulseFormGenerator0Width, 47  
TriggerPulseFormGenerator1Delay, 47  
TriggerPulseFormGenerator1Downscale, 45  
TriggerPulseFormGenerator1DownscalePhase, 46  
TriggerPulseFormGenerator1Width, 47  
TriggerPulseFormGenerator2Delay, 47  
TriggerPulseFormGenerator2Downscale, 45  
TriggerPulseFormGenerator2DownscalePhase, 46  
TriggerPulseFormGenerator2Width, 47  
TriggerPulseFormGenerator3Delay, 47  
TriggerPulseFormGenerator3Downscale, 45  
TriggerPulseFormGenerator3DownscalePhase, 46  
TriggerPulseFormGenerator3Width, 47  
TriggerQueueFillLevel, 44  
TriggerQueueMode, 43  
TriggerState, 32  
TriggerWaveViolation, 8

## **W**

Width, 11