microEnable 5 marathon/LightBridge VCL

AcquisitionApplets User Documentation for

Acq_SingleMediumLineGray

Functional Description
For Framegrabber SDK Usage

Document Number: AW001770
Part Number: 000 (English)
Document Version: 01
Release Date: 01 December 2022
Applet Version 2.2.4.0

Contacting Basler Support Worldwide

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

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

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

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

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

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

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

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

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

Supplemental Information

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

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

Table of Contents

1. Introduction 1
    1.1. Features of Applet Acq_SingleMediumLineGray 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 4
        1.3.3. License 4
    1.4. Camera Interface 4
    1.5. Image Transfer to PC Memory 4
2. Software Interface 5
3. Camera Link 6
    3.1. FG_CAMERA_LINK_CAMTYPE 6
    3.2. FG_USEDVAL 6
    3.3. FG_CAMERA_LINK_CORE_RESET 7
    3.4. FG_CAMERA_LINK_PIXEL_CLOCK_X 8
    3.5. FG_CAMERA_LINK_PIXEL_CLOCK_Y 8
4. Camera 10
    4.1. Events 10
        4.1.1. FG_START_OF_FRAME_CAM_PORT_0 10
        4.1.2. FG_END_OF_FRAME_CAM_PORT_0 10
        4.1.3. FG_START_OF_LINE_CAM_PORT_0 10
        4.1.4. FG_END_OF_LINE_CAM_PORT_0 10
5. Tap Geometry 11
    5.1. FG_TAPGEOMETRY 11
    5.2. FG_SENSORWIDTH 11
    5.3. FG_VANTAGEPOINT 12
6. ROI 14
    6.1. FG_WIDTH 15
    6.2. FG_HEIGHT 15
    6.3. FG_XOFFSET 16
    6.4. FG_YOFFSET 17
7. Digital I/O 18
    7.1. GPO 18
        7.1.1. FG_TRIGGEROUT_GPO_0_SOURCE et al. 18
        7.1.2. FG_TRIGGEROUT_GPO_0_POLARITY et al. 19
        7.1.3. FG_DIGIO_OUTPUT 20
    7.2. GPI 21
        7.2.1. FG_DIGIO_INPUT 21
    7.3. Event Source 22
        7.3.1. FG_CUSTOM_SIGNAL_EVENT_0_SOURCE 22
        7.3.2. FG_CUSTOM_SIGNAL_EVENT_0_POLARITY 23
        7.3.3. FG_CUSTOM_SIGNAL_EVENT_1_SOURCE 23
        7.3.4. FG_CUSTOM_SIGNAL_EVENT_1_POLARITY 24
    7.4. Events 25
        7.4.1. FG_TRIGGER_INPUT0_RISING 25
        7.4.2. FG_TRIGGER_INPUT0_FALLING 25
        7.4.3. FG_CUSTOM_SIGNAL_EVENT_0 25
        7.4.4. FG_CUSTOM_SIGNAL_EVENT_1 25
8. CC Signal Mapping 27
    8.1. FG_CC1_SOURCE et al. 28
    8.2. FG_CC1_POLARITY et al. 28
9. Line Trigger / ExSync 30
    9.1. FG_LINETRIGGERMODE 30
    9.2. FG_EXSYNCON 31
    9.3. Line Trigger Input 32

9.3.1. FG_LINETRIGGERINSRC 33
9.3.2. FG_LINETRIGGERINPOLARITY 34
9.3.3. FG_LINETRIGGERDEBOUNCING 35
9.3.4. Downscale 35
9.3.4.1. FG_LINE_DOWNSCALE 35
9.3.4.2. FG_LINE_DOWNSCALEINIT 36
9.4. Shaft Encoder A/B Filter 37
9.4.1. FG_SHAFTENCODERON 37
9.4.2. FG_SHAFTENCODERMODE 38
9.4.3. FG_SHAFTENCODERINSRC 39
9.4.4. FG_SHAFTENCODERLEADING 40
9.4.5. FG_SHAFTENCODER_COMPENSATION_ENABLE 41
9.4.6. FG_SHAFTENCODER_COMPENSATION_COUNT 42
9.5. ExSync Output 48
9.5.1. FG_LINEPERIODE 48
9.5.2. FG_LINEEXPOSURE 49
9.5.3. FG_EXSYNCPOLARITY 50
9.5.4. FG_LINETRIGGERDELAY 51
10. Image Trigger / Flash 52
10.1. FG_IMGTRIGGERMODE 53
10.2. FG_IMGTRIGGERON 53
10.3. FG_FLASHON 54
10.4. FG_IMGTRIGGER_ASYNC_HEIGHT 54
10.5. FG_IMGTRIGGER_IS_BUSY 55
10.6. Image Trigger Input 56
10.6.1. FG_IMGTRIGGERINSRC 56
10.6.2. FG_IMGTRIGGERINPOLARITY 56
10.6.3. FG_IMGTRIGGERGATEDELAY 57
10.6.4. FG_IMGTRIGGERDEBOUNCING 58
10.6.5. FG_STROBEPULSEDELAY 58
10.6.6. Flash 59
10.6.6.1. FG_FLASH_POLARITY 59
10.6.7. Software Trigger 59
10.6.7.1. FG_SENDSOFTWARETRIGGER 59
10.6.7.2. FG_SETSOFTWARETRIGGER 60
11. Signal Analyzer 61
11.1. FG_SIGNAL_ANALYZER_0_SOURCE et al. 61
11.2. FG_SIGNAL_ANALYZER_0_POLARITY et al. 62
11.3. FG_SIGNAL_ANALYZER_0_PERIOD_CURRENT et al. 63
11.4. FG_SIGNAL_ANALYZER_0_PERIOD_MAX et al. 64
11.5. FG_SIGNAL_ANALYZER_0_PERIOD_MIN et al. 64
11.6. FG_SIGNAL_ANALYZER_0_PULSE_COUNT et al. 65
11.7. FG_SIGNAL_ANALYZER_PULSE_COUNT_DIFFERENCE 65
11.8. FG_SIGNAL_ANALYZER_CLEAR 66
12. Overflow 67
12.1. FG_FILLLEVEL 67
12.2. FG_OVERFLOW 68
12.3. Events 68
12.3.1. FG_OVERFLOW_CAM0 68
13. Image Selector 71
13.1. FG_IMG_SELECT_PERIOD 71
13.2. FG_IMG_SELECT 72
14. Lookup Table 73
14.1. FG_LUT_TYPE 73
14.2. FG_LUT_VALUE 74
14.3. FG_LUT_CUSTOM_FILE 74
14.4. FG_LUT_SAVE_FILE 76
14.5. Applet Properties 76

14.5.1. FG_LUT_IMPLEMENTATION_TYPE 77
14.5.2. FG_LUT_IN_BITS 77
14.5.3. FG_LUT_OUT_BITS 77
15. Processing 79
15.1. FG_PROCESSING_OFFSET 80
15.2. FG_PROCESSING_GAIN 80
15.3. FG_PROCESSING_GAMMA 81
15.4. FG_PROCESSING_INVERT 82
16. Output Format 84
16.1. FG_FORMAT 84
16.2. FG_BITALIGNMENT 85
16.3. FG_PIXELDEPTH 86
16.4. FG_CUSTOM_BIT_SHIFT_RIGHT 86
17. Camera Simulator 88
17.1. FG_CAMERASIMULATOR_ENABLE 88
17.2. FG_CAMERASIMULATOR_WIDTH 89
17.3. FG_CAMERASIMULATOR_LINE_GAP 89
17.4. FG_CAMERASIMULATOR_HEIGHT 90
17.5. FG_CAMERASIMULATOR_FRAME_GAP 91
17.6. FG_CAMERASIMULATOR_PATTERN 91
17.7. FG_CAMERASIMULATOR_PATTERN_OFFSET 92
17.8. FG_CAMERASIMULATOR_ROLL 93
17.9. FG_CAMERASIMULATOR_SELECT_MODE 93
17.10. FG_CAMERASIMULATOR_PIXEL_FREQUENCY 94
17.11. FG_CAMERASIMULATOR_LINERATE 94
17.12. FG_CAMERASIMULATOR_FRAMERATE 95
17.13. FG_CAMERASIMULATOR_TRIGGER_MODE 96
17.14. FG_CAMERASIMULATOR_ACTIVE 96
17.15. FG_CAMERASIMULATOR_PASSIVE 97
18. Miscellaneous 98
18.1. FG_TIMEOUT 98
18.2. FG_APPLET_VERSION 98
18.3. FG_APPLET_REVISION 99
18.4. FG_APPLET_ID 99
18.5. FG_APPLET_BUILD_TIME 99
18.6. FG_HAP_FILE 100
18.7. FG_DMASTATUS 100
18.8. FG_CAMSTATUS 101
18.9. FG_CAMSTATUS_EXTENDED 101
18.10. FG_SYSTEMMONITOR_FPGA_TEMPERATURE 102
18.11. FG_SYSTEMMONITOR_FPGA_VCC_INT 102
18.12. FG_SYSTEMMONITOR_FPGA_VCC_AUX 103
18.13. FG_SYSTEMMONITOR_FPGA_VCC_BRAM 103
18.14. FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH 104
18.15. FG_SYSTEMMONITOR_CURRENT_LINK_SPEED 104
18.16. FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE 105
18.17. FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE 105
18.18. FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE 106
18.19. FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT 106
18.20. FG_ALTERNATIVE_BOARD_DETECTION 107
18.21. FG_SYSTEMMONITOR_POCL_STATE_PORT_A 107
18.22. FG_SYSTEMMONITOR_POCL_STATE_PORT_B 110
18.23. FG_SYSTEMMONITOR_FPGA_DNA 111
18.24. Debug 111
18.24.1. FG_DEBUGSOURCE 112
18.24.2. FG_DEBUGSOURCENAME 112
18.24.3. FG_DEBUGSAVECONFIG 112
18.24.4. FG_DEBUG_VERSION 113

18.24.5. Input 113
18.24.5.1. FG_DEBUGINENABLE 113
18.24.5.2. FG_DEBUGFILE 114
18.24.5.3. FG_DEBUGINSERT 114
18.24.5.4. FG_DEBUGWRITEPIXEL 115
18.24.5.5. FG_DEBUGWRITEFLAG 115
18.24.5.6. FG_DEBUGREADY 116
18.24.6. Output 116
18.24.6.1. FG_DEBUGOUTENABLE 116
18.24.6.2. FG_DEBUGOUTXPOS 117
18.24.6.3. FG_DEBUGOUTYPOS 117
18.24.6.4. FG_DEBUGOUTPIXEL 118
19. Legacy Parameter 119
19.1. FG_CCSEL0 119
19.2. FG_CCSEL1 120
19.3. FG_CCSEL2 122
19.4. FG_CCSEL3 123
Glossary 125
Index 128

Chapter 1. Introduction

This document provides you with detailed information on applet "Acq_SingleMediumLineGray" for microEnable 5 marathon/LightBridge VCL frame grabber.

This document will outline the features and benefits of this applet. Furthermore, the output formats and the software interface is explained. The main part of this document includes the detailed description of all applet modules and their parameters which make the applet adaptable for numerous applications.

1.1. Features of Applet Acq_SingleMediumLineGray

"Acq_SingleMediumLineGray" is an applet for one camera (single-camera applet). You can configure the Camera Link camera interface for Camera Link cameras in Medium-Configuration mode transferring grayscale pixels at bit depths between 8 and 12 bits. A multi-functional line trigger is included in the applet. This allows you to control the camera or external devices using frame grabber generated, external or software generated trigger pulses. Line scan cameras up to a width of 49136 pixels can be processed. The trigger system will generate images of a maximum height of 16777215 pixels. The applet is processing data at a bit depth of 12 bits. An image selector at the camera port facilitates the selection of one image out of a parameterizable sequence of images. This enables the distribution of the images to multiple frame grabber and PCs. The applet supports horizontal tap geometry sorting for 1, 2, 3 and 4 Taps. You can mirror the acquired sensor images in x-direction (horizontal mirror). To enable mirroring, set the vantage point. For reverse operation, you can mirror the image in x-direction before cutting the ROI. Acquired images are buffered in frame grabber 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. You can configure the 12 bit full resolution lookup table either by using a user defined table, or by using a processor. The processor gives you the opportunity to use pre-defined functions such as offset, gain, invert to enhance the image quality.

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.

You can easily include the applet into your own applications using the Silicon Software Framegrabber SDK.

Table 1.1. Feature Summary of Acq_SingleMediumLineGray

Feature	Applet Property
Applet Name	 Acq_SingleMediumLineGray
Type of Applet	AcquisitionApplets
Board	microEnable 5 marathon/LightBridge VCL
Minimum Framegrabber SDK Version	5.5.1
No. of Cameras	1
Camera Type	Camera Link Medium Configuration (3 Tap 8 bit, 3 Tap 10 Bit, 4 Tap 8 Bit, 4 Tap 10 Bit, 4 Tap 12 Bit)
Sensor Type	Line Scan
Camera Format	Grayscale
Processing Bit Depth	12 Bit
Sensor Correction / Tap Sorting	no
Maximum Images Dimensions	49136 * 16777215
ROI Stepsize	x: 8, y: 1
Tap Geometry	Yes, horizontal 1, 2, 3 and 4 Taps 1X, 1X2, 2X, 2XE, 2XM, 1X3, 3X, 1X4, 4X, 4XE, 4X2E, 2X2, 2X2E, 2X2M
Mirroring	Yes, horizontal (Use vantage point)
Image Selector	Yes
Noise Filter	No
Shading Correction	No
Dead Pixel Interpolation	No
Bayer Filter	No
Color White Balancing	Yes
Lookup Table	Full Resolution
DMA	Full Speed
DMA Image Output Format	Grayscale 8 bits, 10 bits packed, 12 bit packed or 16 bit
Event Generation	yes
Overflow Control	yes

1.1.1. Parameterization Order

We recommend to configure the functional blocks which are responsible for sensor setup/correction first. This will be the camera settings, shading correction, and dead pixel interpolation (if available). Afterwards, you can configure other image enhancement functional blocks such as white balancing, noise filter, and lookup table. By default, all presets are configured for receiving images directly.

1.2. Bandwidth

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

Table 1.2. Bandwidth of Acq_SingleMediumLineGray

Description	Bandwidth
Max. CamClk	85 MHz
Peak Bandwidth per Camera	340 MPixel/s
Mean Bandwidth per Camera	340 MPixel/s
DMA Bandwidth	1800 MByte/s (depends on PC mainboard) Approx. 1350 MB/s on LightBridge depending on cable length and PC interface.

Max. CamClk refers to the maximum pixel clock frequency used by the the Camera Link camera. You can use any lower pixel clock frequency. Lower pixel clock frequencies enable longer CameraLink cables.

The peak bandwidth defines the maximum allowed bandwidth for each camera at the camera interface. When transferring data from the camera to the frame grabber at this bandwidth, the buffer on the frame grabber will fill up as the data can be buffered, but not be processed at that speed. The peak bandwidth is related to the Camera Link interface mode and pixel clock frequency. The product of the used Camera Link taps and the pixel frequency must not exceed the peak bandwidth.

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.

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 1800 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 frame grabber license are listed.

1.3.1. Software Requirements

To run this applet, a Silicon Software Framegrabber SDK installation is required. Ensure you use the applet with compatible versions only. You should also take care to use the board firmware and drivers included in the Basler Framegrabber SDK. The minimum Basler Framegrabber SDK version you need for using this applet is version 5.5.1.

For integration in 3rd party applications, check Chapter 2, '*Software Interface*'.

1.3.2. Hardware Requirements

To run applet "Acq_SingleMediumLineGray", a Silicon Software microEnable 5 marathon/LightBridge VCL frame grabber is required.

For PC system requirements, check the frame grabber 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 frame grabbers can run the applet using the Basler Framegrabber SDK.

1.4. Camera Interface

Applet "Acq_SingleMediumLineGray" supports one Camera Link camera in medium configuration mode. The frame grabber has two Camera Link connectors. Connect the camera to port 'A' with its base connector and to port 'B' with its medium connector.

Figure 1.1. Camera Interface and Camera Cable Setup

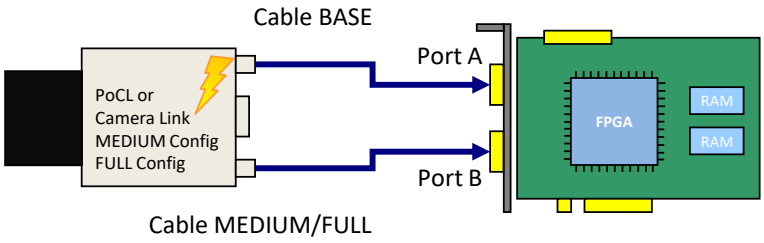
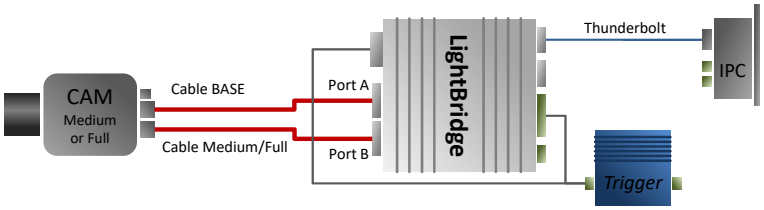


Figure 1.2. Camera Interface and Camera Cable Setup



1.5. Image Transfer to PC Memory

The image transfer between frame grabber 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 16, '*Output Format*'. All outputs are little-endian coded.

DMA Image Tag

The applet does not generate a valid DMA image tag (**FG_IMAGE_TAG**). You may check for lost or corrupted frames using the overflow module described in Chapter 12, '*Overflow*'.

Chapter 2. Software Interface

The software interface of this applet is fully compatible to the Basler Framegrabber SDK. Please read the Basler Framegrabber API manual of the Basler Framegrabber SDK to understand how to include the frame grabbers and their applets into own applications. https://docs.baslerweb.com/frame-grabbers/framegrabber-sdk-overview.html

The Basler Framegrabber SDK includes functional SDK examples which use the features of the Framegrabber SDK. Most of these examples can be used with this AcquisitionApplets. These examples are very useful to learn on how to acquire images, set parameters and use events.

This document is focused on the explanation of the functionality and parameterization of the applet. The next chapters will list all parameters of this applet. Keep in mind that for multi-camera applets, parameters can be set for all cameras individually. The sample source codes parameterize the processing components of the first camera. The index in the source code examples has to be changed for the other cameras.

Amongst others, parameters of the applet are set and read using functions

- int Fg_setParameter(Fg_Struct *Fg, const int Parameter, const void *Value, const unsigned int index)
- int Fg_setParameterWithType(Fg_Struct *Fg, const int Parameter, const void *Value, const unsigned int index, const enum FgParamTypes type)
- int Fg_getParameter(Fg_Struct *Fg, int Parameter, void *Value, const unsigned int index)
- int Fg_getParameterWithType(Fg_Struct *Fg, const int Parameter, void *Value, const unsigned int index, const enum FgParamTypes type)

The index is used to address a DMA channel, a camera index or a processing logic index. It is important to understand the relations between cameras, processes, parameters and DMA channels. This AcquisitionApplets is a single camera applet and is using only one DMA channel. All parameterizations are made using index 0 only.

For applets having multiple DMA channels for each camera, the relation between the indices is more complex. Please check the respective documentation of these applets for more details.

Chapter 3. Camera Link

This AcquisitionApplets can be used with one line scan camera in Camera Link To receive correct image data from your camera, it is crucial that the camera output format matches the selected frame grabber input format. The following parameters configure the grabber's camera interface to match with the individual camera properties. 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 data format.

Ensure that the lines transferred by the camera do not exceed the maximum allowed line length for this applet (49136).

3.1. FG_CAMERA_LINK_CAMTYPE

This parameter specifies the data format of the connected camera.

This camera interface can be configured to support the different data formats specified by the Camera Link standard. In this AcquisitionApplets,the processing data bit depth is 12 bit. The camera interface automatically performs a conversion to the 12 bit format using bit shifting independently from the selected camera format. If the Camera Link bit depth is greater than the processing bit depth, bits will be right shifted to meet the internal bit depth. If the Camera Link 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.

Table 3.1. Parameter properties of FG_CAMERA_LINK_CAMTYPE

Property	Value
Name	**FG_CAMERA_LINK_CAMTYPE**
Display Name	**Input Format**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_CL_MEDIUM_4_TAP_8_BIT** Medium 4 Tap 8bit **FG_CL_MEDIUM_4_TAP_10_BIT** Medium 4 Tap 10bit **FG_CL_MEDIUM_4_TAP_12_BIT** Medium 4 Tap 12bit **FG_CL_MEDIUM_3_TAP_10_BIT** Medium 3 Tap 10bit **FG_CL_MEDIUM_3_TAP_12_BIT** Medium 3 Tap 12bit
Default value	**FG_CL_MEDIUM_8_BIT**

Example 3.1. Usage of FG_CAMERA_LINK_CAMTYPE

```
int result = 0;
int value = FG_CL_MEDIUM_8_BIT;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERA_LINK_CAMTYPE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERA_LINK_CAMTYPE, &value, 0, type)) < 0) {
    /* error handling */
}
```

3.2. FG_USEDVAL

With this parameter it is possible to support cameras that do not fully comply with the Camera Link specification. If **FG_YES** is selected, DVAL, LVAL and FVAL is used to decode valid pixels. If the parameter is set to **FG_NO**,

only LVAL and FVAL is used to decode valid pixels. If you are not sure about the required setting, keep the parameter's value at **FG_YES**.

Table 3.2. Parameter properties of FG_USEDVAL

Property	Value
Name	**FG_USEDVAL**
Display Name	**Use DVAL**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_YES** Yes **FG_NO** No
Default value	**FG_YES**

Example 3.2. Usage of FG_USEDVAL

```
int result = 0;
int value = FG_YES;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_USEDVAL, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_USEDVAL, &value, 0, type)) < 0) {
    /* error handling */
}
```

3.3. FG_CAMERA_LINK_CORE_RESET

This parameter allows to re-initialize the Camera Link core. Under normal circumstances, re-initializing the Camera Link core is not necessary. This parameter is exclusively for use by the Silicon Software Support department. Value range: [0;1]

1 = Enforces a reset of the Camera Link core.

0 = Resets the reset.

Table 3.3. Parameter properties of FG_CAMERA_LINK_CORE_RESET

Property	Value
Name	**FG_CAMERA_LINK_CORE_RESET**
Display Name	**Camera Link Core Reset**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**Minimum** 0 **Maximum** 1 **Stepsize** 1
Default value	**0**
Unit of measure	

Example 3.3. Usage of FG_CAMERA_LINK_CORE_RESET

```
int result = 0;
unsigned int value = 0;
```

```
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERA_LINK_CORE_RESET, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERA_LINK_CORE_RESET, &value, 0, type)) < 0) {
    /* error handling */
}
```

3.4. FG_CAMERA_LINK_PIXEL_CLOCK_X

This parameter displays the value of the pixel clock (in MHz) as meassured on the Camera Link channel X. This is a read only parameter. The value range is [0;85].

Table 3.4. Parameter properties of FG_CAMERA_LINK_PIXEL_CLOCK_X

Property	Value
Name	**FG_CAMERA_LINK_PIXEL_CLOCK_X**
Display Name	**Pixel Clock X**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 85** **Stepsize 1**
Unit of measure	**MHz**

Example 3.4. Usage of FG_CAMERA_LINK_PIXEL_CLOCK_X

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_CAMERA_LINK_PIXEL_CLOCK_X, &value, 0, type)) < 0) {
    /* error handling */
}
```

3.5. FG_CAMERA_LINK_PIXEL_CLOCK_Y

This parameter displays the value of the pixel clock (in MHz) as meassured on the Camera Link channel Y. This is a read only parameter. The value range is [0;85].

Table 3.5. Parameter properties of FG_CAMERA_LINK_PIXEL_CLOCK_Y

Property	Value
Name	**FG_CAMERA_LINK_PIXEL_CLOCK_Y**
Display Name	**Pixel Clock Y**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 85** **Stepsize 1**
Unit of measure	**MHz**

Example 3.5. Usage of FG_CAMERA_LINK_PIXEL_CLOCK_Y

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_CAMERA_LINK_PIXEL_CLOCK_Y, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 4. Camera

This applet Acq_SingleMediumLineGray for the microEnable 5 marathon/LightBridge VCL acquires the sensor data of a line scan camera. When this is performed some sensor dimension depending information can be used to register an event based callback function.

4.1. Events

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

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

4.1.1. FG_START_OF_FRAME_CAM_PORT_0

This event is generated when the first pixel of one camera frame arrives at the applet. Except for the timestamp, the event has no additional data included. Keep in mind that a high framerate can cause high interrupt rates which might slow down the overall PC system. This event can only occur if the acquisition is running.

4.1.2. FG_END_OF_FRAME_CAM_PORT_0

This event is generated right after the last pixel of one camera frame arrives at the applet. Except for the timestamp, the event has no additional data included. Keep in mind that a high framerate can cause high interrupt rates which might slow down the overall PC system. This event can only occur if the acquisition is running.

4.1.3. FG_START_OF_LINE_CAM_PORT_0

This event is generated when the first pixel of camera line arrives at the framegrabber. Except for the timestamp, the event has no additional data included. Keep in mind that a high linerate can cause a critical high interrupt rate which might slow down the overall PC system. Even if the trigger setup will not use this line for a generated frame output this event will occour. This event can only occur if the acquisition is running.

4.1.4. FG_END_OF_LINE_CAM_PORT_0

This event is generated when the last pixel of camera line has arrives at the framegrabber. Except for the timestamp, the event has no additional data included. Keep in mind that a high linerate can cause a critical high interrupt rate which might slow down the overall PC system. This event can only occur if the acquisition is running.

Chapter 5. Tap Geometry

5.1. FG_TAPGEOMETRY

Define the tap geometry with this parameter. The parameter follows the naming defined in the GenICam SFNC (Standard Features Naming Convention).

Table 5.1. Parameter properties of FG_TAPGEOMETRY

Property	Value
Name	**FG_TAPGEOMETRY**
Display Name	**Tap Geometry**
Type	**Enumeration**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**FG_GEOMETRY_1X** 1X **FG_GEOMETRY_1X2** 1X2 **FG_GEOMETRY_1X3** 1X3 **FG_GEOMETRY_1X4** 1X4 **FG_GEOMETRY_2X** 2X **FG_GEOMETRY_2XE** 2XE **FG_GEOMETRY_2XM** 2XM **FG_GEOMETRY_2X2** 2X2 **FG_GEOMETRY_2X2E** 2X2E **FG_GEOMETRY_2X2M** 2X2M **FG_GEOMETRY_3X** 3X **FG_GEOMETRY_4X** 4X **FG_GEOMETRY_4XE** 4XE **FG_GEOMETRY_4X2** 4X2 **FG_GEOMETRY_4X2E** 4X2E
Default value	**FG_GEOMETRY_1X**

Example 5.1. Usage of FG_TAPGEOMETRY

```
int result = 0;
int value = FG_GEOMETRY_1X;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_TAPGEOMETRY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TAPGEOMETRY, &value, 0, type)) < 0) {
    /* error handling */
}
```

5.2. FG_SENSORWIDTH

For tap geometry sorting, the applet needs to be parameterized with the exact sensor width that is transfered from the camera to the frame grabber. If you have set a region of interest (ROI) in the camera, the parameter *FG_SENSORWIDTH* needs to be set to the ROI size. Note that this value has to be a multiple of the taps. For example, if you are using a 10 tap camera, *FG_SENSORWIDTH* will be, for example, 1020 pixels. A value of 1024 is therefore not possible in this example.

The Value of *FG_SENSORWIDTH* Is Not Used, If Only One *x-zone* Is Used and Mirroring Is Not Active

If the tap geometry is configured to one *x-zone* only and no mirroring is configured, the value of the parameter *FG_SENSORWIDTH* is not used. Instead, the sum of *FG_XOFFSET* and *FG_WIDTH* is used. This makes the use of the module easier as an extra configuration is avoided, if defaults are used.

Table 5.2. Parameter properties of FG_SENSORWIDTH

Property	Value
Name	**FG_SENSORWIDTH**
Display Name	**Sensor Width**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum** 8 **Maximum** 49136 **Stepsize** 1
Default value	**1024**
Unit of measure	**pixel**

Example 5.2. Usage of FG_SENSORWIDTH

```
int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SENSORWIDTH, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SENSORWIDTH, &value, 0, type)) < 0) {
    /* error handling */
}
```

5.3. FG_VANTAGEPOINT

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

Table 5.3. Parameter properties of FG_VANTAGEPOINT

Property	Value
Name	**FG_VANTAGEPOINT**
Display Name	**Vantage Point**
Type	**Enumeration**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**FG_VANTAGEPOINT_TOP_LEFT** Top-Left **FG_VANTAGEPOINT_TOP_RIGHT** Top-Right
Default value	**FG_VANTAGEPOINT_TOP_LEFT**

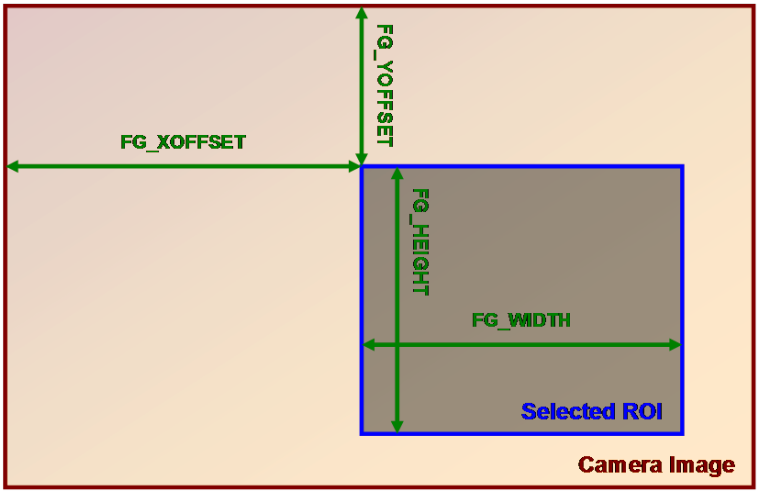
Example 5.3. Usage of FG_VANTAGEPOINT

```
int result = 0;
int value = FG_VANTAGEPOINT_TOP_LEFT;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_VANTAGEPOINT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_VANTAGEPOINT, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 6. ROI

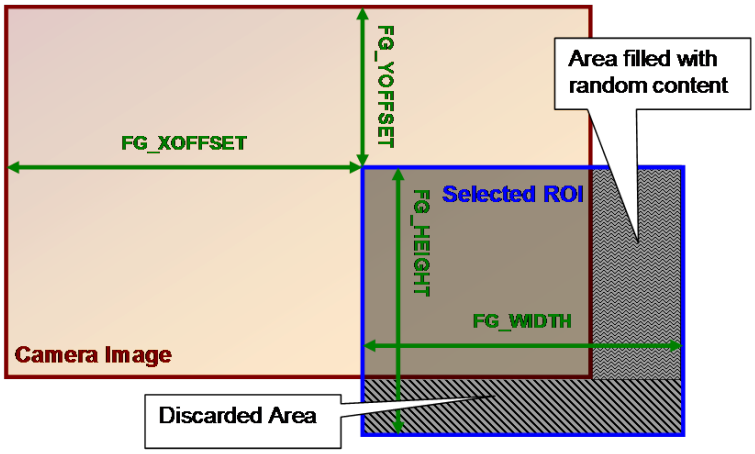
This module allows the definition of a region of interest (ROI), also called area of interest (AOI). A ROI allows the selection of a smaller subset pixel area from the input image. It is defined by using parameters *FG_XOFFSET*, *FG_WIDTH*, *FG_YOFFSET* and *FG_HEIGHT*. The following figure illustrates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

Figure 6.1. 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 6.2. 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 49136 pixels and a maximum image height of 16777215 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.

Camera ROI

Most cameras allow the setting of a ROI inside the camera. The ROI settings described in this section are independent from the camera settings.

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 frame grabber.

6.1. FG_WIDTH

The parameter specifies the width of the ROI. The values of parameters *FG_WIDTH* + *FG_XOFFSET* must not exceed the maximum image width of 49136 pixels.

Table 6.1. Parameter properties of FG_WIDTH

Property	Value
Name	**FG_WIDTH**
Display Name	**Width**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum 8** **Maximum 49136** **Stepsize 8**
Default value	**1024**
Unit of measure	**pixel**

Example 6.1. Usage of FG_WIDTH

```
int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}
```

6.2. FG_HEIGHT

The parameter specifies the height of the ROI. The values of parameters *FG_HEIGHT* + *FG_YOFFSET* must not exceed the maximum image height of 16777215 pixels.

Table 6.2. Parameter properties of FG_HEIGHT

Property	Value
Name	**FG_HEIGHT**
Display Name	**Height**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 16777215** **Stepsize 1**
Default value	**1024**
Unit of measure	**pixel**

Example 6.2. Usage of FG_HEIGHT

```
int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}
```

6.3. FG_XOFFSET

The x-offset is defined by this parameter.

Table 6.3. Parameter properties of FG_XOFFSET

Property	Value
Name	**FG_XOFFSET**
Display Name	**XOffset**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 49128** **Stepsize 8**
Default value	**0**
Unit of measure	**pixel**

Example 6.3. Usage of FG_XOFFSET

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_XOFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_XOFFSET, &value, 0, type)) < 0) {
    /* error handling */
```

```
}
```

6.4. FG_YOFFSET

The y-offset is defined by this parameter.

Table 6.4. Parameter properties of FG_YOFFSET

Property	Value
Name	**FG_YOFFSET**
Display Name	**YOffset**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 16777215** **Stepsize 1**
Default value	**0**
Unit of measure	**pixel**

Example 6.4. Usage of FG_YOFFSET

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_YOFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_YOFFSET, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 7. Digital I/O

The frame grabber provides digital inputs and digital outputs for triggering, light synchronization, hardware control etc. This microEnable 5 marathon/LightBridge VCL frame grabber has

- 8 general purpose digital inputs (GPIs) using the extension board connector of the frame grabber.
- 4 front general purpose digital inputs (Front GPIs) using the SUB-D connector on the frame grabber slot bracket or LightBridge front.
- 8 digital outputs on the GPO connector
- 2 digital outputs on the Front GPO connector
- 4 Camera Control (CC) links on each Camera Link connector

This AcquisitionApplets allows an arbitrary mapping of the inputs to the trigger processing modules of the frame grabber. The same applies for the outputs: Any signal source from the trigger modules or digital inputs can be selected.

- **GND:** Value set to GND, zero. For digital outputs check for possibly inverted outputs.
- **VCC:** Value set to VCC, one. For digital outputs check for possibly inverted outputs.
- **FG_SIGNAL_CAM0_EXSYNC:** The Exsync signal. Usually the line trigger signal used to trigger the camera. Check Chapter 9, '*Line Trigger / ExSync*' for more information.
- **FG_SIGNAL_CAM0_EXSYNC2:** The Exsync 2 signal a delayed exsync signal. Check *FG_LINETRIGGERDELAY* for more information.
- **FG_SIGNAL_CAM0_FLASH:** The flash signal. It is generated once at the start of each frame generated by the trigger module. Check Chapter 10, '*Image Trigger / Flash*' for more information.
- **FG_SIGNAL_CAM0_LVAL:** The line valid signal of the received camera or simulator image data. The signal is high for the duration of the line data transfer.
- **FG_SIGNAL_CAM0_FVAL:** The frame valid signal after the trigger module. The signal is high for the duration of the frame data transfer. Depending on the image trigger mode, the image dimension and timing the signal can vary. See Chapter 10, '*Image Trigger / Flash*' for more information.
- **FG_SIGNAL_GPI_0 to FG_SIGNAL_GPI_7 and FG_SIGNAL_FRONT_GPI_0 to FG_SIGNAL_FRONT_GPI_3:** Direct mapping of the digital input signal after debouncing.
- **FG_SIGNAL_CAM0_LINE_START:** Line start pulse. Use for events and signal analyzer.
- **FG_SIGNAL_CAM0_LINE_END:** Line end pulse. Use for events and signal analyzer.
- **FG_SIGNAL_CAM0_FRAME_START:** Frame start pulse. Use for events and signal analyzer.
- **FG_SIGNAL_CAM0_FRAME_END:** Frame end pulse. Use for events and signal analyzer.

7.1. GPO

7.1.1. FG_TRIGGEROUT_GPO_0_SOURCE et al.

Note

This description applies also to the following parameters: FG_TRIGGEROUT_GPO_1_SOURCE, FG_TRIGGEROUT_GPO_2_SOURCE, FG_TRIGGEROUT_GPO_3_SOURCE, FG_TRIGGEROUT_GPO_4_SOURCE, FG_TRIGGEROUT_GPO_5_SOURCE, FG_TRIGGEROUT_GPO_6_SOURCE, FG_TRIGGEROUT_GPO_7_SOURCE, FG_TRIGGEROUT_FRONT_GPO_0_SOURCE, FG_TRIGGEROUT_FRONT_GPO_1_SOURCE

Select the signal source of the General Purpose Output (GPO) or Front GPO. For further explanation of the available sources see Chapter 7, '*Digital I/O*'.

You can change the polarity using parameter *FG_TRIGGEROUT_GPO_X_POLARITY*.

Table 7.1. Parameter properties of FG_TRIGGEROUT_GPO_0_SOURCE

Property	Value	
Name	**FG_TRIGGEROUT_GPO_0_SOURCE**	
Display Name	**GPO 0 Source**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**GND**	Gnd
	VCC	Vcc
	FG_SIGNAL_CAM0_EXSYNC	Cam0 Exsync
	FG_SIGNAL_CAM0_EXSYNC2	Cam0 Exsync2
	FG_SIGNAL_CAM0_FLASH	Cam0 Flash
	FG_SIGNAL_CAM0_LVAL	Cam0 LVAL
	FG_SIGNAL_CAM0_FVAL	Cam0 FVAL
	FG_SIGNAL_GPI_0	GPI 0
	FG_SIGNAL_GPI_1	GPI 1
	FG_SIGNAL_GPI_2	GPI 2
	FG_SIGNAL_GPI_3	GPI 3
	FG_SIGNAL_GPI_4	GPI 4
	FG_SIGNAL_GPI_5	GPI 5
	FG_SIGNAL_GPI_6	GPI 6
	FG_SIGNAL_GPI_7	GPI 7
	FG_SIGNAL_FRONT_GPI_0	Front GPI 0
	FG_SIGNAL_FRONT_GPI_1	Front GPI 1
	FG_SIGNAL_FRONT_GPI_2	Front GPI 2
	FG_SIGNAL_FRONT_GPI_3	Front GPI 3
Default value	**FG_SIGNAL_CAM0_FLASH**	

Example 7.1. Usage of FG_TRIGGEROUT_GPO_0_SOURCE

```
int result = 0;
int value = FG_SIGNAL_CAM0_FLASH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_TRIGGEROUT_GPO_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGEROUT_GPO_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}
```

7.1.2. FG_TRIGGEROUT_GPO_0_POLARITY et al.

Note

This description applies also to the following parameters: FG_TRIGGEROUT_GPO_1_POLARITY, FG_TRIGGEROUT_GPO_2_POLARITY, FG_TRIGGEROUT_GPO_3_POLARITY, FG_TRIGGEROUT_GPO_4_POLARITY, FG_TRIGGEROUT_GPO_5_POLARITY, FG_TRIGGEROUT_GPO_6_POLARITY, FG_TRIGGEROUT_GPO_7_POLARITY, FG_TRIGGEROUT_FRONT_GPO_0_POLARITY, FG_TRIGGEROUT_FRONT_GPO_1_POLARITY

Select the output polarity the General Purpose Output (GPO) or Front GPO to invert the signal. For further explanation of the available sources see Chapter 7, '*Digital I/O*'.

Table 7.2. Parameter properties of FG_TRIGGEROUT_GPO_0_POLARITY

Property	Value
Name	**FG_TRIGGEROUT_GPO_0_POLARITY**
Display Name	**GPO 0 Polarity**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_LOW** Low Active **FG_HIGH** High Active
Default value	**FG_HIGH**

Example 7.2. Usage of FG_TRIGGEROUT_GPO_0_POLARITY

```
int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_TRIGGEROUT_GPO_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TRIGGEROUT_GPO_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}
```

7.1.3. FG_DIGIO_OUTPUT

Set the output value of outputs 3 and 7 using this parameter. Bit 0 of the parameter refers to the value at output 3, while bit 1 refers to output 7.

Legacy Parameter

This is a legacy parameter. The parameter will overwrite parameters *FG_TRIGGEROUT_GPO_3_SOURCE* and *FG_TRIGGEROUT_GPO_7_SOURCE*. If these parameters are not set to VCC or GND this parameter will return value -1 as it cannot represent the value as a bitmask. Writing value -1 to this parameter will have no effect on the hardware.

Table 7.3. Parameter properties of FG_DIGIO_OUTPUT

Property	Value
Name	**FG_DIGIO_OUTPUT**
Display Name	**Legacy Digital Output**
Type	**Signed Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum -1** **Maximum 3** **Stepsize 1**
Default value	**3**
Unit of measure	

Example 7.3. Usage of FG_DIGIO_OUTPUT

```
int result = 0;
int value = 3;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_DIGIO_OUTPUT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DIGIO_OUTPUT, &value, 0, type)) < 0) {
    /* error handling */
}
```

7.2. GPI

7.2.1. FG_DIGIO_INPUT

Parameter *FG_DIGIO_INPUT* is used to monitor the digital inputs of the frame grabber. This AcquisitionApplets has 12 digital inputs. You can read the current state of these inputs using parameter *FG_DIGIO_INPUT*. Bit 0 of the read value represents input 0, bit 1 represents input 1 and so on. For example, if you obtain the value 37 or hexadecimal 0x25 the frame grabber will have high level on it's digital inputs 0, 2 and 5.

Table 7.4. Parameter properties of FG_DIGIO_INPUT

Property	Value
Name	**FG_DIGIO_INPUT**
Display Name	**Digital Input**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 4095** **Stepsize 1**
Unit of measure	

Example 7.4. Usage of FG_DIGIO_INPUT

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;
```

```
if ((result = Fg_getParameterWithType(fg, FG_DIGIO_INPUT, &value, 0, type)) < 0) {
    /* error handling */
}
```

7.3. Event Source

7.3.1. FG_CUSTOM_SIGNAL_EVENT_0_SOURCE

FIXME_ParameterDocumentation_Missing_FG_CUSTOM_SIGNAL_EVENT_0_SOURCE

Table 7.5. Parameter properties of FG_CUSTOM_SIGNAL_EVENT_0_SOURCE

Property	Value	
Name	**FG_CUSTOM_SIGNAL_EVENT_0_SOURCE**	
Display Name	**Custom Signal Event 0 Source**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**GND**	Gnd
	VCC	Vcc
	FG_SIGNAL_CAM0_EXSYNC	Cam0 Exsync
	FG_SIGNAL_CAM0_EXSYNC2	Cam0 Exsync2
	FG_SIGNAL_CAM0_FLASH	Cam0 Flash
	FG_SIGNAL_CAM0_LVAL	Cam0 LVAL
	FG_SIGNAL_CAM0_FVAL	Cam0 FVAL
	FG_SIGNAL_CAM0_LINE_START	Cam0 Line Transfer Start
	FG_SIGNAL_CAM0_LINE_END	Cam0 Line Transfer End
	FG_SIGNAL_CAM0_FRAME_START	Cam0 Frame Transfer Start
	FG_SIGNAL_CAM0_FRAME_END	Cam0 Frame Transfer End
	FG_SIGNAL_GPI_0	GPI 0
	FG_SIGNAL_GPI_1	GPI 1
	FG_SIGNAL_GPI_2	GPI 2
	FG_SIGNAL_GPI_3	GPI 3
	FG_SIGNAL_GPI_4	GPI 4
	FG_SIGNAL_GPI_5	GPI 5
	FG_SIGNAL_GPI_6	GPI 6
	FG_SIGNAL_GPI_7	GPI 7
	FG_SIGNAL_FRONT_GPI_0	Front GPI 0
	FG_SIGNAL_FRONT_GPI_1	Front GPI 1
	FG_SIGNAL_FRONT_GPI_2	Front GPI 2
	FG_SIGNAL_FRONT_GPI_3	Front GPI 3
Default value	**FG_SIGNAL_CAM0_EXSYNC**	

Example 7.5. Usage of FG_CUSTOM_SIGNAL_EVENT_0_SOURCE

```
int result = 0;
int value = FG_SIGNAL_CAM0_EXSYNC;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}
```

7.3.2. FG_CUSTOM_SIGNAL_EVENT_0_POLARITY

FIXME_ParameterDocumentation_Missing_FG_CUSTOM_SIGNAL_EVENT_0_POLARITY

Table 7.6. Parameter properties of FG_CUSTOM_SIGNAL_EVENT_0_POLARITY

Property	Value
Name	**FG_CUSTOM_SIGNAL_EVENT_0_POLARITY**
Display Name	**Custom Signal Event 0 Polarity**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_LOW** Low Active **FG_HIGH** High Active
Default value	**FG_HIGH**

Example 7.6. Usage of FG_CUSTOM_SIGNAL_EVENT_0_POLARITY

```
int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}
```

7.3.3. FG_CUSTOM_SIGNAL_EVENT_1_SOURCE

FIXME_ParameterDocumentation_Missing_FG_CUSTOM_SIGNAL_EVENT_1_SOURCE

Table 7.7. Parameter properties of FG_CUSTOM_SIGNAL_EVENT_1_SOURCE

Property	Value	
Name	**FG_CUSTOM_SIGNAL_EVENT_1_SOURCE**	
Display Name	**Custom Signal Event 1 Source**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**GND**	Gnd
	VCC	Vcc
	FG_SIGNAL_CAM0_EXSYNC	Cam0 Exsync
	FG_SIGNAL_CAM0_EXSYNC2	Cam0 Exsync2
	FG_SIGNAL_CAM0_FLASH	Cam0 Flash
	FG_SIGNAL_CAM0_LVAL	Cam0 LVAL
	FG_SIGNAL_CAM0_FVAL	Cam0 FVAL
	FG_SIGNAL_CAM0_LINE_START	Cam0 Line Transfer Start
	FG_SIGNAL_CAM0_LINE_END	Cam0 Line Transfer End
	FG_SIGNAL_CAM0_FRAME_START	Cam0 Frame Transfer Start
	FG_SIGNAL_CAM0_FRAME_END	Cam0 Frame Transfer End
	FG_SIGNAL_GPI_0	GPI 0
	FG_SIGNAL_GPI_1	GPI 1
	FG_SIGNAL_GPI_2	GPI 2
	FG_SIGNAL_GPI_3	GPI 3
	FG_SIGNAL_GPI_4	GPI 4
	FG_SIGNAL_GPI_5	GPI 5
	FG_SIGNAL_GPI_6	GPI 6
	FG_SIGNAL_GPI_7	GPI 7
	FG_SIGNAL_FRONT_GPI_0	Front GPI 0
	FG_SIGNAL_FRONT_GPI_1	Front GPI 1
	FG_SIGNAL_FRONT_GPI_2	Front GPI 2
	FG_SIGNAL_FRONT_GPI_3	Front GPI 3
Default value	**FG_SIGNAL_CAM0_FLASH**	

Example 7.7. Usage of FG_CUSTOM_SIGNAL_EVENT_1_SOURCE

```
int result = 0;
int value = FG_SIGNAL_CAM0_FLASH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_1_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_1_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}
```

7.3.4. FG_CUSTOM_SIGNAL_EVENT_1_POLARITY

FIXME_ParameterDocumentation_Missing_FG_CUSTOM_SIGNAL_EVENT_1_POLARITY

Table 7.8. Parameter properties of FG_CUSTOM_SIGNAL_EVENT_1_POLARITY

Property	Value
Name	**FG_CUSTOM_SIGNAL_EVENT_1_POLARITY**
Display Name	**Custom Signal Event 1 Polarity**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_LOW** Low Active **FG_HIGH** High Active
Default value	**FG_HIGH**

Example 7.8. Usage of FG_CUSTOM_SIGNAL_EVENT_1_POLARITY

```
int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_1_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CUSTOM_SIGNAL_EVENT_1_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}
```

7.4. Events

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

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

7.4.1. FG_TRIGGER_INPUT0_RISING

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

7.4.2. FG_TRIGGER_INPUT0_FALLING

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

7.4.3. FG_CUSTOM_SIGNAL_EVENT_0

doc_node_missing!

7.4.4. FG_CUSTOM_SIGNAL_EVENT_1

doc_node_missing!

Chapter 8. CC Signal Mapping

The CameraLink interface specifies four camera input signals, i.e. CC1, CC2, CC3 and CC4. Usually the camera will use one particular CC-signal to trigger the data acquisition and define the exposure time. Please consult the vendor's manual of your camera to identify the required signals and their mapping to the CC1-CC4 signals.

This AcquisitionApplets provides eight different methods to map five source signals to any of the four CC-signal lines:

- **CC_EXSYNC**

  The ExSync signal is used to trigger the line acquisition of the camera. It can be generated periodically, or triggered by one of the eight trigger input signals. When generated periodically by the frame grabber (grabber controlled), it determines the framerate. Depending on the operation mode of the camera, its active-time can determine the exposure time. For details refer to the trigger section of this document.

- **CC_NOT_EXSYNC**

  The inverted signal to **CC_EXSYNC**.

- **CC_EXSYNC2**

  A delayed copy of line trigger ExSync is provided by ExSync2

- **CC_NOT_EXSYNC2**

  The inverted signal to ExSync2.

- **CC_STROBEPULSE**

  A delayed copy of the image trigger is provided by the flash. It has as separate delay and is usually used to control external flashes. For details refer to the Image Trigger System.

- **CC_NOT_STROBEPULSE**

  The inverted signal to **CC_STROBEPULSE**.

- **CC_GND**

  Ground
  , corresponding to a low state.

- **CC_VCC**

  V_{CC}, corresponding to a high state.

- **CC_GPI_X**

  X is trigger input source 0 to 7. No de-bouncing applied to selected general purpose input.

- **CC_NOT_GPI_X**

  X is the inverted trigger input source 0 to 7. No de-bouncing applied to selected general purpose input.

- **CC_FRONT_GPI_X**

  X is trigger input source 0 to 3. No de-bouncing applied to selected general purpose input.

- **CC_NOT_FRONT_GPI_X**

  X is the inverted trigger input source 0 to 3. No de-bouncing applied to selected general purpose input.

8.1. FG_CC1_SOURCE et al.

Note

This description applies also to the following parameters: FG_CC2_SOURCE, FG_CC3_SOURCE, FG_CC4_SOURCE

Select the signal source of the CC trigger signal output. For further explanation of the available sources see Chapter 7, '*Digital I/O*'.

Ensure that the signal has the correct polarity and pulse width. Check the camera user manual if the camera requires high active or low active signals. You can change the polarity using parameter *FG_CCX_POLARITY*.

Table 8.1. Parameter properties of FG_CC1_SOURCE

Property	Value	
Name	**FG_CC1_SOURCE**	
Display Name	**CC 1**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**GND**	Gnd
	VCC	Vcc
	FG_SIGNAL_CAM0_EXSYNC	Cam0 Exsync
	FG_SIGNAL_CAM0_EXSYNC2	Cam0 Exsync2
	FG_SIGNAL_CAM0_FLASH	Cam0 Flash
	FG_SIGNAL_CAM0_LVAL	Cam0 LVAL
	FG_SIGNAL_CAM0_FVAL	Cam0 FVAL
	FG_SIGNAL_GPI_0	GPI 0
	FG_SIGNAL_GPI_1	GPI 1
	FG_SIGNAL_GPI_2	GPI 2
	FG_SIGNAL_GPI_3	GPI 3
	FG_SIGNAL_GPI_4	GPI 4
	FG_SIGNAL_GPI_5	GPI 5
	FG_SIGNAL_GPI_6	GPI 6
	FG_SIGNAL_GPI_7	GPI 7
	FG_SIGNAL_FRONT_GPI_0	Front GPI 0
	FG_SIGNAL_FRONT_GPI_1	Front GPI 1
	FG_SIGNAL_FRONT_GPI_2	Front GPI 2
	FG_SIGNAL_FRONT_GPI_3	Front GPI 3
Default value	**FG_SIGNAL_CAM0_EXSYNC**	

Example 8.1. Usage of FG_CC1_SOURCE

```
int result = 0;
int value = FG_SIGNAL_CAM0_EXSYNC;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CC1_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CC1_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}
```

8.2. FG_CC1_POLARITY et al.

Note

This description applies also to the following parameters: FG_CC2_POLARITY, FG_CC3_POLARITY, FG_CC4_POLARITY

Select the output polarity of the CC trigger signal. For further explanation of the available sources see Chapter 7, '*Digital I/O*'.

Ensure that the signal has the correct polarity and pulse width. Check the camera user manual if the camera requires high active or low active signals.

Table 8.2. Parameter properties of FG_CC1_POLARITY

Property	Value
Name	**FG_CC1_POLARITY**
Display Name	**CC 1 Polarity**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_LOW** Low Active **FG_HIGH** High Active
Default value	**FG_HIGH**

Example 8.2. Usage of FG_CC1_POLARITY

```
int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CC1_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CC1_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 9. Line Trigger / ExSync

The line trigger function block uses signals to control the line scan acquisition of the specific camera. A external synchronization signal or internal generated puls with fixed frequency being sent to the line scan camera is called ExSync. With the help of this signal it is possible to control the exposure of the connected camera.

The camera needs to be configured accordingly to use the ExSync as control signal. Furthermore the camera might expect the ExSync at a particular CC signal and/or polarity.

In most cases for CameraLink the the exposure control is expected at CameraControl signal CC1.

An sensor exposure control based on pulse length/duration is very common. Please make sure that the exposure time is less than the period of the expected maximum line frequency. Consult the camera's manual for more details because these are device specific. More details concerning ExSync can be found in the parameter description of *FG_EXSYNCON*.

Basically two different generation modes for the ExSync signals are available,

- a simple periodical and
- an externally triggered generation.

Additionally, two variants of these are available,

- the first is independent from the image gate,
- and the second is gated by the image gate, which creates ExSync signals only during the actual acquisition.

All details can be found in the parameter description of *FG_LINETRIGGERMODE*.

For the mapping of the ExSync signals to the digital outputs check Chapter 7, '*Digital I/O*'.

9.1. FG_LINETRIGGERMODE

Please choose one of the line trigger modes described here. Make sure that the operation modes of the frame grabber and the camera are the same.

Image independent ExSync modes:

- **Grabber Controlled**

  For the grabber controlled line trigger, the ExSync signal is a simple periodical signal. Its period defines the line frequency and its active time is used by many cameras to define the exposure time.

- **External Trigger**

  The external trigger mode for ExSync generates a single ExSync pulse when the external trigger source becomes active. The ExSync defines the exposure time for the camera. During the exposure time is not possible to re-trigger the ExSync. If the camera needs an additional setup time, it is possible to extend the deadtime of the trigger - the time where no re-trigger is possible - beyond the exposure time. If you want to trigger fewer lines than pulses available at the trigger input, it is possible to downscale the trigger input, e.g. a downscaler of 2 will generate an ExSync every 2nd input pulse, a downscaler of 3 only every third of the input pulses, and so on.

Image gate dependent ExSync modes:

- **Grabber Controlled Gated**

For the grabber controlled gated line trigger, the ExSync signal is generated the very same way as for the grabber controlled mode described above. However, the generator for the ExSync is starting the rising image gate and stops with the image gate becoming inactive. This gives a smaller jitter for the time from the start of the image gate and the generation of the first ExSync, especially for very long ExSync periods.

- **External Trigger Gated**

  For the external trigger gated controlled line trigger, the ExSync signal is generated the very same way as for the external trigger mode described above. However, the generator for the ExSync is starting the rising image gate and stops with the image gate becoming inactive. For this mode two downscalers are available. The first is the downscaler from the beginning of the image gate to the first ExSync, it is called phase. The second is downscaling all succeeding input triggers and is the same as the downscaler used in external trigger mode described above. The options downscale and phase allow further adjustment of the camera trigger with respect to its external source, the trigger input. The value downscale determines the divisor of the input frequency, e.g. a downscale of 16 will produce an ExSync every 16 * n of the input trigger. Furthermore, the phase gives the possibility to shift the camera trigger. A phase shift of 90° is achieved when setting phase to 4, which produces a camera trigger at times 16 * n + 4 of the input trigger signal.

Table 9.1. Parameter properties of FG_LINETRIGGERMODE

Property	Value	
Name	**FG_LINETRIGGERMODE**	
Display Name	**Line Trigger Mode**	
Type	**Enumeration**	
Access policy	**Read/Write**	
Storage policy	**Persistent**	
Allowed values	**GRABBER_CONTROLLED** **ASYNC_TRIGGER** **GRABBER_CONTROLLED_GATED** **ASYNC_GATED**	Grabber controlled Async External Trigger Gated, Grabber controlled Gated, External Triggered
Default value	**GRABBER_CONTROLLED**	

Example 9.1. Usage of FG_LINETRIGGERMODE

```
int result = 0;
int value = GRABBER_CONTROLLED;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LINETRIGGERMODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINETRIGGERMODE, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.2. FG_EXSYNCON

This parameter enables the transmission of ExSync signals to the camera.

Please take care to first start the acquisition before setting this ExSyncOn parameter to On (**FG_ON**) if you want to acquire all lines being generated by the camera. The signal will be sent as soon as the ExSync has been started. As soon as the acquisition is started the used timeout parameter becomes valid independent of the ExSyncOn parameter being On (**FG_ON**) or Off (**FG_OFF**). By switching this parameter On (**FG_ON**) and Off (**FG_OFF**) during an acquisition you can check if the camera is configured to use this external signal for exposure start.

Whether the ExSync is really used by the camera is based on the settings of the camera. Consult the camera's manual for more details because these are device specific.

Table 9.2. Parameter properties of FG_EXSYNCON

Property	Value
Name	**FG_EXSYNCON**
Display Name	**Exsync Enable**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_ON** On **FG_OFF** Off
Default value	**FG_ON**

Example 9.2. Usage of FG_EXSYNCON

```
int result = 0;
int value = FG_ON;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_EXSYNCON, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_EXSYNCON, &value, 0, type)) < 0) {
    /* error handling */
}
```

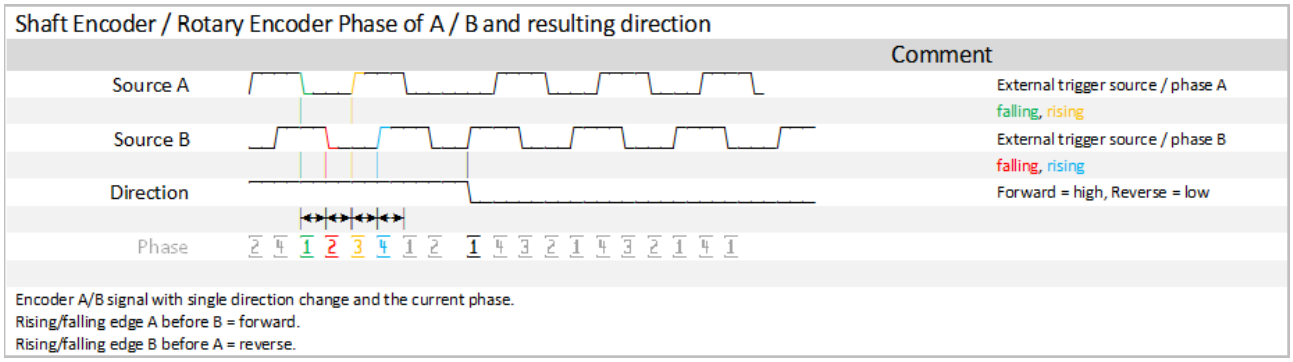
9.3. Line Trigger Input

In the line trigger input category of the line trigger module, the applet is configured for a possible external line trigger input. Here, debouncing times, downscales, polarities and a shaft encoder input are configured.

The external peripheral line trigger source will be in most cases a shaft encoder, also called a rotary encoder. These devices convert the objects movement over an angular motion into relative incremental pulses. The angular motion is taken from the motor axis or a wheel being connected to the translational motion of the scanned object. For most line scan applications it is relevant to get exact feedback of the relative motion between camera and object. By this a certain number of incremental pulses per distance is given to the frame grabber trigger input interface. Depending on the used incremental shaft encoders a certain number (500, 1000, ...) of incremental pulses per rotation is produced.

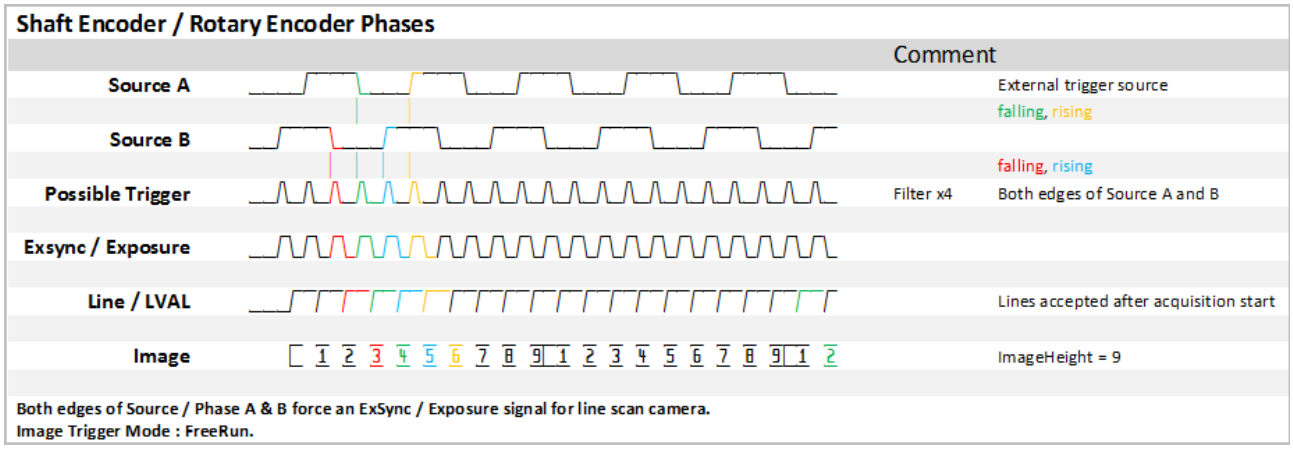
Most incremental shaft encoders provide 2 signals that are called A & B. By using these two signals the relative increments can be seen at the edges of these signals and a direction. In one direction the A-phase high state rises before the B-phase in the other direction, i.e. vice versa. If we do not need a direction for our application, only the A-phase is necessary. A combination of A & B may provide a higher resolution. Please see *FG_SHAFTENCODERMODE* and *FG_SHAFTENCODERON* for this.

Figure 9.1. Shaft Encoder, A & B phase, direction



During an acquisition the shaft encoder signals trigger the ExSync signals and force the sensor to perform an exposure. After the sensor exposure the line is read-out and transfered. The time between exposure and transfer is for most line scan cameras very short.

Figure 9.2. Shaft Encoder, A & B signal, acquisition



The different phases are defined as seen in the following table. A positive phase increment is forward direction, a negative means reverse. This induces rising/falling edge A before B equals forward direction and rising/falling edge B before A means reverse.

Table 9.3. Phases of an A/B Shaft Encoder

Phase	A-state	B-state
1	low	high
2	low	low
3	high	low
4	high	high

Some shaft encoders provide a third signal that is pulsed for each full rotation which is called Z or index. This signal Z could become interesting for an image trigger mode. For more details see Chapter 10, '*Image Trigger / Flash*'.

For most applications and several camera or line scan sensor types it is necessary to have the same resolution in X and Y direction of an image. Due to this the number of pixels per mm in sensor- and motion-direction needs to be the same. In case of an 1024 pixel line scan sensor looking at 10 cm we have 10.24 pixel per mm orthogonal to the web direction. In order to reach an 1:1 scaling we need 10.24 ExSync signals per mm. If a perfectly round object is scanned with an 1:1 scaling then it is exactly round in the image too. When the result becomes elliptic, the scaling is not perfect and some line scan sensor architectures (Bi/Tri-Linear, Dual-Line, ...) will show some additional artefacts.

9.3.1. FG_LINETRIGGERINSRC

This parameter specifies the digital signal source for phase A, which is used to trigger the ExSync signal. If an A/B shaft encoder is used, configure source B at *FG_SHAFTENCODERINSRC*, too. For more details consult the Framegrabber SDK manual.

It is possible to use the shaft encoder A phase only if the direction of scanning is not of interest in the target application. Concerning more details to the shaft encoder please consider the introduction of Section 9.3, 'Line Trigger Input'.

Table 9.4. Parameter properties of FG_LINETRIGGERINSRC

Property	Value	
Name	**FG_LINETRIGGERINSRC**	
Display Name	**Source A**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**TRGINSRC_GPI_0**	GPI Trigger Source 0
	TRGINSRC_GPI_1	GPI Trigger Source 1
	TRGINSRC_GPI_2	GPI Trigger Source 2
	TRGINSRC_GPI_3	GPI Trigger Source 3
	TRGINSRC_GPI_4	GPI Trigger Source 4 (not available on LightBridge)
	TRGINSRC_GPI_5	GPI Trigger Source 5 (not available on LightBridge)
	TRGINSRC_GPI_6	GPI Trigger Source 6 (not available on LightBridge)
	TRGINSRC_GPI_7	GPI Trigger Source 7 (not available on LightBridge)
	TRGINSRC_FRONT_GPI_0	Front GPI Trigger Source 0
	TRGINSRC_FRONT_GPI_1	Front GPI Trigger Source 1
	TRGINSRC_FRONT_GPI_2	Front GPI Trigger Source 2
	TRGINSRC_FRONT_GPI_3	Front GPI Trigger Source 3
Default value	**TRGINSRC_GPI_1**	

Example 9.3. Usage of FG_LINETRIGGERINSRC

```
int result = 0;
int value = TRGINSRC_GPI_1;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LINETRIGGERINSRC, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINETRIGGERINSRC, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.3.2. FG_LINETRIGGERINPOLARITY

The parameter defines the polarity of the external input trigger signal encoder source A and source B. When set to LowActive, the ExSync generator starts on a falling edge of the signal specified by the parameter *FG_LINETRIGGERINSRC*. Otherwise, the ExSync generation starts on a rising edge. This is only relevant if the *FG_LINETRIGGERMODE* is set to an external trigger.

Table 9.5. Parameter properties of FG_LINETRIGGERINPOLARITY

Property	Value	
Name	**FG_LINETRIGGERINPOLARITY**	
Display Name	**Input Polarity**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**HIGH_ON_ZERO_LOW**	Low Active
	HIGH_ON_ZERO_HIGH	High Active
Default value	**HIGH_ACTIVE**	

Example 9.4. Usage of FG_LINETRIGGERINPOLARITY

```
int result = 0;
int value = HIGH_ACTIVE;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LINETRIGGERINPOLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINETRIGGERINPOLARITY, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.3.3. FG_LINETRIGGERDEBOUNCING

This parameter specifies the debouncing time. This is the time for which the input line trigger signals must keep the same value to be detected as such. Fast signal changes within the debouncing time will be filtered out.

Table 9.6. Parameter properties of FG_LINETRIGGERDEBOUNCING

Property	Value
Name	**FG_LINETRIGGERDEBOUNCING**
Display Name	**Debouncing Time**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.008** **Maximum 65.0** **Stepsize 0.008**
Default value	**0.112**
Unit of measure	**µs**

Example 9.5. Usage of FG_LINETRIGGERDEBOUNCING

```
int result = 0;
double value = 0.112;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_LINETRIGGERDEBOUNCING, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINETRIGGERDEBOUNCING, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.3.4. Downscale

9.3.4.1. FG_LINE_DOWNSCALE

Sets the value after how many pulses of the input trigger signal a single one is passed through as ExSync. For example, a value of 2 creates an ExSync pulse at each 2nd input trigger signal. This is only relevant if the *FG_LINETRIGGERMODE* is set to an external trigger mode. The parameter *FG_LINE_DOWNSCALEINIT* selects an initial delay of incoming pulses.

Figure 9.3. Downscale and Init phase behaviour

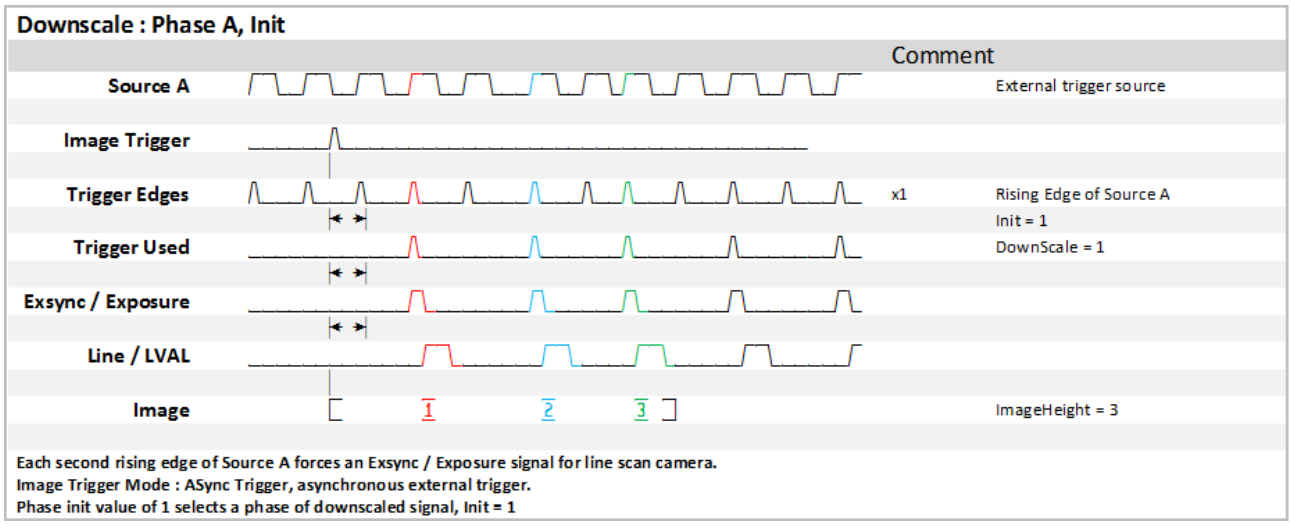


Table 9.7. Parameter properties of FG_LINE_DOWNSCALE

Property	Value
Name	**FG_LINE_DOWNSCALE**
Display Name	**Downscale**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 255** **Stepsize 1**
Default value	**1**
Unit of measure	**pulses**

Example 9.6. Usage of FG_LINE_DOWNSCALE

```
int result = 0;
unsigned int value = 1;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LINE_DOWNSCALE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINE_DOWNSCALE, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.3.4.2. FG_LINE_DOWNSCALEINIT

In addition to the downscale value this parameter sets a phase position. This parameter specifies the number of external input trigger signals, which are needed to generate the first ExSync of a frame. This is only relevant if the *FG_LINETRIGGERMODE* is set to an image gate dependent ExSync mode. This value is applied after the image start pulse. The parameter *FG_LINE_DOWNSCALE* represents the number of possible steps and an explaining figure is found in its description (Init=1).

Table 9.8. Parameter properties of FG_LINE_DOWNSCALEINIT

Property	Value
Name	**FG_LINE_DOWNSCALEINIT**
Display Name	**Phase**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 255** **Stepsize 1**
Default value	**1**
Unit of measure	**pulses**

Example 9.7. Usage of FG_LINE_DOWNSCALEINIT

```
int result = 0;
unsigned int value = 1;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LINE_DOWNSCALEINIT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINE_DOWNSCALEINIT, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.4. Shaft Encoder A/B Filter

With the support of signal A/B for shaft encoders it is possible to detect the rotary direction of an attached encoder and filter the encoder signals accordingly. Also a compensation is performed for up to 16,777,216 reverse encoder signals. A brief description about this feature is found in the shaft encoder documentation.

9.4.1. FG_SHAFTENCODERON

Switch the shaft encoder filter On or Off. This is only relevant if the *FG_LINETRIGGERMODE* is set to an external trigger mode. The functionalities of *FG_SHAFTENCODERMODE*, *FG_SHAFTENCODERINSRC*, *FG_SHAFTENCODERLEADING*, *FG_SHAFTENCODER_COMPENSATION_ENABLE*, *FG_SHAFTENCODER_COMPENSATION_COUNT* become relevant in the case this parameter is set to On = **FG_ON**. When enabling the shaft encoder, a reset of the encoder compensation is performed. If this filter is switched on an correct A & B encoder signal is expected and necessary for correct functionality. Please be aware that the input signal at *FG_SHAFTENCODERINSRC* is interpreted as phase B and the input signal at *FG_LINETRIGGERINSRC* as phase A. A sketch of the signal can be found in the description of parameter *FG_LINETRIGGERINSRC*.

Table 9.9. Parameter properties of FG_SHAFTENCODERON

Property	Value
Name	**FG_SHAFTENCODERON**
Display Name	**Enable**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_ON** On **FG_OFF** Off
Default value	**FG_OFF**

Example 9.8. Usage of FG_SHAFTENCODERON

```
int result = 0;
int value = FG_OFF;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SHAFTENCODERON, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SHAFTENCODERON, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.4.2. FG_SHAFTENCODERMODE

The shaft encoder mode can be run in three operation modes. Please choose the according operation mode for your application. This feature can be used if *FG_SHAFTENCODERON* is switched on. It enables you to adjust the number of increments per rotation of the shaft encoder. Together with the parameter *FG_LINE_DOWNSCALE* you can adjust the increment re-scaling.

The following modes are available:

- Filter x1

  ExSync is generated for a forward rotation of the shaft encoder in single resolution, i.e. a trigger pulse for rising edge of Source A.

- Filter x2

  ExSync is generated for a forward rotation of the shaft encoder in double resolution, i.e. a trigger pulse for a rising and falling edge of Source A, edges of Source B are not used.

- Filter x4

  ExSync is generated for a forward rotation of the shaft encoder in quad resolution, i.e. a trigger pulse for a rising and falling edge of Source A and a rising and falling edge of Source B.

Figure 9.4. Shaft Encoder Mode : Filter x4, x2, x1

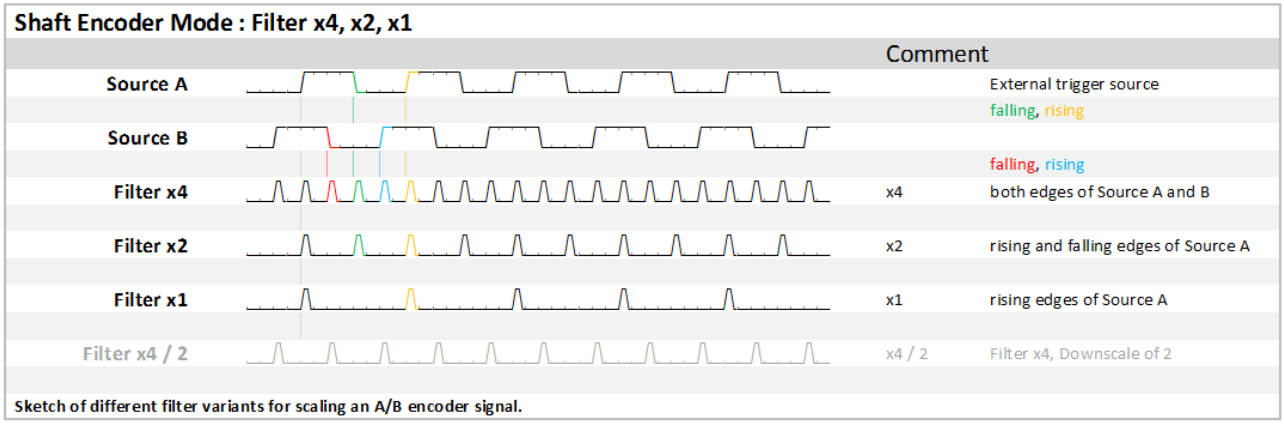


Table 9.10. Parameter properties of FG_SHAFTENCODERMODE

Property	Value
Name	**FG_SHAFTENCODERMODE**
Display Name	**Shaft Encoder Mode**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FILTER_X1** Filter x1 **FILTER_X2** Filter x2 **FILTER_X4** Filter x4
Default value	**FILTER_X1**

Example 9.9. Usage of FG_SHAFTENCODERMODE

```
int result = 0;
int value = FILTER_X1;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SHAFTENCODERMODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SHAFTENCODERMODE, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.4.3. FG_SHAFTENCODERINSRC

Specifies the input signal source / phase B for the shaft encoder filter. Signal source B of the shaft encoder is 90 degree phase shifted to source / phase A. In this document you can get more explanations regarding the input pins in the context of parameter *FG_LINETRIGGERINSRC* and concerning the shaft encoder in the introduction of Section 9.3, 'Line Trigger Input'. Check the hardware documentation of the microEnable trigger board and the Framegrabber SDK manual for more details.

Table 9.11. Parameter properties of FG_SHAFTENCODERINSRC

Property	Value	
Name	**FG_SHAFTENCODERINSRC**	
Display Name	**Source B**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**TRGINSRC_GPI_0**	GPI Trigger Source 0
	TRGINSRC_GPI_1	GPI Trigger Source 1
	TRGINSRC_GPI_2	GPI Trigger Source 2
	TRGINSRC_GPI_3	GPI Trigger Source 3
	TRGINSRC_GPI_4	GPI Trigger Source 4 (not available on LightBridge)
	TRGINSRC_GPI_5	GPI Trigger Source 5 (not available on LightBridge)
	TRGINSRC_GPI_6	GPI Trigger Source 6 (not available on LightBridge)
	TRGINSRC_GPI_7	GPI Trigger Source 7 (not available on LightBridge)
	TRGINSRC_FRONT_GPI_0	Front GPI Trigger Source 0
	TRGINSRC_FRONT_GPI_1	Front GPI Trigger Source 1
	TRGINSRC_FRONT_GPI_2	Front GPI Trigger Source 2
	TRGINSRC_FRONT_GPI_3	Front GPI Trigger Source 3
Default value	**TRGINSRC_GPI_2**	

Example 9.10. Usage of FG_SHAFTENCODERINSRC

```
int result = 0;
int value = TRGINSRC_GPI_2;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SHAFTENCODERINSRC, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SHAFTENCODERINSRC, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.4.4. FG_SHAFTENCODERLEADING

This parameter defines the leading signal (= direction) of the shaft encoder filter. This induces rising/falling edge A before B equals forward direction and rising/falling edge B before A means reverse. The default setting is A as the leading signal. Flipping the input pins or their polarity will have the same effect as changing this to B as the leading signal. It simply defines the valid direction of the scan. An explanation of the direction detection based on an encoder A / B signal is found in Section 9.3, 'Line Trigger Input'.

Table 9.12. Parameter properties of FG_SHAFTENCODERLEADING

Property	Value	
Name	**FG_SHAFTENCODERLEADING**	
Display Name	**Leading Signal**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**SOURCE_A**	Source A
	SOURCE_B	Source B
Default value	**SOURCE_A**	

Example 9.11. Usage of FG_SHAFTENCODERLEADING

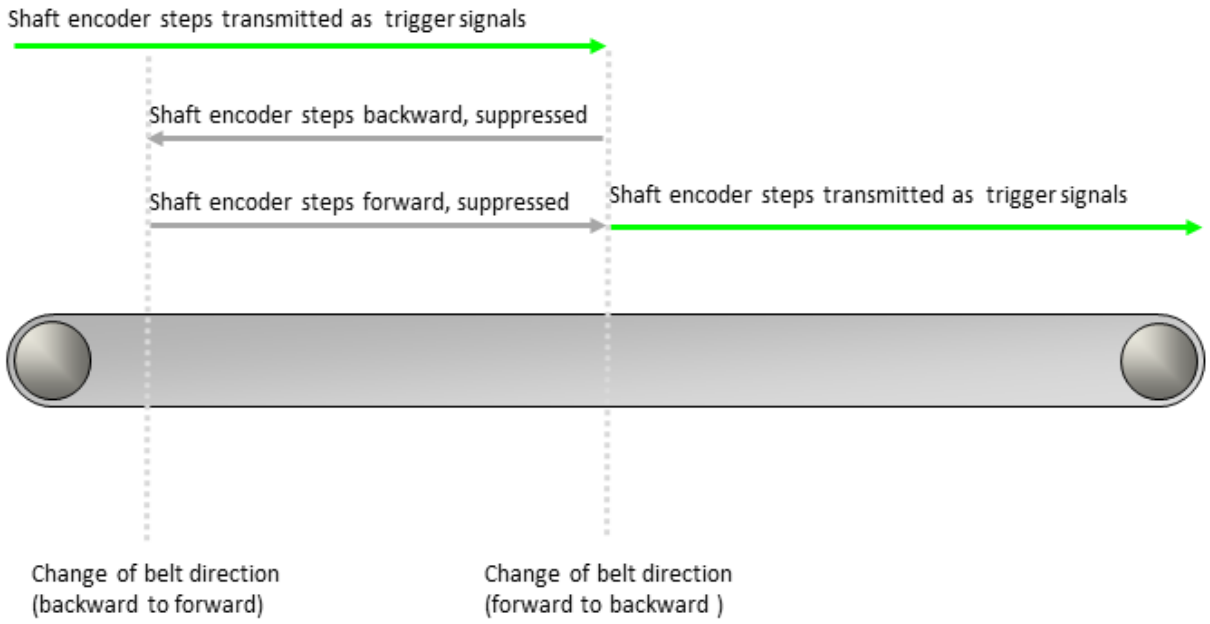
```
int result = 0;
int value = SOURCE_A;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SHAFTENCODERLEADING, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SHAFTENCODERLEADING, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.4.5. FG_SHAFTENCODER_COMPENSATION_ENABLE

The shaft encoder analyzer includes a rollback compensation. In case the rollback compensation is enabled, the module will compensate the reverse movement so that no object is scanned twice. The module will count the number of reverse pulses and will suppress all reverse and forward pulses until position of maximum progress is reached again. If switched to ON, in case of shaft encoder backward movement, the operator counts how many shaft encoder steps the shaft encoder moves backwards. When the shaft encoder moves forwards again, this number of shaft encoder steps (now forward direction) is not transmitted as external trigger signals. Only after the transportation belt is back to the place where the backward movement started, the shaft encoder steps (forward direction) are transmitted as external trigger signals again.

Parameter *FG_SHAFTENCODER_COMPENSATION_ENABLE* switched ON:

Figure 9.5. Shaft Encoder Compensation Enable = ON



In case the rollback compensation is disabled, the shaft encoder analyzer will only suppress reverse pulses but use all forward pulses. If switched to OFF, the operator simply doesn't transmit any trigger signals as long as the transportation belt moves backwards. As soon as the transport belt starts to move forwards again, the operator transmits the shaft encoder steps (forward direction) as trigger signals.

Parameter *FG_SHAFTENCODER_COMPENSATION_ENABLE* switched OFF:

Figure 9.6. Shaft Encoder Compensation Enable = OFF

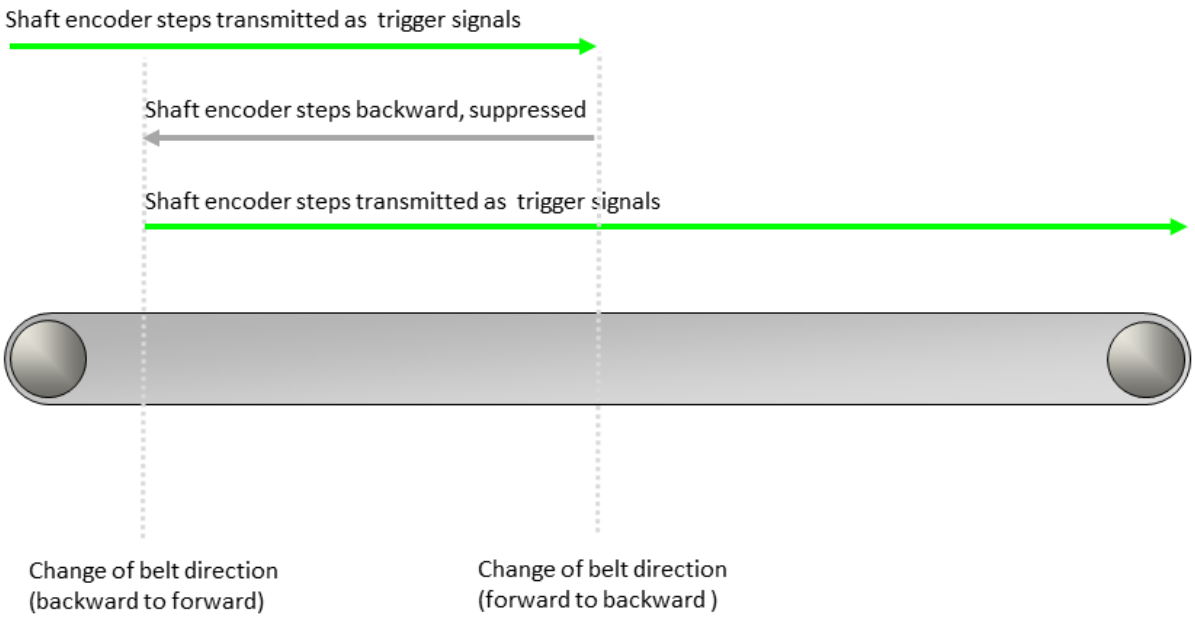


Table 9.13. Parameter properties of FG_SHAFTENCODER_COMPENSATION_ENABLE

Property	Value
Name	**FG_SHAFTENCODER_COMPENSATION_ENABLE**
Display Name	**Compensation Enable**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_ON** On **FG_OFF** Off
Default value	**FG_ON**

Example 9.12. Usage of FG_SHAFTENCODER_COMPENSATION_ENABLE

```
int result = 0;
int value = FG_ON;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SHAFTENCODER_COMPENSATION_ENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SHAFTENCODER_COMPENSATION_ENABLE, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.4.6. FG_SHAFTENCODER_COMPENSATION_COUNT

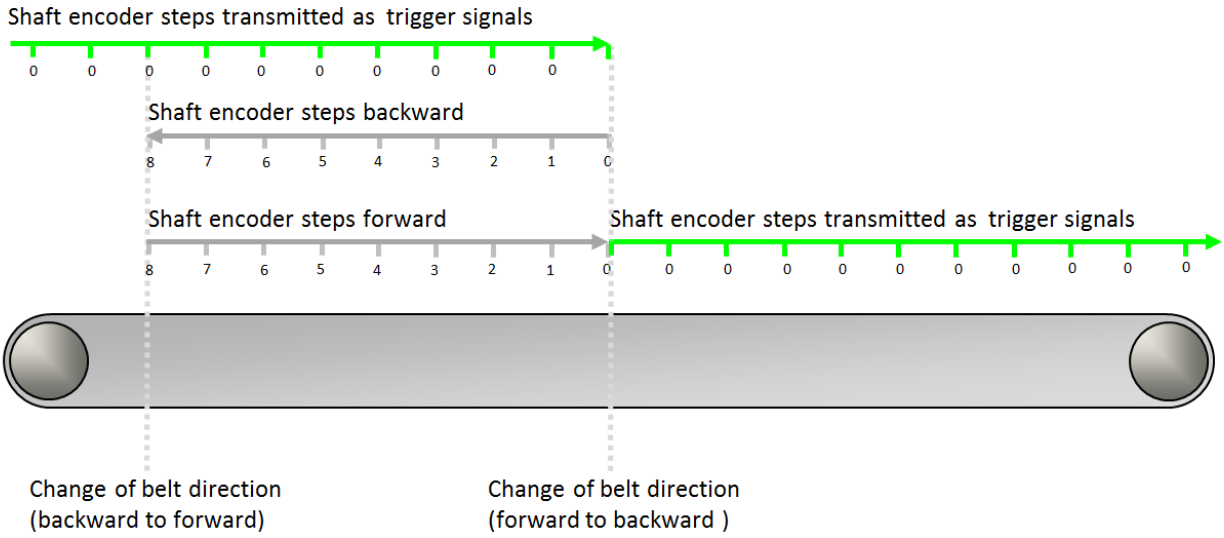
Using this parameter you can read and write the current shaft encoder rollback compensation counter. A compensation value zero indicates that currently no compensation is made. Therefore, you can reset the compensation by writing value zero to this parameter. Any other value will set a new compensation value. By knowing the distance / resolution for every encoder pulse, the compensation distance can be set. Concerning the shaft encoder find some more details in the introduction of Section 9.3, 'Line Trigger Input'.

It is based on a 20bit counter enabling a backward movement of up to 1048575 encoder pulses. An overflow of this value will not occur since it will skip all additional pulses for a compensation state of more than 1048575. By this the count of the rollback compensation is limited by 2 to the power of 20 pulses, what is enough for most applications in practice. As an example we could use a pretty high resolution of 20 pulses per mm, what is already sufficient for a maximum rollback distance of more than 50 meters.

Basic Conditions

If parameter *FG_SHAFTENCODER_COMPENSATION_ENABLE* is set to ON, an internal counter counts the shaft encoder steps the transportation belt moves backwards. This is necessary to be able to compensate the exact number of shaft encoder steps when the transportation belt starts moving forwards again:

Figure 9.7. Shaft Encoder Compensation Enable = ON



The internal counter counts forwards as long as the transportation belt moves backwards. (In figure 9.7, from 0 to 8.)

The internal counter counts backwards while the transportation belt moves forwards. (In figure 9.7, from 8 to 0.)

When the internal counter holds the value 0, the shaft encoder steps are transmitted as trigger signals.

The value the internal counter holds at a given moment is the value of parameter *FG_SHAFTENCODER_COMPENSATION_COUNT*. Only if this value is 0, encoder steps are transmitted as trigger signals. If the value of parameter *FG_SHAFTENCODER_COMPENSATION_COUNT* is not 0, the shaft encoder steps are not transmitted as trigger signals and the value keeps changing with every encoder step until it reaches the value 0 again.

Reading the Parameter

The parameter *FG_SHAFTENCODER_COMPENSATION_COUNT* is a read/write parameter. Therefore, at any given moment, you can always read out the value the counter holds at a given moment.

Defining an Offset

On the other hand, you can always modify the parameter value since you have write access during acquisition. If you need to define an offset to the standard encoder compensation, you can use this parameter to enter the number of steps you need the offset to be.

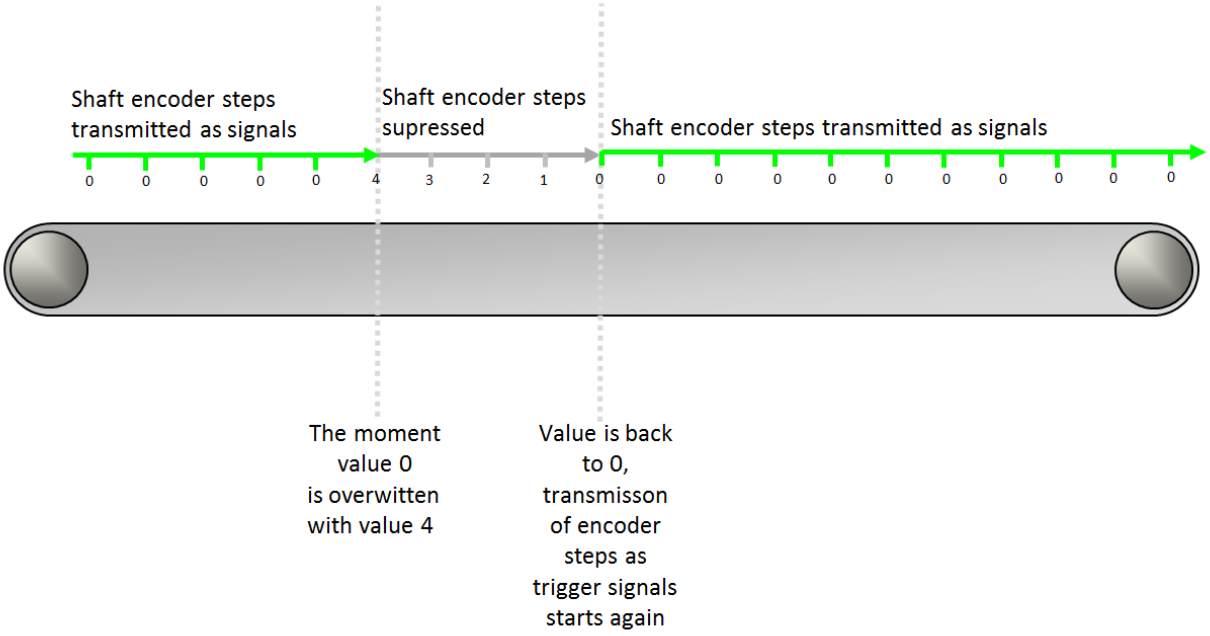
As soon as you enter a value for *FG_SHAFTENCODER_COMPENSATION_COUNT*, this value overwrites the value the parameter holds before.

In the following let's look at some examples for overwriting the current value of *FG_SHAFTENCODER_COMPENSATION_COUNT*:

Example 1:

The transportation belt is moving forward, the shaft encoder steps are transmitted as trigger signals, and the value of *FG_SHAFTENCODER_COMPENSATION_COUNT* is 0. Then, the value 0 of *FG_SHAFTENCODER_COMPENSATION_COUNT* is overwritten by value 4. Result: 4 shaft encoder steps are not transmitted as trigger signals.

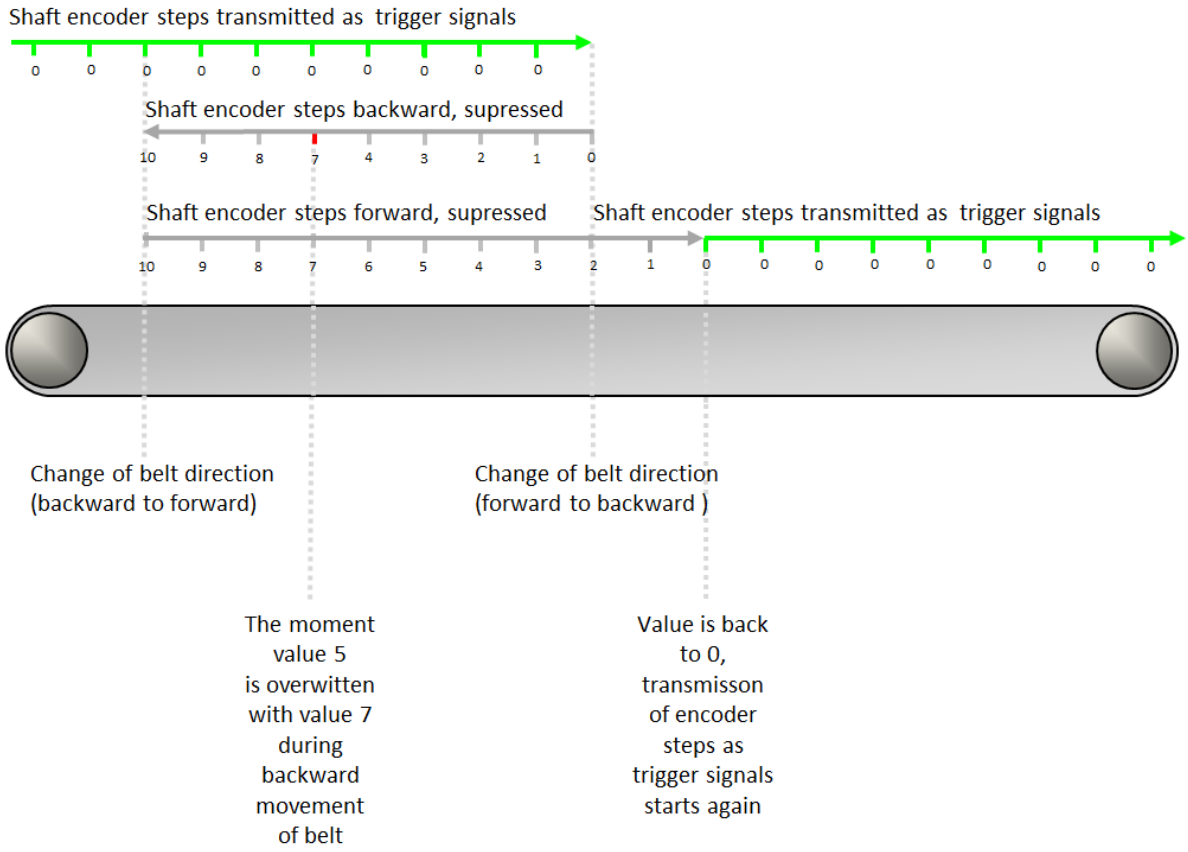
Figure 9.8. Shaft Encoder Compensation Count Example 1



Example 2:

The transportation belt is moving backward, the (backward) shaft encoder steps are suppressed, and the value of *FG_SHAFTENCODER_COMPENSATION_COUNT* is not 0. Then, during backward movement of the transportation belt, the value 5 of *FG_SHAFTENCODER_COMPENSATION_COUNT* is overwritten by value 7. Result: Offset of 2 shaft encoder steps.

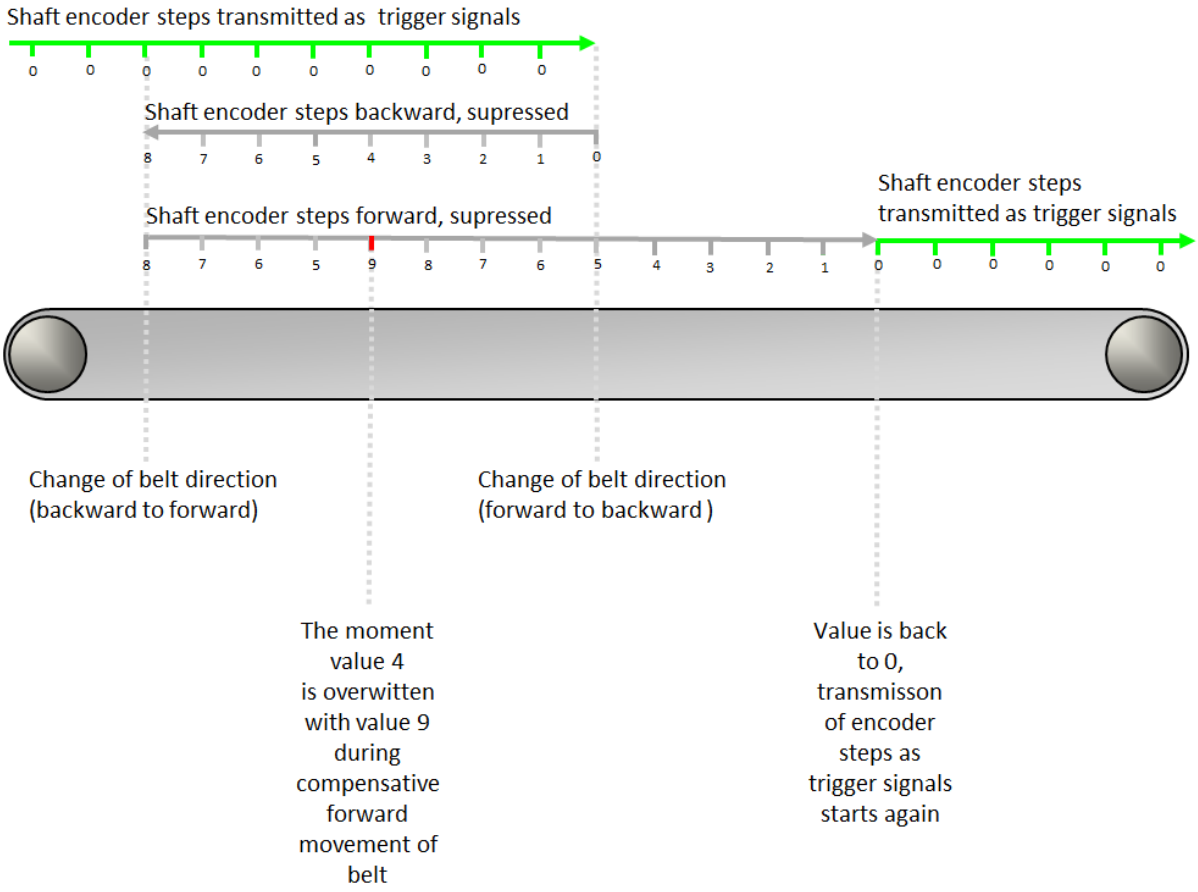
Figure 9.9. Shaft Encoder Compensation Count Example 2



Example 3:

The transportation belt is moving forward during compensation, the (forward) shaft encoder steps are suppressed, and the value of *FG_SHAFTENCODER_COMPENSATION_COUNT* is not 0. Then, during compensative forward movement of the transportation belt, the value 4 of *FG_SHAFTENCODER_COMPENSATION_COUNT* is overwritten with value 9. Result: Offset of 5 shaft encoder steps.

Figure 9.10. Shaft Encoder Compensation Count Example 3



Example 4:

The transportation belt is moving forward during compensation, the (forward) shaft encoder steps are suppressed, and the value of *FG_SHAFTENCODER_COMPENSATION_COUNT* is not 0. Then, during compensative forward movement of the transportation belt, the value 4 of *FG_SHAFTENCODER_COMPENSATION_COUNT* is overwritten with a smaller value, in our case with value 3. Result: Negative offset of -1 shaft encoder step.

Figure 9.11. Shaft Encoder Compensation Count Example 4

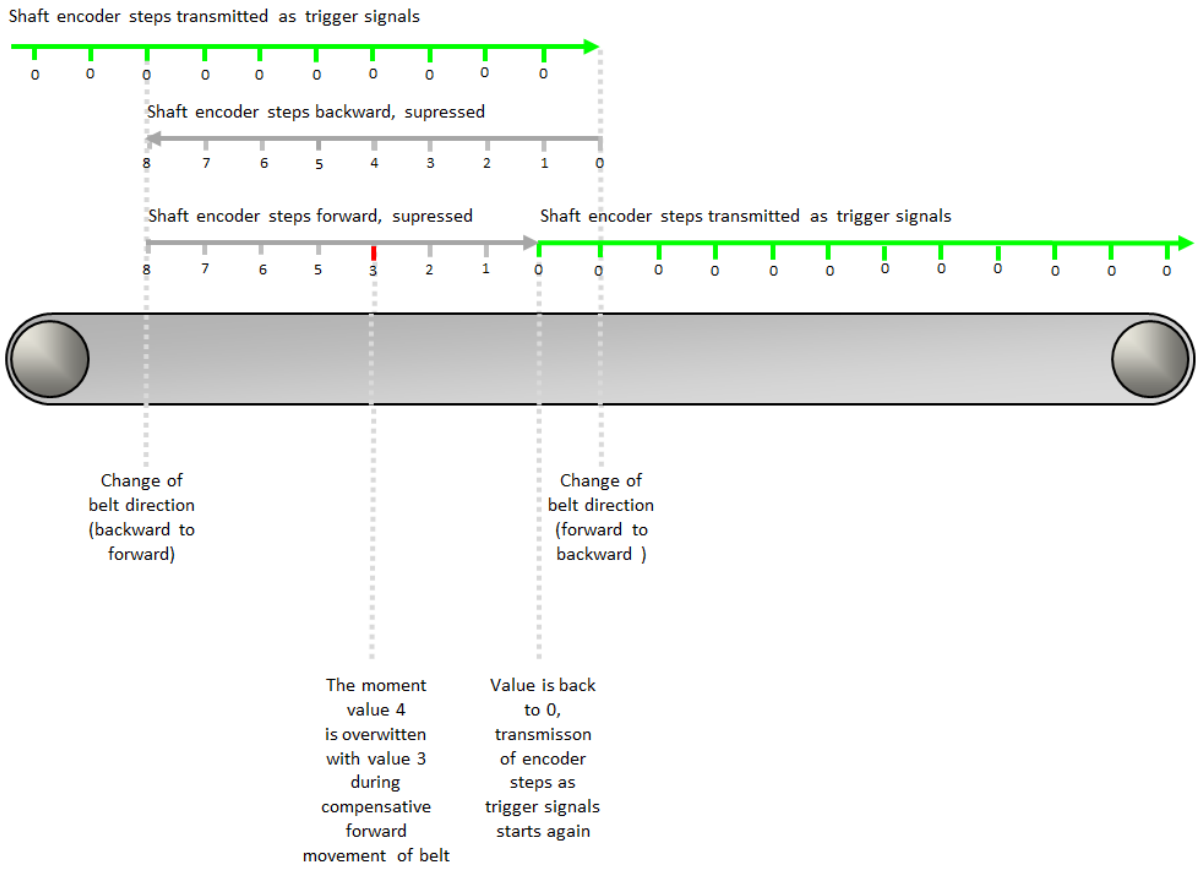


Table 9.14. Parameter properties of FG_SHAFTENCODER_COMPENSATION_COUNT

Property	Value
Name	**FG_SHAFTENCODER_COMPENSATION_COUNT**
Display Name	**Compensation Count**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 1048575** **Stepsize 1**
Default value	**0**
Unit of measure	**pulses**

Example 9.13. Usage of FG_SHAFTENCODER_COMPENSATION_COUNT

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SHAFTENCODER_COMPENSATION_COUNT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SHAFTENCODER_COMPENSATION_COUNT, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.5. ExSync Output

This category includes parameters to specify and parameterize the generated ExSync output signals.

9.5.1. FG_LINEPERIODE

This parameter specifies the period of the ExSync signal. Therefore, it defines the line frequency when using the grabber controlled mode to trigger the connected camera. This period is of interest if a grabber controlled line trigger mode is used; more details for this can be found at *FG_LINETRIGGERMODE*. The line period is not allowed to be shorter than the minimum period - maximum line frequency - being supported by the camera, or in other words:

Please do not try to trigger the camera at a higher frequency than possible.

This maximum frequency is limited by the exposure time and the line scan sensor maximum speed. Please consider the camera manual for more details.

The following equations are mentioned in order to support the setup process if no period for *FG_LINEPERIODE* is mentioned:

- **Frequency**

  The period **T** is the duration of time of one cycle in a repeating event, so the period is the reciprocal of the frequency **f**.

  Equation 9.1. Frequency to Period

$$T = \frac{1}{f}$$

  Equation 9.2. Example: 17.6 kHz to Period

$$\begin{aligned} T &= \frac{1}{F} = \frac{1}{17.6kHz} = \frac{1}{17600Hz} \\ T &= 0.0000568s = 0.0568ms = 56.8\mu s \end{aligned}$$

- **Velocity and Pixel / mm**

  The period **T** is the duration of time of one cycle in a repeating event. At a velocity **v** and a given number **n** of pixels / mm together with the number **n** of pixels / mm being based on the resolution count **r** of the line scan sensor pixels and the width of view **w** in mm the following equations are valid.

  Equation 9.3. Velocity and Resolution to Period

$$\begin{aligned} n &= \frac{r}{w} \\ v &= \frac{distance}{time} \\ f &= v * n \\ T &= \frac{1}{f} \end{aligned}$$

Equation 9.4. Example: v = 53.4 m/min, r = 4096 pixels, w = 19.2 cm Wide Web to Period

$$
\begin{aligned}
n &= \frac{r}{w} = \frac{4096}{19.2cm} = \frac{4096}{192mm} = \frac{21.33}{mm} \\
v &= \frac{distance}{time} = \frac{53.4m}{min} = \frac{53.4m}{60s} = 0.89\frac{m}{s} \\
f &= v * n = 0.89\frac{m}{s} * \frac{21.33}{mm} = 890\frac{mm}{s} * \frac{21.33}{mm} \\
&= \frac{890 * 21.33}{s} = \frac{18983.7}{s} = 18983.7Hz = 18.9837kHz \\
T &= \frac{1}{f} \\
&= \frac{1}{18983.7Hz} = 52.68\mu s
\end{aligned}
$$

Table 9.15. Parameter properties of FG_LINEPERIODE

Property	Value
Name	**FG_LINEPERIODE**
Display Name	**Period**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.512** **Maximum 2097.144** **Stepsize 0.008**
Default value	**200.0**
Unit of measure	**µs**

Example 9.14. Usage of FG_LINEPERIODE

```
int result = 0;
double value = 200.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_LINEPERIODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINEPERIODE, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.5.2. FG_LINEEXPOSURE

This parameter specifies the pulse width of the ExSync signal, which can be used by many cameras to specify the exposure time. It is possible to adjust the exposure time via software, even while grabbing. The value is set in microseconds and may not exceed the period time of the ExSync *FG_LINEPERIODE*. In order to check the polarity simply increase this value and the resulting frame should become brighter. If this behaves in an opposite way check the polarity using *FG_EXSYNCPOLARITY*.

Table 9.16. Parameter properties of FG_LINEEXPOSURE

Property	Value
Name	**FG_LINEEXPOSURE**
Display Name	**Exposure Time**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum** 0.512 **Maximum** 1048.568 **Stepsize** 0.008
Default value	**19.0**
Unit of measure	**µs**

Example 9.15. Usage of FG_LINEEXPOSURE

```
int result = 0;
double value = 19.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_LINEEXPOSURE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINEEXPOSURE, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.5.3. FG_EXSYNCPOLARITY

The parameter adjusts the polarity of the ExSync signal generator. Use Low Active, if the camera opens the shutter on a falling edge, otherwise use High Active. For the mapping of the ExSync signals to the digital outputs check Chapter 7, '*Digital I/O*'.

Table 9.17. Parameter properties of FG_EXSYNCPOLARITY

Property	Value
Name	**FG_EXSYNCPOLARITY**
Display Name	**Polarity**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_LOW** Low Active **FG_HIGH** High Active
Default value	**FG_HIGH**

Example 9.16. Usage of FG_EXSYNCPOLARITY

```
int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_EXSYNCPOLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_EXSYNCPOLARITY, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.5.4. FG_LINETRIGGERDELAY

This parameter specifies the delay between the generated ExSync and ExSync2 signals with respect to an external trigger input. Therefore, the ExSync2 signal is a delayed clone of the ExSync (polarity, period, etc. are the same as for ExSync). For the mapping of the ExSync signals to the digital outputs check Chapter 7, '*Digital I/O*'.

Please note that the line trigger delay needs to be less than the line trigger period. You might need to increase the line period first before increasing the line delay. This constraint also applies for external line trigger modes.

Table 9.18. Parameter properties of FG_LINETRIGGERDELAY

Property	Value
Name	**FG_LINETRIGGERDELAY**
Display Name	**Delay->Exsync**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum** 0.0 **Maximum** 1048.568 **Stepsize** 0.008
Default value	**0.0**
Unit of measure	**µs**

Example 9.17. Usage of FG_LINETRIGGERDELAY

```
int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_LINETRIGGERDELAY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LINETRIGGERDELAY, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 10. Image Trigger / Flash

The image trigger for line-scan cameras is in charge to generate an internal signal called image gate. Lines sent by the camera are only accepted if this image gate is active = open. Therefore, with help of the Image Gate it is possible to define frames by grouping all lines that belong to the same image gate into one frame.

This AcquisitionApplets supports three distinct operation modes of the image trigger:

- Free run

  In free run mode the image gate basically remains active all time. Therefore, all lines sent by the camera are grabbed. Moreover, it cuts the input lines into frames of the height specified by parameter *FG_HEIGHT* of the display module. Also, offsets defined by *FG_YOFFSET* are covered and removed from the camera transfers for each image.

- Async Trigger

  For the external trigger mode of the image trigger, the image gate is inactive = closed until an external trigger signal activates the image gate for *FG_HEIGHT* + *FG_YOFFSET* lines. Therefore, for each external trigger event, the frame grabber records a frame of the specified height.

- Async Trigger Multi Buffer

  For the external trigger mode of the image trigger, the image gate is inactive = closed until an external trigger signal activates the image gate. In contrast to the **ASYNC_TRIGGER** mode, the gate is open for *FG_IMGTRIGGER_ASYNC_HEIGHT* lines while this image is split into smaller chunks of *FG_HEIGHT* lines. Therefore, for each external trigger event, the frame grabber records a frame of a large specified height and split the large image into smaller chunks. The purpose of the mode is to start processing in PC while the image is still recorded.

  The parameter value of *FG_YOFFSET* is without influence in this mode.

- Gated, Trigger

  For the external gated mode of the image trigger, the image gate is active as long as the external trigger source is active, but is becoming inactive when *FG_HEIGHT* + *FG_YOFFSET* lines have been grabbed. Therefore, during an external trigger phase the frame grabber records a frame with a height depending on the duration of active time of the external trigger signal, but is not exceeding an image height of *FG_HEIGHT* + *FG_YOFFSET* lines.

- Gated Multi Buffer, Triggered

  Equal to the 'Gated Trigger' mode, for the 'Gated Multi Buffer Trigger' the image gate is active as long as the external trigger source is active. In contrast, it does not limit the height to *FG_HEIGHT* lines. It will cut the image after *FG_HEIGHT* lines and start a new frame. Thus, for each gate, multiple frames are generated when a gate is active for more lines than defined by *FG_HEIGHT*.

  All images of a generated sequence will have a height of *FG_HEIGHT* lines. However, the last image of each sequence might have a lower number of lines in the image.

  To detect the last image of a sequence in your software. Parameter *FG_IMAGE_TAG* can be used. This parameter is of type unsigned 32 bit integer. The most significant bit i.e. bit 31 includes a flag which is set to one if the respective image is the last image of a multi buffer sequence.

```
uint32_t imageTag = 0;
int returnCode = Fg_getParameterEx(fg, FG_IMAGE_TAG, &imageTag, 0, pmem0, imageNumber);
bool isLastImageOfSequence = imageTagRAW >> 31;
```

  All other bits of parameter *FG_IMAGE_TAG* are fixed to value 0. The image tag parameter does not output the image number as available for older AcquisitionApplets.

Note that the value of parameter *FG_YOFFSET* is not considered if the 'Gated Multi Buffer Trigger' mode is used. An y-offset cannot be set in the applet.

- Camera FVAL Multibuffer

  In this mode, the image trigger of the grabber is disabled. Thus, no external or internal values can define the image height. The applet expects valid FVAL signals from the camera so that the FVAL from the camera defines the image height. It is not possible to gate any camera frames.

  If the FVAL height exceeds the parameterized image height (*FG_HEIGHT*), the applet will cut the incoming frames into chunks of height *FG_HEIGHT*. All images of a generated sequence will have a height of *FG_HEIGHT* lines. However, the last image of each sequence might have a lower number of lines in the image. To detect the last image of a sequence in your software. Parameter *FG_IMAGE_TAG* can be used. This parameter is of type unsigned 32 bit integer. The most significant bit i.e. bit 31 includes a flag which is set to one if the respective image is the last image of a multi buffer sequence.

10.1. FG_IMGTRIGGERMODE

Choose one of the image trigger modes described above. Please make sure that the operation mode of frame grabber and camera is the same.

Table 10.1. Parameter properties of FG_IMGTRIGGERMODE

Property	Value	
Name	**FG_IMGTRIGGERMODE**	
Display Name	**Mode**	
Type	**Enumeration**	
Access policy	**Read/Write**	
Storage policy	**Persistent**	
Allowed values	**FREE_RUN**	Free Run
	ASYNC_TRIGGER	Async External Trigger
	ASYNC_TRIGGER_MULTIFRAME	Async Multi Buffer External Trigger
	ASYNC_GATED	Gated, External Triggered
	ASYNC_GATED_MULTIFRAME	Gated Multi Buffer, External Triggered
	CAMERA_FVAL_MULTIFRAME	Camera FVAL Multiframe
Default value	**FREE_RUN**	

Example 10.1. Usage of FG_IMGTRIGGERMODE

```
int result = 0;
int value = FREE_RUN;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGERMODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGERMODE, &value, 0, type)) < 0) {
    /* error handling */
}
```

10.2. FG_IMGTRIGGERON

The generation of image triggers can be switched on or off by use of this parameter. When the image trigger is disabled and the image trigger is not running in free-run mode, the image acquisition is terminated. If the image trigger is enabled, the acquisition will start immediately.

Table 10.2. Parameter properties of FG_IMGTRIGGERON

Property	Value
Name	**FG_IMGTRIGGERON**
Display Name	**Image Trigger Enable**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_ON** On **FG_OFF** Off
Default value	**FG_ON**

Example 10.2. Usage of FG_IMGTRIGGERON

```
int result = 0;
int value = FG_ON;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGERON, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGERON, &value, 0, type)) < 0) {
    /* error handling */
}
```

10.3. FG_FLASHON

To enable the flash output use this parameter.

For the mapping of the flash signal to the digital IO check Chapter 7, '*Digital I/O*'.

Table 10.3. Parameter properties of FG_FLASHON

Property	Value
Name	**FG_FLASHON**
Display Name	**Flash Enable**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_ON** On **FG_OFF** Off
Default value	**FG_ON**

Example 10.3. Usage of FG_FLASHON

```
int result = 0;
int value = FG_ON;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_FLASHON, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_FLASHON, &value, 0, type)) < 0) {
    /* error handling */
}
```

10.4. FG_IMGTRIGGER_ASYNC_HEIGHT

This parameter only has influence in the image trigger mode *FG_IMGTRIGGERMODE* Async Trigger Multi Frame **ASYNC_TRIGGER_MULTIFRAME**. The value is used to define the image height of the frame after the trigger pulse. Whereas parameter *FG_HEIGHT* defines the chunk height.

If the value of *FG_IMGTRIGGER_ASYNC_HEIGHT* is less than *FG_HEIGHT*, the frame is not split into multiple frames an will result in a smaller output frame.

Table 10.4. Parameter properties of FG_IMGTRIGGER_ASYNC_HEIGHT

Property	Value
Name	**FG_IMGTRIGGER_ASYNC_HEIGHT**
Display Name	**Async Image Trigger Height**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum** 1 **Maximum** 16777216 **Stepsize** 1
Default value	**1024**
Unit of measure	**lines**

Example 10.4. Usage of FG_IMGTRIGGER_ASYNC_HEIGHT

```
int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGER_ASYNC_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGER_ASYNC_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}
```

10.5. FG_IMGTRIGGER_IS_BUSY

The image trigger is busy if the current requested frame from the camera has not been completely transferred to the grabber. This parameter can be used to check if the camera can accept a new software trigger pulse.

Table 10.5. Parameter properties of FG_IMGTRIGGER_IS_BUSY

Property	Value
Name	**FG_IMGTRIGGER_IS_BUSY**
Display Name	**Image Trigger is Busy**
Type	**Enumeration**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**IS_BUSY** Busy Flag is set **IS_NOT_BUSY** Busy Flag is not set

Example 10.5. Usage of FG_IMGTRIGGER_IS_BUSY

```
int result = 0;
int value = IS_NOT_BUSY;
```

```
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGER_IS_BUSY, &value, 0, type)) < 0) {
    /* error handling */
}
```

10.6. Image Trigger Input

This category includes parameters to specify and control the image trigger inputs. The input can either be input pins of the frame grabber's trigger connector or trigger pulses generated by software register accesses.

10.6.1. FG_IMGTRIGGERINSRC

This parameter specifies the signal source, which is used to trigger the image acquisition gate. If a software image trigger has to be used select option **TRGINSOFTWARE**.

Table 10.6. Parameter properties of FG_IMGTRIGGERINSRC

Property	Value	
Name	**FG_IMGTRIGGERINSRC**	
Display Name	**Source**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**TRGINSRC_GPI_0**	GPI Trigger Source 0
	TRGINSRC_GPI_1	GPI Trigger Source 1
	TRGINSRC_GPI_2	GPI Trigger Source 2
	TRGINSRC_GPI_3	GPI Trigger Source 3
	TRGINSRC_GPI_4	GPI Trigger Source 4 (not available on LightBridge)
	TRGINSRC_GPI_5	GPI Trigger Source 5 (not available on LightBridge)
	TRGINSRC_GPI_6	GPI Trigger Source 6 (not available on LightBridge)
	TRGINSRC_GPI_7	GPI Trigger Source 7 (not available on LightBridge)
	TRGINSRC_FRONT_GPI_0	Front GPI Trigger Source 0
	TRGINSRC_FRONT_GPI_1	Front GPI Trigger Source 1
	TRGINSRC_FRONT_GPI_2	Front GPI Trigger Source 2
	TRGINSRC_FRONT_GPI_3	Front GPI Trigger Source 3
	TRGINSOFTWARE	Software Trigger
Default value	**TRGINSRC_GPI_0**	

Example 10.6. Usage of FG_IMGTRIGGERINSRC

```
int result = 0;
int value = TRGINSRC_GPI_0;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGERINSRC, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGERINSRC, &value, 0, type)) < 0) {
    /* error handling */
}
```

10.6.2. FG_IMGTRIGGERINPOLARITY

The parameter defines the polarity of the external input trigger signal.

Table 10.7. Parameter properties of FG_IMGTRIGGERINPOLARITY

Property	Value
Name	**FG_IMGTRIGGERINPOLARITY**
Display Name	**Polarity**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**HIGH_ON_ZERO_LOW** Low Active **HIGH_ON_ZERO_HIGH** High Active
Default value	**HIGH_ACTIVE**

Example 10.7. Usage of FG_IMGTRIGGERINPOLARITY

```
int result = 0;
int value = HIGH_ACTIVE;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGERINPOLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGERINPOLARITY, &value, 0, type)) < 0) {
    /* error handling */
}
```

10.6.3. FG_IMGTRIGGERGATEDELAY

With this parameter, a delay of lines can be configured before the activation of the image gate. This delays the start of the image acquisition. The parameter y-offest (as in free run mode) rejects the first lines from the camera. Delay and y-offset seem to have the same effect, however the difference is, that y-offset doesn't affect the image gate, which is relevant while using the gated line trigger mode.

Table 10.8. Parameter properties of FG_IMGTRIGGERGATEDELAY

Property	Value
Name	**FG_IMGTRIGGERGATEDELAY**
Display Name	**Gate Delay**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum** **0** **Maximum** **65535** **Stepsize** **1**
Default value	**0**
Unit of measure	**lines**

Example 10.8. Usage of FG_IMGTRIGGERGATEDELAY

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGERGATEDELAY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGERGATEDELAY, &value, 0, type)) < 0) {
```

```
    /* error handling */
}
```

10.6.4. FG_IMGTRIGGERDEBOUNCING

This parameter specifies the debouncing time. This is the time for which the input image trigger signal must keep the same value to be detected as such. Fast signal changes within the debounce time will be filtered out.

Table 10.9. Parameter properties of FG_IMGTRIGGERDEBOUNCING

Property	Value
Name	**FG_IMGTRIGGERDEBOUNCING**
Display Name	**Debouncing Time**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.008** **Maximum 65.0** **Stepsize 0.008**
Default value	**0.112**
Unit of measure	**µs**

Example 10.9. Usage of FG_IMGTRIGGERDEBOUNCING

```
int result = 0;
double value = 0.112;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_IMGTRIGGERDEBOUNCING, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMGTRIGGERDEBOUNCING, &value, 0, type)) < 0) {
    /* error handling */
}
```

10.6.5. FG_STROBEPULSEDELAY

This parameter specifies the delay of the generated flash signal with respect to an external trigger input. Therefore, it is possible to synchronize the flash to the external trigger input. The delay is set in image line ticks.

Table 10.10. Parameter properties of FG_STROBEPULSEDELAY

Property	Value
Name	**FG_STROBEPULSEDELAY**
Display Name	**Flash Delay->Image Trigger**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 65535** **Stepsize 1**
Default value	**0**
Unit of measure	**lines**

Example 10.10. Usage of FG_STROBEPULSEDELAY

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_STROBEPULSEDELAY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_STROBEPULSEDELAY, &value, 0, type)) < 0) {
    /* error handling */
}
```

10.6.6. Flash

10.6.6.1. FG_FLASH_POLARITY

The polarity of the generated flash signal can be changed with this parameter. For the mapping of the flash signal to the digital outputs check Chapter 7, '*Digital I/O*'.

Table 10.11. Parameter properties of FG_FLASH_POLARITY

Property	Value
Name	**FG_FLASH_POLARITY**
Display Name	**Flash Polarity**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_LOW** Low Active **FG_HIGH** High Active
Default value	**FG_HIGH**

Example 10.11. Usage of FG_FLASH_POLARITY

```
int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_FLASH_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_FLASH_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}
```

10.6.7. Software Trigger

For the image trigger it is possible to use a software generated trigger signal to replace the external trigger input.

The software trigger control modules allows the to either generate a software trigger pulse or allows to set the state of the software trigger signal to generate a gate i.e. for gated image trigger mode.

To enable the software trigger set parameter *FG_IMGTRIGGERINSRC* to software trigger.

10.6.7.1. FG_SENDSOFTWARETRIGGER

A software trigger pulse can be sent by use of this parameter.

Table 10.12. Parameter properties of FG_SENDSOFTWARETRIGGER

Property	Value
Name	**FG_SENDSOFTWARETRIGGER**
Display Name	**Software Trigger Pulse**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**FG_APPLY** Apply
Default value	**FG_APPLY**

Example 10.12. Usage of FG_SENDSOFTWARETRIGGER

```
int result = 0;
int value = FG_APPLY;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SENDSOFTWARETRIGGER, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SENDSOFTWARETRIGGER, &value, 0, type)) < 0) {
    /* error handling */
}
```

10.6.7.2. FG_SETSOFTWARETRIGGER

The software trigger state can be set to zero = low or one = high.

Table 10.13. Parameter properties of FG_SETSOFTWARETRIGGER

Property	Value
Name	**FG_SETSOFTWARETRIGGER**
Display Name	**Software Trigger State**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_LOW** Low Active **FG_HIGH** High Active
Default value	

Example 10.13. Usage of FG_SETSOFTWARETRIGGER

```
int result = 0;
int value = FG_ZERO;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SETSOFTWARETRIGGER, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SETSOFTWARETRIGGER, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 11. Signal Analyzer

The signal analyzer module computes some information on a signal source. These are

- Pulse Count
- Period (current, min, max)
- Difference between two pulse counters

The module is used to detect unexpected behaviors of the trigger system. For example a bouncing encode signal resulting in overtriggering of the camera. Another example is the detection of trigger lost signals or corrupted camera data which can result in extra lines.

Simply select the analyzer source signal and polarity. The measurement values can be obtained using read-only parameters. All measurements can be cleared synchronously.

Note that the module is available only once for the applet. All cameras share the same module. The camera/DMA index in the setParameter and getParameter functions has no influence.

11.1. FG_SIGNAL_ANALYZER_0_SOURCE et al.

Note

This description applies also to the following parameters: FG_SIGNAL_ANALYZER_1_SOURCE

Select the source signal for the trigger analyzer. For further explanation of the available sources see Chapter 7, '*Digital I/O*'. In addition, the line/frame start/end pulses can be used as signal sources, too.

Table 11.1. Parameter properties of FG_SIGNAL_ANALYZER_0_SOURCE

Property	Value	
Name	**FG_SIGNAL_ANALYZER_0_SOURCE**	
Display Name	**Signal Analyzer 0 Source**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**GND**	Gnd
	VCC	Vcc
	FG_SIGNAL_CAM0_EXSYNC	Cam0 Exsync
	FG_SIGNAL_CAM0_EXSYNC2	Cam0 Exsync2
	FG_SIGNAL_CAM0_FLASH	Cam0 Flash
	FG_SIGNAL_CAM0_LVAL	Cam0 LVAL
	FG_SIGNAL_CAM0_FVAL	Cam0 FVAL
	FG_SIGNAL_CAM0_LINE_START	Cam0 Line Transfer Start
	FG_SIGNAL_CAM0_LINE_END	Cam0 Line Transfer End
	FG_SIGNAL_CAM0_FRAME_START	Cam0 Frame Transfer Start
	FG_SIGNAL_CAM0_FRAME_END	Cam0 Frame Transfer End
	FG_SIGNAL_GPI_0	GPI 0
	FG_SIGNAL_GPI_1	GPI 1
	FG_SIGNAL_GPI_2	GPI 2
	FG_SIGNAL_GPI_3	GPI 3
	FG_SIGNAL_GPI_4	GPI 4
	FG_SIGNAL_GPI_5	GPI 5
	FG_SIGNAL_GPI_6	GPI 6
	FG_SIGNAL_GPI_7	GPI 7
	FG_SIGNAL_FRONT_GPI_0	Front GPI 0
	FG_SIGNAL_FRONT_GPI_1	Front GPI 1
	FG_SIGNAL_FRONT_GPI_2	Front GPI 2
	FG_SIGNAL_FRONT_GPI_3	Front GPI 3
Default value	**FG_SIGNAL_CAM0_EXSYNC**	

Example 11.1. Usage of FG_SIGNAL_ANALYZER_0_SOURCE

```
int result = 0;
int value = FG_SIGNAL_CAM0_EXSYNC;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SIGNAL_ANALYZER_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_0_SOURCE, &value, 0, type)) < 0) {
    /* error handling */
}
```

11.2. FG_SIGNAL_ANALYZER_0_POLARITY et al.

Note

This description applies also to the following parameters: FG_SIGNAL_ANALYZER_1_POLARITY

Select the polarity for the signal analyzer of the selected source. With this parameter you can invert the signal. The signal analyzer module will only measure on rising edges.

Table 11.2. Parameter properties of FG_SIGNAL_ANALYZER_0_POLARITY

Property	Value
Name	**FG_SIGNAL_ANALYZER_0_POLARITY**
Display Name	**Signal Analyzer 0 Polarity**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_LOW** Low Active **FG_HIGH** High Active
Default value	**FG_HIGH**

Example 11.2. Usage of FG_SIGNAL_ANALYZER_0_POLARITY

```
int result = 0;
int value = FG_HIGH;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SIGNAL_ANALYZER_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_0_POLARITY, &value, 0, type)) < 0) {
    /* error handling */
}
```

11.3. FG_SIGNAL_ANALYZER_0_PERIOD_CURRENT et al.

Note

This description applies also to the following parameters: FG_SIGNAL_ANALYZER_1_PERIOD_CURRENT

This read-only parameter returns the last measured period of the selected signal source. Keep in mind that the module requires two rising edges to obtain a measurement result. Selecting a new source or changing the acquisition states can result in very long periods.

Table 11.3. Parameter properties of FG_SIGNAL_ANALYZER_0_PERIOD_CURRENT

Property	Value
Name	**FG_SIGNAL_ANALYZER_0_PERIOD_CURRENT**
Display Name	**Signal Analyzer 0 Current Period**
Type	**Double**
Access policy	**Read-Only**
Storage policy	**Persistent**
Allowed values	**Minimum** 0.008 **Maximum** 3.435973836E7 **Stepsize** 0.008
Unit of measure	**µs**

Example 11.3. Usage of FG_SIGNAL_ANALYZER_0_PERIOD_CURRENT

```
int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_0_PERIOD_CURRENT, &value, 0, type)) < 0) {
    /* error handling */
}
```

11.4. FG_SIGNAL_ANALYZER_0_PERIOD_MAX et al.

Note

This description applies also to the following parameters: FG_SIGNAL_ANALYZER_1_PERIOD_MAX

This read-only parameter returns the maximum measured period after the last reset. Keep in mind that selecting a new source or changing the acquisition states can result in very long periods.

Table 11.4. Parameter properties of FG_SIGNAL_ANALYZER_0_PERIOD_MAX

Property	Value
Name	**FG_SIGNAL_ANALYZER_0_PERIOD_MAX**
Display Name	**Signal Analyzer 0 Max Period**
Type	**Double**
Access policy	**Read-Only**
Storage policy	**Persistent**
Allowed values	**Minimum 0.008** **Maximum 3.435973836E7** **Stepsize 0.008**
Unit of measure	**µs**

Example 11.4. Usage of FG_SIGNAL_ANALYZER_0_PERIOD_MAX

```
int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_0_PERIOD_MAX, &value, 0, type)) < 0) {
    /* error handling */
}
```

11.5. FG_SIGNAL_ANALYZER_0_PERIOD_MIN et al.

Note

This description applies also to the following parameters: FG_SIGNAL_ANALYZER_1_PERIOD_MIN

This read-only parameter returns the minimum measured period after the last reset.

Table 11.5. Parameter properties of FG_SIGNAL_ANALYZER_0_PERIOD_MIN

Property	Value
Name	**FG_SIGNAL_ANALYZER_0_PERIOD_MIN**
Display Name	**Signal Analyzer 0 Min Period**
Type	**Double**
Access policy	**Read-Only**
Storage policy	**Persistent**
Allowed values	**Minimum 0.008** **Maximum 3.435973836E7** **Stepsize 0.008**
Unit of measure	**µs**

Example 11.5. Usage of FG_SIGNAL_ANALYZER_0_PERIOD_MIN

```
int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_0_PERIOD_MIN, &value, 0, type)) < 0) {
    /* error handling */
}
```

11.6. FG_SIGNAL_ANALYZER_0_PULSE_COUNT et al.

Note

This description applies also to the following parameters: FG_SIGNAL_ANALYZER_1_PULSE_COUNT

Returns the counter value of the selected source. For each rising edge the counter is increased. This, after the first pulse, the counter value will be one. On counter overflow, it will start from 0 again.

Table 11.6. Parameter properties of FG_SIGNAL_ANALYZER_0_PULSE_COUNT

Property	Value
Name	**FG_SIGNAL_ANALYZER_0_PULSE_COUNT**
Display Name	**Signal Analyzer 0 Pulse Count**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 4294967295** **Stepsize 1**
Unit of measure	**µs**

Example 11.6. Usage of FG_SIGNAL_ANALYZER_0_PULSE_COUNT

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_0_PULSE_COUNT, &value, 0, type)) < 0) {
    /* error handling */
}
```

11.7. FG_SIGNAL_ANALYZER_PULSE_COUNT_DIFFERENCE

Use this read only parameter to check the difference of the signal analyzer 0 and 1 pulse counter values (Analyzer 0 - Analyzer 1 value). This can be used to check for trigger lost signals if analyzer 0 will count the exsyc pulses and analyzer 1 the returned camera lines. In this case the difference is between 0 and 1 for single line cameras with no extra delay. If the difference exceeds 1, the camera did not return a line for all trigger pulses i.e. a trigger is lost or ignored due to overtriggering. If the difference is less than 0 an additional camera line was generated and received by the frame grabber. The reason for this can be a noisy trigger cable which added extra spikes or a corrupted data transfer which split the data into several parts.

Table 11.7. Parameter properties of FG_SIGNAL_ANALYZER_PULSE_COUNT_DIFFERENCE

Property	Value
Name	**FG_SIGNAL_ANALYZER_PULSE_COUNT_DIFFERENCE**
Display Name	**Signal Analyzer Pulse Count Difference**
Type	**Signed Integer (64 Bit)**
Access policy	**Read-Only**
Storage policy	**Persistent**
Allowed values	**Minimum** -4294967296 **Maximum** 4294967295 **Stepsize** 1
Unit of measure	**µs**

Example 11.7. Usage of FG_SIGNAL_ANALYZER_PULSE_COUNT_DIFFERENCE

```
int result = 0;
int64_t value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_INT64_T;

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_PULSE_COUNT_DIFFERENCE, &value, 0, type)) < 0) {
    /* error handling */
}
```

11.8. FG_SIGNAL_ANALYZER_CLEAR

To clear all signal analyzer measurement results and counters use this parameter. All counters will be reset synchronously and are ready to restart immediately.

Table 11.8. Parameter properties of FG_SIGNAL_ANALYZER_CLEAR

Property	Value
Name	**FG_SIGNAL_ANALYZER_CLEAR**
Display Name	**Signal Analyzer Clear**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_APPLY** Apply
Default value	**FG_APPLY**

Example 11.8. Usage of FG_SIGNAL_ANALYZER_CLEAR

```
int result = 0;
int value = FG_APPLY;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_SIGNAL_ANALYZER_CLEAR, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_SIGNAL_ANALYZER_CLEAR, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 12. Overflow

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 frame grabber. However, the transfer bandwidth to the PC via DMA channel can vary caused by interrupts, other hardware and the current CPU load. Furthermore, if operated in **selective mode**, it is possible to queue buffer slower than the camera offers new images and therefore generate an overflow condition on the frame grabber. Also, the camera frame rate can vary due to an fluctuating trigger. For these cases, the applet is equipped with a memory to buffer the input frames. The fill level of the buffer can be obtained by reading from parameter *FG_FILLLEVEL*.

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

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

- Corrupted Frames:

  The transfer of a current frame is interrupted by an overflow. This means, the first pixels or lines of the frame were transfered into the buffer, but not the full frame. The output of the applet i.e. the DMA transfer will be shorter. The output image will not have it's full height.

- Lost Frames:

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

A way to detect the overflows is to read parameter *FG_OVERFLOW* or check for event *FG_OVERFLOW_CAM0*. Reading from the parameter will provide information about an overflow condition. As soon as the parameter is read, it will reset. Using the parameter an overflow condition can be detect, but it is not possible to obtain the exact image number and the moment. For this, the overflow event can be used.

12.1. FG_FILLLEVEL

The fill-level of the frame grabber 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 12.1. Parameter properties of FG_FILLLEVEL

Property	Value
Name	**FG_FILLLEVEL**
Display Name	**Buffer fill level**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 100** **Stepsize 1**
Unit of measure	**%**

Example 12.1. Usage of FG_FILLLEVEL

```
int result = 0;
```

```
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_FILLLEVEL, &value, 0, type)) < 0) {
    /* error handling */
}
```

12.2. FG_OVERFLOW

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

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

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

Table 12.2. Parameter properties of FG_OVERFLOW

Property	Value
Name	**FG_OVERFLOW**
Display Name	**Buffer overflow**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 1** **Stepsize 1**

Example 12.2. Usage of FG_OVERFLOW

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_OVERFLOW, &value, 0, type)) < 0) {
    /* error handling */
}
```

12.3. Events

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

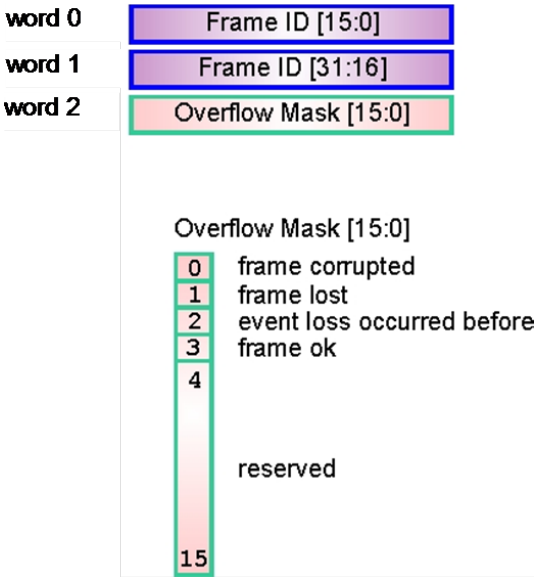
The Basler Framegrabber SDK and pylon SDK via GenTL enables an application to get these event notifications about certain state changes at the data flow from camera to RAM and the image and trigger processing as well. Please consult the Basler Framegrabber SDK, pylon SDK or GenTL documentation for more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

12.3.1. FG_OVERFLOW_CAM0

Overflow events are generated for each corrupted or lost frame. In contrast to the other events presented in this document, the overflow event transports data, namely the type of overflow, the image number and the

timestamp. The following figure illustrates the event data. Data is included in a 64 Bit data packet. The first 32 Bit include the frame number. Bits 32 to 47 include an overflow mask.

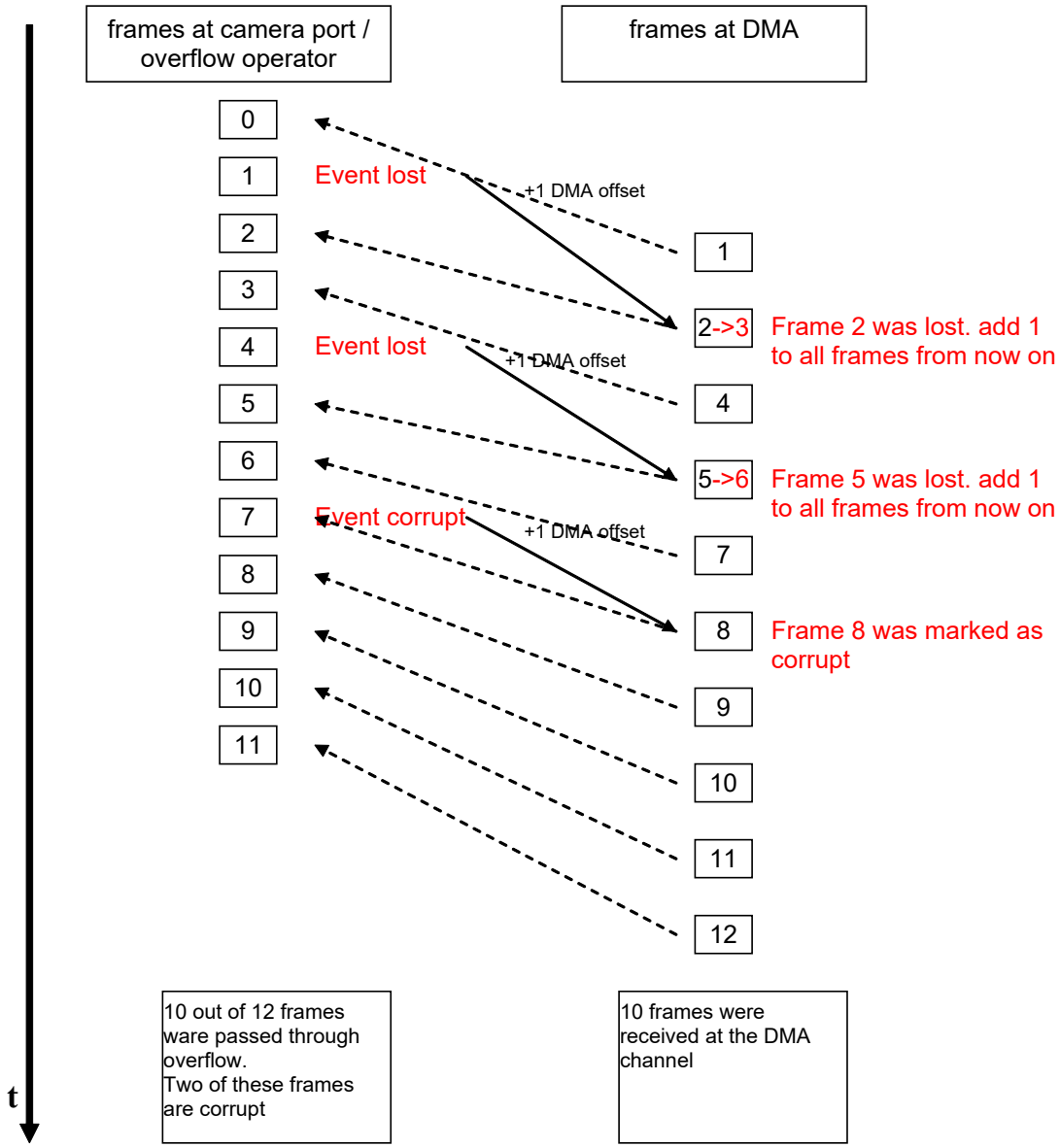
Figure 12.1. Illustration of Overflow Data Packet



Note that the frame number is reset on acquisition start. Also note that the first frame will have frame number zero, while a DMA transfer starts with frame number one. The frame number is a 32 Bit value. If it's maximum is reached, it will start from zero again. Keep in mind that on a 64 Bit runtime, the DMA transfer number will be a 64 Bit value. If the frame corrupted flag is set, the frame with the frame number in the event is corrupted i.e. it will not have it's full length but is still transfered via DMA channel. If the frame lost flag is set, the frame with the frame number in the event was fully discarded. No DMA transfer will exist for this frame. The corrupted frame flag and the frame lost flag will never occur for the same event. The flag "event loss occured before" is an additional security mechanism. It means that an event has been lost. This can only happen at very high event rates and should not happen under normal conditions.

The analysis of the overflow events depends on the user requirements. In the following, an example is shown on how to ensure the integrity if the DMA data by analyzing the events and DMA transfers.

Figure 12.2. Analysis of Overflow Data



In the example, two frames got lost and one is marked as corrupted. As the events are not synchronous with the DMA transfers, for analysis a software queue (push and pull) is required to allocate the events to the DMA transfers.

Chapter 13. Image Selector

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

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

1. Grabber 0:
   - *FG_IMG_SELECT_PERIOD* = 2

     FG_IMG_SELECT = 0
2. Grabber 1:
   - *FG_IMG_SELECT_PERIOD* = 2

     FG_IMG_SELECT = 1

Ensure that both grabbers are used synchronously. This is possible with a triggered camera. To do so, initialize and configure both frame grabbers. Configure the camera for external trigger and the trigger system of master grabber which is directly connected to the camera.

13.1. FG_IMG_SELECT_PERIOD

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

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

Table 13.1. Parameter properties of FG_IMG_SELECT_PERIOD

Property	Value
Name	**FG_IMG_SELECT_PERIOD**
Display Name	**Image select period**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 256** **Stepsize 1**
Default value	**1**
Unit of measure	**image**

Example 13.1. Usage of FG_IMG_SELECT_PERIOD

```
int result = 0;
unsigned int value = 1;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMG_SELECT_PERIOD, &value, 0, type)) < 0) {
    /* error handling */
}
```

```
if ((result = Fg_getParameterWithType(fg, FG_IMG_SELECT_PERIOD, &value, 0, type)) < 0) {
    /* error handling */
}
```

13.2. FG_IMG_SELECT

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

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

Table 13.2. Parameter properties of FG_IMG_SELECT

Property	Value
Name	**FG_IMG_SELECT**
Display Name	**Image select**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 255** **Stepsize 1**
Default value	**0**
Unit of measure	**image**

Example 13.2. Usage of FG_IMG_SELECT

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_IMG_SELECT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_IMG_SELECT, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 14. Lookup Table

This Acquisition Applet includes a full resolution lookup table (LUT). Settings are applied to the acquired images just before transferring them to the host PC. Thus, it is the last pre-processing step on the frame grabber.

A lookup table includes one entry for every allowed input pixel value. The pixel value will be replaced by the value of the lookup table element. In other words, a new value is assigned to each pixel value. This can be used for image quality enhancements such as an added offset, a gain factor or gamma correction which can be performed by use of the processing module of this applet in a convenient way (see Module Chapter 15, '*Processing*'). The lookup table can also be loaded with custom values. Application areas are custom image enhancements or correct pixel classifications.

This applet is processing data with a resolution of 12 bits. Thus, the lookup table has 12 input bits i.e. pixel values can be in the range [0, 4095]. For each of these 4096 elements, a table entry exists containing a new output value. The new values are in the range from 0 to 4095.

In the following the parameters to use the lookup table are explained. Parameter *FG_LUT_TYPE* is important to be set correctly as it defines the lookup table operation mode.

14.1. FG_LUT_TYPE

There exist two basic possibilities to use and configure the lookup table. One possibility is to use the internal processor which allows a convenient way to improve the image quality using parameters such as offset, gain and gamma. Check category Chapter 15, '*Processing*' for more detailed documentation. Set this parameter to **LUT_TYPE_PROCESSING** to use the processor.

The second possibility to use the lookup table is to load a file containing custom values to the lookup table. Set the parameter to **LUT_TYPE_CUSTOM** to enable the possibility to load a custom file with lookup table entries.

Beside these two possibilities it is always possible to directly write to the lookup table entries using the field parameter *FG_LUT_VALUE*. The use of these parameters will overwrite the settings made with the processor or the custom input file. Vice versa, changing a processing parameter or loading a custom lookup table file, will overwrite the settings made by the field parameters.

Table 14.1. Parameter properties of FG_LUT_TYPE

Property	Value	
Name	**FG_LUT_TYPE**	
Display Name	**Type**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**LUT_TYPE_PROCESSING** **LUT_TYPE_CUSTOM**	Processor User file
Default value	**LUT_TYPE_PROCESSING**	

Example 14.1. Usage of FG_LUT_TYPE

```
int result = 0;
int value = LUT_TYPE_PROCESSING;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_LUT_TYPE, &value, 0, type)) < 0) {
    /* error handling */
}
```

```
if ((result = Fg_getParameterWithType(fg, FG_LUT_TYPE, &value, 0, type)) < 0) {
    /* error handling */
}
```

14.2. FG_LUT_VALUE

Table 14.2. Parameter properties of FG_LUT_VALUE

Property	Value
Name	**FG_LUT_VALUE**
Display Name	**LUT Values**
Type	**Unsigned Integer Field**
Field Size	**4096**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Default value	**0**

Example 14.2. Usage of FG_LUT_VALUE

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

for (unsigned int i = 0; i < 4096; ++i)
{
    access.index = i;
    access.value = 0;

    if ((result = Fg_setParameterWithType(fg, FG_LUT_VALUE, &access, 0, type)) < 0) {
        /* error handling */
    }

    if ((result = Fg_getParameterWithType(fg, FG_LUT_VALUE, &access, 0, type)) < 0) {
        /* error handling */
    }
}
```

14.3. FG_LUT_CUSTOM_FILE

If parameter *FG_LUT_TYPE* is set to **LUT_TYPE_CUSTOM**, the according path and filename to the file containing the custom lookup table entries can be set here. If the file is valid, the file values will be loaded to the lookup table. If the file is invalid, the call to this parameter will return an error.

A convenient way of getting a draft file, is to save the current lookup table settings to file using parameter *FG_LUT_SAVE_FILE*.

Please make sure to activate the Type of LUT *FG_LUT_TYPE* to "UserFile"/**LUT_TYPE_CUSTOM** in order to make the changes and file names taking effect.

This section describes the file formats which are in use to fill the so called look-up tables (LUT). The purpose of a LUT is a transformation of pixel values from a input (source) image to the pixel values of an output image. This transformation is done by a kind of table, which contains the assignment between these pixel values (input pixel values - output pixel values). Basically the LUT is defined for gray format and color formats as well. When defining a LUT for color formats, the definition of tables has to be done for each color component. The LUT file format consists of 2 parts:

- Header section containing control and description information.

- Main section containing the assignment table for transforming pixel values form a source (input) image to a destination (output) image.

The following example shows how a grey scale lookup table description could look like:

```
# Lut data file v1.1
id=3;
nrOfElements=4096;
format=0;
number=0;
0,0;
1,1;
2,2;
3,3;
4,4;
5,5;
6,6;
…
4095,4095;
```

General Properties:

- File format extension should be ".lut"
- LUT file format is an ASCII file format consisting of multiple lines of data.
- Lines are defined by a line separator a <CR> <LF> line feed (0x3D 0x0D 0x0A).
- Lines consist of key / value pairs. Key and value are separated by "=". The value has to be followed by a semicolon ; (0x3B)
- Formats consist of header data, containing control information and the assignment table for a specific color component (gray / red, green, blue).
- Basically the LUT file color format follows the same rules as the gray image format. In addition, due to the fact, that each color component can has its own transformation, the definitions are repeated for each color component.

The following example shows how a color scale lookup table description could look like:

```
# Lut data file v1.1
[red]
id=0;
nrOfElements=256;
format=0;
number=0;
0,0;
1,1;
..
255,255;
[green]
id=1;
nrOfElements=256;
format=0;
number=0;
0,0;
1,1;
..
255,255;
[blue]
id=2;
nrOfElements=256;
format=0;
number=0;
0,0;
1,1;
..
255,255;
```

A more detailed explanation of the lookup table file format can be found in the Basler Framegrabber API manual.

Table 14.3. Parameter properties of FG_LUT_CUSTOM_FILE

Property	Value
Name	**FG_LUT_CUSTOM_FILE**
Display Name	**Load File**
Type	**String**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Default value	**""**

Example 14.3. Usage of FG_LUT_CUSTOM_FILE

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_setParameterWithType(fg, FG_LUT_CUSTOM_FILE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LUT_CUSTOM_FILE, &value, 0, type)) < 0) {
    /* error handling */
}
```

14.4. FG_LUT_SAVE_FILE

To save the current lookup table configuration to a file, write the according output filename to this parameter. Keep in mind that you need to have full write access to the specified path.

Writing the current lookup table settings to a file is also a convenient way to exploit the settings made by the processor. Moreover, you will get a draft version of the lookup table file format. The values in the output file can directly be used to be loaded to the lookup table again using parameter *FG_LUT_CUSTOM_FILE*.

Table 14.4. Parameter properties of FG_LUT_SAVE_FILE

Property	Value
Name	**FG_LUT_SAVE_FILE**
Display Name	**Save File**
Type	**String**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Default value	**""**

Example 14.4. Usage of FG_LUT_SAVE_FILE

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_setParameterWithType(fg, FG_LUT_SAVE_FILE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_LUT_SAVE_FILE, &value, 0, type)) < 0) {
    /* error handling */
}
```

14.5. Applet Properties

In the following, some properties of the lookup table implementation are listed.

14.5.1. FG_LUT_IMPLEMENTATION_TYPE

In this applet, a full lookup table is implemented and can be setup in a custom way. By default a linear representation is performed.

Table 14.5. Parameter properties of FG_LUT_IMPLEMENTATION_TYPE

Property	Value
Name	**FG_LUT_IMPLEMENTATION_TYPE**
Display Name	**LUT Implementation Type**
Type	**Enumeration**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**LUT_IMPLEMENTATION_FULL_LUT** Full LUT **LUT_IMPLEMENTATION_KNEELUT** KneeLUT

Example 14.5. Usage of FG_LUT_IMPLEMENTATION_TYPE

```
int result = 0;
int value = LUT_IMPLEMENTATION_FULL_LUT;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_LUT_IMPLEMENTATION_TYPE, &value, 0, type)) < 0) {
    /* error handling */
}
```

14.5.2. FG_LUT_IN_BITS

This applet is using 12 lookup table input bits.

Table 14.6. Parameter properties of FG_LUT_IN_BITS

Property	Value
Name	**FG_LUT_IN_BITS**
Display Name	**LUT In Bits**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Unit of measure	**bit**

Example 14.6. Usage of FG_LUT_IN_BITS

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_LUT_IN_BITS, &value, 0, type)) < 0) {
    /* error handling */
}
```

14.5.3. FG_LUT_OUT_BITS

This applet is using 12 lookup table output bits.

Table 14.7. Parameter properties of FG_LUT_OUT_BITS

Property	Value
Name	**FG_LUT_OUT_BITS**
Display Name	**LUT Out Bits**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Unit of measure	**bit**

Example 14.7. Usage of FG_LUT_OUT_BITS

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_LUT_OUT_BITS, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 15. Processing

A convenient way to improve the image quality are the processing parameters. Using these parameters an offset, gain and gamma correction can be performed. Moreover, the image can be inverted.

Processor Activation

The processing parameters use the lookup table for determination of the correction values. For activation of the processing parameters, set *FG_LUT_TYPE* of category lookup table to **LUT_TYPE_PROCESSING** . Otherwise, parameter changes will have no effect.

All transformations apply in the following order:

1. Offset Correction, range [-1.0, +1.0], identity = 0
2. Gain Correction, range [0, 2^12[, identity = 1.0
3. Gamma Correction, range]0, inf], identity = 1.0
4. Invert, identity = 'off'

In this applet, a full lookup table with m = 12 input bits and n = 12 outputs bits is used to perform the corrections. Values are determined by

Equation 15.1. LUT Processor without Inversion

$$Output(x) = \left[\left[gain * \left(\frac{x}{2^{12}-1} + offset\right)\right]^{\frac{1}{gamma}}\right] * \left(2^{12}-1\right).$$

If the inversion is used, output values are determined by

Equation 15.2. LUT Processor with Inversion

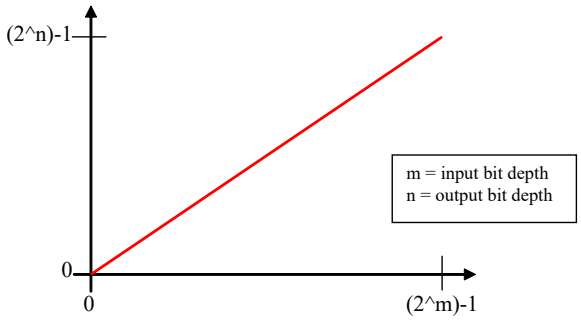
$$Output(x) = 2^{12} - 1 - \left[\left[gain * \left(\frac{x}{2^{12}-1} + offset\right)\right]^{\frac{1}{gamma}}\right] * \left(2^{12}-1\right),$$

where x represents the input pixel value i.e. is in the range from 0 to 2^12 - 1. If the determined output value is less than 0, it will be set to 0. If the determined output value is greater than 2^12 - 1 it is set to 2^12 - 1.

Color applets process each color component separately using the same processing parameters for each component.

If no parameters are changed, i.e. they are set to identity, the output values will be equal to the input values as shown in the figure below. In the following, you will find detailed explanations for all processing parameters.

Figure 15.1. Lookup Table Processing: Identity



15.1. FG_PROCESSING_OFFSET

The offset is a relative value added to each pixel, which leads to a behavior similar to a brightness controller. A relative offset means, that e. g. 0.5 adds half of the total brightness to each pixel. In absolute numbers when using 8 bit/pixel, 128 is added to each pixel (0.5 x 255 = 127.5). If you rather want to add an absolute value to each pixel do the following calculation: e. g. add -51 to an 8 bit/pixel offset = -51 / 255 = -0.2.Figure 15.2 shows an example of an offset.

Figure 15.2. Lookup Table Processing: Offset

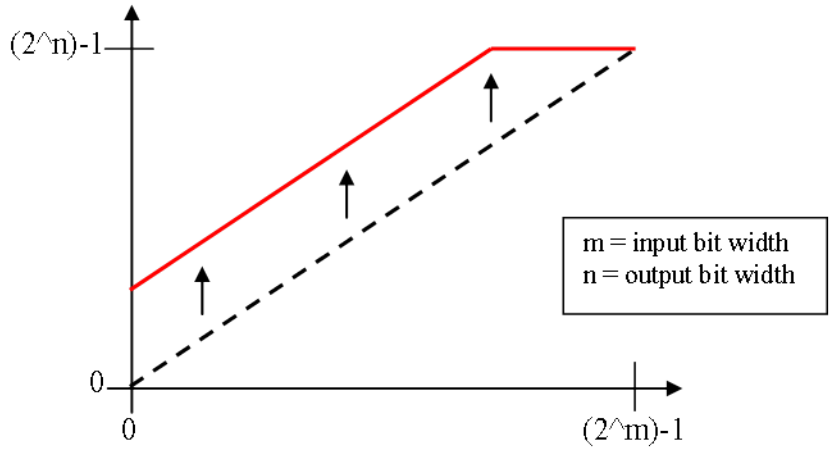


Table 15.1. Parameter properties of FG_PROCESSING_OFFSET

Property	Value
Name	**FG_PROCESSING_OFFSET**
Display Name	**Offset**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum -1.0** **Maximum 1.0** **Stepsize 2.220446049250313E-16**
Default value	**0.0**

Example 15.1. Usage of FG_PROCESSING_OFFSET

```
int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_PROCESSING_OFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_PROCESSING_OFFSET, &value, 0, type)) < 0) {
    /* error handling */
}
```

15.2. FG_PROCESSING_GAIN

The gain is a multiplicative coefficient applied to each pixel, which leads to a behavior similar to a contrast controller. Each pixel value will be multiplied with the given value. For identity select value 1.0.

Figure 15.3. Lookup Table Processing: Gain

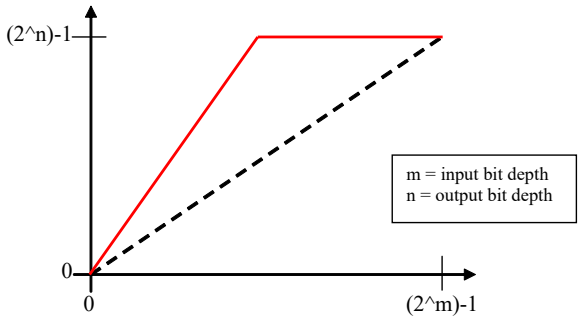


Table 15.2. Parameter properties of FG_PROCESSING_GAIN

Property	Value
Name	**FG_PROCESSING_GAIN**
Display Name	**Gain**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum** 0.0 **Maximum** 4096.0 **Stepsize** 2.220446049250313E-16
Default value	**1.0**

Example 15.2. Usage of FG_PROCESSING_GAIN

```
int result = 0;
double value = 1.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_PROCESSING_GAIN, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_PROCESSING_GAIN, &value, 0, type)) < 0) {
    /* error handling */
}
```

15.3. FG_PROCESSING_GAMMA

The gamma correction is a power-law transformation applied to each pixel. Normalized pixel values p ranging [0, 1.0] transform like $p' = p^{1/gamma}$.

Figure 15.4. Lookup Table Processing: Gamma

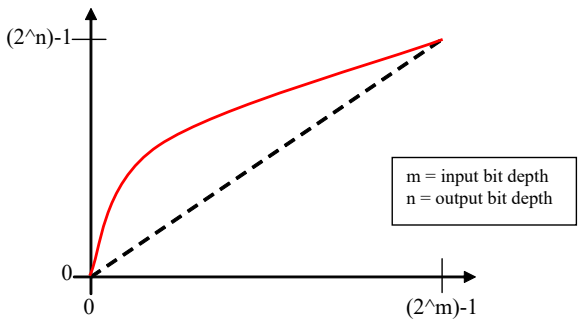


Table 15.3. Parameter properties of FG_PROCESSING_GAMMA

Property	Value
Name	**FG_PROCESSING_GAMMA**
Display Name	**Gamma**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum** -1000.0 **Maximum** 1000.0 **Stepsize** 2.220446049250313E-16
Default value	**1.0**

Example 15.3. Usage of FG_PROCESSING_GAMMA

```
int result = 0;
double value = 1.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_PROCESSING_GAMMA, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_PROCESSING_GAMMA, &value, 0, type)) < 0) {
    /* error handling */
}
```

15.4. FG_PROCESSING_INVERT

When *FG_PROCESSING_INVERT* is set to **FG_ON**, the output is the negative of the input. Normalized pixel values p ranging [0, 1.0] transform to $p' = 1 - p$.

Figure 15.5. Lookup Table Processing: Invert

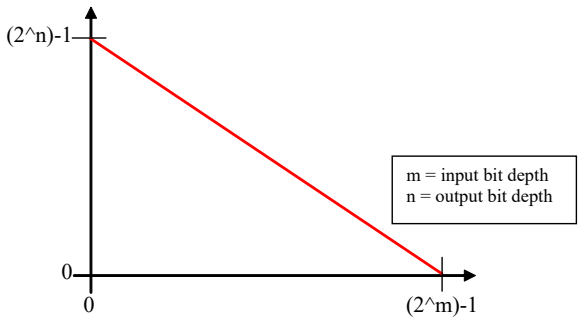


Table 15.4. Parameter properties of FG_PROCESSING_INVERT

Property	Value
Name	**FG_PROCESSING_INVERT**
Display Name	**Invert**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_ON** On **FG_OFF** Off
Default value	**FG_OFF**

Example 15.4. Usage of FG_PROCESSING_INVERT

```
int result = 0;
int value = FG_OFF;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_PROCESSING_INVERT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_PROCESSING_INVERT, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 16. Output Format

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

16.1. FG_FORMAT

Parameter *FG_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 12 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 *FG_BITALIGNMENT*.

This applet has no integrated color converter. If you select a different color pixel format between the input and output no valid output data can be generated.

This applet supports the following output formats:

- **FG_MONO8**: 8 bit grayscale format
- **FG_MONO10**: 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.

- **FG_MONO12**: 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.

- **FG_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 16.1. Parameter properties of FG_FORMAT

Property	Value
Name	**FG_FORMAT**
Display Name	**Output Format**
Type	**Enumeration**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**FG_GRAY** Gray 8bit **FG_GRAY10** Gray 10bit **FG_GRAY12** Gray 12bit **FG_GRAY16** Gray 16bit
Default value	**FG_GRAY**

Example 16.1. Usage of FG_FORMAT

```
int result = 0;
int value = FG_GRAY;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_FORMAT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_FORMAT, &value, 0, type)) < 0) {
    /* error handling */
}
```

16.2. FG_BITALIGNMENT

The bit alignment is used to map the pixel bits of the internal processing with a depth of 12 bit to the configured DMA output bit depth defined by parameter *FG_FORMAT*.

You can select three different modes: Left aligned, right aligned and a custom shift mode. If you select left aligned, the applet will map the upper bits of the internal processing bit width to the available output bits. If you select right aligned, the applet will map the lower bits of the internal processing bit width to the available output bits. If you want to define a custom bit shift, you'll need to set the parameter to CustomBitShift and use parameter *FG_CUSTOM_BIT_SHIFT_RIGHT* 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 16.2. Parameter properties of FG_BITALIGNMENT

Property	Value
Name	**FG_BITALIGNMENT**
Display Name	**Alignment**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_LEFT_ALIGNED** Left Aligned **FG_RIGHT_ALIGNED** Right Aligned **FG_CUSTOM_BIT_SHIFT_MODE** Custom Bit Shift
Default value	**FG_LEFT_ALIGNED**

Example 16.2. Usage of FG_BITALIGNMENT

```
int result = 0;
int value = FG_LEFT_ALIGNED;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_BITALIGNMENT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_BITALIGNMENT, &value, 0, type)) < 0) {
    /* error handling */
}
```

16.3. FG_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 16.3. Parameter properties of FG_PIXELDEPTH

Property	Value
Name	**FG_PIXELDEPTH**
Display Name	**Pixel Depth**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 128** **Stepsize 1**
Unit of measure	**bit**

Example 16.3. Usage of FG_PIXELDEPTH

```
int result = 0;
unsigned int value = 8;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_PIXELDEPTH, &value, 0, type)) < 0) {
    /* error handling */
}
```

16.4. FG_CUSTOM_BIT_SHIFT_RIGHT

This parameter can only be used if parameter *FG_BITALIGNMENT* is set to **FG_CUSTOM_BIT_SHIFT_MODE**. If it is enabled, you can define a custom right bit shift value for the DMA output of the frame grabber. 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 12 bits.

Table 16.4. Parameter properties of FG_CUSTOM_BIT_SHIFT_RIGHT

Property	Value
Name	**FG_CUSTOM_BIT_SHIFT_RIGHT**
Display Name	**Bit Shift Right**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 15** **Stepsize 1**
Default value	**0**
Unit of measure	**bit**

Example 16.4. Usage of FG_CUSTOM_BIT_SHIFT_RIGHT

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CUSTOM_BIT_SHIFT_RIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CUSTOM_BIT_SHIFT_RIGHT, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 17. 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.

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

Figure 17.1. 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.

17.1. FG_CAMERASIMULATOR_ENABLE

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. Note that an activated simulator will have effect on parameter *FG_CAMSTATUS*.

The camera simulator will use 8bit values for all supported pixel formats.

Table 17.1. Parameter properties of FG_CAMERASIMULATOR_ENABLE

Property	Value	
Name	**FG_CAMERASIMULATOR_ENABLE**	
Display Name	**Image Source**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**FG_CAMPORT** **FG_CAMERASIMULATOR**	Camera Simulator
Default value	**FG_CAMPORT**	

Example 17.1. Usage of FG_CAMERASIMULATOR_ENABLE

```
int result = 0;
int value = FG_CAMPORT;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_ENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_ENABLE, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.2. FG_CAMERASIMULATOR_WIDTH

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 17.2. Parameter properties of FG_CAMERASIMULATOR_WIDTH

Property	Value
Name	**FG_CAMERASIMULATOR_WIDTH**
Display Name	**Width**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 65536** **Stepsize 1**
Default value	**1024**
Unit of measure	**pixel**

Example 17.2. Usage of FG_CAMERASIMULATOR_WIDTH

```
int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.3. FG_CAMERASIMULATOR_LINE_GAP

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 *FG_CAMERASIMULATOR_SELECT_MODE* is set to **FG_PIXEL_FREQUENCY**.

Table 17.3. Parameter properties of FG_CAMERASIMULATOR_LINE_GAP

Property	Value
Name	**FG_CAMERASIMULATOR_LINE_GAP**
Display Name	**Line Gap**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 65536** **Stepsize 1**
Default value	**0**
Unit of measure	**pixel**

Example 17.3. Usage of FG_CAMERASIMULATOR_LINE_GAP

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_LINE_GAP, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_LINE_GAP, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.4. FG_CAMERASIMULATOR_HEIGHT

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 17.4. Parameter properties of FG_CAMERASIMULATOR_HEIGHT

Property	Value
Name	**FG_CAMERASIMULATOR_HEIGHT**
Display Name	**Height**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 65536** **Stepsize 1**
Default value	**1024**
Unit of measure	**pixel**

Example 17.4. Usage of FG_CAMERASIMULATOR_HEIGHT

```
int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}
```

```
if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.5. FG_CAMERASIMULATOR_FRAME_GAP

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 *FG_CAMERASIMULATOR_SELECT_MODE* is set to **FG_FRAMERATE**.

Table 17.5. Parameter properties of FG_CAMERASIMULATOR_FRAME_GAP

Property	Value
Name	**FG_CAMERASIMULATOR_FRAME_GAP**
Display Name	**Frame Gap**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 65536** **Stepsize 1**
Default value	**0**
Unit of measure	**pixel**

Example 17.5. Usage of FG_CAMERASIMULATOR_FRAME_GAP

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_FRAME_GAP, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_FRAME_GAP, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.6. FG_CAMERASIMULATOR_PATTERN

The simulator will generate pixel value ramps from 0 to 255.

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

- **FG_HORIZONTAL**

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

- **FG_VERTICAL**

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

- **FG_DIAGONAL**

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

Table 17.6. Parameter properties of FG_CAMERASIMULATOR_PATTERN

Property	Value
Name	**FG_CAMERASIMULATOR_PATTERN**
Display Name	**Pattern**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_HORIZONTAL** Horizontal **FG_VERTICAL** Vertical **FG_DIAGONAL** Diagonal
Default value	**FG_DIAGONAL**

Example 17.6. Usage of FG_CAMERASIMULATOR_PATTERN

```
int result = 0;
int value = FG_DIAGONAL;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_PATTERN, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_PATTERN, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.7. FG_CAMERASIMULATOR_PATTERN_OFFSET

Using this parameter, an offset value can be added to 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 17.7. Parameter properties of FG_CAMERASIMULATOR_PATTERN_OFFSET

Property	Value
Name	**FG_CAMERASIMULATOR_PATTERN_OFFSET**
Display Name	**Pattern Offset**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum** 0 **Maximum** 255 **Stepsize** 1
Default value	**0**
Unit of measure	**pixel value**

Example 17.7. Usage of FG_CAMERASIMULATOR_PATTERN_OFFSET

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_PATTERN_OFFSET, &value, 0, type)) < 0) {
    /* error handling */
}
```

```
if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_PATTERN_OFFSET, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.8. FG_CAMERASIMULATOR_ROLL

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 17.8. Parameter properties of FG_CAMERASIMULATOR_ROLL

Property	Value
Name	**FG_CAMERASIMULATOR_ROLL**
Display Name	**Roll**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_ON** On **FG_OFF** Off
Default value	**FG_ON**

Example 17.8. Usage of FG_CAMERASIMULATOR_ROLL

```
int result = 0;
int value = FG_ON;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_ROLL, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_ROLL, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.9. FG_CAMERASIMULATOR_SELECT_MODE

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 17.9. Parameter properties of FG_CAMERASIMULATOR_SELECT_MODE

Property	Value
Name	**FG_CAMERASIMULATOR_SELECT_MODE**
Display Name	**Speed Mode**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_PIXEL_FREQUENCY** Pixel Frequency **FG_LINERATE** Line Rate **FG_FRAMERATE** Frame Rate
Default value	**FG_LINERATE**

Example 17.9. Usage of FG_CAMERASIMULATOR_SELECT_MODE

```
int result = 0;
int value = FG_LINERATE;
```

```
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_SELECT_MODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_SELECT_MODE, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.10. FG_CAMERASIMULATOR_PIXEL_FREQUENCY

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 *FG_CAMERASIMULATOR_SELECT_MODE* to **FG_PIXEL_FREQUENCY**.

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 17.10. Parameter properties of FG_CAMERASIMULATOR_PIXEL_FREQUENCY

Property	Value
Name	**FG_CAMERASIMULATOR_PIXEL_FREQUENCY**
Display Name	**Pixel Frequency**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum** 0.12499999999999997 **Maximum** -1.0 **Stepsize** 0.25
Default value	**39.375**
Unit of measure	**MHz**

Example 17.10. Usage of FG_CAMERASIMULATOR_PIXEL_FREQUENCY

```
int result = 0;
double value = 39.375;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_PIXEL_FREQUENCY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_PIXEL_FREQUENCY, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.11. FG_CAMERASIMULATOR_LINERATE

This parameter sets the line rate of the generated images.

To set the line rate, you will need to set parameter *FG_CAMERASIMULATOR_SELECT_MODE* to **FG_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 17.11. Parameter properties of FG_CAMERASIMULATOR_LINERATE

Property	Value
Name	**FG_CAMERASIMULATOR_LINERATE**
Display Name	**Line Rate**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.15** **Maximum -35714.28571428572** **Stepsize 7.0E-11**
Default value	**10240.0**
Unit of measure	**Hz**

Example 17.11. Usage of FG_CAMERASIMULATOR_LINERATE

```
int result = 0;
double value = 10240.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_LINERATE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_LINERATE, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.12. FG_CAMERASIMULATOR_FRAMERATE

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

To set the frame rate, you will need to set parameter *FG_CAMERASIMULATOR_SELECT_MODE* to **FG_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 17.12. Parameter properties of FG_CAMERASIMULATOR_FRAMERATE

Property	Value
Name	**FG_CAMERASIMULATOR_FRAMERATE**
Display Name	**Framerate**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.15** **Maximum -35714.28571428572** **Stepsize 7.0E-11**
Default value	**10.0**
Unit of measure	**Hz**

Example 17.12. Usage of FG_CAMERASIMULATOR_FRAMERATE

```
int result = 0;
double value = 10.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_FRAMERATE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_FRAMERATE, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.13. FG_CAMERASIMULATOR_TRIGGER_MODE

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 is a Camera Link applet, output CC1 of the respective camera port is used as camera simulator trigger input. The rising edge of the trigger will be used. Thus you should set the output polarity of the trigger module to high active.

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 17.13. Parameter properties of FG_CAMERASIMULATOR_TRIGGER_MODE

Property	Value
Name	**FG_CAMERASIMULATOR_TRIGGER_MODE**
Display Name	**Trigger Mode**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**SIMULATION_FREE_RUN** Free Run **RISING_EDGE_TRIGGERS_LINE** Rising Edge Triggers Line **RISING_EDGE_TRIGGERS_FRAME** Rising Edge Triggers Frame
Default value	**SIMULATION_FREE_RUN**

Example 17.13. Usage of FG_CAMERASIMULATOR_TRIGGER_MODE

```
int result = 0;
int value = SIMULATION_FREE_RUN;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CAMERASIMULATOR_TRIGGER_MODE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_TRIGGER_MODE, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.14. FG_CAMERASIMULATOR_ACTIVE

Table 17.14. Parameter properties of FG_CAMERASIMULATOR_ACTIVE

Property	Value
Name	**FG_CAMERASIMULATOR_ACTIVE**
Display Name	**Active Parts**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 2000** **Stepsize 1**
Unit of measure	**pixel**

Example 17.14. Usage of FG_CAMERASIMULATOR_ACTIVE

```
int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_ACTIVE, &value, 0, type)) < 0) {
    /* error handling */
}
```

17.15. FG_CAMERASIMULATOR_PASSIVE

Table 17.15. Parameter properties of FG_CAMERASIMULATOR_PASSIVE

Property	Value
Name	**FG_CAMERASIMULATOR_PASSIVE**
Display Name	**Passive Parts**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 2000** **Stepsize 1**
Unit of measure	**pixel**

Example 17.15. Usage of FG_CAMERASIMULATOR_PASSIVE

```
int result = 0;
unsigned int value = 1024;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_CAMERASIMULATOR_PASSIVE, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 18. 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.

18.1. FG_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. Use the timeout value described in the Framegrabber API or microDisplay for acquisition in order to handle the functionality correctly.

Table 18.1. Parameter properties of FG_TIMEOUT

Property	Value
Name	**FG_TIMEOUT**
Display Name	**Timeout**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 2** **Maximum 2147483646** **Stepsize 1**
Default value	**1000000**
Unit of measure	**seconds**

Example 18.1. Usage of FG_TIMEOUT

```
int result = 0;
unsigned int value = 1000000;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_TIMEOUT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_TIMEOUT, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.2. FG_APPLET_VERSION

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

Table 18.2. Parameter properties of FG_APPLET_VERSION

Property	Value
Name	**FG_APPLET_VERSION**
Display Name	**Applet version**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 18.2. Usage of FG_APPLET_VERSION

```
int result = 0;
```

```
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_APPLET_VERSION, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.3. FG_APPLET_REVISION

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

Table 18.3. Parameter properties of FG_APPLET_REVISION

Property	Value
Name	**FG_APPLET_REVISION**
Display Name	**Applet revision**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 18.3. Usage of FG_APPLET_REVISION

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_APPLET_REVISION, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.4. FG_APPLET_ID

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

Table 18.4. Parameter properties of FG_APPLET_ID

Property	Value
Name	**FG_APPLET_ID**
Display Name	**Applet Id**
Type	**String**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 18.4. Usage of FG_APPLET_ID

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_getParameterWithType(fg, FG_APPLET_ID, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.5. FG_APPLET_BUILD_TIME

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 18.5. Parameter properties of FG_APPLET_BUILD_TIME

Property	Value
Name	**FG_APPLET_BUILD_TIME**
Display Name	**Build Time**
Type	**String**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 18.5. Usage of FG_APPLET_BUILD_TIME

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_getParameterWithType(fg, FG_APPLET_BUILD_TIME, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.6. FG_HAP_FILE

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 18.6. Parameter properties of FG_HAP_FILE

Property	Value
Name	**FG_HAP_FILE**
Display Name	**HAP file**
Type	**String**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 18.6. Usage of FG_HAP_FILE

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_getParameterWithType(fg, FG_HAP_FILE, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.7. FG_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 18.7. Parameter properties of FG_DMASTATUS

Property	Value
Name	**FG_DMASTATUS**
Display Name	**DMA Status**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 18.7. Usage of FG_DMASTATUS

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_DMASTATUS, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.8. FG_CAMSTATUS

The camera status shows whether the camera clock signal can be recognized by frame grabber or not. If value "1" is determined from this read parameter, the grabber recognized a camera clock signal.

Table 18.8. Parameter properties of FG_CAMSTATUS

Property	Value
Name	**FG_CAMSTATUS**
Display Name	**Camera Status**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 1** **Stepsize 1**

Example 18.8. Usage of FG_CAMSTATUS

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_CAMSTATUS, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.9. FG_CAMSTATUS_EXTENDED

This parameter provides extended information on the pixel clock from the camera, LVAL and FVAL, as well as the camera trigger signals, external trigger signals, buffer overflow status and buffer status. Each bit of the eight bit output word represents one parameter listed in the following:

- 0 = CameraClk, provided by CameraLink interface. Shows if CL PixelClock is available.
- 1 = CameraLval, provided by CameraLink interface. Shows if CameraLink LVAL is available, representing a line being transferred into frame grabber.
- 2 = CameraFval, provided by CameraLink interface. Shows if CameraLink FVAL is available, representing frames being transferred into frame grabber. Not relevant for standard line scan applets.
- 3 = Camera CC1 Signal, NOT provided by this frame grabber.
- 4 = ExTrg / external trigger, NOT provided by this frame grabber.
- 5 = BufferOverflow
- 6 = BufStatus, LSB

- 7 = BufStatus, MSB

Table 18.9. Parameter properties of FG_CAMSTATUS_EXTENDED

Property	Value
Name	**FG_CAMSTATUS_EXTENDED**
Display Name	**Camera Status Extended**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 255** **Stepsize 1**

Example 18.9. Usage of FG_CAMSTATUS_EXTENDED

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_CAMSTATUS_EXTENDED, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.10. FG_SYSTEMMONITOR_FPGA_TEMPERATURE

Returns the current FGPA die temperature.

Table 18.10. Parameter properties of FG_SYSTEMMONITOR_FPGA_TEMPERATURE

Property	Value
Name	**FG_SYSTEMMONITOR_FPGA_TEMPERATURE**
Display Name	**FGPA Temperature**
Type	**Double**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0.0** **Maximum 1000.0** **Stepsize 0.0**
Unit of measure	**Celsius**

Example 18.10. Usage of FG_SYSTEMMONITOR_FPGA_TEMPERATURE

```
int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_FPGA_TEMPERATURE, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.11. FG_SYSTEMMONITOR_FPGA_VCC_INT

Returns the current FPGA internal voltage.

Table 18.11. Parameter properties of FG_SYSTEMMONITOR_FPGA_VCC_INT

Property	Value
Name	**FG_SYSTEMMONITOR_FPGA_VCC_INT**
Display Name	**FGPA Vcc Int**
Type	**Double**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum -1000.0** **Maximum 1000.0** **Stepsize 0.0**
Unit of measure	**V**

Example 18.11. Usage of FG_SYSTEMMONITOR_FPGA_VCC_INT

```
int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_FPGA_VCC_INT, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.12. FG_SYSTEMMONITOR_FPGA_VCC_AUX

Returns the current FPGA Vcc auxiliary voltage.

Table 18.12. Parameter properties of FG_SYSTEMMONITOR_FPGA_VCC_AUX

Property	Value
Name	**FG_SYSTEMMONITOR_FPGA_VCC_AUX**
Display Name	**FGPA Vcc Aux**
Type	**Double**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum -1000.0** **Maximum 1000.0** **Stepsize 0.0**
Unit of measure	**V**

Example 18.12. Usage of FG_SYSTEMMONITOR_FPGA_VCC_AUX

```
int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_FPGA_VCC_AUX, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.13. FG_SYSTEMMONITOR_FPGA_VCC_BRAM

Returns the current FPGA Vcc of the BlockRAM voltage.

Table 18.13. Parameter properties of FG_SYSTEMMONITOR_FPGA_VCC_BRAM

Property	Value
Name	**FG_SYSTEMMONITOR_FPGA_VCC_BRAM**
Display Name	**FGPA Vcc BRAM**
Type	**Double**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum -1000.0** **Maximum 1000.0** **Stepsize 0.0**
Unit of measure	**V**

Example 18.13. Usage of FG_SYSTEMMONITOR_FPGA_VCC_BRAM

```
int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_FPGA_VCC_BRAM, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.14. FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH

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

Table 18.14. Parameter properties of FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH

Property	Value
Name	**FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH**
Display Name	**Current Link Width**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 15** **Stepsize 0**
Unit of measure	**lanes**

Example 18.14. Usage of FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.15. FG_SYSTEMMONITOR_CURRENT_LINK_SPEED

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

Table 18.15. Parameter properties of FG_SYSTEMMONITOR_CURRENT_LINK_SPEED

Property	Value
Name	**FG_SYSTEMMONITOR_CURRENT_LINK_SPEED**
Display Name	**Current Link Speed**
Type	**Double**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum** 0.0 **Maximum** 1000.0 **Stepsize** 0.0
Unit of measure	**GB/s**

Example 18.15. Usage of FG_SYSTEMMONITOR_CURRENT_LINK_SPEED

```
int result = 0;
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_CURRENT_LINK_SPEED, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.16. FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE

Returns if PCIe generation 2 is supported by current applet of the frame grabber.

Table 18.16. Parameter properties of FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE

Property	Value
Name	**FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE**
Display Name	**PCIe Link Gen 2 Capable**
Type	**Enumeration**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**FG_YES** Yes **FG_NO** No

Example 18.16. Usage of FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE

```
int result = 0;
int value = FG_YES;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.17. FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE

Returns if the expected PCIe generation 2 is supported by the partner. The partner would be the mainboard or in detail the corresponding PCIe interface on the host side.

Table 18.17. Parameter properties of FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE

Property	Value
Name	**FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE**
Display Name	**PCIe Link Partner Gen 2 Capable**
Type	**Enumeration**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**FG_YES** Yes **FG_NO** No

Example 18.17. Usage of FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE

```
int result = 0;
int value = FG_YES;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.18. FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE

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

Table 18.18. Parameter properties of FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE

Property	Value
Name	**FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE**
Display Name	**PCIe Trained Payload Size**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum** 0 **Maximum** 1024 **Stepsize** 0
Unit of measure	**byte**

Example 18.18. Usage of FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.19. FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT

Returns if a extension connector is present on the frame grabber board.

Table 18.19. Parameter properties of FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT

Property	Value
Name	**FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT**
Display Name	**Extension Connector Present**
Type	**Enumeration**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**FG_YES** Yes **FG_NO** No

Example 18.19. Usage of FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT

```
int result = 0;
int value = FG_YES;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.20. FG_ALTERNATIVE_BOARD_DETECTION

Returns the current state of the alternative frame grabber PCIe board detection algorithm. If value = FG_OFF, the Silicon Software default algorithm is used. If value = FG_ON, an alternative board detection algorithm is used.

This parameter is used for support purposes only.

Table 18.20. Parameter properties of FG_ALTERNATIVE_BOARD_DETECTION

Property	Value
Name	**FG_ALTERNATIVE_BOARD_DETECTION**
Display Name	**Alternative Board Detection**
Type	**Enumeration**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**FG_ON** On **FG_OFF** Off

Example 18.20. Usage of FG_ALTERNATIVE_BOARD_DETECTION

```
int result = 0;
int value = FG_OFF;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_ALTERNATIVE_BOARD_DETECTION, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.21. FG_SYSTEMMONITOR_POCL_STATE_PORT_A

The parameter allows to read the current state of the Power over Camera Link (PoCL) state machine on the Port A connector.

The individual states indicate the following:

FG_INITIALIZE : This state has a duration of 100 ms. During this period, PoCL detection as well as PoCL operation is off. This way, the board establishes a defined initial state with no voltage applied.

FG_POCL_CONNECTION_SENSE : This state has a duration of 600 ms. It follows directly of state Initialize. During this state, the controller finds out if the connected camera is PoCL-capable or not.

- If a PoCL camera is detected, the PoCL state machine switches to state PoCL_Wait_for_Connection.
- If a CL camera without PoCL support is detected, the PoCL state machine switches to state CL_Wait_for_Connection.
- If a disconnect or disturbances are detected, the state machine switches back to state Initialize and starts again.

(The controller applies a test current and waits for 600 ms. Then, the voltage drop is measured. There are two thresholds: Is the measured value between both thresholds, the PoCL state machine switches to state PoCL_Wait_for_Connection. Is the measured value lower than both thresholds, the PoCL state machine switches to state CL_Wait_for_Connection. Is the measured value higher than both thresholds, the connection is either broken or disturbed. In this case, the PoCL state machine switches to state Initialize and starts again.)

FG_POCL_WAIT_FOR_CONNECTION : This state has a duration of 1.8 seconds. It follows directly of state **FG_POCL_CONNECTION_SENSE** in case a a power-over capable camera is detected. During this time, the controller waits and checks if the information about the availability of a power-over capable camera remains stable:

- If it is stable, the state machine switches to state **FG_POCL_CAMERA_DETECTED** and powers the camera.
- If it is not stable, the state machine switches back to state Initialize and starts again.

To ensure that a PoCL camera was not detected erroneously (due to disturbances), during state **FG_POCL_WAIT_FOR_CONNECTION** the controller checks if the measured voltage drop remains stable between the two thresholds values during the 1.8 seconds. If the measured voltage remains stable, a PoCL-capable camera is assumed, the state machine switches to state **FG_POCL_CAMERA_DETECTED** , and the camera is powered. Rises the measured voltage higher the upper threshold value, or falls it below the lower threshold value, there is a disturbance. The state machine switches to state Initialize and starts again.

FG_POCL_CAMERA_DETECTED : This state has a duration of up to 4 seconds. The camera is powered. The controller waits for the camera to get ready and for receiving a clock signal from the camera.

- If a clock is detected (within maximally 4s), the camera is ready for operation. The state machine switches to state **FG_POCL_CAMERA_CLOCK_DETECTED** .
- If no clock is detected (during maximally 4s), the state machine switches to state Initialize and starts again.

FG_POCL_CAMERA_CLOCK_DETECTED : The camera is ready for operation.

- As long as the state machine receives the clock signal from the camera, the state machine remains in this state.
- If there is no clock signal for more than 400 ms, the state machine switches to state Initialize. (It is assumed that either the camera has been disconnected, or an error has occurred.)

FG_CL_WAIT_FOR_CONNECTION : This state has a duration of 100ms. It follows directly of state FG_POCL_CONNECTION_SENSE in case a CL camera without PoCL support is detected. The test current is switched off. The system waits for 100ms to allow the charges to drain slowly. After this timespan, the state machine switches to state **FG_CL_CAMERA_DETECTED** , and ground (GND) is connected.

FG_CL_CAMERA_DETECTED : This state has a duration of up to 4 seconds. The connected camera has been identified as not PoCL-capable. The controller waits for the camera to get ready and for receiving a clock signal from the camera.

- If a clock is detected (within maximally 4s), the camera is ready for operation. The state machine switches to state **FG_CL_CAMERA_CLOCK_DETECTED** .

- If during 4s no clock is detected, the camera is not ready for operation. The state machine switches to state Initialize and starts again.

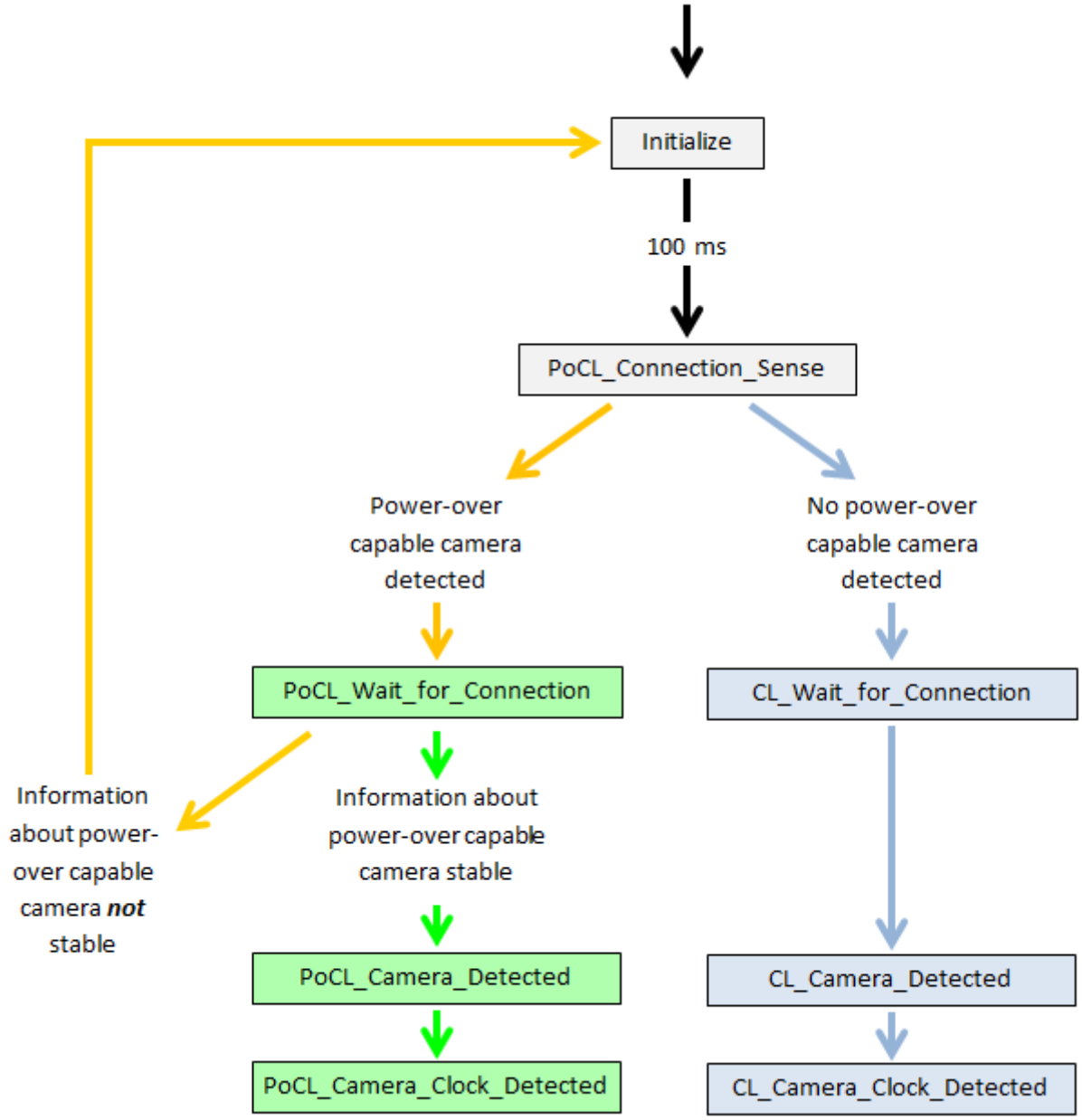
FG_CL_CAMERA_CLOCK_DETECTED : The camera is ready for operation.

- As long as the state machine receives the clock signal from the camera, the state machine remains in this state.
- If there is no clock signal for more than 1 s, the state machine switches to state Initialize. (It is assumed that either the camera has been disconnected, or an error has occurred.)

FG_POCL_DISABLE : PoCL is completely disabled for this the frame grabber. A PoCL camera needs to be powered from external power supply.

PoCL state machine decision flow:

Figure 18.1. PoCL States



Power Watchdog

Additional security mechanism: The voltage level is permanently monitored.

- Is the voltage for 2 ms higher than the the lower threshold value while **no** PoCL voltage is applied: A short circuit is assumed. The state machine switches to state Initialize.
- Is the voltage for 2 ms lower than the the upper threshold value while PoCL voltage **is** applied: The occurrence of an error is assumed. The state machine switches to state Initialize.

Table 18.21. Parameter properties of FG_SYSTEMMONITOR_POCL_STATE_PORT_A

Property	Value	
Name	**FG_SYSTEMMONITOR_POCL_STATE_PORT_A**	
Display Name	**PoCL State Port A**	
Type	**Enumeration**	
Access policy	**Read-Only**	
Storage policy	**Transient**	
Allowed values	**FG_INITIALIZE**	Initialize
	FG_POCL_CONNECTION_SENSE	PoCL Connection Sense
	FG_POCL_WAIT_FOR_CONNECTION	PoCL wait for Connection
	FG_POCL_CAMERA_DETECTED	PoCL Camera and Cable Detected
	FG_POCL_CAMERA_CLOCK_DETECTED	PoCL Camera Clock Detected
	FG_CL_WAIT_FOR_CONNECTION	CL wait for Connection
	FG_CL_CAMERA_DETECTED	CL Camera Detected
	FG_CL_CAMERA_CLOCK_DETECTED	CL Camera Clock Detected
	FG_POCL_DISABLED	PoCL Disabled

Example 18.21. Usage of FG_SYSTEMMONITOR_POCL_STATE_PORT_A

```
int result = 0;
int value = FG_INITIALIZE;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_POCL_STATE_PORT_A, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.22. FG_SYSTEMMONITOR_POCL_STATE_PORT_B

Returns the Power over CameraLink state of CameraLink Port B. Please see documentation of Section 18.21, 'FG_SYSTEMMONITOR_POCL_STATE_PORT_A'.

Table 18.22. Parameter properties of FG_SYSTEMMONITOR_POCL_STATE_PORT_B

Property	Value	
Name	**FG_SYSTEMMONITOR_POCL_STATE_PORT_B**	
Display Name	**PoCL State Port B**	
Type	**Enumeration**	
Access policy	**Read-Only**	
Storage policy	**Transient**	
Allowed values	**FG_INITIALIZE**	Initialize
	FG_POCL_CONNECTION_SENSE	PoCL Connection Sense
	FG_POCL_WAIT_FOR_CONNECTION	PoCL wait for Connection
	FG_POCL_CAMERA_DETECTED	PoCL Camera and Cable Detected
	FG_POCL_CAMERA_CLOCK_DETECTED	PoCL Camera Clock Detected
	FG_CL_WAIT_FOR_CONNECTION	CL wait for Connection
	FG_CL_CAMERA_DETECTED	CL Camera Detected
	FG_CL_CAMERA_CLOCK_DETECTED	CL Camera Clock Detected
	FG_POCL_DISABLED	PoCL Disabled

Example 18.22. Usage of FG_SYSTEMMONITOR_POCL_STATE_PORT_B

```
int result = 0;
int value = FG_INITIALIZE;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_POCL_STATE_PORT_B, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.23. FG_SYSTEMMONITOR_FPGA_DNA

The parameter *FG_SYSTEMMONITOR_FPGA_DNA* provides the 57 bit unique FPGA DNA as an integer value.

Table 18.23. Parameter properties of FG_SYSTEMMONITOR_FPGA_DNA

Property	Value	
Name	**FG_SYSTEMMONITOR_FPGA_DNA**	
Display Name	**FPGA DNA**	
Type	**Unsigned Integer (64 Bit)**	
Access policy	**Read-Only**	
Storage policy	**Transient**	
Allowed values	**Minimum**	**0**
	Maximum	**144115188075855872**
	Stepsize	**0**

Example 18.23. Usage of FG_SYSTEMMONITOR_FPGA_DNA

```
int result = 0;
uint64_t value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT64_T;

if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_FPGA_DNA, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24. Debug

18.24.1. FG_DEBUGSOURCE

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.24. Parameter properties of FG_DEBUGSOURCE

Property	Value
Name	**FG_DEBUGSOURCE**
Display Name	**Debug Source**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 10** **Stepsize 1**
Default value	**0**

Example 18.24. Usage of FG_DEBUGSOURCE

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGSOURCE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGSOURCE, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.2. FG_DEBUGSOURCENAME

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.25. Parameter properties of FG_DEBUGSOURCENAME

Property	Value
Name	**FG_DEBUGSOURCENAME**
Display Name	**Debug Source Name**
Type	**String**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 18.25. Usage of FG_DEBUGSOURCENAME

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_getParameterWithType(fg, FG_DEBUGSOURCENAME, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.3. FG_DEBUGSAVECONFIG

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.26. Parameter properties of FG_DEBUGSAVECONFIG

Property	Value
Name	**FG_DEBUGSAVECONFIG**
Display Name	**Debug Save Config**
Type	**String**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Default value	**""**

Example 18.26. Usage of FG_DEBUGSAVECONFIG

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGSAVECONFIG, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGSAVECONFIG, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.4. FG_DEBUG_VERSION

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.27. Parameter properties of FG_DEBUG_VERSION

Property	Value
Name	**FG_DEBUG_VERSION**
Display Name	**Version**
Type	**String**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 18.27. Usage of FG_DEBUG_VERSION

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_getParameterWithType(fg, FG_DEBUG_VERSION, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.5. Input

18.24.5.1. FG_DEBUGINENABLE

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.28. Parameter properties of FG_DEBUGINENABLE

Property	Value
Name	**FG_DEBUGINENABLE**
Display Name	**Debug Input Mode**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_ON** On **FG_OFF** Off
Default value	**FG_OFF**

Example 18.28. Usage of FG_DEBUGINENABLE

```
int result = 0;
int value = FG_OFF;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGINENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGINENABLE, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.5.2. FG_DEBUGFILE

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.29. Parameter properties of FG_DEBUGFILE

Property	Value
Name	**FG_DEBUGFILE**
Display Name	**Debug File**
Type	**String**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Default value	**""**

Example 18.29. Usage of FG_DEBUGFILE

```
int result = 0;
char* value = "";
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGFILE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGFILE, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.5.3. FG_DEBUGINSERT

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.30. Parameter properties of FG_DEBUGINSERT

Property	Value
Name	**FG_DEBUGINSERT**
Display Name	**Debug Insert Image**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**FG_APPLY** Apply
Default value	**FG_APPLY**

Example 18.30. Usage of FG_DEBUGINSERT

```
int result = 0;
int value = FG_APPLY;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGINSERT, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGINSERT, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.5.4. FG_DEBUGWRITEPIXEL

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.31. Parameter properties of FG_DEBUGWRITEPIXEL

Property	Value
Name	**FG_DEBUGWRITEPIXEL**
Display Name	**Debug Write Pixel**
Type	**Unsigned Integer (64 Bit)**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**Minimum** 0 **Maximum** 144115188075855872 **Stepsize** 1
Default value	**0**

Example 18.31. Usage of FG_DEBUGWRITEPIXEL

```
int result = 0;
uint64_t value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT64_T;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGWRITEPIXEL, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGWRITEPIXEL, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.5.5. FG_DEBUGWRITEFLAG

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.32. Parameter properties of FG_DEBUGWRITEFLAG

Property	Value
Name	**FG_DEBUGWRITEFLAG**
Display Name	**Debug Write Flag**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**FG_ENDOFLINE** EndOfLine **FG_ENDOFFRAME** EndOfFrame
Default value	**FG_ENDOFLINE**

Example 18.32. Usage of FG_DEBUGWRITEFLAG

```
int result = 0;
int value = FG_ENDOFLINE;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGWRITEFLAG, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGWRITEFLAG, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.5.6. FG_DEBUGREADY

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.33. Parameter properties of FG_DEBUGREADY

Property	Value
Name	**FG_DEBUGREADY**
Display Name	**Debug Write Ready**
Type	**Enumeration**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**FG_YES** Yes **FG_NO** No

Example 18.33. Usage of FG_DEBUGREADY

```
int result = 0;
int value = FG_NOE;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_getParameterWithType(fg, FG_DEBUGREADY, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.6. Output

18.24.6.1. FG_DEBUGOUTENABLE

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.34. Parameter properties of FG_DEBUGOUTENABLE

Property	Value
Name	**FG_DEBUGOUTENABLE**
Display Name	**Debug Output Mode**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_ON** On **FG_OFF** Off
Default value	**FG_OFF**

Example 18.34. Usage of FG_DEBUGOUTENABLE

```
int result = 0;
int value = FG_OFF;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_DEBUGOUTENABLE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_DEBUGOUTENABLE, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.6.2. FG_DEBUGOUTXPOS

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.35. Parameter properties of FG_DEBUGOUTXPOS

Property	Value
Name	**FG_DEBUGOUTXPOS**
Display Name	**Debug Output XPosition**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum** 8 **Maximum** 49136 **Stepsize** 1
Unit of measure	**pixel**

Example 18.35. Usage of FG_DEBUGOUTXPOS

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_DEBUGOUTXPOS, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.6.3. FG_DEBUGOUTYPOS

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.36. Parameter properties of FG_DEBUGOUTYPOS

Property	Value
Name	**FG_DEBUGOUTYPOS**
Display Name	**Debug Output YPosition**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 1** **Maximum 16777215** **Stepsize 1**
Unit of measure	**pixel**

Example 18.36. Usage of FG_DEBUGOUTYPOS

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_DEBUGOUTYPOS, &value, 0, type)) < 0) {
    /* error handling */
}
```

18.24.6.4. FG_DEBUGOUTPIXEL

This parameter is for internal testing. Please DON'T use this parameter.

Table 18.37. Parameter properties of FG_DEBUGOUTPIXEL

Property	Value
Name	**FG_DEBUGOUTPIXEL**
Display Name	**Debug Output Pixel**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum -1** **Stepsize 1**
Unit of measure	**pixel**

Example 18.37. Usage of FG_DEBUGOUTPIXEL

```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_getParameterWithType(fg, FG_DEBUGOUTPIXEL, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 19. Legacy Parameter

This category includes legacy parameter. They can be used for compatibility to previous software versions. However all functions are replaced by new parameters. Hence, legacy parameters should not be used anymore.

19.1. FG_CCSEL0

Select one of the options described above.

Table 19.1. Parameter properties of FG_CCSEL0

Property	Value	
Name	**FG_CCSEL0**	
Display Name	**CC 1**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**CC_EXSYNC**	Exsync
	CC_NOT_EXSYNC	!Exsync
	CC_PRESCALER	Prescaler
	CC_NOT_PRESCALER	!Prescaler
	CC_STROBEPULSE	Flash
	CC_NOT_STROBEPULSE	!Flash
	CC_GND	Gnd
	CC_VCC	Vcc
	CC_GPI_0	Input 0
	CC_NOT_GPI_0	Not Input 0
	CC_GPI_1	Input 1
	CC_NOT_GPI_1	Not Input 1
	CC_GPI_2	Input 2
	CC_NOT_GPI_2	Not Input 2
	CC_GPI_3	Input 3
	CC_NOT_GPI_3	Not Input 3
	CC_GPI_4	Input 4
	CC_NOT_GPI_4	Not Input 4
	CC_GPI_5	Input 5
	CC_NOT_GPI_5	Not Input 5
	CC_GPI_6	Input 6
	CC_NOT_GPI_6	Not Input 6
	CC_GPI_7	Input 7
	CC_NOT_GPI_7	Not Input 7
	CC_FRONT_GPI_0	Front Input 0
	CC_NOT_FRONT_GPI_0	Not Front Input 0
	CC_FRONT_GPI_1	Front Input 1
	CC_NOT_FRONT_GPI_1	Not Front Input 1
	CC_FRONT_GPI_2	Front Input 2
	CC_NOT_FRONT_GPI_2	Not Front Input 2
	CC_FRONT_GPI_3	Front Input 3
	CC_NOT_FRONT_GPI_3	Not Front Input 3
	FG_OTHER	Other
Default value	**CC_EXSYNC**	

Example 19.1. Usage of FG_CCSEL0

```
int result = 0;
int value = CC_EXSYNC;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CCSEL0, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CCSEL0, &value, 0, type)) < 0) {
    /* error handling */
}
```

19.2. FG_CCSEL1

Select one of the options described above.

Table 19.2. Parameter properties of FG_CCSEL1

Property	Value	
Name	**FG_CCSEL1**	
Display Name	**CC 2**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**CC_EXSYNC**	Exsync
	CC_NOT_EXSYNC	!Exsync
	CC_PRESCALER	Prescaler
	CC_NOT_PRESCALER	!Prescaler
	CC_STROBEPULSE	Flash
	CC_NOT_STROBEPULSE	!Flash
	CC_GND	Gnd
	CC_VCC	Vcc
	CC_GPI_0	Input 0
	CC_NOT_GPI_0	Not Input 0
	CC_GPI_1	Input 1
	CC_NOT_GPI_1	Not Input 1
	CC_GPI_2	Input 2
	CC_NOT_GPI_2	Not Input 2
	CC_GPI_3	Input 3
	CC_NOT_GPI_3	Not Input 3
	CC_GPI_4	Input 4
	CC_NOT_GPI_4	Not Input 4
	CC_GPI_5	Input 5
	CC_NOT_GPI_5	Not Input 5
	CC_GPI_6	Input 6
	CC_NOT_GPI_6	Not Input 6
	CC_GPI_7	Input 7
	CC_NOT_GPI_7	Not Input 7
	CC_FRONT_GPI_0	Front Input 0
	CC_NOT_FRONT_GPI_0	Not Front Input 0
	CC_FRONT_GPI_1	Front Input 1
	CC_NOT_FRONT_GPI_1	Not Front Input 1
	CC_FRONT_GPI_2	Front Input 2
	CC_NOT_FRONT_GPI_2	Not Front Input 2
	CC_FRONT_GPI_3	Front Input 3
	CC_NOT_FRONT_GPI_3	Not Front Input 3
	FG_OTHER	Other
Default value	**CC_VCC**	

Example 19.2. Usage of FG_CCSEL1

```
int result = 0;
int value = CC_VCC;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CCSEL1, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CCSEL1, &value, 0, type)) < 0) {
    /* error handling */
}
```

19.3. FG_CCSEL2

Select one of the options described above.

Table 19.3. Parameter properties of FG_CCSEL2

Property	Value	
Name	**FG_CCSEL2**	
Display Name	**CC 3**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**CC_EXSYNC**	Exsync
	CC_NOT_EXSYNC	!Exsync
	CC_PRESCALER	Prescaler
	CC_NOT_PRESCALER	!Prescaler
	CC_STROBEPULSE	Flash
	CC_NOT_STROBEPULSE	!Flash
	CC_GND	Gnd
	CC_VCC	Vcc
	CC_GPI_0	Input 0
	CC_NOT_GPI_0	Not Input 0
	CC_GPI_1	Input 1
	CC_NOT_GPI_1	Not Input 1
	CC_GPI_2	Input 2
	CC_NOT_GPI_2	Not Input 2
	CC_GPI_3	Input 3
	CC_NOT_GPI_3	Not Input 3
	CC_GPI_4	Input 4
	CC_NOT_GPI_4	Not Input 4
	CC_GPI_5	Input 5
	CC_NOT_GPI_5	Not Input 5
	CC_GPI_6	Input 6
	CC_NOT_GPI_6	Not Input 6
	CC_GPI_7	Input 7
	CC_NOT_GPI_7	Not Input 7
	CC_FRONT_GPI_0	Front Input 0
	CC_NOT_FRONT_GPI_0	Not Front Input 0
	CC_FRONT_GPI_1	Front Input 1
	CC_NOT_FRONT_GPI_1	Not Front Input 1
	CC_FRONT_GPI_2	Front Input 2
	CC_NOT_FRONT_GPI_2	Not Front Input 2
	CC_FRONT_GPI_3	Front Input 3
	CC_NOT_FRONT_GPI_3	Not Front Input 3
	FG_OTHER	Other
Default value	**CC_VCC**	

Example 19.3. Usage of FG_CCSEL2

```
int result = 0;
int value = CC_VCC;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;

if ((result = Fg_setParameterWithType(fg, FG_CCSEL2, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CCSEL2, &value, 0, type)) < 0) {
```

```
    /* error handling */
}
```

19.4. FG_CCSEL3

Select one of the options described above.

Table 19.4. Parameter properties of FG_CCSEL3

Property	Value	
Name	**FG_CCSEL3**	
Display Name	**CC 4**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**CC_EXSYNC**	Exsync
	CC_NOT_EXSYNC	!Exsync
	CC_PRESCALER	Prescaler
	CC_NOT_PRESCALER	!Prescaler
	CC_STROBEPULSE	Flash
	CC_NOT_STROBEPULSE	!Flash
	CC_GND	Gnd
	CC_VCC	Vcc
	CC_GPI_0	Input 0
	CC_NOT_GPI_0	Not Input 0
	CC_GPI_1	Input 1
	CC_NOT_GPI_1	Not Input 1
	CC_GPI_2	Input 2
	CC_NOT_GPI_2	Not Input 2
	CC_GPI_3	Input 3
	CC_NOT_GPI_3	Not Input 3
	CC_GPI_4	Input 4
	CC_NOT_GPI_4	Not Input 4
	CC_GPI_5	Input 5
	CC_NOT_GPI_5	Not Input 5
	CC_GPI_6	Input 6
	CC_NOT_GPI_6	Not Input 6
	CC_GPI_7	Input 7
	CC_NOT_GPI_7	Not Input 7
	CC_FRONT_GPI_0	Front Input 0
	CC_NOT_FRONT_GPI_0	Not Front Input 0
	CC_FRONT_GPI_1	Front Input 1
	CC_NOT_FRONT_GPI_1	Not Front Input 1
	CC_FRONT_GPI_2	Front Input 2
	CC_NOT_FRONT_GPI_2	Not Front Input 2
	CC_FRONT_GPI_3	Front Input 3
	CC_NOT_FRONT_GPI_3	Not Front Input 3
	FG_OTHER	Other
Default value	**CC_VCC**	

Example 19.4. Usage of FG_CCSEL3

```
int result = 0;
int value = CC_VCC;
const enum FgParamTypes type = FG_PARAM_TYPE_INT32_T;
```

```
if ((result = Fg_setParameterWithType(fg, FG_CCSEL3, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_CCSEL3, &value, 0, type)) < 0) {
    /* error handling */
}
```

Glossary

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

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

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

Index

A
Area of Interest, 14

B
Bandwidth, 2

C
Camera, 10
  Events, 10
  Format, 6
  Interface, 4, 10
Camera Link, 6, 6
Camera Simulator, 88, 88
Camera Trigger Source, 18, 21, 22
Camera::Events, 10
CC Signal Mapping, 27

D
Debugging, 61
Digital I/O, 18, 18
Digital I/O::Event Source, 22
Digital I/O::Events, 25
Digital I/O::GPI, 21
Digital I/O::GPO, 18

E
Events
  Camera, 10
  Overflow, 68
  Trigger, 25

F
Features, 1
FG_ALTERNATIVE_BOARD_DETECTION, 107
FG_APPLET_BUILD_TIME, 99
FG_APPLET_ID, 99
FG_APPLET_REVISION, 99
FG_APPLET_VERSION, 98
FG_BITALIGNMENT, 85
FG_CAMERASIMULATOR_ACTIVE, 96
FG_CAMERASIMULATOR_ENABLE, 88
FG_CAMERASIMULATOR_FRAMERATE, 95
FG_CAMERASIMULATOR_FRAME_GAP, 91
FG_CAMERASIMULATOR_HEIGHT, 90
FG_CAMERASIMULATOR_LINERATE, 94
FG_CAMERASIMULATOR_LINE_GAP, 89
FG_CAMERASIMULATOR_PASSIVE, 97
FG_CAMERASIMULATOR_PATTERN, 91
FG_CAMERASIMULATOR_PATTERN_OFFSET, 92
FG_CAMERASIMULATOR_PIXEL_FREQUENCY, 94
FG_CAMERASIMULATOR_ROLL, 93
FG_CAMERASIMULATOR_SELECT_MODE, 93
FG_CAMERASIMULATOR_TRIGGER_MODE, 96
FG_CAMERASIMULATOR_WIDTH, 89

FG_CAMERA_LINK_CAMTYPE, 6
FG_CAMERA_LINK_CORE_RESET, 7
FG_CAMERA_LINK_PIXEL_CLOCK_X, 8
FG_CAMERA_LINK_PIXEL_CLOCK_Y, 8
FG_CAMSTATUS, 101
FG_CAMSTATUS_EXTENDED, 101
FG_CC1_POLARITY, 28
FG_CC1_SOURCE, 28
FG_CC2_POLARITY, 29
FG_CC2_SOURCE, 28
FG_CC3_POLARITY, 29
FG_CC3_SOURCE, 28
FG_CC4_POLARITY, 29
FG_CC4_SOURCE, 28
FG_CCSEL0, 119
FG_CCSEL1, 120
FG_CCSEL2, 122
FG_CCSEL3, 123
FG_CUSTOM_BIT_SHIFT_RIGHT, 86
FG_CUSTOM_SIGNAL_EVENT_0, 25
FG_CUSTOM_SIGNAL_EVENT_0_POLARITY, 23
FG_CUSTOM_SIGNAL_EVENT_0_SOURCE, 22
FG_CUSTOM_SIGNAL_EVENT_1, 25
FG_CUSTOM_SIGNAL_EVENT_1_POLARITY, 24
FG_CUSTOM_SIGNAL_EVENT_1_SOURCE, 23
FG_DEBUGFILE, 114
FG_DEBUGINENABLE, 113
FG_DEBUGINSERT, 114
FG_DEBUGOUTENABLE, 116
FG_DEBUGOUTPIXEL, 118
FG_DEBUGOUTXPOS, 117
FG_DEBUGOUTYPOS, 117
FG_DEBUGREADY, 116
FG_DEBUGSAVECONFIG, 112
FG_DEBUGSOURCE, 112
FG_DEBUGSOURCENAME, 112
FG_DEBUGWRITEFLAG, 115
FG_DEBUGWRITEPIXEL, 115
FG_DEBUG_VERSION, 113
FG_DIGIO_INPUT, 21
FG_DIGIO_OUTPUT, 20
FG_DMASTATUS, 100
FG_END_OF_FRAME_CAM_PORT_0, 10
FG_END_OF_LINE_CAM_PORT_0, 10
FG_EXSYNCON, 31
FG_EXSYNCPOLARITY, 50
FG_FILLLEVEL, 67
FG_FLASHON, 54
FG_FLASH_POLARITY, 59
FG_FORMAT, 84
FG_HAP_FILE, 100
FG_HEIGHT, 15
FG_IMGTRIGGERDEBOUNCING, 58
FG_IMGTRIGGERGATEDELAY, 57
FG_IMGTRIGGERINPOLARITY, 56
FG_IMGTRIGGERINSRC, 56
FG_IMGTRIGGERMODE, 53
FG_IMGTRIGGERON, 53

FG_IMGTRIGGER_ASYNC_HEIGHT, 54
FG_IMGTRIGGER_IS_BUSY, 55
FG_IMG_SELECT, 72
FG_IMG_SELECT_PERIOD, 71
FG_LINEEXPOSURE, 49
FG_LINEPERIODE, 48
FG_LINETRIGGERDEBOUNCING, 35
FG_LINETRIGGERDELAY, 51
FG_LINETRIGGERINPOLARITY, 34
FG_LINETRIGGERINSRC, 33
FG_LINETRIGGERMODE, 30
FG_LINE_DOWNSCALE, 35
FG_LINE_DOWNSCALEINIT, 36
FG_LUT_CUSTOM_FILE, 74
FG_LUT_IMPLEMENTATION_TYPE, 77
FG_LUT_IN_BITS, 77
FG_LUT_OUT_BITS, 77
FG_LUT_SAVE_FILE, 76
FG_LUT_TYPE, 73
FG_LUT_VALUE, 74
FG_OVERFLOW, 68
FG_OVERFLOW_CAM0, 68
FG_PIXELDEPTH, 86
FG_PROCESSING_GAIN, 80
FG_PROCESSING_GAMMA, 81
FG_PROCESSING_INVERT, 82
FG_PROCESSING_OFFSET, 80
FG_SENDSOFTWARETRIGGER, 59
FG_SENSORWIDTH, 11
FG_SETSOFTWARETRIGGER, 60
FG_SHAFTENCODERINSRC, 39
FG_SHAFTENCODERLEADING, 40
FG_SHAFTENCODERMODE, 38
FG_SHAFTENCODERON, 37
FG_SHAFTENCODER_COMPENSATION_COUNT, 42
FG_SHAFTENCODER_COMPENSATION_ENABLE, 41
FG_SIGNAL_ANALYZER_0_PERIOD_CURRENT, 63
FG_SIGNAL_ANALYZER_0_PERIOD_MAX, 64
FG_SIGNAL_ANALYZER_0_PERIOD_MIN, 64
FG_SIGNAL_ANALYZER_0_POLARITY, 62
FG_SIGNAL_ANALYZER_0_PULSE_COUNT, 65
FG_SIGNAL_ANALYZER_0_SOURCE, 61
FG_SIGNAL_ANALYZER_1_PERIOD_CURRENT, 63
FG_SIGNAL_ANALYZER_1_PERIOD_MAX, 64
FG_SIGNAL_ANALYZER_1_PERIOD_MIN, 64
FG_SIGNAL_ANALYZER_1_POLARITY, 62
FG_SIGNAL_ANALYZER_1_PULSE_COUNT, 65
FG_SIGNAL_ANALYZER_1_SOURCE, 61
FG_SIGNAL_ANALYZER_CLEAR, 66
FG_SIGNAL_ANALYZER_PULSE_COUNT_DIFFERENCE, 65
FG_START_OF_FRAME_CAM_PORT_0, 10
FG_START_OF_LINE_CAM_PORT_0, 10
FG_STROBEPULSEDELAY, 58
FG_SYSTEMMONITOR_CURRENT_LINK_SPEED, 104
FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH, 104
FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT, 106
FG_SYSTEMMONITOR_FPGA_DNA, 111
FG_SYSTEMMONITOR_FPGA_TEMPERATURE, 102

FG_SYSTEMMONITOR_FPGA_VCC_AUX, 103
FG_SYSTEMMONITOR_FPGA_VCC_BRAM, 103
FG_SYSTEMMONITOR_FPGA_VCC_INT, 102
FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE, 105
FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE, 105
FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE, 106
FG_SYSTEMMONITOR_POCL_STATE_PORT_A, 107
FG_SYSTEMMONITOR_POCL_STATE_PORT_B, 110
FG_TAPGEOMETRY, 11
FG_TIMEOUT, 98
FG_TRIGGEROUT_FRONT_GPO_0_POLARITY, 19
FG_TRIGGEROUT_FRONT_GPO_0_SOURCE, 18
FG_TRIGGEROUT_FRONT_GPO_1_POLARITY, 20
FG_TRIGGEROUT_FRONT_GPO_1_SOURCE, 18
FG_TRIGGEROUT_GPO_0_POLARITY, 19
FG_TRIGGEROUT_GPO_0_SOURCE, 18
FG_TRIGGEROUT_GPO_1_POLARITY, 19
FG_TRIGGEROUT_GPO_1_SOURCE, 18
FG_TRIGGEROUT_GPO_2_POLARITY, 19
FG_TRIGGEROUT_GPO_2_SOURCE, 18
FG_TRIGGEROUT_GPO_3_POLARITY, 19
FG_TRIGGEROUT_GPO_3_SOURCE, 18
FG_TRIGGEROUT_GPO_4_POLARITY, 19
FG_TRIGGEROUT_GPO_4_SOURCE, 18
FG_TRIGGEROUT_GPO_5_POLARITY, 19
FG_TRIGGEROUT_GPO_5_SOURCE, 18
FG_TRIGGEROUT_GPO_6_POLARITY, 19
FG_TRIGGEROUT_GPO_6_SOURCE, 18
FG_TRIGGEROUT_GPO_7_POLARITY, 19
FG_TRIGGEROUT_GPO_7_SOURCE, 18
FG_TRIGGER_INPUT0_FALLING, 25
FG_TRIGGER_INPUT0_RISING, 25
FG_USEDVAL, 6
FG_VANTAGEPOINT, 12
FG_WIDTH, 15
FG_XOFFSET, 16
FG_YOFFSET, 17
Format, 84

G

Generator, 88

I

Image Select, 71
Image Selector, 71
Image Transfer, 4
Image Trigger / Flash, 52
Image Trigger / Flash::Image Trigger Input, 56
Image Trigger / Flash::Image Trigger Input::Flash, 59
Image Trigger / Flash::Image Trigger Input::Software Trigger, 59

L

Legacy Parameter, 119
Line Trigger / ExSync, 30
Line Trigger / ExSync::ExSync Output, 48
Line Trigger / ExSync::Line Trigger Input, 32
Line Trigger / ExSync::Line Trigger Input::Downscale, 35
Line Trigger / ExSync::Shaft Encoder A/B Filter, 37

Lookup Table, 73, 73
Lookup Table::Applet Properties, 76

M

Miscellaneous, 98
Miscellaneous::Debug, 111
Miscellaneous::Debug::Input, 113
Miscellaneous::Debug::Output, 116

O

Output Format, 84
Overflow, 67, 67
  Events, 68
Overflow::Events, 68

P

PC Interface, 4
Processing, 79
Processor, 79

R

Region of Interest, 14
ROI, 14

S

Sensor Correction , 11
Signal Analyzer, 61, 61
Software Interface, 5
Specifications, 1

T

Tap Geometry, 11, 11
Tap Sorting, 11
Trigger
  Digital Input, 21
  Events, 25
  Input, 21