SILICONSOFTWARE GmbH

User documentation

AcquisitionApplets Advanced

Acq_MediumAreaRGB36

for microEnable IV AD4-CL

AcquisitionApplets Advanced

User documentation

Acq_MediumAreaRGB36
for microEnable IV AD4-CL

Applet Characteristics

Applet Name	Acq_MediumAreaRGB36
Applet Version	1.0
Type of Applet	AcquisitionApplets Advanced
Frame Grabber	microEnable IV AD4-CL
No. of Cameras	1
Camera Interface	Camera Link MEDIUM Configuration
Sensor Type	Area Scan
Color	RGB
Processing Bit Depth	12 Bit per Color Component

Supplemental Information

1. AcquisitionApplets Advanced Release Notes: Please refer to the Runtime Software Documentation, section *AcquisitionApplets Advanced - Release Notes*
2. Runtime Software Documentation: Is part of the Runtime Software Installation [InstDir]/doc
3. Software Updates and Support (engl.): www.Silicon-Software.com/download.html
4. Software Updates and Support (dt.): www.Silicon-Software.de/download.html

Silicon Software Headquarter: Steubenstrasse 46 - D-68163 Mannheim - Germany
Tel. +49 (0)621 - 789 50 70, Email: info@silicon-software.com

Copyright © 2012 Silicon Software GmbH

All rights reserved. All trademarks and registered trademarks are the property of their respective owners. Any information without obligation. Technical specifications and scope of delivery are liability-free and valid until revocation. Changes and errors are excepted.

Table of Contents

1. Introduction 1
    1.1. Silicon Software AcquisitionApplets Advanced 1
    1.2. Features of Applet Acq_MediumAreaRGB36 2
        1.2.1. Parameterization Order 4
    1.3. Bandwidth 5
    1.4. Camera Interface 5
    1.5. Image Transfer to PC Memory 6
2. Software Interface 7
3. Camera Link 8
    3.1. FG_CAMERA_LINK_CAMTYPE 8
    3.2. FG_USEDVAL 8
4. ROI 10
    4.1. FG_WIDTH 11
    4.2. FG_HEIGHT 12
    4.3. FG_XOFFSET 12
    4.4. FG_YOFFSET 13
5. Trigger 14
    5.1. Features and Functional Blocks of Area Trigger 14
    5.2. Pin allocation 17
    5.3. Event Overview 18
    5.4. Trigger Scenarios 19
        5.4.1. Internal Frequency Generator 19
        5.4.2. External Trigger Signals 21
        5.4.3. Control of Three Flash Lights 25
        5.4.4. Software Trigger 29
        5.4.5. Software Trigger with Trigger Queue 34
        5.4.6. External Trigger with Trigger Queue 36
        5.4.7. Bypass External Trigger Signals 37
        5.4.8. Multi Camera Applications 37
        5.4.9. Hardware System Analysis and Error Detection 38
    5.5. Parameters 39
        5.5.1. FG_AREATRIGGERMODE 39
        5.5.2. FG_TRIGGERSTATE 40
        5.5.3. FG_TRIGGER_FRAMESPERSECOND 41
        5.5.4. FG_TRIGGER_EXCEEDED_PERIOD_LIMITS 42
        5.5.5. FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CLEAR 43
        5.5.6. FG_TRIGGER_LEGACY_MODE 43
        5.5.7. FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CAM0 44
        5.5.8. Legacy 44
            5.5.8.1. FG_TRIGGERMODE 44
            5.5.8.2. FG_EXSYNCON 45
            5.5.8.3. FG_FLASHON 45
            5.5.8.4. FG_EXPOSURE 46
            5.5.8.5. FG_EXSYNCDELAY 46
            5.5.8.6. FG_EXSYNCPOLARITY 47
            5.5.8.7. FG_STROBEPULSEDELAY 47
            5.5.8.8. FG_FLASH_POLARITY 48
            5.5.8.9. FG_PRESCALER 49
            5.5.8.10. FG_CCSEL0 et al. 49
            5.5.8.11. FG_DIGIO_OUTPUT 50
        5.5.9. Trigger Input 50
            5.5.9.1. External 50
                5.5.9.1.1. FG_TRIGGERIN_DEBOUNCE 50

5.5.9.1.2. FG_DIGIO_INPUT 52
5.5.9.1.3. FG_TRIGGERIN_SRC 52
5.5.9.1.4. FG_TRIGGERIN_POLARITY 53
5.5.9.1.5. FG_TRIGGERIN_DOWNSCALE 53
5.5.9.1.6. FG_TRIGGERIN_DOWNSCALE_PHASE 54
5.5.9.1.7. FG_TRIGGERIN_BYPASS_SRC 55
5.5.9.2. Software Trigger 56
5.5.9.2.1. FG_SENDSOFTWARETRIGGER 56
5.5.9.2.2. FG_SOFTWARETRIGGER_IS_BUSY 57
5.5.9.2.3. FG_SOFTWARETRIGGER_QUEUE_FILLLEVEL 57
5.5.9.3. Statistics 58
5.5.9.3.1. FG_TRIGGERIN_STATS_PULSECOUNT 58
5.5.9.3.2. FG_TRIGGERIN_STATS_PULSECOUNT_CLEAR 58
5.5.9.3.3. FG_TRIGGERIN_STATS_FREQUENCY 59
5.5.9.3.4. FG_TRIGGERIN_STATS_MINFREQUENCY 59
5.5.9.3.5. FG_TRIGGERIN_STATS_MAXFREQUENCY 60
5.5.9.3.6. FG_TRIGGERIN_STATS_MINMAXFREQUENCY_CLEAR 60
5.5.9.3.7. FG_TRIGGER_INPUT0_RISING et al. 61
5.5.9.3.8. FG_TRIGGER_INPUT0_FALLING et al. 61
5.5.10. Sequencer 61
5.5.10.1. FG_TRIGGER_MULTIPLY_PULSES 62
5.5.11. Queue 62
5.5.11.1. FG_TRIGGERQUEUE_MODE 63
5.5.11.2. FG_TRIGGERQUEUE_FILLLEVEL 63
5.5.11.3. FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_ON_THRESHOLD 64
5.5.11.4. FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_OFF_THRESHOLD 64
5.5.11.5. FG_TRIGGER_QUEUE_FILLLEVEL_THRESHOLD_CAM0_ON 65
5.5.11.6. FG_TRIGGER_QUEUE_FILLLEVEL_THRESHOLD_CAM0_OFF 65
5.5.12. Pulse Form Generator 0 65
5.5.12.1. FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE et al. 66
5.5.12.2. FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE_PHASE et al. 67
5.5.12.3. FG_TRIGGER_PULSEFORMGEN0_DELAY et al. 68
5.5.12.4. FG_TRIGGER_PULSEFORMGEN0_WIDTH et al. 69
5.5.13. Pulse Form Generator 1 69
5.5.14. Pulse Form Generator 2 70
5.5.15. Pulse Form Generator 3 70
5.5.16. CC Signal Mapping 70
5.5.16.1. FG_TRIGGERCC_SELECT0 et al. 70
5.5.17. Digital Output 71
5.5.17.1. FG_TRIGGEROUT_SELECT0 et al. 72
5.5.17.2. Statistics 72
5.5.17.2.1. FG_TRIGGEROUT_STATS_SOURCE 72
5.5.17.2.2. FG_TRIGGEROUT_STATS_PULSECOUNT 73
5.5.17.2.3. FG_TRIGGEROUT_STATS_PULSECOUNT_CLEAR 73
5.5.17.2.4. FG_MISSING_CAMERA_FRAME_RESPONSE 74
5.5.17.2.5. FG_MISSING_CAMERA_FRAME_RESPONSE_CLEAR 75
5.5.17.2.6. FG_MISSING_CAM0_FRAME_RESPONSE 76

5.5.18. Output Event 76
5.5.18.1. FG_TRIGGER_OUTPUT_EVENT_SELECT 76
5.5.18.2. FG_TRIGGER_OUTPUT_CAM0 77
6. Overflow 78
6.1. FG_FILLLEVEL 78
6.2. FG_OVERFLOW 79
6.3. FG_OVERFLOW_CAM0 80
7. Image Selector 82
7.1. FG_IMG_SELECT_PERIOD 83
7.2. FG_IMG_SELECT 83
8. Shading Correction 85
8.1. FG_SHADING_OFFSET_ENABLE 86
8.2. FG_SHADING_GAIN_ENABLE 87
8.3. Auto Shading Correction 87
8.3.1. FG_SHADING_BLACK_FILENAME 88
8.3.2. FG_SHADING_GRAY_FILENAME 88
8.3.3. FG_SHADING_GAIN_CORRECTION_MODE 89
8.3.4. FG_SHADING_APPLY_SETTINGS 89
9. Shading Custom Values 91
10. White Balance 93
10.1. FG_SCALINGFACTOR_RED 93
10.2. FG_SCALINGFACTOR_BLUE 93
10.3. FG_SCALINGFACTOR_GREEN 94
11. Lookup Table 95
11.1. FG_LUT_TYPE 95
11.2. FG_LUT_VALUE_RED 96
11.3. FG_LUT_VALUE_GREEN 96
11.4. FG_LUT_VALUE_BLUE 97
11.5. FG_LUT_CUSTOM_FILE 97
11.6. FG_LUT_SAVE_FILE 98
11.7. Applet Properties 99
11.7.1. FG_LUT_IMPLEMENTATION_TYPE 99
11.7.2. FG_LUT_IN_BITS 99
11.7.3. FG_LUT_OUT_BITS 99
12. Processing 101
12.1. FG_PROCESSING_OFFSET 102
12.2. FG_PROCESSING_GAIN 102
12.3. FG_PROCESSING_GAMMA 103
12.4. FG_PROCESSING_INVERT 104
13. Output Format 106
13.1. FG_FORMAT 106
13.2. FG_BITALIGNMENT 106
13.3. FG_PIXELDEPTH 107
14. Camera Simulator 108
14.1. FG_GEN_ENABLE 108
14.2. FG_GEN_START 109
14.3. FG_GEN_WIDTH 109
14.4. FG_GEN_HEIGHT 110
14.5. FG_GEN_LINE_GAP 111
14.6. FG_GEN_FREQ 111
14.7. FG_GEN_ACCURACY 112
14.8. FG_GEN_TAP1 et al. 113
14.9. FG_GEN_ROLL 113
14.10. FG_GEN_ACTIVE 114
14.11. FG_GEN_PASSIVE 114

14.12. FG_GEN_LINE_WIDTH 115
15. Miscellaneous 116
15.1. FG_TIMEOUT 116
15.2. FG_APPLET_VERSION 116
15.3. FG_APPLET_REVISION 117
15.4. FG_APPLET_ID 117
15.5. FG_HAP_FILE 117
15.6. FG_DMASTATUS 118
15.7. FG_CAMSTATUS 118
15.8. FG_CAMSTATUS_EXTENDED 119
16. Revision History 120
Glossary 121
Index 123

Chapter 1. Introduction

This document provides detailed information on the Silicon Software AcquisitionApplets Advanced "Acq_MediumAreaRGB36" for microEnable IV AD4-CL frame grabbers. Starting with an explanation on Silicon Software AcquisitionApplets Advanced in general, the document will outline the features and benefits of this AcquisitionApplets Advanced. 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 AcquisitionApplets Advanced adaptable for numerous applications.

1.1. Silicon Software AcquisitionApplets Advanced

AcquisitionApplets Advanced is an AcquisitionApplets set incorporating ready-to-use, real-time image processing functions. Each single applet comprises a specific number and arrangement of image processing functions, realized as a data flow, supporting a certain camera interface implementation. Depending on the target camera there is a subset of applets available, representing suitable image processing functions. Target hardware for AcquisitionApplets Advanced for Camera Link applications are the following A-Series and V-Series frame grabber:

- microEnable IV AD4-CL / -PoCL
- microEnable IV VD4-CL / -PoCL

Image processing highlights are:

- Shading Correction including Dead Pixel Interpolation (DPI)
- Image Selector
- Noise Filter
- Trigger Functions
- Region of Interest (RoI)
- High-Quality Bayer Pattern de-mosaicing and White Balancing
- Look-Up Table

The AcquisitionApplets Advanced set does not substitute the well-established AcquisitionApplets set, introduced some years ago. AcquisitionApplets Advanced can be used in the same manner as all other Silicon Software applet types:

- You can load Applets onto a frame grabber and control (start, stop and configuration) a certain acquisition setup using the viewer software microDisplay,

  OR

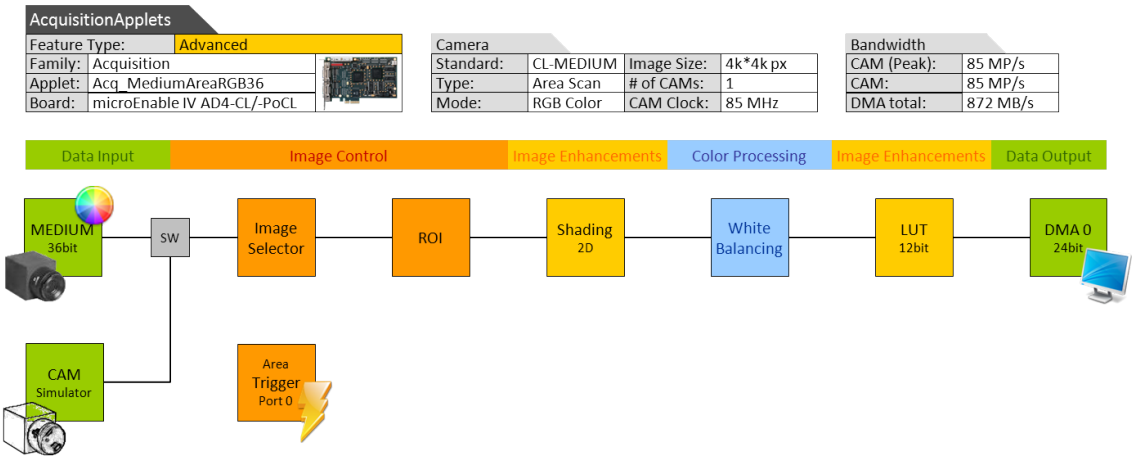
- You can load Applets onto a frame grabber and control (start, stop and configuration) a certain acquisition setup using functions of the Silicon Software API, the powerful Software Development Kit (SDK).

Figure 1.1. Silicon Software AcquisitionApplets Advanced

	Board	Img. Size	CamPort		Bit Depth		Image Pre-processing							Bandwidth			CamClk
	mE4	Max.			Proc.	Output	Image Select.	Noise Filter	Shading Corr.	Dead Pixel Interpol.	Bayer Color Recon.	White Balance	LUT	Peak per CAM	per CAM	per DMA	Max.
	CL	[Px]	#	Type	[bit]	[bit]		Median	Offset + Gain		HQ			[MP/s]	[MP/s]	[MB/s]	[MHz]
Area Scan																	
Acq_BaseAreaBayer12	AD4	4k*4k	1	BASE	12	24/48	yes		2D		5*5	Yes	Full Res	248	225	872	85
Acq_DualBaseAreaBayer8	AD4	4k*4k	2	BASE	8	24	Yes		2D		5*5	Yes	Full Res	170	170	872	85
Acq_DualBaseAreaGray12	AD4	4k*4k	2	BASE	12	8/16	Yes	3*3	2D	Yes			Full Res	248	225	872	85
Acq_DualBaseAreaRGB24	AD4	4k*4k	2	BASE	24	24/48	Yes		2D*			Yes	Full Res	85	85	872	85
Acq_MediumAreaBayer12	AD4	4k*4k	1	MEDIUM	12	24/48	Yes		2D		5*5	Yes	Full Res	340	288	872	85
Acq_MediumAreaGray12	AD4	4k*4k	1	MEDIUM	12	8/16	Yes	3*3	2D	Yes			Full Res	340	340	872	85
Acq_MediumAreaRGB36	AD4	4k*4k	1	MEDIUM	36	24/48	Yes		2D			Yes	Full Res	85	85	872	85
Acq_FullAreaBayer8	AD4	4k*4k	1	FULL	8	24/48	Yes		2D*		5*5	Yes	Full Res	850	288	872	85
Acq_FullAreaGray8	AD4	4k*4k	1	FULL	8	8/16	Yes	3*3	2D	Yes			Full Res	850	850	872	85
Acq_BaseAreaBayer12	VD4	4k*4k	1	BASE	12	24/48	Yes		2D		5*5	Yes	Full Res	248	225	872	85
Acq_DualBaseAreaBayer8	VD4	4k*4k	2	BASE	8	24	Yes		2D		5*5	Yes	Full Res	170	170	872	85
Acq_DualBaseAreaGray12	VD4	4k*4k	2	BASE	12	8/16	Yes	3*3	2D	Yes			Full Res	248	225	872	85
Acq_DualBaseAreaRGB24	VD4	4k*4k	2	BASE	24	24/48	Yes		2D			Yes	Full Res	85	85	872	85
Acq_MediumAreaBayer12	VD4	4k*4k	1	MEDIUM	12	24/48	Yes		2D		5*5	Yes	Full Res	340	288	872	85
Acq_MediumAreaGray12	VD4	4k*4k	1	MEDIUM	12	8/16	Yes	3*3	2D	Yes			Full Res	340	340	872	85
Acq_MediumAreaRGB36	VD4	4k*4k	1	MEDIUM	36	24/48	Yes		2D			Yes	Full Res	85	85	872	85
Acq_FullAreaBayer8	VD4	4k*4k	1	FULL	8	24/48	Yes		2D		5*5	Yes	Full Res	850	288	872	85
Acq_FullAreaGray8	VD4	4k*4k	1	FULL	8	8/16	Yes	3*3	2D	Yes			Full Res	850	850	872	85

1.2. Features of Applet Acq_MediumAreaRGB36

Figure 1.2. Block Diagram of AcquisitionApplets Advanced Acq_MediumAreaRGB36



AcquisitionApplets Advanced "Acq_MediumAreaRGB36" is a single camera applet. The Camera Link camera interface can be configured for Camera Link cameras in Medium-Configuration mode transferring RGB pixels at bit depths between 30 and 36 bits. A multi-functional area trigger is included to control the camera or external devices using grabber generated, external or software generated trigger pulses. Area scan cameras transferring images with a resolution of up to 4096 by 4096 pixels are supported. 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 grabbers and PCs. This AcquisitionApplets Advanced is equipped with a two-dimensional area shading correction. It consists of a subtractive offset correction

and a multiplicative gain correction. Correction values can be defined for each pixel. Moreover an auto-shading correction can be performed by loading black and gray reference images into the shading module. The shading correction is using 16 Bit for offset correction including 4 fraction bits. The gain correction also uses 16 Bits, where 12 Bits are used for fractional representation. Acquired images are buffered in frame grabber memory. A ROI can be selected for further processing. The stepsize of the ROI width is 4 pixel and the ROI stepsize for the image height is 1 lines. A white balancing module for individual gain of each color component enhances image quality. The full resolution lookup table can either be configured by using a user defined table or by using a processor. The processor gives the opportunity to use pre-defined functions such as offset, gain, and gamma to enhance the image quality. The color components are processed individually.

Processed image data is output via a high speed DMA channel. The output pixel format is 24 bits or 48 bits per pixel in BGR color component order.

The applet can easily be included into user applications using the Silicon Software runtime.

Table 1.1. Feature Summary of Acq_MediumAreaRGB36

Feature	Applet Property
Applet Name	Acq_MediumAreaRGB36
Type of Applet	AcquisitionApplets Advanced
Board	microEnable IV AD4-CL
No. of Cameras	1 , asynchronous or synchronous
Camera Type	Camera Link Medium Configuration (30 Bit RGB, 36 Bit RGB)
Sensor Type	Area Scan
Camera Format	RGB
Processing Bit Depth	12 Bit per color component
Maximum Images Dimensions	4096 * 4096
ROI Stepsize	x: 4, y: 1
Image Selector	Yes
Noise Filter	No
Shading Correction	Yes, Area , Half X-Resolution Offset Correction Value Range: [0, 4096[, Stepsize 1/16 (12 integer bits, 4 fractional bits) Gain Correction Value Range: [0, 16[, Stepsize 1/4096 (4 integer bits, 12 fractional bits)
Dead Pixel Interpolation	No
Bayer Filter	No
Color White Balancing	Yes
Lookup Table	Full Resolution
DMA	High Speed
DMA Image Output Format	BGR 24 or 48 bits
Event Generation	Yes
Overflow Control	Yes

1.2.1. Parameterization Order

It is recommended to configure the functional blocks which are responsible for sensor correction first. This will be the camera settings, shading correction and dead pixel interpolation, if available. Afterwards, other image enhancement functional blocks can be configured, such as the white balancing, noise filter and lookup table.

1.3. Bandwidth

The maximum bandwidths of applet Acq_MediumAreaRGB36 are listed in the following table.

Table 1.2. Bandwidth of Acq_MediumAreaRGB36

Description	Bandwidth
Max. CamClk	85 MHz
Peak Bandwidth per Camera	85 MPixel/s
Mean Bandwidth per Camera	85 MPixel/s
DMA Bandwidth	872 MByte/s (depends on PC mainboard)

Max. CamClk refers to the maximum pixel clock frequency used by the the Camera Link camera. You can use any lower pixel clock frequency.

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 being processed at that speed. The peak bandwidth is related to the Canera 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 maximum allowed mean bandwidth for each camera at the camera interface. It is the product of the framerate and the image pixels. For example, for a 1 Megapixel image and a framerate of 100 fps, the mean bandwidth will be 100 MPixel/s.

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 depends on the applet and the parameterization. 1 MByte is 1,000,000 Byte.

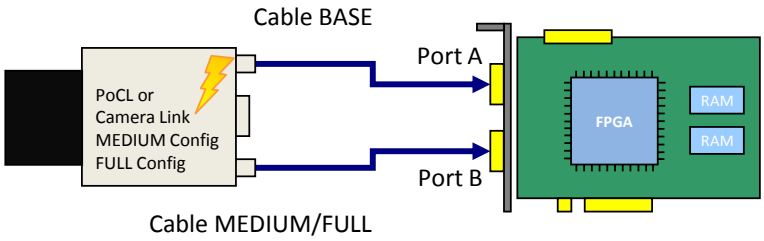
Bandwidth Varies

The exact maximum DMA bandwidth depends on the used PC system. The camera bandwidth depends on the image size. The given values might vary.

1.4. Camera Interface

AcquisitionApplets Advanced "Acq_MediumAreaRGB36" 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 1.3. Camera Interface and Camera Cable Setup



1.5. Image Transfer to PC Memory

Image transfer between the frame grabber and the PC is performed by DMA transfers. In this applet, only one DMA channel exists to transfer the image data. The DMA channel has index 0. The applet output format can be set with the parameters of the output format module. See Chapter 13, *Output Format*. All outputs are little endian coded.

DMA Image Tag

The applet does not generate a valid DMA image tag (*FG_IMAGE_TAG*). Check for lost or corrupted frames using the overflow module described in Chapter 6, *Overflow*. Moreover the trigger system offers possibilities to detect lost trigger signals. See Chapter 5, *Trigger* for more information.

Chapter 2. Software Interface

The software interface is fully compatible to the Silicon Software runtime. Please read the SDK manual of the Silicon Software runtime software to understand how to include the microEnable frame grabbers and their applets into own applications.

The runtime includes functional SDK examples which use the features of the runtime. Most of these examples can be used with this AcquisitionApplets Advanced. 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 chapter will list all parameters of this applet. Keep in mind that for multi-camera applets, parameters are can be set for all cameras individually. The sample source codes in parameterize the processing components of the first camera. The index in the source code examples has to be changed for the other cameras.

Amongest 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 Advanced 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 Advanced can be used with one 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.

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 Advanced,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**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_CL_MEDIUM_RGB_30** 30bit RGB Medium **FG_CL_MEDIUM_RGB_36** 36bit RGB Medium
Default value	**FG_CL_MEDIUM_RGB_30**

Example 3.1. Usage of FG_CAMERA_LINK_CAMTYPE

```
int result = 0;
unsigned int value = FG_CL_MEDIUM_RGB_30;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

if ((result = Fg_getParameterWithType(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**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_YES** Yes **FG_NO** No
Default value	**FG_YES**

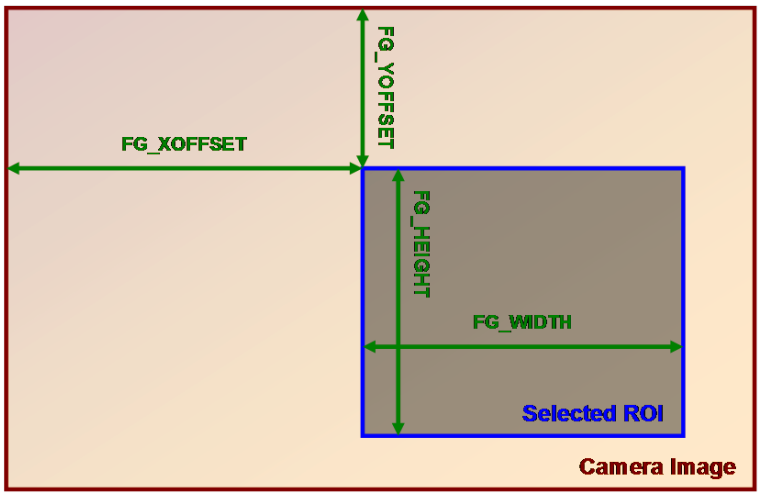
Example 3.2. Usage of FG_USEDVAL

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

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

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

Chapter 4. ROI

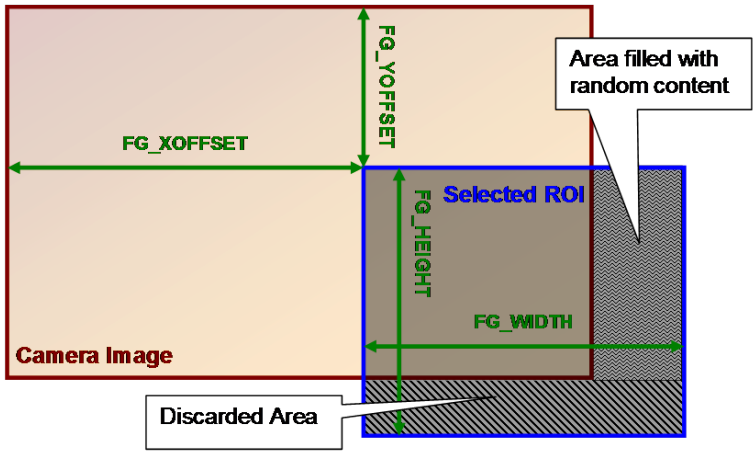
This module allows the definition of a region of interest (ROI), also called area of interest (AOI). A ROI allows the selection of a smaller subset pixel area from the input image. It is defined by using parameters *FG_XOFFSET*, *FG_WIDTH*, *FG_YOFFSET* and *FG_HEIGTH*. The following figure illustrates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

Figure 4.1. 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 grater 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 4.2. Region of Interest Selection Outside the Input Image Dimensions



Furthermore, mind that the image send by the camera must not exceed the maximum allowed image dimensions. This applet allows a maximum image width of 4096 pixels and a maximum image height of 4096 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 4 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.

4.1. 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 4096 pixels.

Table 4.1. Parameter properties of FG_WIDTH

Property	Value
Name	**FG_WIDTH**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum 4** **Maximum 4096** **Stepsize 4**
Default value	**1024**
Unit of measure	**pixel**

Example 4.1. 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_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}

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

4.2. 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 4096 pixels.

Table 4.2. Parameter properties of FG_HEIGHT

Property	Value
Name	**FG_HEIGHT**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum 2** **Maximum 4096** **Stepsize 1**
Default value	**1024**
Unit of measure	**pixel**

Example 4.2. 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_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

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

4.3. FG_XOFFSET

The x-offset is defined by this parameter.

Table 4.3. Parameter properties of FG_XOFFSET

Property	Value
Name	**FG_XOFFSET**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 4092** **Stepsize 4**
Default value	**0**
Unit of measure	**pixel**

Example 4.3. 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_XOFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

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

4.4. FG_YOFFSET

The y-offset is defined by this parameter.

Table 4.4. Parameter properties of FG_YOFFSET

Property	Value
Name	**FG_YOFFSET**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 4095** **Stepsize 1**
Default value	**0**
Unit of measure	**pixel**

Example 4.4. Usage of FG_YOFFSET

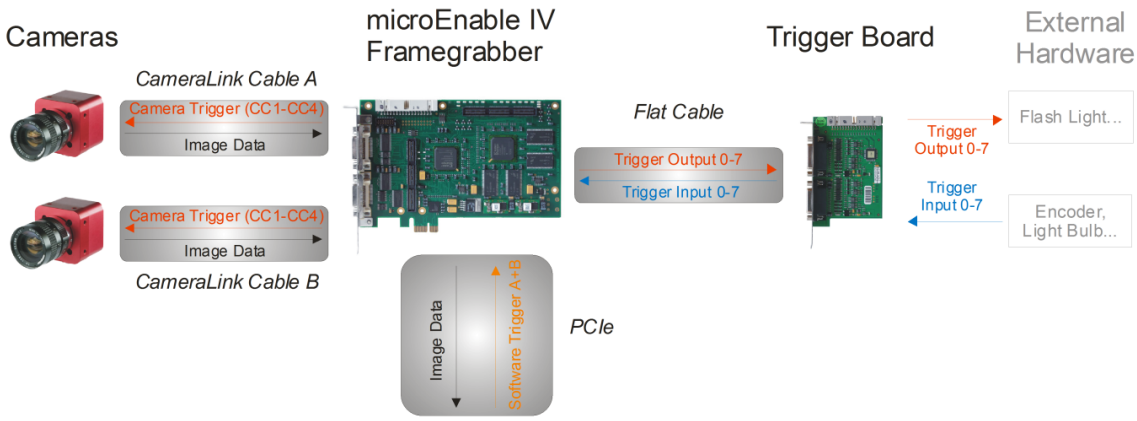
```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

Chapter 5. Trigger

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

Figure 5.1 shows an overview of the signals involved. There are eight different input signals available to the trigger system plus a software trigger input. The external trigger inputs are connected to the frame grabber through a SiliconSoftware trigger IO board. The microEnable IV generates the desired trigger outputs and control signals from the input events according to the trigger system's parameterization. The trigger system outputs can be routed to the camera via the CameraLink's CC signals and additionally, outputs can be routed to the trigger board outputs for control of external devices such as flash lights, for synchronizing or for debugging.

Figure 5.1. microEnable IV Trigger System



In the following an introduction into the Silicon Software microEnable IV trigger system is presented. Several trigger scenarios will show the possibilities and functionalities and will help to understand the trigger system. The documentation includes the parameter reference where all parameters of the trigger system are listed and their functionality is explained in detail.

5.1. Features and Functional Blocks of Area Trigger

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

- Trigger signal generation for cameras and external devices.
- **External devices** such as encoders and light barriers can be used to source the trigger system and control the trigger signal generation.
- In alternative an internal **frequency generator** can be used to generate trigger pulses.
- The trigger signal generation can be fully controlled by **software** . Single pulses or sequences of pulses can be generated. The trigger system will automatically control and limit the output frequency.

- Input **signal monitoring** .
- Input signal **frequency analysis** and **pulse counting** .
- Inpput signal **debouncing**
- Inpput signal **downscaling**
- **Pulse multiplication** using a sequencer and controllable maximum output frequency. Make up to 65,000 output pulses out of a single input pulse.
- **Trigger pulse queue** for buffering up to 2000 pulses and control the output using a maximum frequency valve.
- Four **pulse form generators** for individual controlling of pulse widths, delays and output downscaling.
- **Eight outputs** plus the CameraLink Control outputs (CC-outputs).
- A **bypass** option to keep the pulse forms of the input signals and forward them to outputs and cameras.
- **Event generation** for input and output monitoring by application software.
- Trigger state events for fill level monitoring, trigger busy states and lost trigger signals give full control of the system.
- Camera **frame loss notification** .
- Full **trigger signal reliability** and easy error detections.

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

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

1. Trigger Input:

   Trigger inputs can be external signals, as well as software generated inputs and the frequency generator. Moreover, if you are using a multi-camera applet, the cameras can be synchronized and can be controlled by the trigger configuration of the first camera.

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

2. Input Pulse Processing:

   The second main block of the trigger system is the Input Pulse Processing. External inputs as well as software trigger generated pulses can be queued and multiplied in a sequencer if desired. All external trigger pulses are processed in a maximum frequency valve. Pulses are only processed by this valve if their frequency is higher than the previously parameterized limit. If a higher frequency is present at the

input, pulses will be rejected or the trigger pulse queue is filled if activated. The maximum frequency valve ensures that the output-pulses will not exceed the maximum possible frequency which can be processed by the camera.

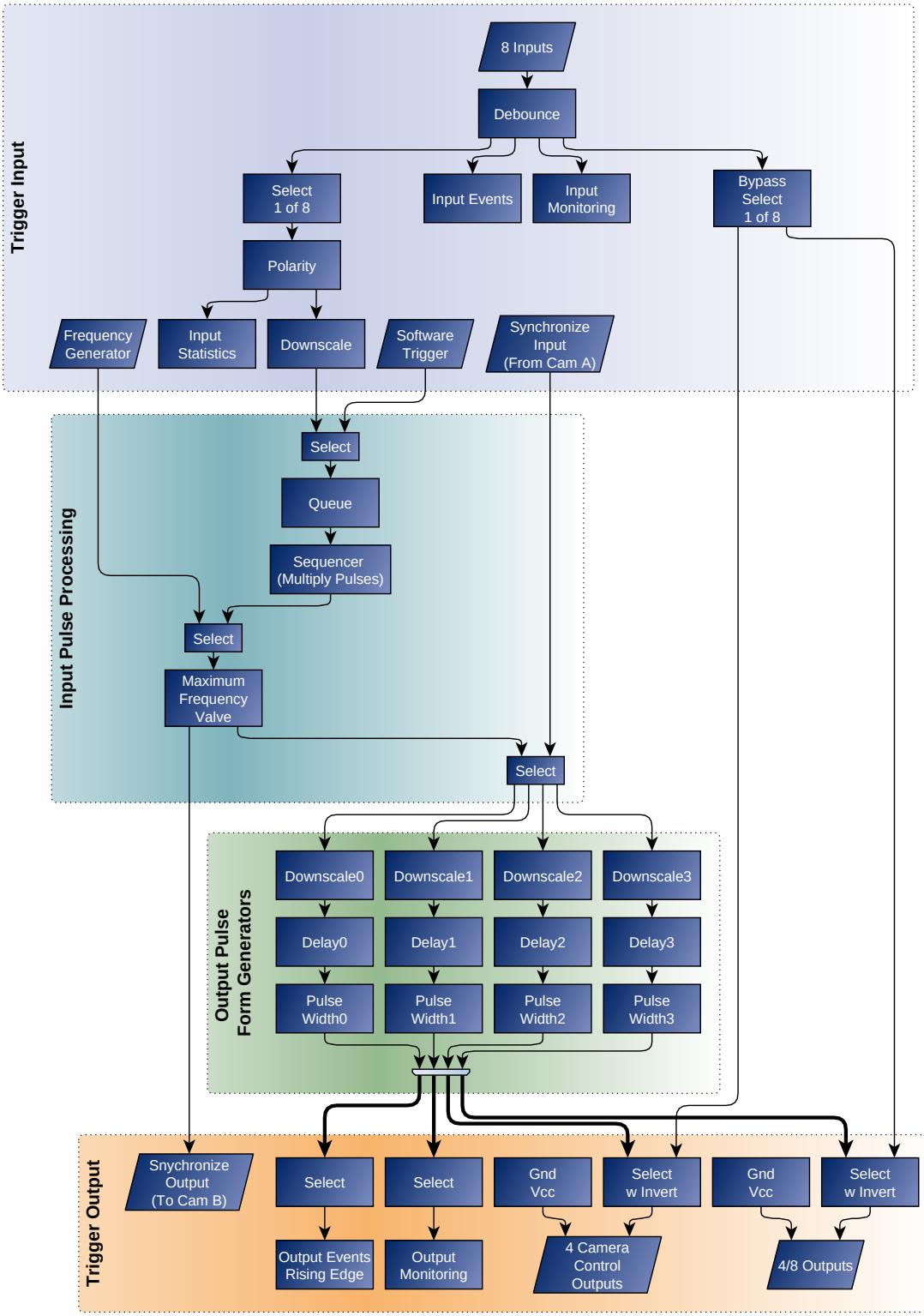
3. Output Pulse Form Generators:

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

4. Trigger Output:

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

Figure 5.2. microEnable IV Trigger System



5.2. Pin allocation

The trigger system supports eight digital inputs and eight digital outputs. These inputs and outputs are allocated to the pins of the trigger IO boards. In the following trigger system explanations, the inputs are numbered from 0 to 7. In the documentation of the trigger IO boards and microEnable IV AD4-CL frame grabber the allocation between these inputs and outputs can be found.

The outputs are also numbered from 0 to 7. Here, the allocation depends on the camera index. If a single camera applet is used, all eight outputs can be used. The allocation between the output indices and the output pins can be found in the documentation of the hardware. In dual camera applets, the first four outputs 0 to 3 can be used by the triggers system of the first camera, while outputs 4 to 7 can only be used by the trigger system of the second camera. This applet is a single camera applet. Thus, all eight outputs can be used.

Trigger IO Boards

The given trigger digital inputs and outputs refer to the inputs and outputs of the microEnable IV AD4-CL frame grabber. For an explanation of the mapping of these IOs to trigger an on boards, please check the respective documentations.

5.3. Event Overview

In the following, a list of all events of the trigger system is presented. Detailed explanations can be found in the respective module descriptions.

- *FG_TRIGGER_INPUT0_RISING*, *FG_TRIGGER_INPUT0_FALLING* to *FG_TRIGGER_INPUT7_RISING*, *FG_TRIGGER_INPUT7_FALLING*

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

  See Section 5.5.9.3.7, “FG_TRIGGER_INPUT0_RISING et al.” and Section 5.5.9.3.8, “FG_TRIGGER_INPUT0_FALLING et al.”

- *FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CAM0*

  The event is generated for each lost input trigger pulse.

  See Section 5.5.7, “FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CAM0”

- *FG_TRIGGER_QUEUE_FILLLEVEL_THRESHOLD_CAM0_ON* and *FG_TRIGGER_QUEUE_FILLLEVEL_THRESHOLD_CAM0_OFF*

  The trigger queue exceeded the upper "on" threshold or got less than the "off" threshold.

  See Section 5.5.7, “FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CAM0”

- *FG_TRIGGER_OUTPUT_CAM0*

  Events for trigger output.

  See Section 5.5.18.2, “FG_TRIGGER_OUTPUT_CAM0”

- *FG_MISSING_CAM0_FRAME_RESPONSE*

The event is generated for a missing camera frame response.

See Section 5.5.17.2.6, “FG_MISSING_CAM0_FRAME_RESPONSE”

5.4. Trigger Scenarios

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

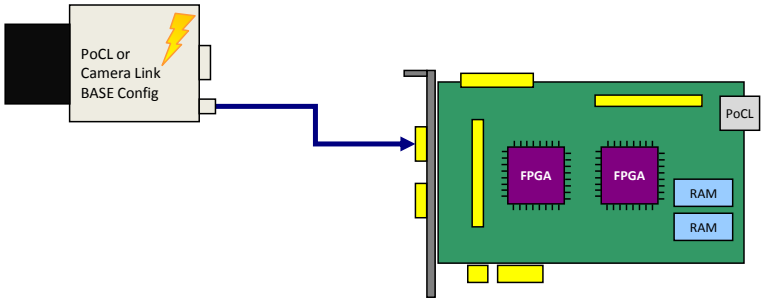
The scenarios show real life configurations. They explain the requirements, illustrate the inputs and outputs and list the required parameters and their values.

5.4.1. Internal Frequency Generator

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

Figure 5.3 shows the hardware setup. Only the camera connected to the frame grabber is required.

Figure 5.3. Generator Controlled Trigger Scenario



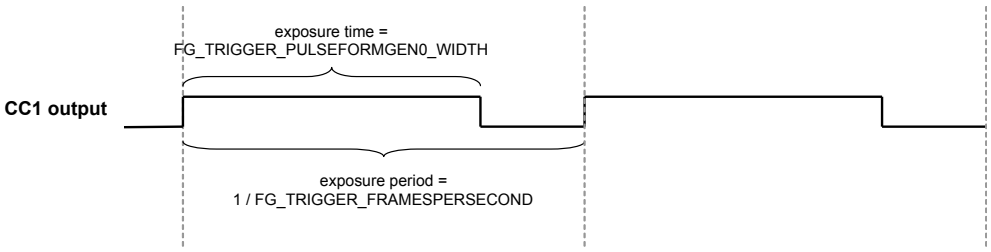
To put this scenario into practice, you will need to set your camera into an external trigger mode. Consult the vendor's user manual for more information. In general Camera Link cameras are configured, using a serial interface. The frame grabber provides this interface and will forward all commands to the camera via the CameraLink cable. Some camera manufacturers provide software tools for camera configuration. Other cameras are configured using command line input. Use the Silicon Software tool clshell in these cases.

After the camera is set to an external trigger mode, the exposure period and the exposure time can be controlled by one of the camera control inputs (CC1 to CC4). Most cameras use CC1 input. The names of the camera trigger modes vary. You will need to use an external trigger mode, where the exposure period is programmable. If you also want to define the exposure time using the frame grabber, the respective trigger mode needs to support this, too.

In the following, a waveform is shown which illustrates the frame grabber trigger output. Most cameras will start the acquisition on the rising or falling edge of the signal. The exposure time is defined by the length of the signal. Note that some

cameras use inverted inputs. In this case, the signal has to be 'low active' instead of being 'high active'. Thus the frame grabber output has to be inverted which is explained later on.

Figure 5.4. Waveform of Generator Controlled Trigger Scenario



After hardware setup and camera configuration we can start parameterizing the frame grabber's trigger system. Parameters can be changed in your own application, in microDisplay or by editing a microEnable Configuration File (mcf). For more information on how to parameterize applets, please consult the Silicon Software runtime documentation.

In the following, all required parameters and their values are listed. In Figure 5.5, “Area Trigger Block Diagram with Highlighted Blocks for Generator Mode” the block diagram of the trigger system is shown, where only the relevant parts are highlighted.

- *FG_AREATRIGGERMODE* = `ATM_GENERATOR`

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

- *FG_TRIGGER_FRAMESPERSECOND* = 10

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

- *FG_TRIGGER_PULSEFORMGEN0_WIDTH* = 200

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

  Suppose, that we require an exposure time of 200µs. Thus, we will set the parameter to value 200.00. Note that the unit of the parameter is µs.

- *FG_CCSEL0* = `CC_PULSEGEN0`

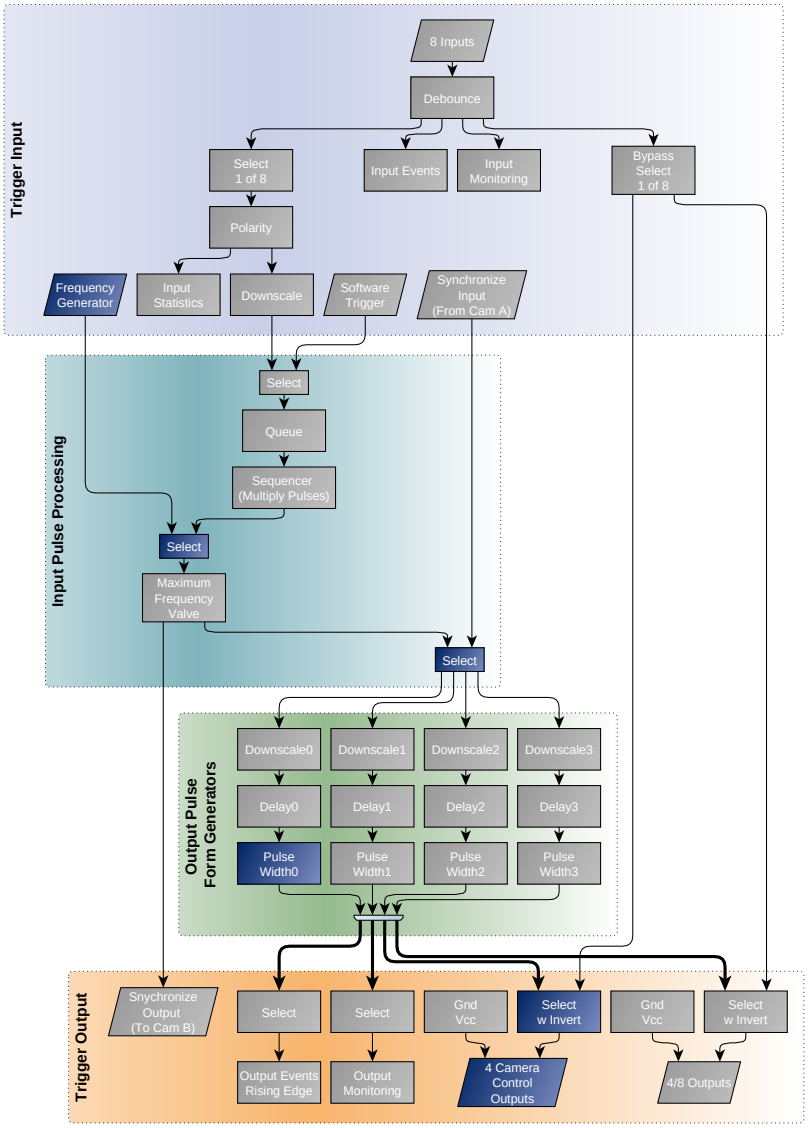
  The only thing left to do is to allocate the output of pulse form generator 0 to output CC1. If your camera requires low active signals, choose CC_NOT_PULSEGEN0 instead.

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

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

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

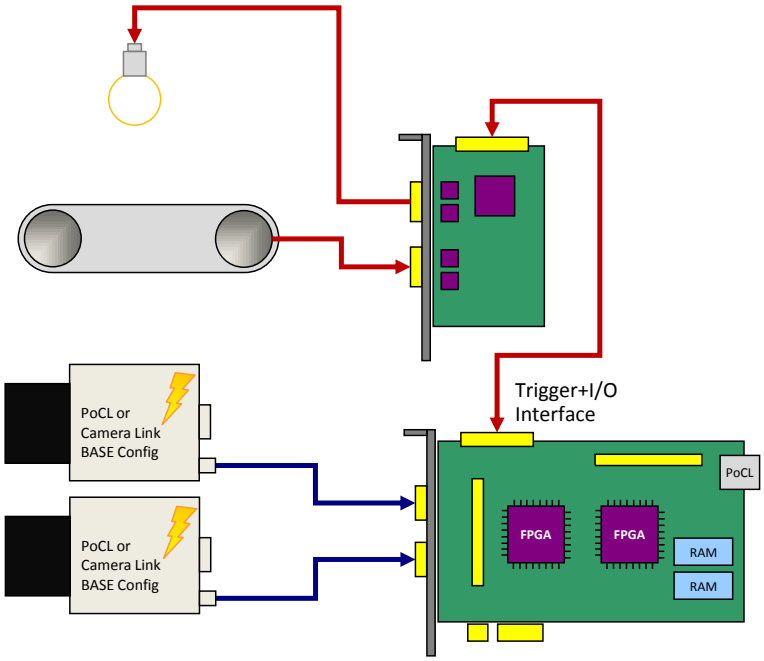
Figure 5.5. Area Trigger Block Diagram with Highlighted Blocks for Generator Mode



5.4.2. External Trigger Signals

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

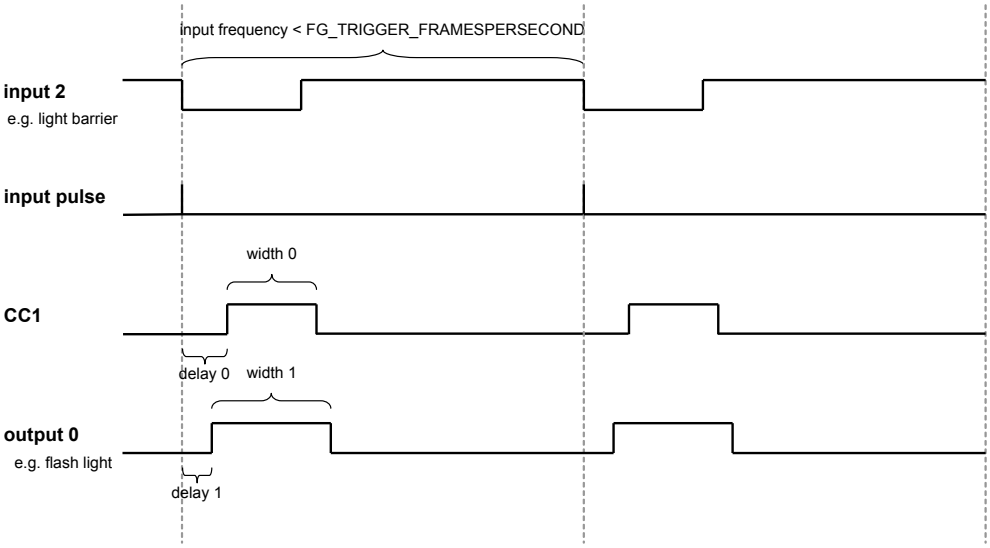
Figure 5.6. External Controlled Trigger Scenario



An exemplary waveform (Figure 5.7) provides information on the input signal and shows the desired output signals. The input is shown on top. As you can see, the falling edge of the signal defines the moment which is used for trigger generation. Thus, the signal is 'low active'. Mind that the pulse length of any external input is ignored (second row), only falling edges are considered.

The output to the camera is shown in the third row. Here we can see an inserted delay. This delay will compensate the positions of the light barrier and the camera. The signal width at output CC1 defines the exposure time, if the camera is configured to the respective trigger mode. Control of the flash light is done using trigger output 0. Again, a delay is added. Depending on the requirements of the flash light, this delay has to be shorter or longer than the CC1 output delay. Similarly, the required pulse length varies for different hardware.

Figure 5.7. Waveform of External Trigger Scenario



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

In this example, we have to parameterize the trigger mode, the input source and we have to configure two trigger outputs. Figure 5.8, "Area Trigger Block Diagram with Highlighted Blocks for External Trigger Scenario" shows the relevant blocks in the area trigger block diagram.

- *FG_AREATRIGGERMODE* = `ATM_EXTERNAL`

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

- *FG_TRIGGERIN_SRC* = `TRGINSRC_2`

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

- *FG_TRIGGERIN_POLARITY* = `FG_LOW_ACTIVE`

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

- *FG_TRIGGER_FRAMESPERSECOND* = 500

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

  The advantage of setting this limit is the information on lost trigger signals. Let's suppose the frequency of the external trigger signals will get to high for the camera or the applet. In this case, you will loose images or obtain corrupted images. If you have set a correct frequency limit in the trigger system, the trigger system will provide you with information of these exceeding line periods. This information can be obtained by register polling or you can use the event

system. Thus you always have the possibility to prevent your application of getting into a bad, probably undefined state and you will always get the information of when an how many pulses got lost. Check the explanations of parameters *FG_TRIGGER_FRAMESPERSECOND* and *FG_TRIGGER_EXCEEDED_PERIOD_LIMITS* as well as the event *FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CAM0* for more information.

More information on error detection and analysis can be found in scenario Section 5.4.9, “Hardware System Analysis and Error Detection”

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

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

- *FG_TRIGGER_PULSEFORMGEN0_WIDTH* = 200

  So far, we have set the trigger system to accept external signals and generate the trigger pulses out of these signals. Next, we need to output these pulses. For realization, we need to define the pulse form of the output signals. Just as shown in the previous scenario, we use pulse form generator 0 for generating the pulse form of the CC1 signals. We set a pulse width of 200µs.

- *FG_TRIGGER_PULSEFORMGEN0_DELAY* = 50

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

- *FG_TRIGGER_PULSEFORMGEN1_WIDTH* = 250

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

- *FG_TRIGGER_PULSEFORMGEN1_DELAY* = 25

  A delay for the flash output is set, too.

- *FG_CCSEL0* = `CC_PULSEGEN0`

  Finally, we have to allocate output CC1 with the pulse form generator 0.

- *FG_TRIGGEROUT_SELECT0* = `PULSEGEN1`

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

- *FG_TRIGGEROUT_SELECT1* = `PULSEGEN0`

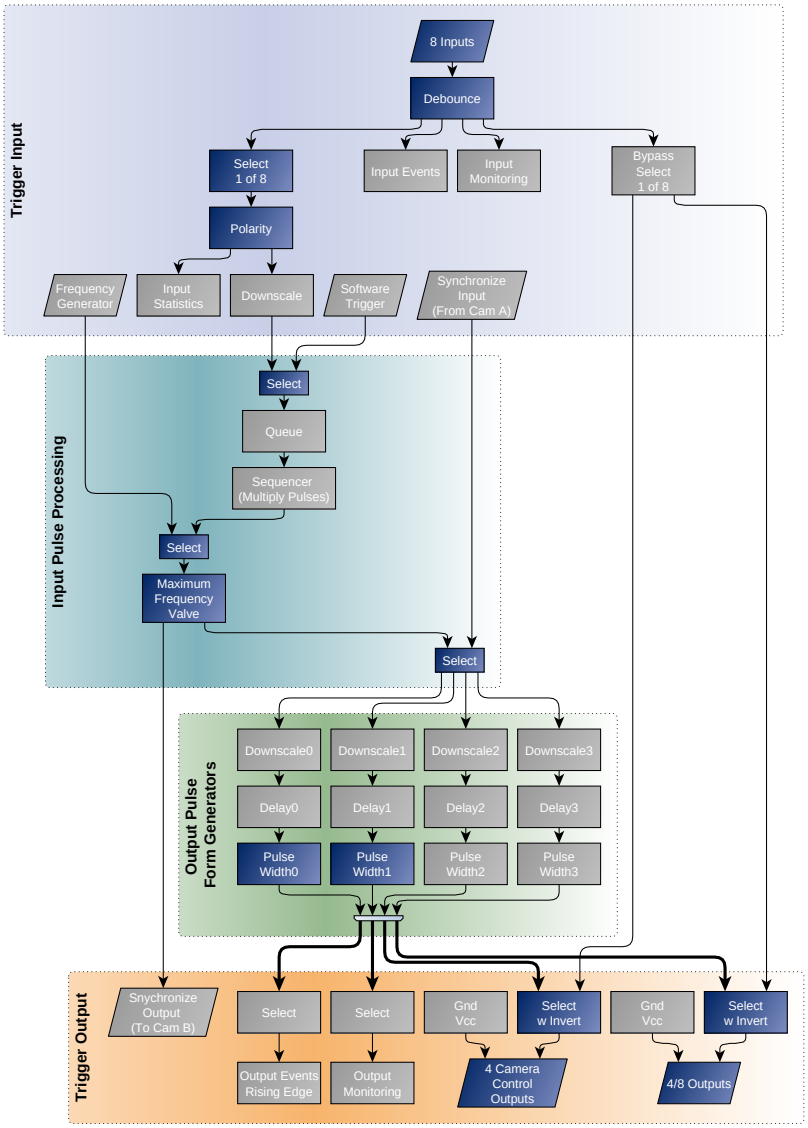
  Let's assume that it is necessary to measure the camera trigger output using a logic analyzer. Hence, we allocate output 1 to pulse form generator 0 as well.

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

You will now receive images from the camera for each external trigger pulse. Compare the number of external pulses with the generated

trigger signals and the received images for verification. Use parameter *FG_TRIGGERIN_STATS_PULSECOUNT* of category Trigger Input -> Input Statistics and parameter *FG_TRIGGEROUT_STATS_PULSECOUNT* of the output statistics parameters to get the number of input pulses and generated pulses. You can compare these values with the received image numbers.

Figure 5.8. Area Trigger Block Diagram with Highlighted Blocks for External Trigger Scenario



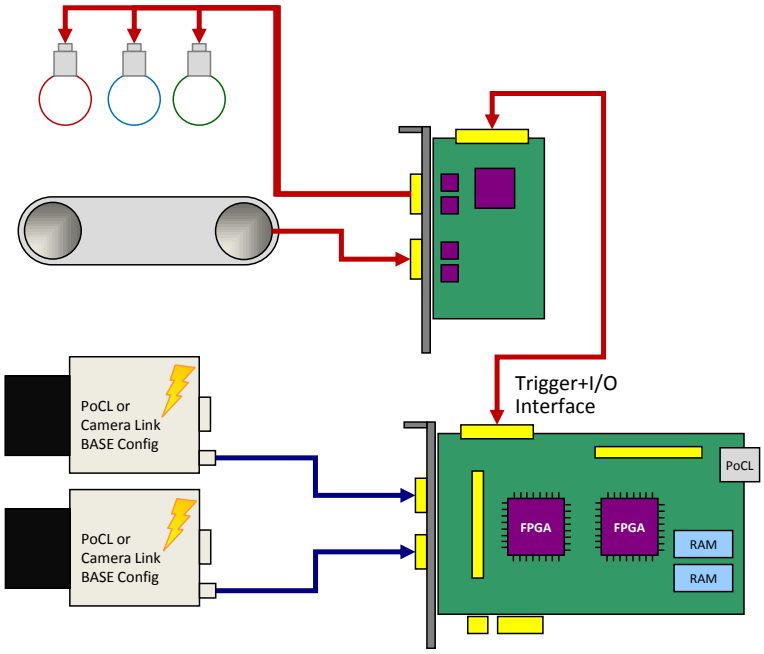
5.4.3. Control of Three Flash Lights

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

The application idea behind this scenario is that an object is acquired using different light sources. This could result in a HDR image or switching between normal and

infrared illumination. The following figure illustrates the hardware setup. As you can see, we have three light sources this time. The objects move in front of the camera. The light barrier will provide the information on when to trigger the camera. Let's suppose that the objects stop in front of the camera or the movement is slow enough to generate three images with the different illuminations.

Figure 5.9. External Controlled Trigger Scenario



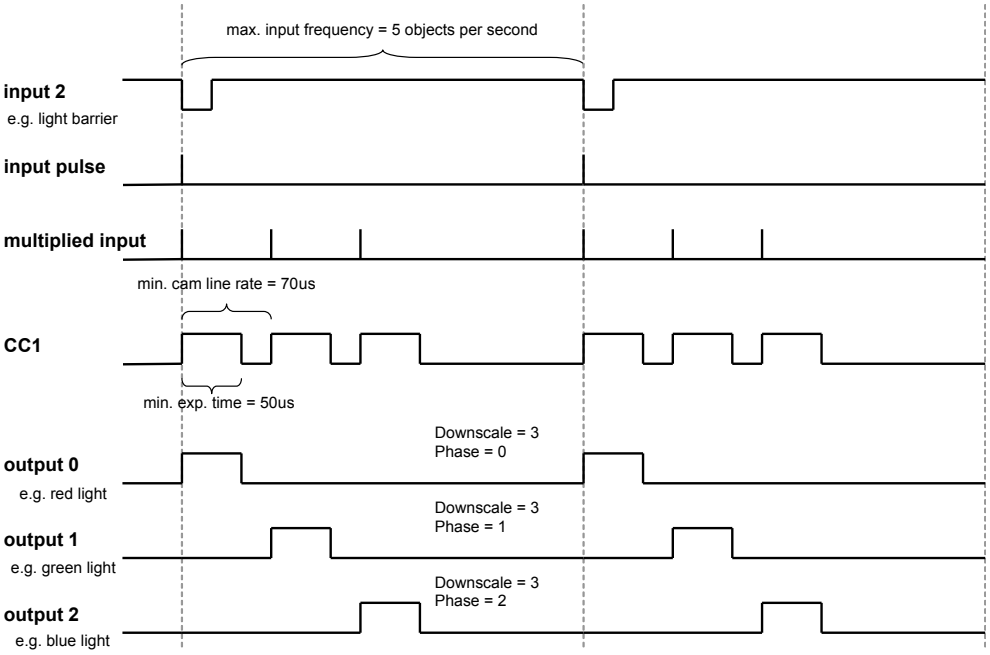
Before looking at the waveform, let's have a look at our fictitious hardware specifications.

Table 5.1. Fictitious Hardware Specifications of Trigger Scenario Three Light Sources

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

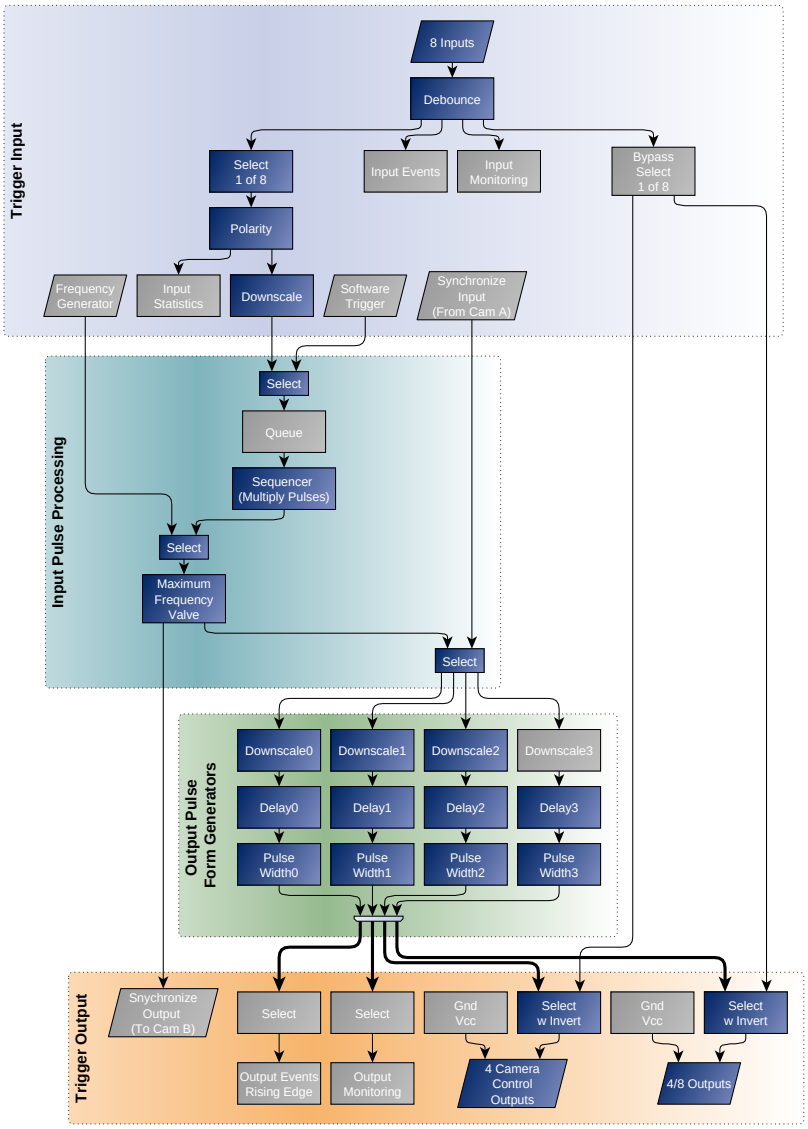
The object speed is 100 objects per second. The minimum camera exposure time is 50µs at a minimum camera frame period of 70µs. Thus we only need 210µs to acquire the three images. The following waveform shows the input and output signals, as well as the multiplied input signals. The first row shows the input. Each falling edge represents the light barrier event as marked in the second row. The third row shows the multiplied input pulses with a gap of 70µs between the pulses. CC1 signal is generated for each of these pulses, however the trigger outputs 0, 1 and 2 are downscaled by three and a delay is added.

Figure 5.10. Waveform of External Trigger Scenario Controlling Three Flash Lights



Parameterization is similar to the previous example. In contrast, this time, we have to set the trigger pulse sequencer using a multiplication factor and we have to use all four pulse form generators. A look at the block diagram (Figure 5.11, “Area Trigger Block Diagram with Highlighted Blocks for Three Flash Lights Trigger Scenario ”) will show us the relevant blocks.

Figure 5.11. Area Trigger Block Diagram with Highlighted Blocks for Three Flash Lights Trigger Scenario



- *FG_AREATRIGGERMODE* = ATM_EXTERNAL
- *FG_TRIGGERIN_SRC* = 2
- *FG_TRIGGERIN_POLARITY* = FG_LOW_ACTIVE
- *FG_TRIGGER_MULTIPLY_PULSES* = 3

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

- *FG_TRIGGER_FRAMESPERSECOND* = 14285

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

- *FG_TRIGGER_PULSEFORMGEN0_WIDTH* = 50

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

- *FG_TRIGGER_PULSEFORMGEN1_WIDTH* = 50

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

- *FG_TRIGGER_PULSEFORMGEN2_WIDTH* = 50
- *FG_TRIGGER_PULSEFORMGEN3_WIDTH* = 50
- *FG_TRIGGER_PULSEFORMGEN1_DOWNSCALE* = 3

  The flash outputs need a downscale of three. This is the same for all flash pulse form generators.

- *FG_TRIGGER_PULSEFORMGEN2_DOWNSCALE* = 3
- *FG_TRIGGER_PULSEFORMGEN3_DOWNSCALE* = 3
- *FG_TRIGGER_PULSEFORMGEN1_DOWNSCALE_PHASE* = 0

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

- *FG_TRIGGER_PULSEFORMGEN2_DOWNSCALE_PHASE* = 1

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

- *FG_TRIGGER_PULSEFORMGEN3_DOWNSCALE_PHASE* = 2

  The phase shift of pulse form generator 3 i.e. the third flash light is 2.

- *FG_CCSEL0* = `PULSEGEN0`

  The output allocation is as usual.

- *FG_TRIGGEROUT_SELECT0* = `PULSEGEN1`
- *FG_TRIGGEROUT_SELECT1* = `PULSEGEN2`
- *FG_TRIGGEROUT_SELECT2* = `PULSEGEN3`

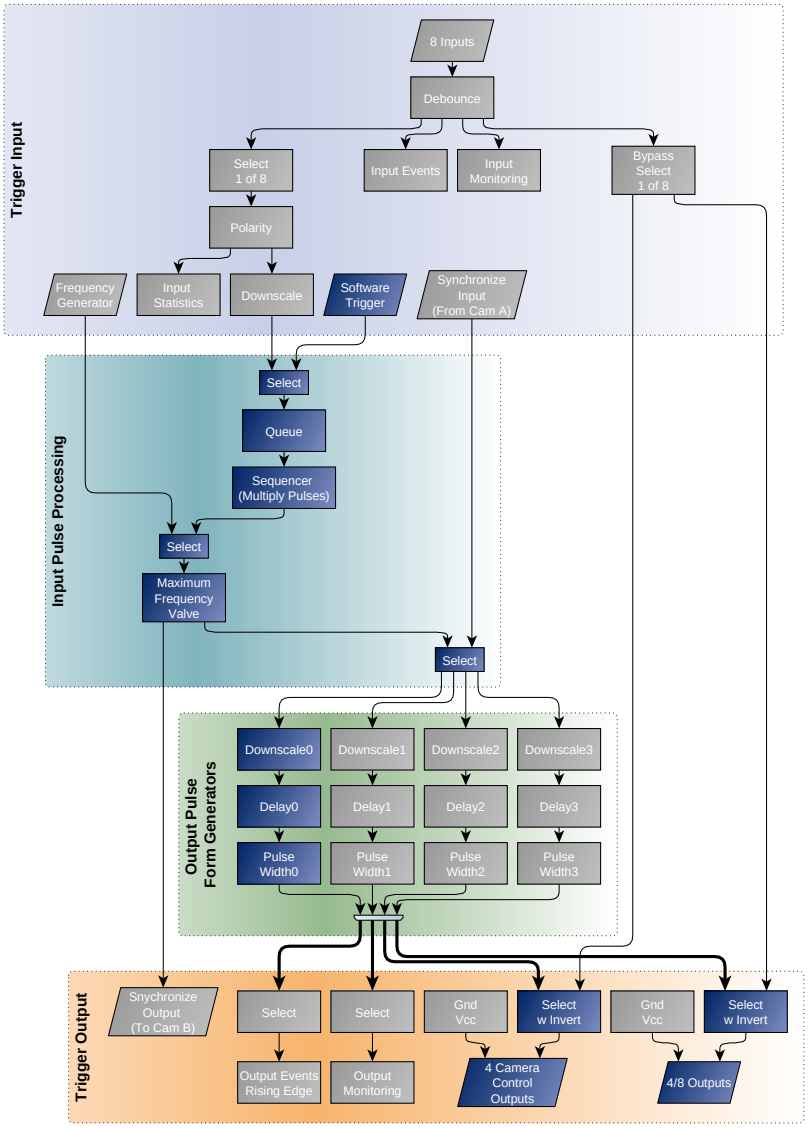
Start the trigger system using parameter *FG_TRIGGERSTATE* as usual. You will notice that you get thrice the number of images from the frame grabber than external trigger pulses have been generated by the light barrier. Equally to the previous example, check for exceeding line periods at the input when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

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

5.4.4. Software Trigger

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

Figure 5.12. Area Trigger Block Diagram with Highlighted Blocks for Software Trigger Scenario



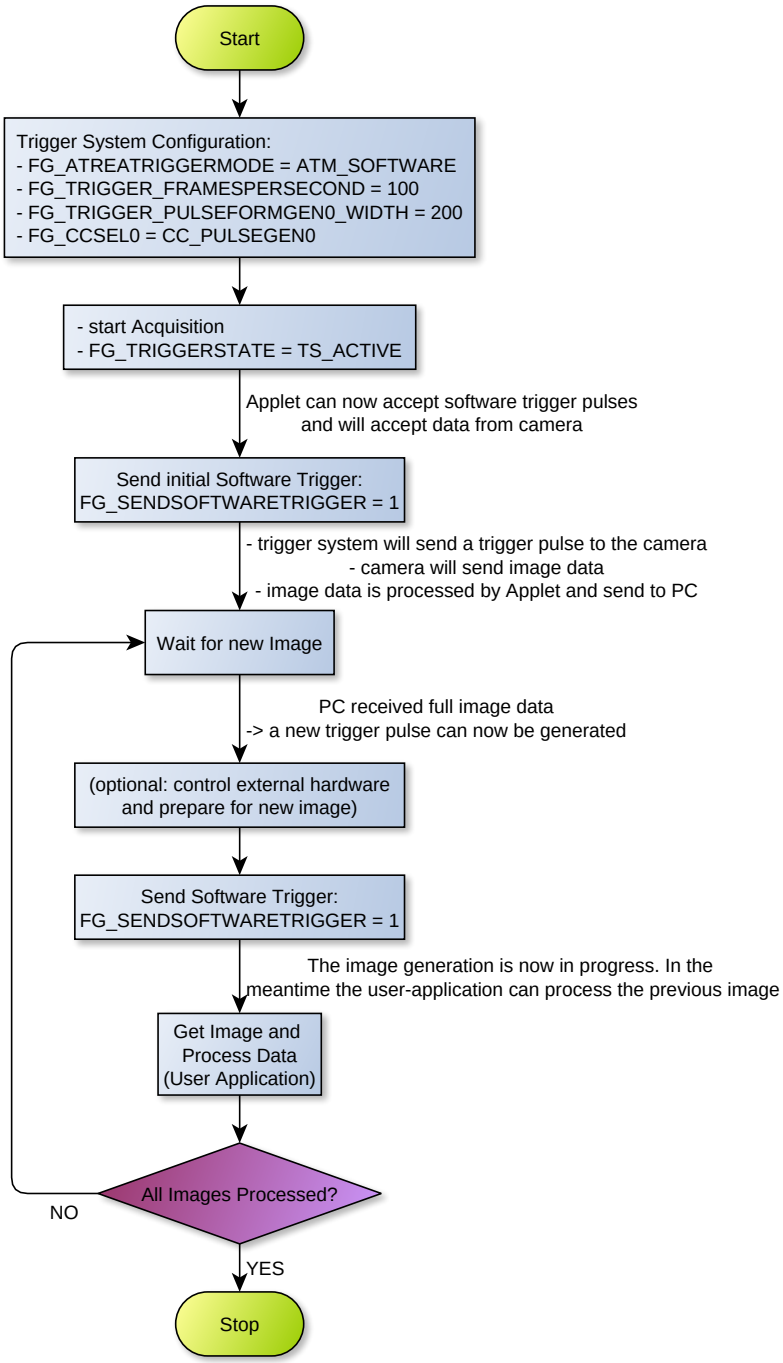
Keep in mind that the time between two pulses has to be larger than 1 / *FG_TRIGGER_FRAMESPERSECOND* as this will limit the maximum trigger frequency. The trigger system offers the possibility to check if a new software trigger pulse can be accepted i.e. the trigger system is not busy anymore. Read parameter *FG_SOFTWARETRIGGER_IS_BUSY* to check it's state. While the parameter has value IS_BUSY, writing to parameter *FG_SENDSOFTWARETRIGGER* is not allowed and will

be ignored. You should always check if the system is not busy before writing a pulse. To check if you lost a pulse, read parameter *FG_TRIGGER_EXCEEDED_PERIOD_LIMITS*.

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

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

Figure 5.13. Flowchart of Software Application Using the Software Trigger

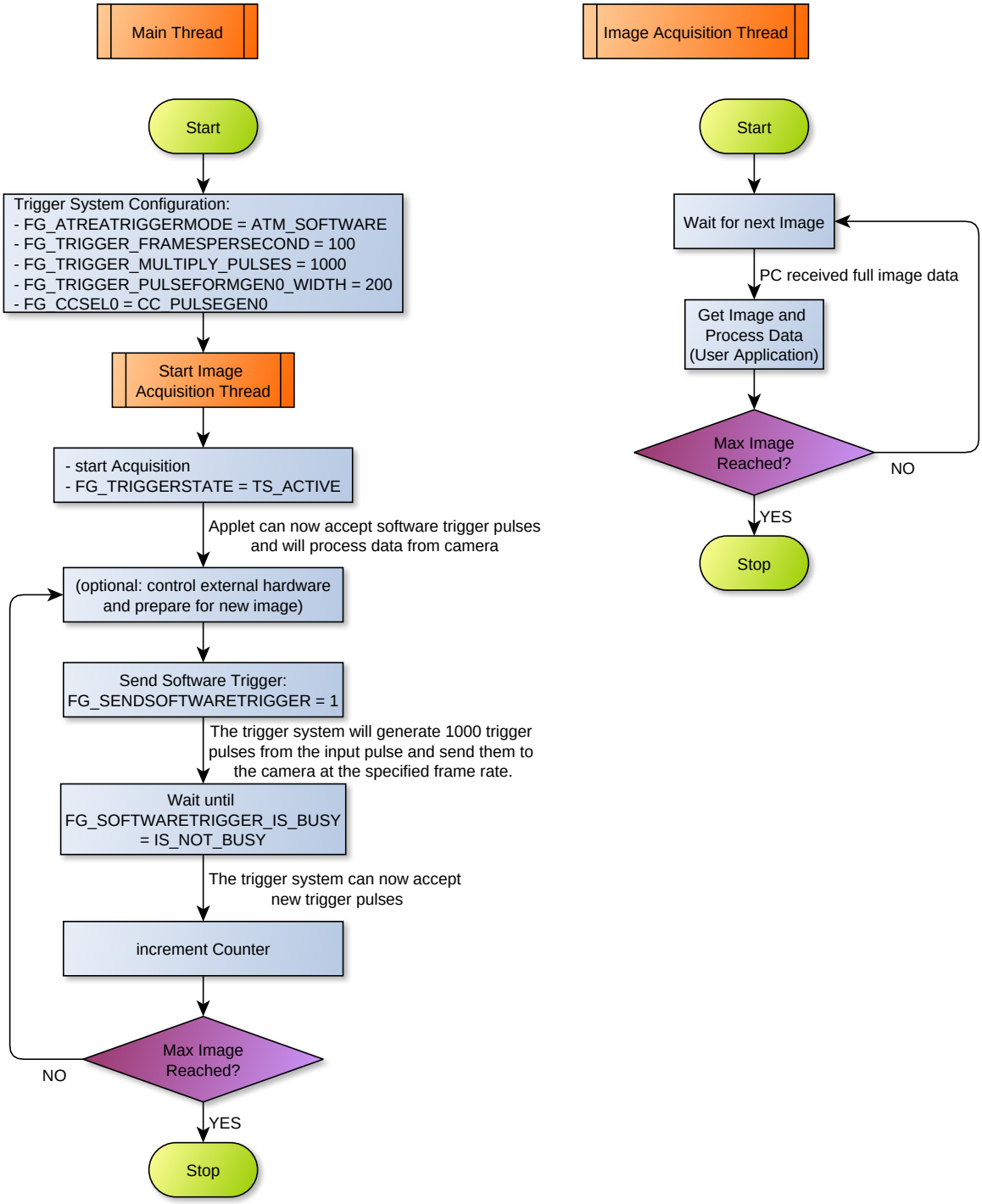


In the shown sample application, it is ensured that the trigger system is not busy after you received the image. Therefore, we do not need to check for the software trigger busy flag in this example. One drawback of the shown example is that we might not acquire the frames at the maximum speed. This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

The next example uses two threads. One thread for trigger generation and one thread for image acquisition and processing. In comparison to the previous example, we use

the trigger sequencer for pulse multiplication and we will have to use the busy flag. This will allow an acquisition at a higher frame rate.

Figure 5.14. Flowchart of Software Application Using the Software Trigger with a Sequencer



The main thread will configure and start the trigger system and the acquisition. For each software trigger pulse we send to the frame grabber, 1000 pulses are generated and send to the camera at the framerate specified by *FG_TRIGGER_FRAMESPERSECOND*. After sending a software trigger pulse to the frame grabber we wait until the software is not busy anymore by polling on register

FG_SOFTWARETRIGGER_IS_BUSY. To control the number of generated trigger pulses we count each successful sequence generation. If more images are required we can send another software trigger pulse to the frame grabber to start a new sequence.

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

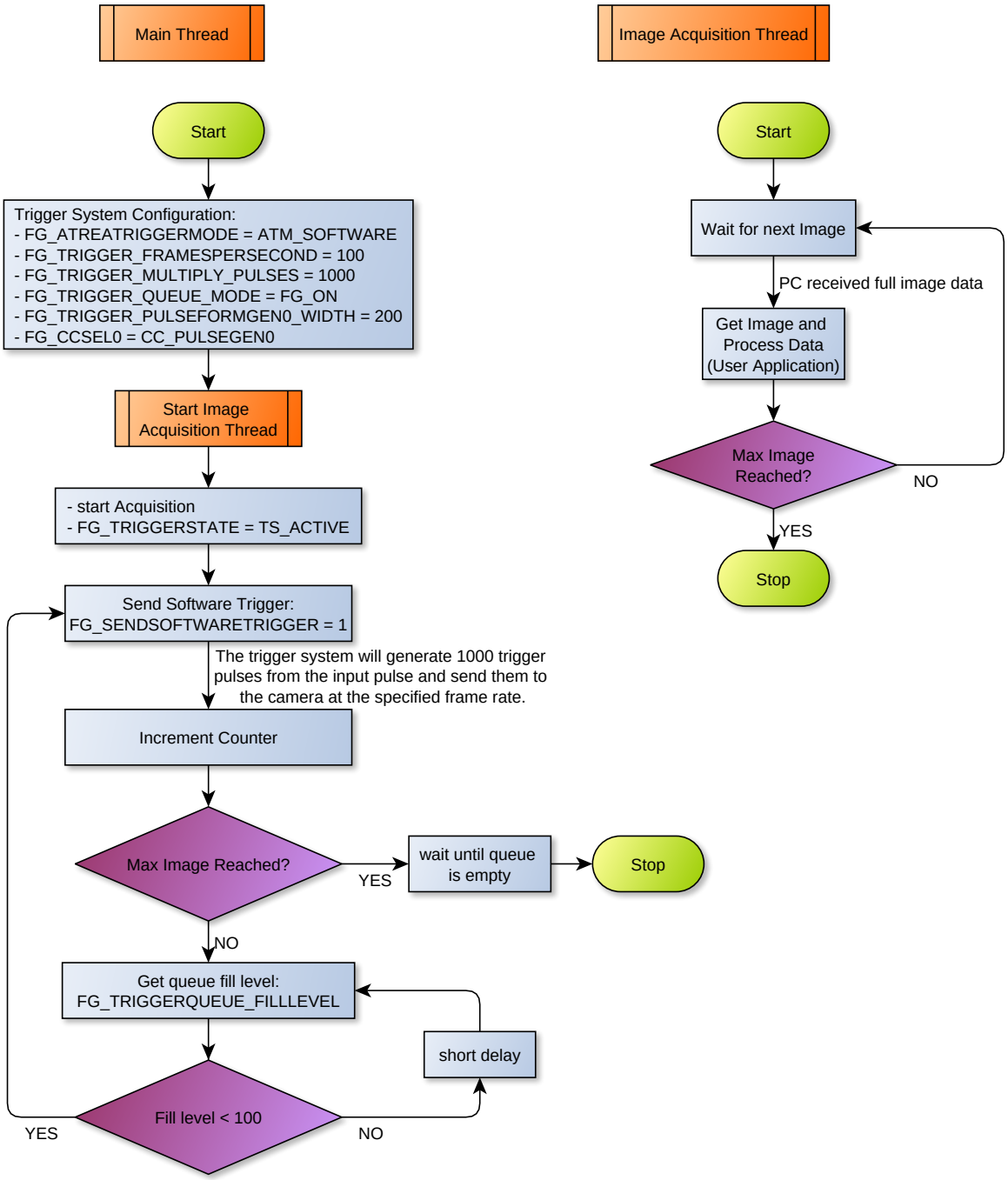
5.4.5. Software Trigger with Trigger Queue

In the following we will have a look at the software trigger once again. This time, we use the trigger queue. The trigger queue enables the buffering of trigger pulses from external sources or from the software trigger and will output these pulses at the maximum allowed frequency specified by *FG_TRIGGER_FRAMESPERSECOND*. Therefore, we can write to *FG_SENDSOFTWARETRIGGER* multiple times even if the trigger system is still busy. Parameter *FG_SOFTWARETRIGGER_IS_BUSY* will only have value IS_BUSY if the queue is full. Instead of writing multiple times to *FG_SENDSOFTWARETRIGGER* you can directly write the number of required pulses to the parameter.

The trigger queue can buffer 2040 pulses. This buffer does not include the sequencer. Thus if you have a sequence length of 1000 pulses and currently 200 pulses in the queue, the trigger system has 200,000 remaining pulses to output. You can check the fill level by reading parameter *FG_TRIGGERQUEUE_FILLLEVEL*.

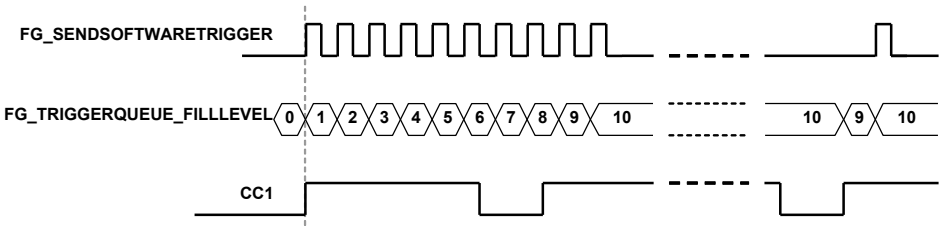
In the following flow chart you can see a queue fill level minimum limit of 10 pulses. In our supposed application we will check the queue fill level and compare it with our limit. If less pulses are in the queue, we generate a new software trigger pulse. Thus, on startup, the queue will fill-up until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

Figure 5.15. Flowchart of Software Application Using the Software Trigger with Trigger Queue



When having a look at the waveform (Figure 5.16) we can see the initialization phase where the queue is filled. After fill level value 10 has been reached, no more software trigger pulses are written to the applet. The system will now continue the output of trigger pulses. As our sequence length is 1000 pulses we have to wait for 1000 pulses to be generated until a change in the fill level will occur. After the 1000th pulse has been completely generated, the fill level will change to 9. This will cause the generation of another software trigger pulse by our sample application which will cause a fill level of 10 again.

Figure 5.16. Waveform Illustrating Software Trigger with Queue Example"



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

5.4.6. External Trigger with Trigger Queue

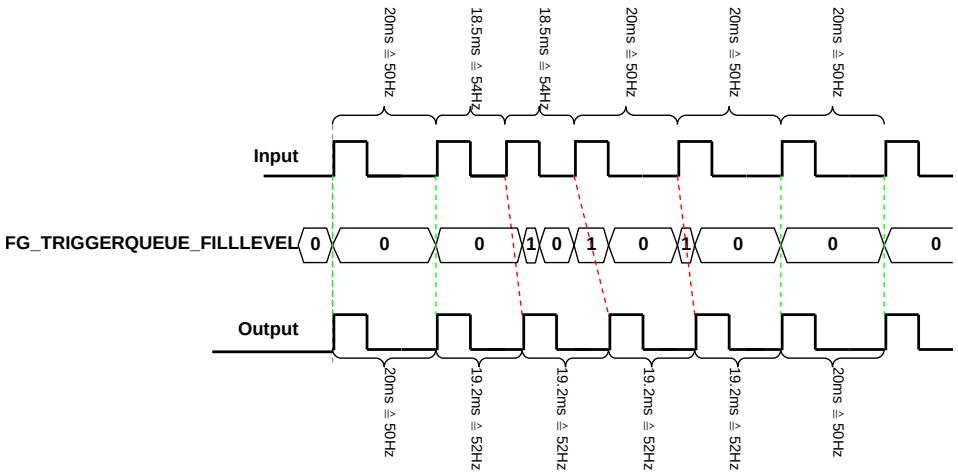
Of course, we can use the trigger queue with external triggers, too. This will give us a possibility to buffer 'jumpy' external encoders or any other external trigger signal generators. Let's suppose an external encoder which is configured to generate trigger pulses with a frequency of 50Hz and a camera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

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

To enable the queue, just write value `FG_ON` to *FG_TRIGGERQUEUE_MODE*.

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

Figure 5.17. Using External Trigger Signal Sources together with the Trigger Queue



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

5.4.7. Bypass External Trigger Signals

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

It is also possible to mix the bypass with the generator or software trigger mode. Thus you can output the bypass signal to CC2, while CC1 includes pulses of the internal frequency generator or from another external source.

Use parameter *FG_TRIGGERIN_BYPASS_SRC* to select the trigger input and configure the outputs to use the bypass i.e. set parameters *FG_CCSEL0* to *FG_CCSEL3* and *FG_TRIGGEROUT_SELECT0* to *FG_TRIGGEROUT_SELECT3* to value `INPUT_BYPASS` or `NOT_INPUT_BYPASS` if required.

5.4.8. Multi Camera Applications

A basic application is that multiple cameras at one or more frame grabbers are connected to the same trigger source. If all cameras have to acquire images for every trigger pulse. Simply connect the trigger source to all frame grabbers and set the same trigger configuration for all cameras. Some applets support more than one camera. In this case, the same parameters for all cameras should be set. They may share the same trigger input.

If you do not have an external trigger source, but use the generator or the software trigger you can synchronize the triggers to ensure camera exposures at the same moment. Simply output the camera control signal on a digital trigger output and connect this output to a digital input of other frame grabbers which have to be synchronized with the master. In the slave applets bypass the input to the camera control (CC) outputs. If an applets is used which supports more than one camera, a special trigger mode called `SYNCHRONIZED` can be used. This mode can be chosen

for all cameras except the first one. The "slave" cameras use trigger pulses at the output of the first camera as input source. However, users will still have to configure the pulse form generators in the trigger system of the slave cameras and will still have to allocate them to the trigger outputs.

5.4.9. Hardware System Analysis and Error Detection

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

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

- Trigger Input Statistics

  The parameters of the trigger input statistics category allow an analysis of the external trigger pulses. Parameter *FG_TRIGGERIN_STATS_FREQUENCY* performs a continuous frequency measurement of the input signals. Compare this value with the expected trigger input frequency. If the measured frequency is much higher or lower than the expected frequency, check your external hardware. Also check if the correct trigger input has been chosen by parameter *FG_TRIGGERIN_SRC* and if the pulse width of the input is long enough to be detected by the hardware.

  To validate a constant input frequency, the trigger system will also show the maximum and minimum detected frequencies using parameters *FG_TRIGGERIN_STATS_MAXFREQUENCY* and *FG_TRIGGERIN_STATS_MINFREQUENCY*. On startup, you will have a very low frequency as no external pulses might have been detected so far. Therefore, you have to clear the measurement using parameter *FG_TRIGGERIN_STATS_MINMAXFREQUENCY_CLEAR* first. If you detect an unwanted deviation from the expected values or the difference between the minimum and maximum frequency is comparably high, your external trigger generating hardware might be 'jumpy', skips pulses or is 'bouncing' which causes pulse multiplication. In this case, you might be able to compensate the problem using a higher debouncing value, set a lower maximum allowed frequency (see Section 5.4.2, “External Trigger Signals”) or use the trigger queue (see Section 5.4.6, “External Trigger with Trigger Queue”).

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

- Trigger Output Statistics

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

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

It is also possible to count the input and output pulses with the input events and the output event *FG_TRIGGER_OUTPUT_CAM0*.

- Camera Response Check

  Trigger pulses might get lost in the link to the camera or the trigger frequency is to high to be processed by the camera. In this case, the number of frames received by the frame grabber differs from the send trigger pulses. For this error, the trigger system includes the missing camera frame response detection module. The module can detect missing frames and generate an event for each lost frame or set a register. Check Section 5.5.17.2, “Statistics” for more information and usage.

- Acquired Image Compare

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

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

5.5. Parameters

5.5.1. FG_AREATRIGGERMODE

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

- Generator

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

- External

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

- Software

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

Free-Run Mode

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

Allowed Frequencies

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

Table 5.2. Parameter properties of FG_AREATRIGGERMODE

Property	Value
Name	**FG_AREATRIGGERMODE**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**ATM_GENERATOR** Generator **ATM_EXTERNAL** External **ATM_SOFTWARE** Software
Default value	**ATM_GENERATOR**

Example 5.1. Usage of FG_AREATRIGGERMODE

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

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

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

5.5.2. FG_TRIGGERSTATE

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

Table 5.3. Parameter properties of FG_TRIGGERSTATE

Property	Value	
Name	**FG_TRIGGERSTATE**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**TS_ACTIVE** **TS_ASYNC_STOP** **TS_SYNC_STOP**	Active Async Stop Sync Stop
Default value	**TS_SYNC_STOP**	

Example 5.2. Usage of FG_TRIGGERSTATE

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

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

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

5.5.3. FG_TRIGGER_FRAMESPERSECOND

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

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

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

Equation 5.1. Dependency of Frequency and Pulse Form Generators

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

- fps = *FG_TRIGGER_FRAMESPERSECOND*

- WIDTH[0..3] = *FG_TRIGGER_PULSEFORMGEN[0..3]_WIDTH*
- DELAY[0..3] = *FG_TRIGGER_PULSEFORMGEN[0..3]_DELAY*
- DOWNSCALE[0..3] = *FG_TRIGGER_PULSEFORMGEN[0..3]_DOWNSCALE*

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

Table 5.4. Parameter properties of FG_TRIGGER_FRAMESPERSECOND

Property	Value
Name	**FG_TRIGGER_FRAMESPERSECOND**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum** **NaN** **Maximum** **6250000.0** **Stepsize** **2.220446049250313E-16**
Default value	**8.0**
Unit of measure	**Hz**

Example 5.3. Usage of FG_TRIGGER_FRAMESPERSECOND

```
int result = 0;
double value = 8.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

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

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

5.5.4. FG_TRIGGER_EXCEEDED_PERIOD_LIMITS

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

Table 5.5. Parameter properties of FG_TRIGGER_EXCEEDED_PERIOD_LIMITS

Property	Value
Name	**FG_TRIGGER_EXCEEDED_PERIOD_LIMITS**
Type	**Enumeration**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**FG_YES** Yes **FG_NO** No

Example 5.4. Usage of FG_TRIGGER_EXCEEDED_PERIOD_LIMITS

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

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

5.5.5. FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CLEAR

Reset *FG_TRIGGER_EXCEEDED_PERIOD_LIMITS* with this parameter.

Table 5.6. Parameter properties of FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CLEAR

Property	Value
Name	**FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CLEAR**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**FG_APPLY** Apply
Default value	**FG_APPLY**

Example 5.5. Usage of FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CLEAR

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

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

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

5.5.6. FG_TRIGGER_LEGACY_MODE

The trigger system of this applet offers powerful features. Numerous parameters are used to configure the module. For users changing from an older Silicon Software AcquisitionApplets Applet (Applets before Runtime 5.2) to this applet (Acq_MediumAreaRGB36), the trigger parameters of the older applets can be reused for convenience. If parameter *FG_TRIGGER_LEGACY_MODE* is set to FG_ON, the parameters of previous applets which do not exist in the current trigger system anymore, can be reused. Setting the trigger system into the legacy mode, will disable all new parameters for writing. However, reading is possible to see the mapping of the old parameters to the new ones. Furthermore, a change of this parameter will reset the trigger system to it's defaults.

A documentation of the legacy parameters is not given in this document. For detailed explanations check the documentation of the older applets or of the runtime.

Use the Current Parameter Set

Silicon Software strongly recommends the use of the current parameter set. The support for the legacy parameters might be discontinued in future

and the exact replication of the old trigger system functionality cannot be guaranteed for all possible combinations of parameter settings.

Table 5.7. Parameter properties of FG_TRIGGER_LEGACY_MODE

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

Example 5.6. Usage of FG_TRIGGER_LEGACY_MODE

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

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

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

5.5.7. FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CAM0

The event is generated for each lost input trigger pulse. A trigger loss can occur if the input frequency is higher than the maximum allowed frequency set by parameter *FG_TRIGGER_FRAMESPERSECOND* (see Chapter 5.5.3). If the trigger queue is enabled, the events will only be generated if the queue is full i.e. for overflows. In generator trigger modes, the events will not be generated. Except for the timestamp, the event has no additional data included.

5.5.8. Legacy

This category includes the trigger legacy parameters. For users changing from older AcquisitionApplets to this applets, the trigger parameters of the older applets can be reused. A full description on how to reuse these parameters can be found in Section 5.5.6, “FG_TRIGGER_LEGACY_MODE”.

The following paramrters will not have any description texts. For an explanation, check the trigger documentation of the old AcquisitionApplets.

5.5.8.1. FG_TRIGGERMODE

Table 5.8. Parameter properties of FG_TRIGGERMODE

Property	Value	
Name	**FG_TRIGGERMODE**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**FREE_RUN** **GRABBER_CONTROLLED** **ASYNC_TRIGGER** **ASYNC_SOFTWARE_TRIGGER**	Free Run Grabber controlled Async Trigger Async Software Trigger
Default value	**FREE_RUN**	

Example 5.7. Usage of FG_TRIGGERMODE

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

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

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

5.5.8.2. FG_EXSYNCON

Table 5.9. Parameter properties of FG_EXSYNCON

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

Example 5.8. Usage of FG_EXSYNCON

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

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

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

5.5.8.3. FG_FLASHON

Table 5.10. Parameter properties of FG_FLASHON

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

Example 5.9. Usage of FG_FLASHON

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

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

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

5.5.8.4. FG_EXPOSURE

Table 5.11. Parameter properties of FG_EXPOSURE

Property	Value
Name	**FG_EXPOSURE**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum** **10** **Maximum** **124990** **Stepsize** **10**
Default value	**4000**

Example 5.10. Usage of FG_EXPOSURE

```
int result = 0;
unsigned int value = 4000;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

5.5.8.5. FG_EXSYNCDELAY

Table 5.12. Parameter properties of FG_EXSYNCDELAY

Property	Value
Name	**FG_EXSYNCDELAY**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.0** **Maximum 10230.0** **Stepsize 10.0**
Default value	**0.0**
Unit of measure	**microseconds**

Example 5.11. Usage of FG_EXSYNCDELAY

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

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

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

5.5.8.6. FG_EXSYNCPOLARITY

Table 5.13. Parameter properties of FG_EXSYNCPOLARITY

Property	Value
Name	**FG_EXSYNCPOLARITY**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_ZERO** Low Active **FG_ONE** High Active
Default value	**FG_ZERO**

Example 5.12. Usage of FG_EXSYNCPOLARITY

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

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

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

5.5.8.7. FG_STROBEPULSEDELAY

Table 5.14. Parameter properties of FG_STROBEPULSEDELAY

Property	Value
Name	**FG_STROBEPULSEDELAY**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.0** **Maximum 10230.0** **Stepsize 10.0**
Default value	**0.0**
Unit of measure	**microseconds**

Example 5.13. Usage of FG_STROBEPULSEDELAY

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

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

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

5.5.8.8. FG_FLASH_POLARITY

Table 5.15. Parameter properties of FG_FLASH_POLARITY

Property	Value
Name	**FG_FLASH_POLARITY**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_ZERO** Low Active **FG_ONE** High Active
Default value	**FG_ZERO**

Example 5.14. Usage of FG_FLASH_POLARITY

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

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

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

5.5.8.9. FG_PRESCALER

Table 5.16. Parameter properties of FG_PRESCALER

Property	Value
Name	**FG_PRESCALER**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum** **0.032** **Maximum** **1048.56** **Stepsize** **0.016**
Default value	**10.0**
Unit of measure	**microseconds**

Example 5.15. Usage of FG_PRESCALER

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

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

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

5.5.8.10. FG_CCSEL0 et al.

Note

This description applies also to the following parameters: FG_CCSEL1, FG_CCSEL2, FG_CCSEL3

Table 5.17. Parameter properties of FG_CCSEL0

Property	Value
Name	**FG_CCSEL0**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**CC_EXSYNC** Exsync **CC_NOT_EXSYNC** !Exsync **CC_STROBEPULSE** Flash **CC_NOT_STROBEPULSE** !Flash **CC_GND** Gnd **CC_VCC** Vcc
Default value	**CC_EXSYNC**

Example 5.16. Usage of FG_CCSEL0

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

if ((result = Fg_setParameterWithType(FG_CCSEL0, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(FG_CCSEL0, &value, 0, type)) < 0) {
    /* error handling */
}
```

5.5.8.11. FG_DIGIO_OUTPUT

Table 5.18. Parameter properties of FG_DIGIO_OUTPUT

Property	Value
Name	**FG_DIGIO_OUTPUT**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 3** **Stepsize 1**
Default value	**3**

Example 5.17. Usage of FG_DIGIO_OUTPUT

```
int result = 0;
unsigned int value = 3;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

5.5.9. Trigger Input

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

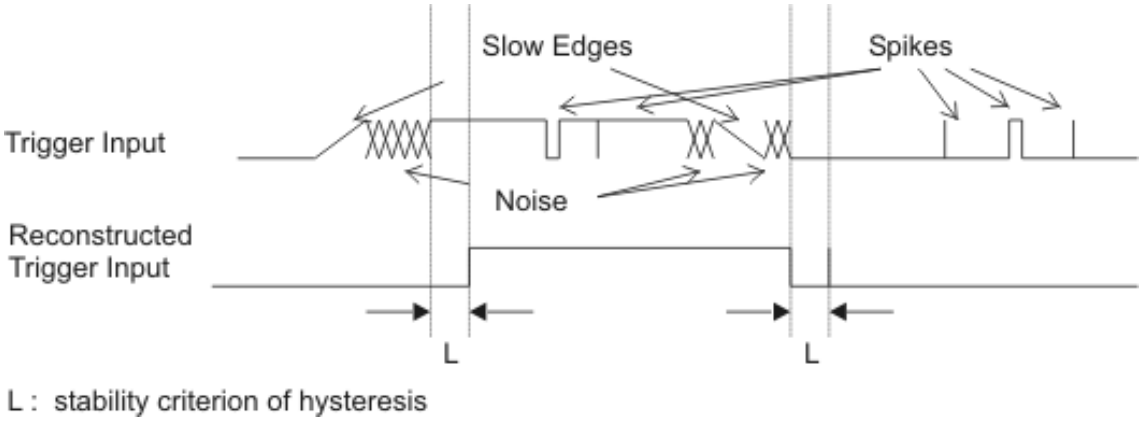
5.5.9.1. External

5.5.9.1.1. FG_TRIGGERIN_DEBOUNCE

In general, a perfect and steady trigger input signal can not be guaranteed in practice. A transfer using long cable connections and the operation in bad shielded

environments might have a distinct influence on the signal quality. Typical problems are strong flattening of the digital's signal edges, occurring interferences during toggling and inducing of short jamming pulses (spikes). In the following figure, some of the influences are illustrated.

Figure 5.18. Faulty Signal and it's Reconstruction



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

Table 5.19. Parameter properties of FG_TRIGGERIN_DEBOUNCE

Property	Value
Name	**FG_TRIGGERIN_DEBOUNCE**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.096** **Maximum 3.024** **Stepsize 0.048**
Default value	**1.0**
Unit of measure	**us**

Example 5.18. Usage of FG_TRIGGERIN_DEBOUNCE

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

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

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

5.5.9.1.2. FG_DIGIO_INPUT

This AcquisitionApplets Advanced has eight 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.

Note that you can also use the input events for monitoring of the digital inputs. REFERENZ EINFÜGEN!

Table 5.20. Parameter properties of FG_DIGIO_INPUT

Property	Value
Name	**FG_DIGIO_INPUT**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 255** **Stepsize 1**

Example 5.19. 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_DIGIO_INPUT, &value, 0, type)) < 0) {
    /* error handling */
}
```

5.5.9.1.3. FG_TRIGGERIN_SRC

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

Table 5.21. Parameter properties of FG_TRIGGERIN_SRC

Property	Value
Name	**FG_TRIGGERIN_SRC**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**TRGINSRC_0** Trigger Source 0 **TRGINSRC_1** Trigger Source 1 **TRGINSRC_2** Trigger Source 2 **TRGINSRC_3** Trigger Source 3 **TRGINSRC_4** Trigger Source 4 **TRGINSRC_5** Trigger Source 5 **TRGINSRC_6** Trigger Source 6 **TRGINSRC_7** Trigger Source 7
Default value	**TRGINSRC_0**

Example 5.20. Usage of FG_TRIGGERIN_SRC

```
int result = 0;
unsigned int value = TRGINSRC_0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

5.5.9.1.4. FG_TRIGGERIN_POLARITY

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

Table 5.22. Parameter properties of FG_TRIGGERIN_POLARITY

Property	Value
Name	**FG_TRIGGERIN_POLARITY**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**LOW_ACTIVE** Low Active **HIGH_ACTIVE** High Active
Default value	**HIGH_ACTIVE**

Example 5.21. Usage of FG_TRIGGERIN_POLARITY

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

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

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

5.5.9.1.5. FG_TRIGGERIN_DOWNSCALE

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

Table 5.23. Parameter properties of FG_TRIGGERIN_DOWNSCALE

Property	Value
Name	**FG_TRIGGERIN_DOWNSCALE**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 256** **Stepsize 1**
Default value	**1**

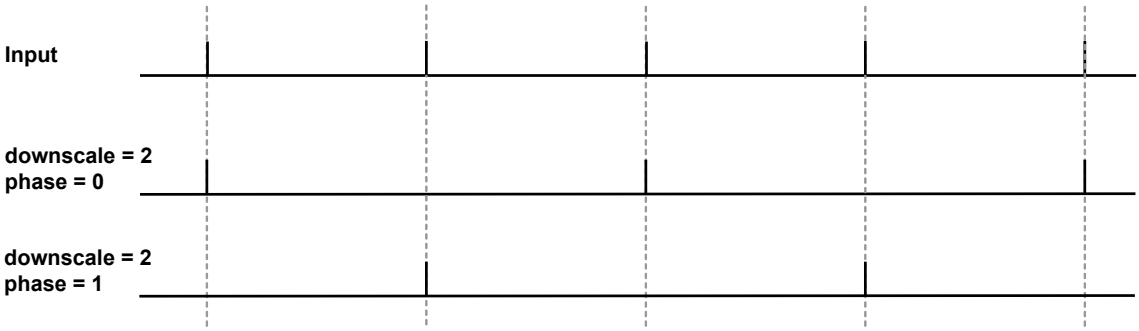
Example 5.22. Usage of FG_TRIGGERIN_DOWNSCALE

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

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

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

5.5.9.1.6. FG_TRIGGERIN_DOWNSCALE_PHASE

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

Figure 5.19. Triggerin Dowscale



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

Table 5.24. Parameter properties of FG_TRIGGERIN_DOWNSCALE_PHASE

Property	Value
Name	**FG_TRIGGERIN_DOWNSCALE_PHASE**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 255** **Stepsize 1**
Default value	**0**

Example 5.23. Usage of FG_TRIGGERIN_DOWNSCALE_PHASE

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

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

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

5.5.9.1.7. FG_TRIGGERIN_BYPASS_SRC

Select an external trigger source used to bypass the signal to an output. The bypass will keep the pulse form, i.e. the signal length of the external signal. For every output and CC signal output. The selected bypass can be selected. See Chapter 5.5.16and Chapter 5.5.17 for output configuration.

Table 5.25. Parameter properties of FG_TRIGGERIN_BYPASS_SRC

Property	Value
Name	**FG_TRIGGERIN_BYPASS_SRC**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**TRGINSRC_0** Trigger Source 0 **TRGINSRC_1** Trigger Source 1 **TRGINSRC_2** Trigger Source 2 **TRGINSRC_3** Trigger Source 3 **TRGINSRC_4** Trigger Source 4 **TRGINSRC_5** Trigger Source 5 **TRGINSRC_6** Trigger Source 6 **TRGINSRC_7** Trigger Source 7
Default value	**TRGINSRC_0**

Example 5.24. Usage of FG_TRIGGERIN_BYPASS_SRC

```
int result = 0;
unsigned int value = TRGINSRC_0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

5.5.9.2. Software Trigger

5.5.9.2.1. FG_SENDSOFTWARETRIGGER

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

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

Table 5.26. Parameter properties of FG_SENDSOFTWARETRIGGER

Property	Value
Name	**FG_SENDSOFTWARETRIGGER**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**Minimum 1** **Maximum 1** **Stepsize 1**
Default value	**1**
Unit of measure	**pulses**

Example 5.25. Usage of FG_SENDSOFTWARETRIGGER

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

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

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

5.5.9.2.2. FG_SOFTWARETRIGGER_IS_BUSY

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

Table 5.27. Parameter properties of FG_SOFTWARETRIGGER_IS_BUSY

Property	Value
Name	**FG_SOFTWARETRIGGER_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 5.26. Usage of FG_SOFTWARETRIGGER_IS_BUSY

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

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

5.5.9.2.3. FG_SOFTWARETRIGGER_QUEUE_FILLLEVEL

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

Table 5.28. Parameter properties of FG_SOFTWARETRIGGER_QUEUE_FILLLEVEL

Property	Value
Name	**FG_SOFTWARETRIGGER_QUEUE_FILLLEVEL**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum** **0** **Maximum** **2040** **Stepsize** **1**
Unit of measure	**pulses**

Example 5.27. Usage of FG_SOFTWARETRIGGER_QUEUE_FILLLEVEL

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

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

5.5.9.3. Statistics

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

5.5.9.3.1. FG_TRIGGERIN_STATS_PULSECOUNT

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

Table 5.29. Parameter properties of FG_TRIGGERIN_STATS_PULSECOUNT

Property	Value
Name	**FG_TRIGGERIN_STATS_PULSECOUNT**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 65535** **Stepsize 1**
Unit of measure	**pulses**

Example 5.28. Usage of FG_TRIGGERIN_STATS_PULSECOUNT

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

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

5.5.9.3.2. FG_TRIGGERIN_STATS_PULSECOUNT_CLEAR

Clear the input pulse counter by writing to this register.

Table 5.30. Parameter properties of FG_TRIGGERIN_STATS_PULSECOUNT_CLEAR

Property	Value
Name	**FG_TRIGGERIN_STATS_PULSECOUNT_CLEAR**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**FG_APPLY** Apply
Default value	**FG_APPLY**

Example 5.29. Usage of FG_TRIGGERIN_STATS_PULSECOUNT_CLEAR

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

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

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

5.5.9.3.3. FG_TRIGGERIN_STATS_FREQUENCY

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

Table 5.31. Parameter properties of FG_TRIGGERIN_STATS_FREQUENCY

Property	Value
Name	**FG_TRIGGERIN_STATS_FREQUENCY**
Type	**Double**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0.0** **Maximum 6.25E7** **Stepsize 2.220446049250313E-16**
Unit of measure	**Hz**

Example 5.30. Usage of FG_TRIGGERIN_STATS_FREQUENCY

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

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

5.5.9.3.4. FG_TRIGGERIN_STATS_MINFREQUENCY

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

Table 5.32. Parameter properties of FG_TRIGGERIN_STATS_MINFREQUENCY

Property	Value
Name	**FG_TRIGGERIN_STATS_MINFREQUENCY**
Type	**Double**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0.0** **Maximum 6.25E7** **Stepsize 2.220446049250313E-16**
Unit of measure	**Hz**

Example 5.31. Usage of FG_TRIGGERIN_STATS_MINFREQUENCY

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

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

5.5.9.3.5. FG_TRIGGERIN_STATS_MAXFREQUENCY

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

Table 5.33. Parameter properties of FG_TRIGGERIN_STATS_MAXFREQUENCY

Property	Value
Name	**FG_TRIGGERIN_STATS_MAXFREQUENCY**
Type	**Double**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0.0** **Maximum 6.25E7** **Stepsize 2.220446049250313E-16**
Unit of measure	**Hz**

Example 5.32. Usage of FG_TRIGGERIN_STATS_MAXFREQUENCY

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

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

5.5.9.3.6. FG_TRIGGERIN_STATS_MINMAXFREQUENCY_CLEAR

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

Table 5.34. Parameter properties of FG_TRIGGERIN_STATS_MINMAXFREQUENCY_CLEAR

Property	Value
Name	**FG_TRIGGERIN_STATS_MINMAXFREQUENCY_CLEAR**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**FG_APPLY** Apply
Default value	**FG_APPLY**

Example 5.33. Usage of FG_TRIGGERIN_STATS_MINMAXFREQUENCY_CLEAR

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

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

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

5.5.9.3.7. FG_TRIGGER_INPUT0_RISING et al.

Note

This description applies also to the following events: FG_TRIGGER_INPUT1_RISING, FG_TRIGGER_INPUT2_RISING, FG_TRIGGER_INPUT3_RISING, FG_TRIGGER_INPUT4_RISING, FG_TRIGGER_INPUT5_RISING, FG_TRIGGER_INPUT6_RISING, FG_TRIGGER_INPUT7_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.

5.5.9.3.8. FG_TRIGGER_INPUT0_FALLING et al.

Note

This description applies also to the following events: FG_TRIGGER_INPUT1_FALLING, FG_TRIGGER_INPUT2_FALLING, FG_TRIGGER_INPUT3_FALLING, FG_TRIGGER_INPUT4_FALLING, FG_TRIGGER_INPUT5_FALLING, FG_TRIGGER_INPUT6_FALLING, FG_TRIGGER_INPUT7_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.

5.5.10. Sequencer

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

Let's have a look at an example. Suppose you have an external trigger source generating a pulse once every second. Your input frequency will then be 1Hz. Assume

that the sequencer is set to a multiplication factor of 2 and the maximum frequency defined by *FG_TRIGGER_FRAMESPERSECOND* is set to 2.1Hz.

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

5.5.10.1. FG_TRIGGER_MULTIPLY_PULSES

Set the trigger input multiplication factor.

Table 5.35. Parameter properties of FG_TRIGGER_MULTIPLY_PULSES

Property	Value
Name	**FG_TRIGGER_MULTIPLY_PULSES**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 65535** **Stepsize 1**
Default value	**1**

Example 5.34. Usage of FG_TRIGGER_MULTIPLY_PULSES

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

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

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

5.5.11. Queue

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

Activate the trigger queue using parameter *FG_TRIGGERQUEUE_MODE*.

The queue fill level can be monitored by parameter *FG_TRIGGERQUEUE_FILLLEVEL*. Moreover, two events allow the monitoring of the fill level.

Using parameters *FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_ON_THRESHOLD* and *FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_OFF_THRESHOLD* it is possible to set two threshold. If the fill level exceeds the ON-threshold the respective event *FG_TRIGGER_QUEUE_FILLLEVEL_THRESHOLD_CAM0_ON* is generated. If the fill level gets less or equal than the OFF-threshold the event *FG_TRIGGER_QUEUE_FILLLEVEL_THRESHOLD_CAM0_OFF* is generated.

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

5.5.11.1. FG_TRIGGERQUEUE_MODE

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

Table 5.36. Parameter properties of FG_TRIGGERQUEUE_MODE

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

Example 5.35. Usage of FG_TRIGGERQUEUE_MODE

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

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

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

5.5.11.2. FG_TRIGGERQUEUE_FILLLEVEL

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

Table 5.37. Parameter properties of FG_TRIGGERQUEUE_FILLLEVEL

Property	Value
Name	**FG_TRIGGERQUEUE_FILLLEVEL**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 2040** **Stepsize 1**
Unit of measure	**pulses**

Example 5.36. Usage of FG_TRIGGERQUEUE_FILLLEVEL

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

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

5.5.11.3. FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_ON_THRESHOLD

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

Table 5.38. Parameter properties of FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_ON_THRESHOLD

Property	Value
Name	**FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_ON_THRESHOLD**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 2** **Maximum 2047** **Stepsize 1**
Default value	**2047**
Unit of measure	**pulses**

Example 5.37. Usage of FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_ON_THRESHOLD

```
int result = 0;
unsigned int value = 2047;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

5.5.11.4. FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_OFF_THRESHOLD

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

Table 5.39. Parameter properties of FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_OFF_THRESHOLD

Property	Value
Name	**FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_OFF_THRESHOLD**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 2** **Maximum 2047** **Stepsize 1**
Default value	**2**
Unit of measure	**pulses**

Example 5.38. Usage of FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_OFF_THRESHOLD

```
int result = 0;
unsigned int value = 2;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

5.5.11.5. FG_TRIGGER_QUEUE_FILLLEVEL_THRESHOLD_CAM0_ON

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

5.5.11.6. FG_TRIGGER_QUEUE_FILLLEVEL_THRESHOLD_CAM0_OFF

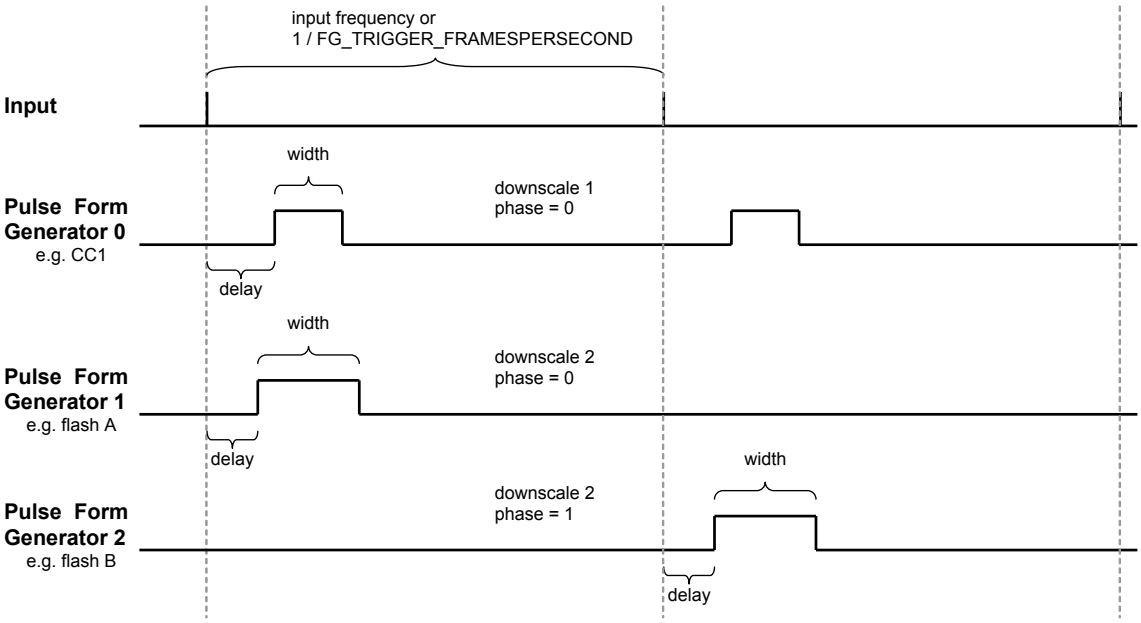
The event is generated if the queue fill level gets less or equal than the OFF-threshold set by parameter *FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_OFF_THRESHOLD*. Except for the timestamp, the event has no additional data included.

5.5.12. Pulse Form Generator 0

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

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

Figure 5.20. Pulse Form Generators



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

Equation 5.2. Dependency of Frequency and Pulse Form Generators

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

- fps = *FG_TRIGGER_FRAMESPERSECOND*
- WIDTH[0..3] = *FG_TRIGGER_PULSEFORMGEN[0..3]_WIDTH*
- DELAY[0..3] = *FG_TRIGGER_PULSEFORMGEN[0..3]_DELAY*
- DOWNSCALE[0..3] = *FG_TRIGGER_PULSEFORMGEN[0..3]_DOWNSCALE*

5.5.12.1. FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE et al.

Note

This description applies also to the following parameters: FG_TRIGGER_PULSEFORMGEN1_DOWNSCALE, FG_TRIGGER_PULSEFORMGEN2_DOWNSCALE, FG_TRIGGER_PULSEFORMGEN3_DOWNSCALE

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

Table 5.40. Parameter properties of FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE

Property	Value
Name	**FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 7** **Stepsize 1**
Default value	**1**

Example 5.39. Usage of FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE

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

if ((result = Fg_setParameterWithType(FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE, &value, 0, type)) < 0) {
    /* error handling */
}
```

5.5.12.2. FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE_PHASE et al.

Note

This description applies also to the following parameters: FG_TRIGGER_PULSEFORMGEN1_DOWNSCALE_PHASE, FG_TRIGGER_PULSEFORMGEN2_DOWNSCALE_PHASE, FG_TRIGGER_PULSEFORMGEN3_DOWNSCALE_PHASE

Parameters *FG_TRIGGER_PULSEFORMGEN[0..3]_DOWNSCALE* and *FG_TRIGGER_PULSEFORMGEN[0..3]_DOWNSCALE_PHASE* are used to downscale the trigger pulses. The downscale value represents the factor. For example value three will remove two out of three successive trigger pulses. The phase is used to make the selection of the pulse in the sequence. For the given example, a phase set to value zero will forward the first pulse and will remove pulses two and three of a sequence of three pulses. Check Section 5.5.12, “Pulse Form Generator 0” for more information.

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

Table 5.41. Parameter properties of FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE_PHASE

Property	Value
Name	**FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE_PHASE**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 6** **Stepsize 1**
Default value	**0**

Example 5.40. Usage of FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE_PHASE

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

if ((result = Fg_setParameterWithType(FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE_PHASE, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE_PHASE, &value, 0, type)) < 0) {
    /* error handling */
}
```

5.5.12.3. FG_TRIGGER_PULSEFORMGEN0_DELAY et al.

Note

This description applies also to the following parameters: FG_TRIGGER_PULSEFORMGEN1_DELAY, FG_TRIGGER_PULSEFORMGEN2_DELAY, FG_TRIGGER_PULSEFORMGEN3_DELAY

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

Table 5.42. Parameter properties of FG_TRIGGER_PULSEFORMGEN0_DELAY

Property	Value
Name	**FG_TRIGGER_PULSEFORMGEN0_DELAY**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.0** **Maximum 3.4E7** **Stepsize 0.016**
Default value	**0.0**
Unit of measure	**us**

Example 5.41. Usage of FG_TRIGGER_PULSEFORMGEN0_DELAY

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

if ((result = Fg_setParameterWithType(FG_TRIGGER_PULSEFORMGEN0_DELAY, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(FG_TRIGGER_PULSEFORMGEN0_DELAY, &value, 0, type)) < 0) {
    /* error handling */
}
```

5.5.12.4. FG_TRIGGER_PULSEFORMGEN0_WIDTH et al.

Note

This description applies also to the following parameters: FG_TRIGGER_PULSEFORMGEN1_WIDTH, FG_TRIGGER_PULSEFORMGEN2_WIDTH, FG_TRIGGER_PULSEFORMGEN3_WIDTH

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

Table 5.43. Parameter properties of FG_TRIGGER_PULSEFORMGEN0_WIDTH

Property	Value
Name	**FG_TRIGGER_PULSEFORMGEN0_WIDTH**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 1.0** **Maximum 6.8E7** **Stepsize 0.016**
Default value	**4000.0**
Unit of measure	**us**

Example 5.42. Usage of FG_TRIGGER_PULSEFORMGEN0_WIDTH

```
int result = 0;
double value = 4000.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

if ((result = Fg_setParameterWithType(FG_TRIGGER_PULSEFORMGEN0_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(FG_TRIGGER_PULSEFORMGEN0_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}
```

5.5.13. Pulse Form Generator 1

The settings for pulse form generator 1 are equal to those of pulse form generator 0. Please read Section 5.5.12, “Pulse Form Generator 0” for a detailed description.

5.5.14. Pulse Form Generator 2

The settings for pulse form generator 2 are equal to those of pulse form generator 0. Please read Section 5.5.12, “Pulse Form Generator 0” for a detailed description.

5.5.15. Pulse Form Generator 3

The settings for pulse form generator 3 are equal to those of pulse form generator 0. Please read Section 5.5.12, “Pulse Form Generator 0” for a detailed description.

5.5.16. 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-input-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 lines.

The trigger system of this AcquisitionApplets Advanced provides several possibilities of mapping sources to the CC lines:

- Pulse form generators 0 to 3

  The pulse form generators are the main output sources of the trigger system. You can either directly one of the four sources to a CC line or invert the signal if you need low active signals.

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

- The input bypass

  The trigger system will ignore the signal length of the input signals. If you want to directly bypass one of the inputs to a CC line, you can set the CC line to bypass or the inverted bypass. The source is selected by parameter *FG_TRIGGERIN_BYPASS_SRC*.

5.5.16.1. FG_TRIGGERCC_SELECT0 et al.

Note

This description applies also to the following parameters: FG_TRIGGERCC_SELECT1, FG_TRIGGERCC_SELECT2, FG_TRIGGERCC_SELECT3

Table 5.44. Parameter properties of FG_TRIGGERCC_SELECT0

Property	Value	
Name	**FG_TRIGGERCC_SELECT0**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**CC_PULSEGEN0**	Pulse Generator 0
	CC_PULSEGEN1	Pulse Generator 1
	CC_PULSEGEN2	Pulse Generator 2
	CC_PULSEGEN3	Pulse Generator 3
	CC_GND	Gnd
	CC_VCC	Vcc
	CC_NOT_PULSEGEN0	!Pulse Generator 0
	CC_NOT_PULSEGEN1	!Pulse Generator 1
	CC_NOT_PULSEGEN2	!Pulse Generator 2
	CC_NOT_PULSEGEN3	!Pulse Generator 3
	CC_INPUT_BYPASS	Input Bypass
	CC_NOT_INPUT_BYPASS	!Input Bypass
Default value	**CC_NOT_PULSEGEN0**	

Example 5.43. Usage of FG_TRIGGERCC_SELECT0

```
int result = 0;
unsigned int value = CC_NOT_PULSEGEN0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(FG_TRIGGERCC_SELECT0, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(FG_TRIGGERCC_SELECT0, &value, 0, type)) < 0) {
    /* error handling */
}
```

5.5.17. Digital Output

The microEnable IV AD4-CL frame grabber has eight digital outputs. This applet is a single camera applet. You can use outputs 0 to 3 only.

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

- Pulse form generators 0 to 4

  The pulse form generators are the main output sources of the trigger system. You can either directly one of the four sources to a digtal output or invert the signal if you need low active signals.

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

- The input bypass

  The trigger system will ignore the signal length of the input signals. If you want to directly bypass one of the inputs to a digital output, you can set the digital output to 'bypass' or the 'inverted bypass' Input signals will then be directly output. The source is selected by parameter *FG_TRIGGERIN_BYPASS_SRC*.

5.5.17.1. FG_TRIGGEROUT_SELECT0 et al.

Note

This description applies also to the following parameters: FG_TRIGGEROUT_SELECT1, FG_TRIGGEROUT_SELECT2, FG_TRIGGEROUT_SELECT3, FG_TRIGGEROUT_SELECT4, FG_TRIGGEROUT_SELECT5, FG_TRIGGEROUT_SELECT6, FG_TRIGGEROUT_SELECT7

Table 5.45. Parameter properties of FG_TRIGGEROUT_SELECT0

Property	Value	
Name	**FG_TRIGGEROUT_SELECT0**	
Type	**Enumeration**	
Access policy	**Read/Write/Change**	
Storage policy	**Persistent**	
Allowed values	**PULSEGEN0**	Pulse Generator 0
	PULSEGEN1	Pulse Generator 1
	PULSEGEN2	Pulse Generator 2
	PULSEGEN3	Pulse Generator 3
	GND	Gnd
	VCC	Vcc
	NOT_PULSEGEN0	!Pulse Generator 0
	NOT_PULSEGEN1	!Pulse Generator 1
	NOT_PULSEGEN2	!Pulse Generator 2
	NOT_PULSEGEN3	!Pulse Generator 3
	INPUT_BYPASS	Input Bypass
	NOT_INPUT_BYPASS	!Input Bypass
Default value	**NOT_PULSEGEN0**	

Example 5.44. Usage of FG_TRIGGEROUT_SELECT0

```
int result = 0;
unsigned int value = NOT_PULSEGEN0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

if ((result = Fg_setParameterWithType(FG_TRIGGEROUT_SELECT0, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(FG_TRIGGEROUT_SELECT0, &value, 0, type)) < 0) {
    /* error handling */
}
```

5.5.17.2. Statistics

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

5.5.17.2.1. FG_TRIGGEROUT_STATS_SOURCE

Table 5.46. Parameter properties of FG_TRIGGEROUT_STATS_SOURCE

Property	Value
Name	**FG_TRIGGEROUT_STATS_SOURCE**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**PULSEGEN0** Pulse Generator 0 **PULSEGEN1** Pulse Generator 1 **PULSEGEN2** Pulse Generator 2 **PULSEGEN3** Pulse Generator 3
Default value	**PULSEGEN0**

Example 5.45. Usage of FG_TRIGGEROUT_STATS_SOURCE

```
int result = 0;
unsigned int value = PULSEGEN0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

5.5.17.2.2. FG_TRIGGEROUT_STATS_PULSECOUNT

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

Table 5.47. Parameter properties of FG_TRIGGEROUT_STATS_PULSECOUNT

Property	Value
Name	**FG_TRIGGEROUT_STATS_PULSECOUNT**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum** **0** **Maximum** **65535** **Stepsize** **1**
Unit of measure	**pulses**

Example 5.46. Usage of FG_TRIGGEROUT_STATS_PULSECOUNT

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

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

5.5.17.2.3. FG_TRIGGEROUT_STATS_PULSECOUNT_CLEAR

Output pulse count register clear.

Table 5.48. Parameter properties of FG_TRIGGEROUT_STATS_PULSECOUNT_CLEAR

Property	Value
Name	**FG_TRIGGEROUT_STATS_PULSECOUNT_CLEAR**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**FG_APPLY** Apply
Default value	**FG_APPLY**

Example 5.47. Usage of FG_TRIGGEROUT_STATS_PULSECOUNT_CLEAR

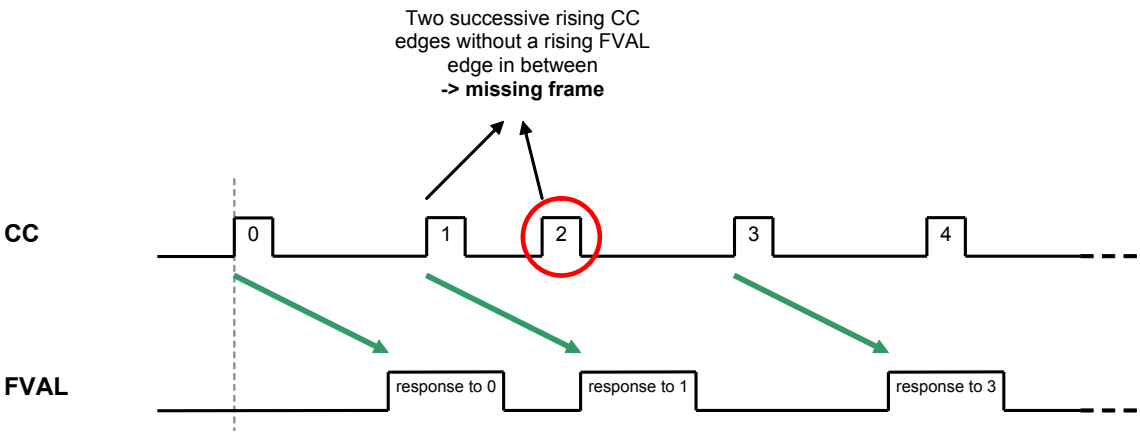
```
int result = 0;
unsigned int value = FG_APPLY;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

5.5.17.2.4. FG_MISSING_CAMERA_FRAME_RESPONSE

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

The idea of the frame loss detection is that for every trigger pulse generated by the trigger system, the camera will send a frame to the frame grabber. If a trigger pulse gets lost, or the camera cannot send a frame, this register will be set to FG_YES. Technically, between two CC output signal edges, a rising FVAL edge has to exist. Or in other words: There must not be two or more successive CC edges without a FVAL edge in between. The following figure illustrates the behavior.

Figure 5.21. Missing Camera Frame Response



The pulse form generator allocated to the CC signal line carrying the image trigger pulses has to be selected by *FG_TRIGGEROUT_STATS_SOURCE*. The missing frame

response system might not work correct for all camera models due to different timings.

Select CC Signal Line

Don't forget to select the pulse form generator feeding the CC signal line which carries the image trigger pulses by setting parameter *FG_TRIGGEROUT_STATS_SOURCE* to the respective source.

Acquisition Start Before Trigger Activation

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

Events for Missing Frame Response

If you want to monitor the exact moment of a missing frame responses, and the exact number of missing frames use event *FG_MISSING_CAM0_FRAME_RESPONSE* .

Table 5.49. Parameter properties of FG_MISSING_CAMERA_FRAME_RESPONSE

Property	Value
Name	**FG_MISSING_CAMERA_FRAME_RESPONSE**
Type	**Enumeration**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**FG_YES** Yes **FG_NO** No

Example 5.48. Usage of FG_MISSING_CAMERA_FRAME_RESPONSE

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

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

5.5.17.2.5. FG_MISSING_CAMERA_FRAME_RESPONSE_CLEAR

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

Table 5.50. Parameter properties of FG_MISSING_CAMERA_FRAME_RESPONSE_CLEAR

Property	Value
Name	**FG_MISSING_CAMERA_FRAME_RESPONSE_CLEAR**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**FG_APPLY** Apply
Default value	**FG_APPLY**

Example 5.49. Usage of FG_MISSING_CAMERA_FRAME_RESPONSE_CLEAR

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

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

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

5.5.17.2.6. FG_MISSING_CAM0_FRAME_RESPONSE

The missing camera frame response event is generated for each camera output trigger pulse with no frame response. Please read the description of the detection and the documentation on how to configure the mechanism in Chapter 5.5.17.2.4 carefully. If the mechanism is compatible with the used camera and set up correct, the number of events is equal to the number of lost frames. Except for the timestamp, the event has no additional data included.

5.5.18. Output Event

5.5.18.1. FG_TRIGGER_OUTPUT_EVENT_SELECT

Select the source for the output event *FG_TRIGGER_OUTPUT_CAM0* with this register. One of the pulse form generators can be selected.

Table 5.51. Parameter properties of FG_TRIGGER_OUTPUT_EVENT_SELECT

Property	Value
Name	**FG_TRIGGER_OUTPUT_EVENT_SELECT**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**PULSEGEN0** Pulse Generator 0 **PULSEGEN1** Pulse Generator 1 **PULSEGEN2** Pulse Generator 2 **PULSEGEN3** Pulse Generator 3
Default value	**PULSEGEN0**

Example 5.50. Usage of FG_TRIGGER_OUTPUT_EVENT_SELECT

```
int result = 0;
unsigned int value = PULSEGEN0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

5.5.18.2. FG_TRIGGER_OUTPUT_CAM0

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

Chapter 6. Overflow

The applet processes image data as fast as possible. Any image data sent by the camera is immediately processed and send 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. 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. This buffer can store up to 32,000 image lines. 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 1.3, "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.

6.1. FG_FILLLEVEL

The fill-level of the frame grabber buffers used in this AcquisitionApplets Advanced 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 6.1. Parameter properties of FG_FILLLEVEL

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

Example 6.1. 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_FILLLEVEL, &value, 0, type)) < 0) {
    /* error handling */
}
```

6.2. 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 the loose of images. To avoid overflows reduce the mean input bandwidth.

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

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

Table 6.2. Parameter properties of FG_OVERFLOW

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

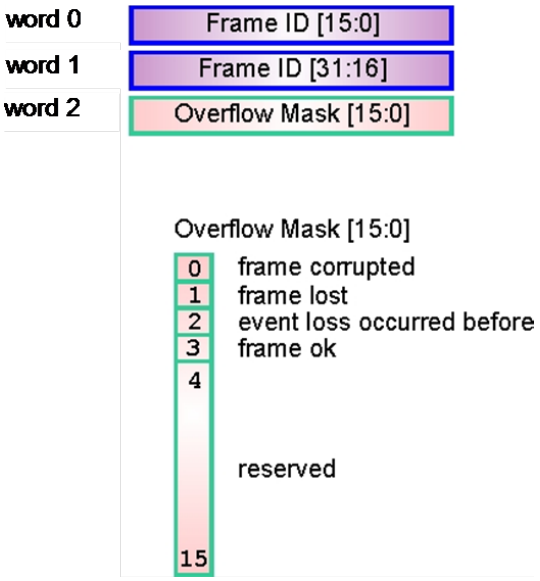
Example 6.2. Usage of FG_OVERFLOW

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

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

6.3. 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.

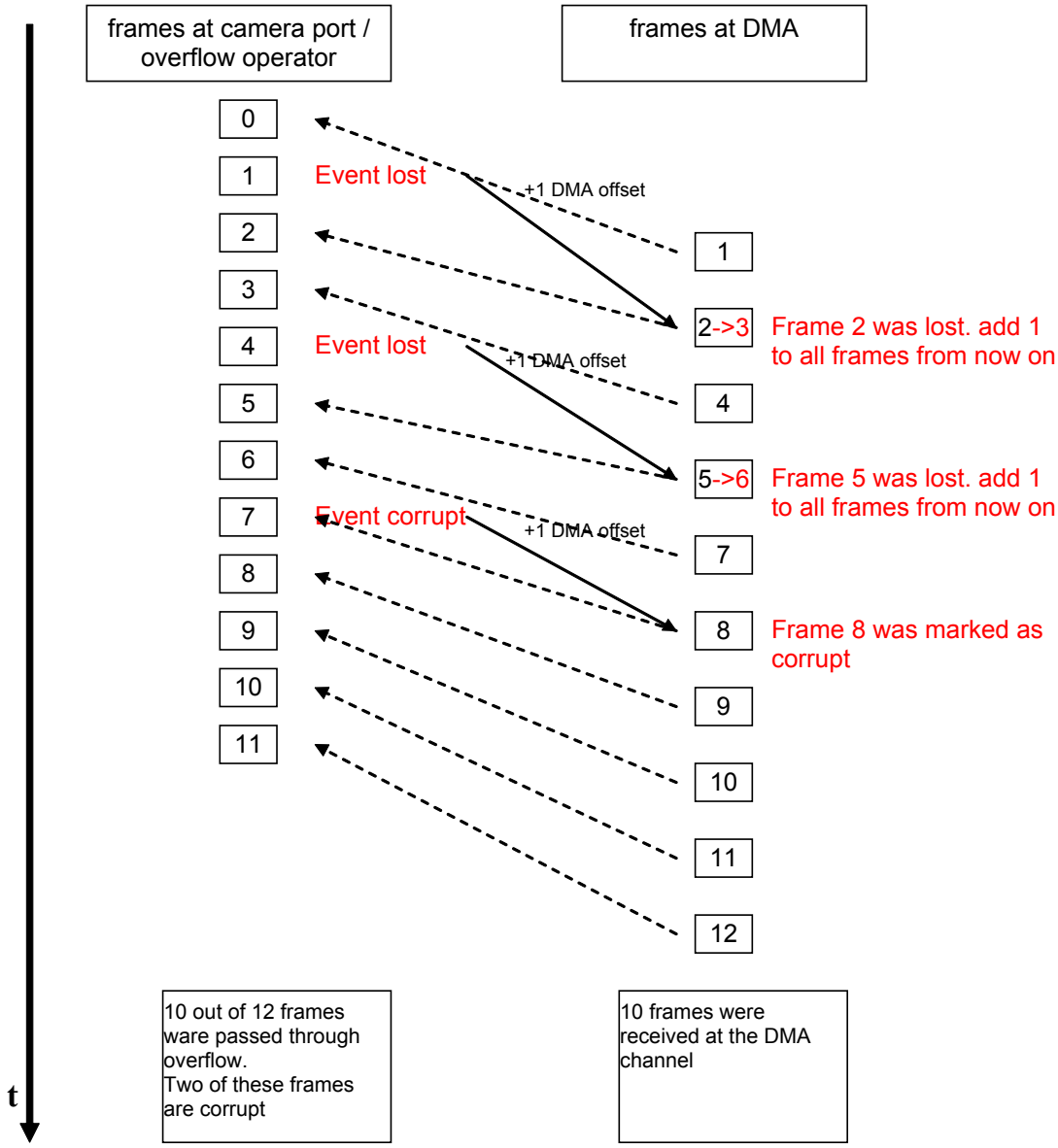
Figure 6.1. Illustration of Overflow Data Packet



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 6.2. 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.

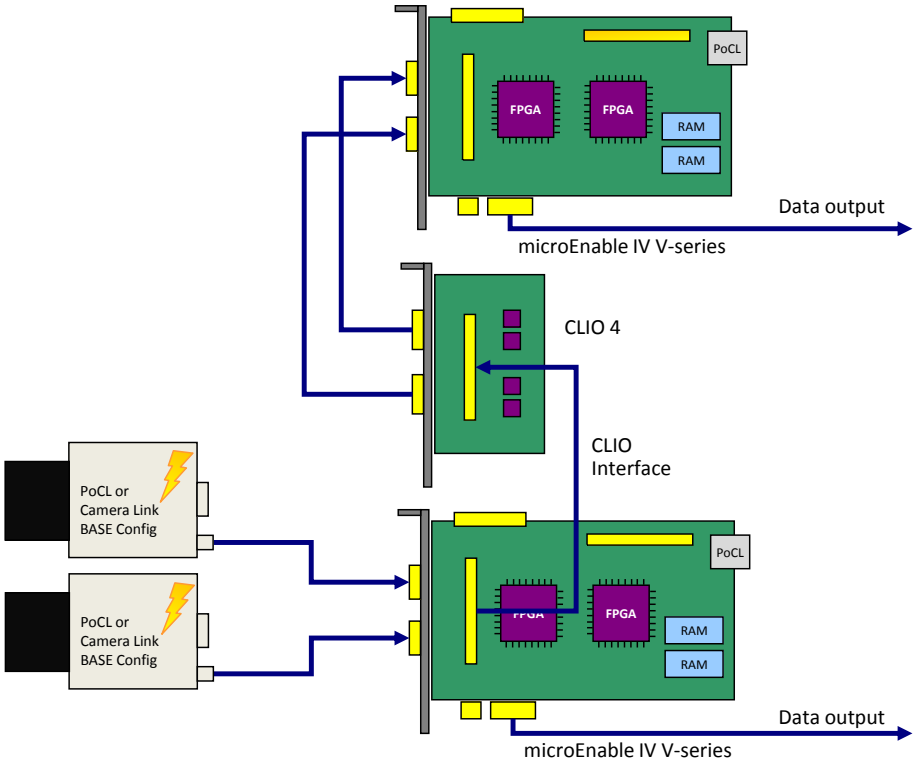
Chapter 7. Image Selector

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.

A typical setup to use this module would be the following: 2 frame grabbers and 1 CameraLink IO board (CLIO) in one PC or two PCs.

The CLIO module distributes the camera output to *p* frame grabbers by copying the image data. The image selector at each frame grabber is programmed to accept only one of *p* images, which decreases the processing frame rate in the grabber and thus, the frame rate at the PCI Express interface. This can be useful if a PC cannot compute the full data rate or if different pre-processing is required to be performed in the frame grabbers.

Figure 7.1. CameraLink IO Board Setup



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 which are connected via a CLIO board to one camera. 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 synchronously used. 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 grabber 0 i.e. the grabber which is directly connected to the camera. Next, the acquisitions of both grabbers have to be started and finally, the trigger generation has to be enabled (generally by setting *FG_TRIGGERSTATE* to active). Now the camera will start sending image data and the grabbers run synchronously. More information can be found in the trigger chapter. Chapter 5, *Trigger*

7.1. 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 7.1. Parameter properties of FG_IMG_SELECT_PERIOD

Property	Value
Name	**FG_IMG_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 7.1. 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_IMG_SELECT_PERIOD, &value, 0, type)) < 0) {
    /* error handling */
}

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

7.2. 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 7.2. Parameter properties of FG_IMG_SELECT

Property	Value
Name	**FG_IMG_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 7.2. 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_IMG_SELECT, &value, 0, type)) < 0) {
    /* error handling */
}

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

Chapter 8. Shading Correction

This AcquisitionApplets Advanced includes a two-dimensional shading correction, also called flat-field correction. It consists of a subtractive i.e. offset shading correction and a multiplicative i.e. gain shading correction .

- The offset (subtractive) correction eliminates the dark current in your camera sensor and adjust a fixed pattern imprinted into your sensor.
- The gain (multiplicative) correction counterbalances inhomogeneities of the used light source and optics.

Shading Resolution

In this applet (Acq_MediumAreaRGB36) it is not possible to define a correction value for each pixel. Neighbored pixels share shading correction values. In this particular applet 2 successive pixels of the same line share the correction values. The configuration progress for the user is unchanged. Values are defined for each image pixel or images at full resolution are loaded into the shading module. However, the applet will calculate the mean value of the neighbored pixels and will only write one correction value to hardware. Of course, color components are not mixed. The mean value is determined for each component separately.

Usage

Specify your own correction values or use the automatic shading correction function.

You will have two possibilities to use the shading correction. The first possibility is to define correction values for each frame pixel position i.e. an offset and a gain correction value can be loaded for each frame pixel position. The second possibility to use the shading correction is the automatic shading correction functionality, where a grabbed black and gray image are uploaded to the applet and are used as references. The applet determines the correction values itself.

To learn on how to use both correction modes read Section 8.3, "Auto Shading Correction" and Chapter 9, *Shading Custom Values*. But before, we want to give you some technical information of the underlying calculations in the. These information is not required to use the shading correction, but will help interested users to understand the functionality of the implementation and will allow the possibility to recompute the frame grabber data output.

The following equation represents the shading correction calculation in the applet.

Equation 8.1. Shading Correction

$$I^*(u,v) = \begin{cases} (I(u,v) - offset(u,v)) * gain(u,v) & \text{if } I(u,v) - offset(u,v) > 0 \\ 0 & \text{else} \end{cases}$$

where I represents the input image and I^* the output image. u and v represent the image coordinates. Depending on the used applet, the performed calculations use varying bit depths.

This AcquisitionApplets Advanced is processing image data with a bit depth of 12Bit. The offset correction is using a 16Bit fixed point calculation. The 12Bit image pixels are

converted into 16bit fixed point values with 12 integer bits and 4 fractional bits (left shift). Thus, the precision of the offset correction is $\frac{1}{2^4} = \frac{1}{16}$ which also represents the stepsize of the offset correction values. The offset correction values have the range from 0 to $\frac{2^{16}-1}{2^4}$. The results of the offset correction i.e. the results after subtraction might include negative values. These negative values are clipped to 0. The gain correction in this applet also uses 16bit fixed point values. The correction values enclose 4 integer bits and 12 fractional bits. Thus they have a range from 0 to $\frac{2^{16}-1}{2^{12}} = 15.99$ and the stepsize is $\frac{1}{2^{12}} = \frac{1}{4096}$. The shading correction output is 12Bit again.

Let's assume the following example: We have a pixel value of 453, an offset correction value of 12.5 and a gain correction value of 1.2. The result will then be:

Equation 8.2. Shading Correction - Sample Calculation

$$I^* = round\left(\frac{\left(453 * 2^4 - round\left(12.5 * 2^4\right)\right) * round\left(1.2 * 2^{12}\right)}{2^{16}}\right) = 529$$

Table 8.1. Shading Correction Bit Depths

Correction	Correction Value From	Correction Value To	Stepsize	Integer Bits	Fractional Bits
Offset	0	4096 - 1/16	1/16	12	4
Gain	0	16 - 1/4096	1/4096	4	12

8.1. FG_SHADING_OFFSET_ENABLE

This parameter is used to enable or disable the offset correction. An offset correction can only be performed if correction values have been successfully uploaded to the frame grabber. This can either be done with the auto white balancing functionality or by manually setting the correction values table. The offset correction can be enabled or disabled at any time, even during a running acquisition.

Table 8.2. Parameter properties of FG_SHADING_OFFSET_ENABLE

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

Example 8.1. Usage of FG_SHADING_OFFSET_ENABLE

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

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

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

8.2. FG_SHADING_GAIN_ENABLE

This parameter is used to enable or disable the gain correction. A gain correction can only be performed if correction values have been successfully uploaded to the frame grabber. This can either be done with the auto white balancing functionality or by manually setting the correction values table. The gain correction can be enabled or disabled at any time, even during a running acquisition.

Table 8.3. Parameter properties of FG_SHADING_GAIN_ENABLE

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

Example 8.2. Usage of FG_SHADING_GAIN_ENABLE

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

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

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

8.3. Auto Shading Correction

The automatic shading correction functionality allows you to easily perform a shading correction for your camera. All you need are two reference images, namely a 'black' and a 'gray' image. These images have to be stored in files and have to use the common tiff image format. Various possibilities exist to generate the images. The easiest will be the use of the program microDisplay which is included in the SiliconSoftware runtime. Your camera has to be run in the desired operation modes. For recording the reference image use this applet, and do not apply any pre-processing as you will need the RAW data from the camera.

The 'black' image is used to calculate the offset correction and to eliminate the dark-current of the camera's image sensor. To record the image, cover the camera's lense to avoid any light getting on the sensor. The pixel values of the black image represent the offset correction values. They will be subtracted from the acquired image which eliminates the camera sensor's dark-current. The 'gray' image should be recorded under normal conditions with full illumination, but avoid the camera to be in saturation. The gain correction values are determined using the gray image reference file. The auto shading correction determines the required multiplication factors for each pixel in the gray reference image to bring all image pixels to the same reference value. You will have three possibilities to define this white reference value using parameter *FG_SHADING_GAIN_CORRECTION_MODE*.

- Maximum Value:

  The white reference is set to the pixel value of the pixel which has the the maximum value in the gray reference image file.

- Mean Value:

  The white reference is defined by the mean value of all pixel values in the gray reference imge file.

- Maximum Range:

  The white reference is defined by the maximum possible value i.e. 2^12 - 1 for this applet. Hence, the auto shading correction will try to find a gain correction value which will bring every white value to the saturation edge.

The applet can now be parameterized with the just recorded reference image filenames which is shown in the following sections. Note that all settings only take effect after writing to parameter *FG_SHADING_APPLY_SETTINGS*.

8.3.1. FG_SHADING_BLACK_FILENAME

Specify the filename and path to your black reference file.

Table 8.4. Parameter properties of FG_SHADING_BLACK_FILENAME

Property	Value
Name	**FG_SHADING_BLACK_FILENAME**
Type	**String**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Default value	**""**

Example 8.3. Usage of FG_SHADING_BLACK_FILENAME

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

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

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

8.3.2. FG_SHADING_GRAY_FILENAME

Specify the filename and path to your gray reference file.

Table 8.5. Parameter properties of FG_SHADING_GRAY_FILENAME

Property	Value
Name	**FG_SHADING_GRAY_FILENAME**
Type	**String**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Default value	**""**

Example 8.4. Usage of FG_SHADING_GRAY_FILENAME

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

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

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

8.3.3. FG_SHADING_GAIN_CORRECTION_MODE

Parameterize the method of how the auto white balancing core will calculate the gain correction values. An explanation of the modes is given in Section 8.3, "Auto Shading Correction".

Table 8.6. Parameter properties of FG_SHADING_GAIN_CORRECTION_MODE

Property	Value
Name	**FG_SHADING_GAIN_CORRECTION_MODE**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_MAX_VALUE** Maximum Value **FG_MEAN_VALUE** Mean Value **FG_MAX_RANGE** Maximum Range
Default value	**FG_MAX_VALUE**

Example 8.5. Usage of FG_SHADING_GAIN_CORRECTION_MODE

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

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

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

8.3.4. FG_SHADING_APPLY_SETTINGS

After all settings have been made, the correction values can be determined and uploaded to the frame grabber. This is initialized by writing FG_APPLY to this parameter. Mind that the calculation and upload might require some seconds.

Table 8.7. Parameter properties of FG_SHADING_APPLY_SETTINGS

Property	Value
Name	**FG_SHADING_APPLY_SETTINGS**
Type	**Enumeration**
Access policy	**Read/Write**
Storage policy	**Transient**
Allowed values	**FG_APPLY** Apply
Default value	**FG_APPLY**

Example 8.6. Usage of FG_SHADING_APPLY_SETTINGS

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

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

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

Chapter 9. Shading Custom Values

As mentioned before, the shading correction can be used by defining values for each pixel or by using the automatic shading correction which will determine the values using reference images. In the following, a description on how to load used defined correction values into the shading correction module is presented. Basically, the shading functions of the SDK are used. Have a look in the Silicon Software Runtime user manual for a detailed description of these functions.

This applet (Acq_MediumAreaRGB36) is an applet to process rgb camera images. Therefore, for each pixel three offset and a gain correction values can be specified. One for each component. The following source code shows an SDK example on how to load custom values into the shading correction module.

```
// ... Fg_Init(...)

int shadingSet = 0; // Always 0 for this applet
int cameraIndex = 0; // Select the index of the camera

//In this examples, the same correction values are applied for all pixels:
float offset_red = 0.1; // An offset of 0.1 results in a subtraction of 0.1 * 2^12
float offset_green = 0.2; // An offset of 0.2 results in a subtraction of 0.2 * 2^12
float offset_blue = 0.1; // An offset of 0.1 results in a subtraction of 0.1 * 2^12
float gain_red = 1.2; // Each pixel value will be multiplied with value 1.2
float gain_red = 1.5; // Each pixel value will be multiplied with value 1.5
float gain_red = 1.3; // Each pixel value will be multiplied with value 1.3

ShadingMaster* sm = Fg_AllocShading(fg, shadingSet, port); ❶

if (sm == 0)
    return -1;

if (Shad_GetAccess(fg, sm) != FG_OK)
    return -1;

int height = Shad_GetMaxLine(fg, sm); // Determines the maximum image height (4096 for this applet)

if (height <= 0)
    return -1;

int width = 4096;

for (int y = 0; y < height; ++y) ❷
{
    //  --- change values in the current line
    for (int x = 0; x < width; ++x)
    {
        if (Shad_SetSubValueLine(fg, sm, x, 0, offset) != FG_OK) //red ❸
            return -1;
        if (Shad_SetMultValueLine(fg, sm, x, 0, gain) != FG_OK) //red ❹
            return -1;
        if (Shad_SetSubValueLine(fg, sm, x, 1, offset) != FG_OK) //green
            return -1;
        if (Shad_SetMultValueLine(fg, sm, x, 1, gain) != FG_OK) //green
            return -1;
        if (Shad_SetSubValueLine(fg, sm, x, 2, offset) != FG_OK) //blue
            return -1;
        if (Shad_SetMultValueLine(fg, sm, x, 2, gain) != FG_OK) //blue
            return -1;
    }

    // store the current line as line y (invalidates the values in the current line for the next usage)
        Shad_WriteActLine(fg, sm, y); ❺
}

// write the stored lines to the hardware
if (Shad_FreeAccess(fg, sm) != FG_OK) ❻
```

```
    return -1;

if (Fg_FreeShading(fg, sm) != FG_OK)
    return -1;
```

❶ Allocates a temporary buffer for shading value.
❷ A loop over all pixel values is performed in this example. There is no need to set a correction value for each pixel in the allowed range. It is possible to set a value for the used ROI only, or for the failing pixel only.
❸ Set the value for offset correction.
❹ Set the value for gain correction.
❺ Finish setting values for the current line.
❻ Up to now, no values have been loaded to the frame grabber. The free access function will start the upload of the values to the frame grabber. Keep in mind that this could last a few seconds.

Keep in mind that the corrections still have to be activated on the frame grabber by using parameters *FG_SHADING_OFFSET_ENABLE* and *FG_SHADING_GAIN_ENABLE*.

Chapter 10. White Balance

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

White Balancing using the Shading Correction

Please note, that an individual white balancing for each pixel value can be realized with the 2D shading correction. The shading correction also enables an automatic white balancing. See Chapter 8, *Shading Correction* for more information. If the shading calibration is used, perform the calibration before changing the white balancing parameters.

10.1. FG_SCALINGFACTOR_RED

Table 10.1. Parameter properties of FG_SCALINGFACTOR_RED

Property	Value
Name	**FG_SCALINGFACTOR_RED**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.0** **Maximum 3.9990234375** **Stepsize 9.765625E-4**
Default value	**1.0**

Example 10.1. Usage of FG_SCALINGFACTOR_RED

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

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

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

10.2. FG_SCALINGFACTOR_BLUE

Table 10.2. Parameter properties of FG_SCALINGFACTOR_BLUE

Property	Value
Name	**FG_SCALINGFACTOR_BLUE**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.0** **Maximum 3.9990234375** **Stepsize 9.765625E-4**
Default value	**1.0**

Example 10.2. Usage of FG_SCALINGFACTOR_BLUE

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

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

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

10.3. FG_SCALINGFACTOR_GREEN

Table 10.3. Parameter properties of FG_SCALINGFACTOR_GREEN

Property	Value
Name	**FG_SCALINGFACTOR_GREEN**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.0** **Maximum 3.9990234375** **Stepsize 9.765625E-4**
Default value	**1.0**

Example 10.3. Usage of FG_SCALINGFACTOR_GREEN

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

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

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

Chapter 11. Lookup Table

This AcquisitionApplets Advanced includes a full resolution lookup table (LUT) for each of the three color components. 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 12). The lookup table can also be loaded with custom values. Application areas are: custom image enhancements or 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 4096. All color components are treated separately.

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.

11.1. 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 12, *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 parameters *FG_LUT_VALUE_RED*, *FG_LUT_VALUE_GREEN* and *FG_LUT_VALUE_BLUE*. 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 11.1. Parameter properties of FG_LUT_TYPE

Property	Value	
Name	**FG_LUT_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 11.1. Usage of FG_LUT_TYPE

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

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

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

11.2. FG_LUT_VALUE_RED

Table 11.2. Parameter properties of FG_LUT_VALUE_RED

Property	Value
Name	**FG_LUT_VALUE_RED**
Type	**Unsigned Integer Field**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Default value	**0**

Example 11.2. Usage of FG_LUT_VALUE_RED

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

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

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

11.3. FG_LUT_VALUE_GREEN

Table 11.3. Parameter properties of FG_LUT_VALUE_GREEN

Property	Value
Name	**FG_LUT_VALUE_GREEN**
Type	**Unsigned Integer Field**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Default value	**0**

Example 11.3. Usage of FG_LUT_VALUE_GREEN

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

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

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

11.4. FG_LUT_VALUE_BLUE

Table 11.4. Parameter properties of FG_LUT_VALUE_BLUE

Property	Value
Name	**FG_LUT_VALUE_BLUE**
Type	**Unsigned Integer Field**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Default value	**0**

Example 11.4. Usage of FG_LUT_VALUE_BLUE

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

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

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

11.5. 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 detailed explanation of the lookup table file format can be found in the runtime user manual.

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

Table 11.5. Parameter properties of FG_LUT_CUSTOM_FILE

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

Example 11.5. 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_LUT_CUSTOM_FILE, &value, 0, type)) < 0) {
    /* error handling */
}

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

11.6. FG_LUT_SAVE_FILE

To save the current lookup table configuration to 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 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 11.6. Parameter properties of FG_LUT_SAVE_FILE

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

Example 11.6. 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_LUT_SAVE_FILE, &value, 0, type)) < 0) {
    /* error handling */
}

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

11.7. Applet Properties

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

11.7.1. FG_LUT_IMPLEMENTATION_TYPE

In this applet, a full lookup table is implemented.

Table 11.7. Parameter properties of FG_LUT_IMPLEMENTATION_TYPE

Property	Value
Name	**FG_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 11.7. Usage of FG_LUT_IMPLEMENTATION_TYPE

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

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

11.7.2. FG_LUT_IN_BITS

This applet is using 12 lookup table input bits.

Table 11.8. Parameter properties of FG_LUT_IN_BITS

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

Example 11.8. 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_LUT_IN_BITS, &value, 0, type)) < 0) {
    /* error handling */
}
```

11.7.3. FG_LUT_OUT_BITS

This applet is using 12 lookup table output bits.

Table 11.9. Parameter properties of FG_LUT_OUT_BITS

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

Example 11.9. 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_LUT_OUT_BITS, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 12. 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 12.1. 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 12.2. LUT Processor with Inversion

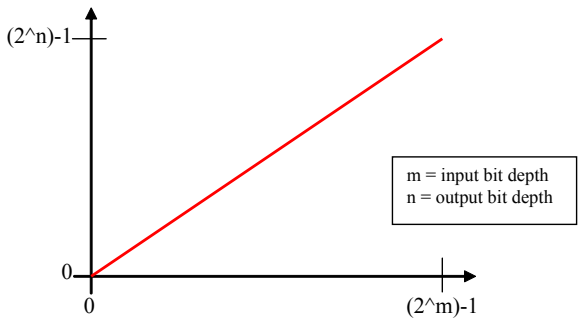
$$Output(x) = 2^{12} - 1 - \left[\left[gain * \left(\frac{x}{2^{12}-1} + offset\right)\right]^{\frac{1}{gamma}}\right] * \left(2^{12}-1\right)$$

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.

This applet processes 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 following figure. In the following, you will find detailed explanations for all processing parameters.

Figure 12.1. Lookup Table Processing: Identity



12.1. 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 The following figure shows an example of an offset. Identity = 0.0

Figure 12.2. Lookup Table Processing: Offset

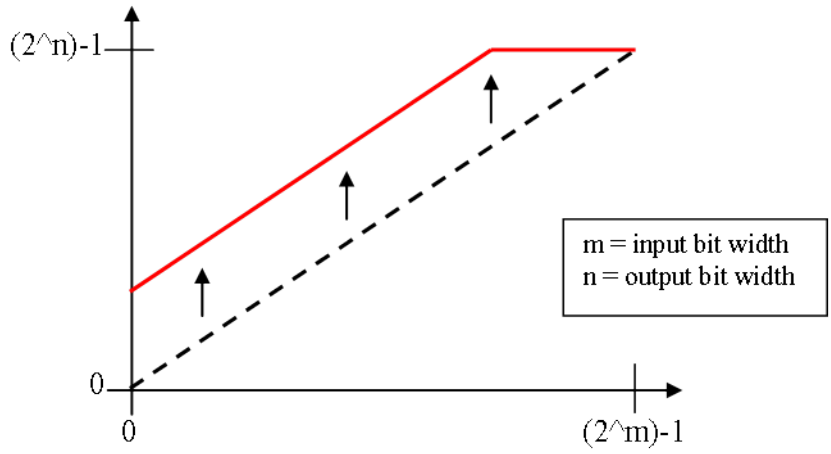


Table 12.1. Parameter properties of FG_PROCESSING_OFFSET

Property	Value
Name	**FG_PROCESSING_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 12.1. 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_PROCESSING_OFFSET, &value, 0, type)) < 0) {
    /* error handling */
}

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

12.2. 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 12.3. Lookup Table Processing: Gain

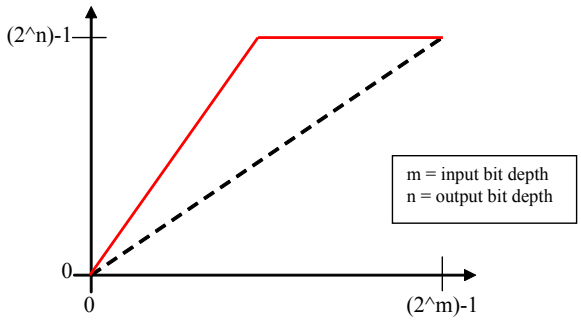


Table 12.2. Parameter properties of FG_PROCESSING_GAIN

Property	Value
Name	**FG_PROCESSING_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 12.2. 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_PROCESSING_GAIN, &value, 0, type)) < 0) {
    /* error handling */
}

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

12.3. 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 $p' = p^{1/gamma}$. Identity = 1.0

Figure 12.4. Lookup Table Processing: Gamma

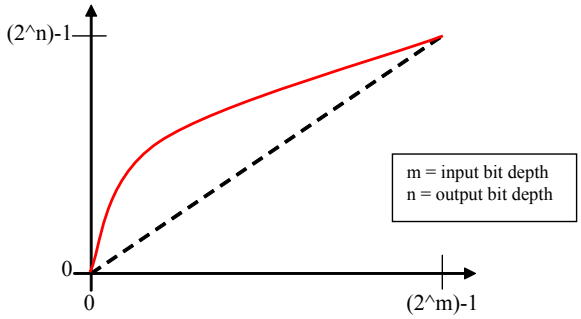


Table 12.3. Parameter properties of FG_PROCESSING_GAMMA

Property	Value
Name	**FG_PROCESSING_GAMMA**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum** 2.220446049250313E-16 **Maximum** 1.7976931348623157E308 **Stepsize** 2.220446049250313E-16
Default value	**1.0**

Example 12.3. 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_PROCESSING_GAMMA, &value, 0, type)) < 0) {
    /* error handling */
}

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

12.4. FG_PROCESSING_INVERT

When 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 12.5. Lookup Table Processing: Invert

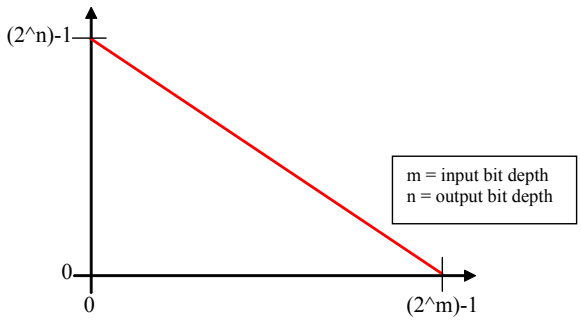


Table 12.4. Parameter properties of FG_PROCESSING_INVERT

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

Example 12.4. Usage of FG_PROCESSING_INVERT

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

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

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

Chapter 13. Output Format

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

13.1. 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. A possibly differing applet processing bit depth from the output format is compensated by adaption to the output format bit depth.

This applet supports the following output formats:

Table 13.1. Parameter properties of FG_FORMAT

Property	Value
Name	**FG_FORMAT**
Type	**Enumeration**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**FG_COL24** Color 24bit **FG_COL48** Color 48bit
Default value	**FG_COL24**

Example 13.1. Usage of FG_FORMAT

```
int result = 0;
unsigned int value = FG_COL24;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

13.2. FG_BITALIGNMENT

The bit alignment is used to map the pixel bits of the internal processing bit depth (12 for the this respective AcquisitionApplets Advanced) to the DMA output bit depth. If the output bit depth is greater than the internal processing bit depth, the bits can either be left- or right-aligned in the output. For example, an internal processing bit depth of 12 bit has to be mapped into a 16 bit output. If the parameter is set to **FG_LEFT_ALIGNED**, the output bits 4 to 11 will transport the pixel data, while bits 0 to 3 are set to zero. If the parameter is set to **FG_RIGHT_ALIGNED**, the output bits 0 to 11 will transport the pixel data. Now, the upper four bits are set to zero.

If the output bit depth is less than the internal processing bit depth, the parameter has no effect as the pixel are automatically left shifted.

If the DMA output bit depth is less than the internal bit depth, values will be rounded. For example the 12 bit value 3785 has to be mapped into an 8-bit output. The value will be divided by 16 (= 236.5625) and rounded to value 237.

Table 13.2. Parameter properties of FG_BITALIGNMENT

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

Example 13.2. Usage of FG_BITALIGNMENT

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

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

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

13.3. FG_PIXELDEPTH

The pixel depth delivers the information of the number of valid bits/pixel. E.g. when using a 10 bit/pixel camera and a 16 bit/pixel output format with right-alignment, then pixel depth is 10 to indicate that the lower 10 bit of the 16 bit value cover the pixel.

Table 13.3. Parameter properties of FG_PIXELDEPTH

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

Example 13.3. 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_PIXELDEPTH, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 14. 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 frequency. 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 patterns will use the full bit depth available for the applet i.e. they will have a bit depth of 12Bit for this applet.

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

Figure 14.1. Generator Pattern

14.1. FG_GEN_ENABLE

The generator is enabled with this parameter. Note that an activated generator will have effect on parameter *FG_CAMSTATUS*.

Table 14.1. Parameter properties of FG_GEN_ENABLE

Property	Value
Name	**FG_GEN_ENABLE**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_CAMPORT** Camera **FG_GENERATOR** Generator
Default value	**FG_CAMPORT**

Example 14.1. Usage of FG_GEN_ENABLE

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

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

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

14.2. FG_GEN_START

This will enable or disable the generation of image data.

Table 14.2. Parameter properties of FG_GEN_START

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

Example 14.2. Usage of FG_GEN_START

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

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

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

14.3. FG_GEN_WIDTH

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

Table 14.3. Parameter properties of FG_GEN_WIDTH

Property	Value
Name	**FG_GEN_WIDTH**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 65535** **Stepsize 1**
Default value	**1024**
Unit of measure	**pixel**

Example 14.3. Usage of FG_GEN_WIDTH

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

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

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

14.4. FG_GEN_HEIGHT

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

Table 14.4. Parameter properties of FG_GEN_HEIGHT

Property	Value
Name	**FG_GEN_HEIGHT**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 65535** **Stepsize 1**
Default value	**1024**
Unit of measure	**pixel**

Example 14.4. Usage of FG_GEN_HEIGHT

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

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

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

14.5. FG_GEN_LINE_GAP

The line gap can be specified with this parameter. A high value reduces the line rate of the camera simulator.

Table 14.5. Parameter properties of FG_GEN_LINE_GAP

Property	Value
Name	**FG_GEN_LINE_GAP**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 65535** **Stepsize 1**
Default value	**4**
Unit of measure	**pixel**

Example 14.5. Usage of FG_GEN_LINE_GAP

```
int result = 0;
unsigned int value = 4;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

14.6. FG_GEN_FREQ

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.

Using the pixel frequency, the resulting frame rate (fps) can be obtained.

Equation 14.1. Shading Correction

$$fps = \frac{FG_GEN_FREQ}{(FG_GEN_WIDTH + FG_GEN_LINE_GAP) * FG_GEN_HEIGHT}$$

Table 14.6. Parameter properties of FG_GEN_FREQ

Property	Value
Name	**FG_GEN_FREQ**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 3.0** **Maximum 125.0** **Stepsize 1.0**
Default value	**40.0**
Unit of measure	**MHz**

Example 14.6. Usage of FG_GEN_FREQ

```
int result = 0;
double value = 40.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

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

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

14.7. FG_GEN_ACCURACY

The accuracy enhances the allowed frequency range, but downgrades the accuracy itself.

Table 14.7. Parameter properties of FG_GEN_ACCURACY

Property	Value
Name	**FG_GEN_ACCURACY**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 65535** **Stepsize 1**
Default value	**25**

Example 14.7. Usage of FG_GEN_ACCURACY

```
int result = 0;
unsigned int value = 25;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

14.8. FG_GEN_TAP1 et al.

Note

This description applies also to the following parameters: FG_GEN_TAP2, FG_GEN_TAP3, FG_GEN_TAP4

If a tap is disabled, it will output black pixels instead of the generated pattern.

Table 14.8. Parameter properties of FG_GEN_TAP1

Property	Value
Name	**FG_GEN_TAP1**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_ON** On **FG_OFF** Off
Default value	**FG_ON**

Example 14.8. Usage of FG_GEN_TAP1

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

if ((result = Fg_setParameterWithType(FG_GEN_TAP1, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(FG_GEN_TAP1, &value, 0, type)) < 0) {
    /* error handling */
}
```

14.9. FG_GEN_ROLL

The generated pattern can be 'rolled'. The speed is controlled by this parameter.

Table 14.9. Parameter properties of FG_GEN_ROLL

Property	Value
Name	**FG_GEN_ROLL**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum** **0** **Maximum** **255** **Stepsize** **1**
Default value	**0**

Example 14.9. Usage of FG_GEN_ROLL

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

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

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

14.10. FG_GEN_ACTIVE

Table 14.10. Parameter properties of FG_GEN_ACTIVE

Property	Value
Name	**FG_GEN_ACTIVE**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 65535** **Stepsize 1**

Example 14.10. Usage of FG_GEN_ACTIVE

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

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

14.11. FG_GEN_PASSIVE

Table 14.11. Parameter properties of FG_GEN_PASSIVE

Property	Value
Name	**FG_GEN_PASSIVE**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 65535** **Stepsize 1**

Example 14.11. Usage of FG_GEN_PASSIVE

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

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

14.12. FG_GEN_LINE_WIDTH

Table 14.12. Parameter properties of FG_GEN_LINE_WIDTH

Property	Value
Name	**FG_GEN_LINE_WIDTH**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 65535** **Stepsize 1**
Unit of measure	**pixel**

Example 14.12. Usage of FG_GEN_LINE_WIDTH

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

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

Chapter 15. 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.

15.1. FG_TIMEOUT

This parameter is used to set a timeout for DMA transfers. After a timeout the acquisition is stopped.

Table 15.1. Parameter properties of FG_TIMEOUT

Property	Value
Name	**FG_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 15.1. 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_TIMEOUT, &value, 0, type)) < 0) {
    /* error handling */
}

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

15.2. FG_APPLET_VERSION

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

Table 15.2. Parameter properties of FG_APPLET_VERSION

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

Example 15.2. 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_APPLET_VERSION, &value, 0, type)) < 0) {
    /* error handling */
}
```

15.3. FG_APPLET_REVISION

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

Table 15.3. Parameter properties of FG_APPLET_REVISION

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

Example 15.3. 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_APPLET_REVISION, &value, 0, type)) < 0) {
    /* error handling */
}
```

15.4. FG_APPLET_ID

This value is for internal use only.

Table 15.4. Parameter properties of FG_APPLET_ID

Property	Value
Name	**FG_APPLET_ID**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.4. Usage of FG_APPLET_ID

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

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

15.5. FG_HAP_FILE

Please report this read-only string parameter for any support case of the AcquisitionApplets Advanced.

Table 15.5. Parameter properties of FG_HAP_FILE

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

Example 15.5. Usage of FG_HAP_FILE

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

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

15.6. 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 15.6. Parameter properties of FG_DMASTATUS

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

Example 15.6. 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_DMASTATUS, &value, 0, type)) < 0) {
    /* error handling */
}
```

15.7. 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 15.7. Parameter properties of FG_CAMSTATUS

Property	Value
Name	**FG_CAMSTATUS**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.7. 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_CAMSTATUS, &value, 0, type)) < 0) {
    /* error handling */
}
```

15.8. 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
- 1 = CameraLval
- 2 = CameraFval
- 3 = Camera CC1 Signal
- 4 = ExTrg
- 5 = BufferOverflow
- 6 = BufStatus
- 7 = BufStatus

Table 15.8. Parameter properties of FG_CAMSTATUS_EXTENDED

Property	Value
Name	**FG_CAMSTATUS_EXTENDED**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.8. 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_CAMSTATUS_EXTENDED, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 16. Revision History

Table 16.1. Revision History

Version	Release Date	Change Log
1.0	12.06.2012	Initital version

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). SiliconSoftware 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.
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)	A part a frame. Mostly rectengular and within the original image boundaries. Defined by coordinates. The frame grabber cuts the region of interest from the camera

	image. A region of interest might reduce or increase the required bandwidth.
Sensor Tap	See Tap.
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 simulatiously 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 tranfer 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.
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 triger IO. The first output has index 1. Check mapping of output pins to logic outputs in the harware documentation.

Index

A
Area of Interest, 10

B
Bandwidth, 5

C
Camera
  Format, 8
  Interface, 5
Camera Link, 8, 8
Camera Simulator, 108, 108

F
Features, 2
FG_APPLET_ID, 117
FG_APPLET_REVISION, 117
FG_APPLET_VERSION, 116
FG_AREATRIGGERMODE, 39
FG_BITALIGNMENT, 106
FG_CAMERA_LINK_CAMTYPE, 8
FG_CAMSTATUS, 118
FG_CAMSTATUS_EXTENDED, 119
FG_CCSEL0, 49
FG_CCSEL1, 49
FG_CCSEL2, 49
FG_CCSEL3, 49
FG_DIGIO_INPUT, 52
FG_DIGIO_OUTPUT, 50
FG_DMASTATUS, 118
FG_EXPOSURE, 46
FG_EXSYNCDELAY, 46
FG_EXSYNCON, 45
FG_EXSYNCPOLARITY, 47
FG_FILLLEVEL, 78
FG_FLASHON, 45
FG_FLASH_POLARITY, 48
FG_FORMAT, 106
FG_GEN_ACCURACY, 112
FG_GEN_ACTIVE, 114
FG_GEN_ENABLE, 108
FG_GEN_FREQ, 111
FG_GEN_HEIGHT, 110
FG_GEN_LINE_GAP, 111
FG_GEN_LINE_WIDTH, 115
FG_GEN_PASSIVE, 114
FG_GEN_ROLL, 113
FG_GEN_START, 109
FG_GEN_TAP1, 113
FG_GEN_TAP2, 113
FG_GEN_TAP3, 113

FG_GEN_TAP4, 113
FG_GEN_WIDTH, 109
FG_HAP_FILE, 117
FG_HEIGHT, 12
FG_IMG_SELECT, 83
FG_IMG_SELECT_PERIOD, 83
FG_LUT_CUSTOM_FILE, 97
FG_LUT_IMPLEMENTATION_TYPE, 99
FG_LUT_IN_BITS, 99
FG_LUT_OUT_BITS, 99
FG_LUT_SAVE_FILE, 98
FG_LUT_TYPE, 95
FG_LUT_VALUE_BLUE, 97
FG_LUT_VALUE_GREEN, 96
FG_LUT_VALUE_RED, 96
FG_MISSING_CAM0_FRAME_RESPONSE, 76
FG_MISSING_CAMERA_FRAME_RESPONSE, 74
FG_MISSING_CAMERA_FRAME_RESPONSE_CLEAR, 75
FG_OVERFLOW, 79
FG_OVERFLOW_CAM0, 80
FG_PIXELDEPTH, 107
FG_PRESCALER, 49
FG_PROCESSING_GAIN, 102
FG_PROCESSING_GAMMA, 103
FG_PROCESSING_INVERT, 104
FG_PROCESSING_OFFSET, 102
FG_SCALINGFACTOR_BLUE, 93
FG_SCALINGFACTOR_GREEN, 94
FG_SCALINGFACTOR_RED, 93
FG_SENDSOFTWARETRIGGER, 56
FG_SHADING_APPLY_SETTINGS, 89
FG_SHADING_BLACK_FILENAME, 88
FG_SHADING_GAIN_CORRECTION_MODE, 89
FG_SHADING_GAIN_ENABLE, 87
FG_SHADING_GRAY_FILENAME, 88
FG_SHADING_OFFSET_ENABLE, 86
FG_SOFTWARETRIGGER_IS_BUSY, 57
FG_SOFTWARETRIGGER_QUEUE_FILLLEVEL, 57
FG_STROBEPULSEDELAY, 47
FG_TIMEOUT, 116
FG_TRIGGERCC_SELECT0, 70
FG_TRIGGERCC_SELECT1, 70
FG_TRIGGERCC_SELECT2, 70
FG_TRIGGERCC_SELECT3, 70
FG_TRIGGERIN_BYPASS_SRC, 55
FG_TRIGGERIN_DEBOUNCE, 50
FG_TRIGGERIN_DOWNSCALE, 53
FG_TRIGGERIN_DOWNSCALE_PHASE, 54
FG_TRIGGERIN_POLARITY, 53
FG_TRIGGERIN_SRC, 52
FG_TRIGGERIN_STATS_FREQUENCY, 59
FG_TRIGGERIN_STATS_MAXFREQUENCY, 60
FG_TRIGGERIN_STATS_MINFREQUENCY, 59
FG_TRIGGERIN_STATS_MINMAXFREQUENCY_CLEAR, 60
FG_TRIGGERIN_STATS_PULSECOUNT, 58

FG_TRIGGERIN_STATS_PULSECOUNT_CLEAR, 58
FG_TRIGGERMODE, 44
FG_TRIGGEROUT_SELECT0, 72
FG_TRIGGEROUT_SELECT1, 72
FG_TRIGGEROUT_SELECT2, 72
FG_TRIGGEROUT_SELECT3, 72
FG_TRIGGEROUT_SELECT4, 72
FG_TRIGGEROUT_SELECT5, 72
FG_TRIGGEROUT_SELECT6, 72
FG_TRIGGEROUT_SELECT7, 72
FG_TRIGGEROUT_STATS_PULSECOUNT, 73
FG_TRIGGEROUT_STATS_PULSECOUNT_CLEAR, 73
FG_TRIGGEROUT_STATS_SOURCE, 72
FG_TRIGGERQUEUE_FILLLEVEL, 63
FG_TRIGGERQUEUE_MODE, 63
FG_TRIGGERSTATE, 40
FG_TRIGGER_EXCEEDED_PERIOD_LIMITS, 42
FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CAM0, 44
FG_TRIGGER_EXCEEDED_PERIOD_LIMITS_CLEAR, 43
FG_TRIGGER_FRAMESPERSECOND, 23, 41
FG_TRIGGER_INPUT0_FALLING, 61
FG_TRIGGER_INPUT0_RISING, 61
FG_TRIGGER_INPUT1_FALLING, 61
FG_TRIGGER_INPUT1_RISING, 61
FG_TRIGGER_INPUT2_FALLING, 61
FG_TRIGGER_INPUT2_RISING, 61
FG_TRIGGER_INPUT3_FALLING, 61
FG_TRIGGER_INPUT3_RISING, 61
FG_TRIGGER_INPUT4_FALLING, 61
FG_TRIGGER_INPUT4_RISING, 61
FG_TRIGGER_INPUT5_FALLING, 61
FG_TRIGGER_INPUT5_RISING, 61
FG_TRIGGER_INPUT6_FALLING, 61
FG_TRIGGER_INPUT6_RISING, 61
FG_TRIGGER_INPUT7_FALLING, 61
FG_TRIGGER_INPUT7_RISING, 61
FG_TRIGGER_LEGACY_MODE, 43
FG_TRIGGER_MULTIPLY_PULSES, 28, 62
FG_TRIGGER_OUTPUT_CAM0, 77
FG_TRIGGER_OUTPUT_EVENT_SELECT, 76
FG_TRIGGER_PULSEFORMGEN0_DELAY, 68
FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE, 66
FG_TRIGGER_PULSEFORMGEN0_DOWNSCALE_PHASE, 67
FG_TRIGGER_PULSEFORMGEN0_WIDTH, 69
FG_TRIGGER_PULSEFORMGEN1_DELAY, 68
FG_TRIGGER_PULSEFORMGEN1_DOWNSCALE, 66
FG_TRIGGER_PULSEFORMGEN1_DOWNSCALE_PHASE, 67
FG_TRIGGER_PULSEFORMGEN1_WIDTH, 69
FG_TRIGGER_PULSEFORMGEN2_DELAY, 68
FG_TRIGGER_PULSEFORMGEN2_DOWNSCALE, 66
FG_TRIGGER_PULSEFORMGEN2_DOWNSCALE_PHASE, 67
FG_TRIGGER_PULSEFORMGEN2_WIDTH, 69
FG_TRIGGER_PULSEFORMGEN3_DELAY, 68
FG_TRIGGER_PULSEFORMGEN3_DOWNSCALE, 66
FG_TRIGGER_PULSEFORMGEN3_DOWNSCALE_PHASE, 67

FG_TRIGGER_PULSEFORMGEN3_WIDTH, 69
FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_OFF_THRESHOLD, 64
FG_TRIGGER_QUEUE_FILLLEVEL_EVENT_ON_THRESHOLD, 64
FG_TRIGGER_QUEUE_FILLLEVEL_THRESHOLD_CAM0_OFF, 65
FG_TRIGGER_QUEUE_FILLLEVEL_THRESHOLD_CAM0_ON, 65
FG_USEDVAL, 8
FG_WIDTH, 11
FG_XOFFSET, 12
FG_YOFFSET, 13
Flat Field Correction (see Shading Correction)
Format, 106

G

Generator, 108

I

Image Select, 82
Image Selector, 82
Image Transfer, 6

L

Lookup Table, 95, 95
Lookup Table::Applet Properties, 99

M

Miscellaneous, 116

O

Output Format, 106
Overflow, 78, 78

P

PC Interface, 6
Processing, 101
Processor, 101

R

Region of Interest, 10
ROI, 10

S

Serial Interface, 19
Shading Correction, 85, 85
  Automatic, 87
  Custom Values, 91
Shading Correction::Auto Shading Correction, 87
Shading Custom Values, 91
Software Interface, 7
Specifications, 2

T

Trigger, 14, 14
  Activate, 40
  Busy, 57

Bypass, 37, 55
CC Signal Mapping, 70
Debounce, 50
Digital Input, 17, 52
Digital Output, 17, 21, 71
Downscale Input, 53
Encoder, 21
Error Detection, 38
Events, 18
Exceeded Period Limits, 42
Exsync, 21
External, 21, 39, 50
Flash, 21, 25
Frame Rate, 19
Framerate, 41
Generator, 19, 39
Grabber Controlled, 19
Image Trigger, 21
Input, 50, 52
Input Statistics, 58
Length, 20
Lost Trigger, 38, 42
Missing Frame Response, 74
Mode, 39
Multi Camera, 37
Multiply Pulses, 61
Output, 71
Output Event, 76
Output Statistics, 72
Period, 41
Pin Allocation, 17
Polarity Input, 53
Pulse Form Generator, 65
Pulse Multiplication, 25
Queue, 34, 36, 62
Sequencer, 25, 61
Signal Length, 20
Signal Width, 20
Software Controlled, 56
Software Trigger, 29, 39, 56
Start, 40
Stop, 40
Synchronized, 37
System Analysis, 38
Trigger IO, 17
Width, 20
Trigger::CC Signal Mapping, 70
Trigger::Digital Output, 71
Trigger::Digital Output::Statistics, 72
Trigger::Legacy, 44
Trigger::Output Event, 76
Trigger::Pulse Form Generator 0, 65
Trigger::Pulse Form Generator 1, 69
Trigger::Pulse Form Generator 2, 70
Trigger::Pulse Form Generator 3, 70

Trigger::Queue, 62
Trigger::Sequencer, 61
Trigger::Trigger Input, 50
Trigger::Trigger Input::External, 50
Trigger::Trigger Input::Software Trigger, 56
Trigger::Trigger Input::Statistics, 58

W

White Balance, 93, 93