microEnable 5 marathon ACX DP

Test Applet User Documentation for

FrameGrabberTest

Functional Description
For Framegrabber SDK Usage

Document Number: AW001766
Part Number: 000 (English)
Document Version: 01
Release Date: 01 December 2022
Applet Version 2.2.4.0

Contacting Basler Support Worldwide

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

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

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

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

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

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

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

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

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

Supplemental Information

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

Table of Contents

Chapter 1. Introduction

This document provides detailed information on the Silicon Software Test Applet "FrameGrabberTest" for microEnable 5 marathon ACX DP frame grabbers.

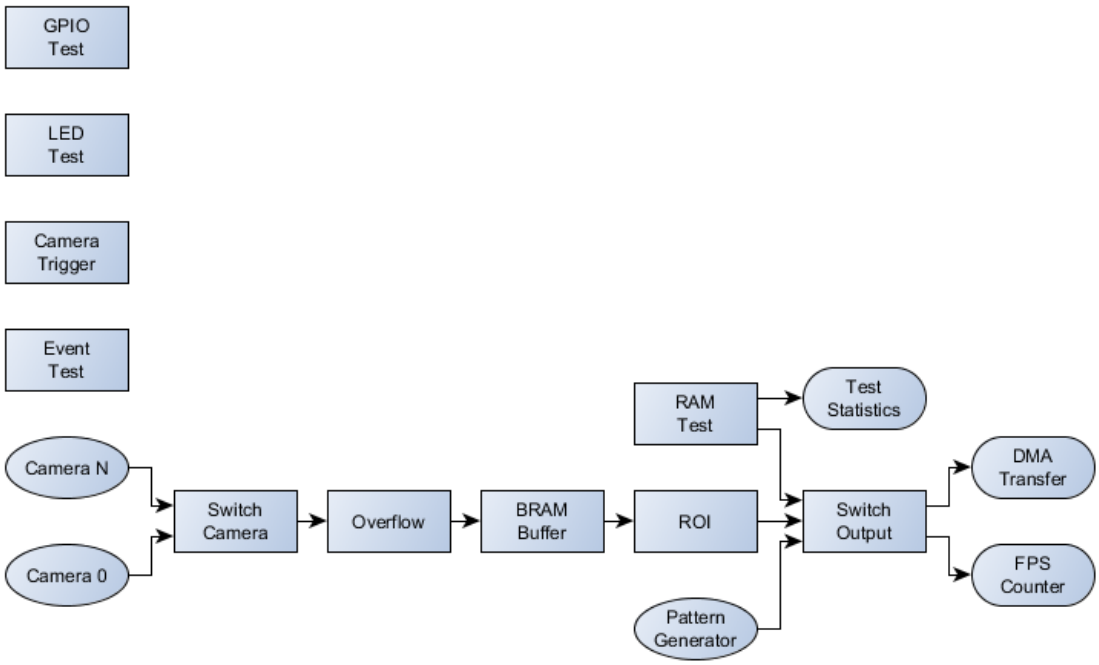
This applet is a frame grabber test applet. Its intention is to test the hardware. You shall not use this applet for your final image processing application. Use AcquisitionApplets or VA Applets instead!

The applet comprises the following functions:

- DMA Performance test: Different image dimensions for varying memory sizes and interrupt rates
- RAM Test: Check for errors and processing
- Camera: Check camera port image acquisition
- Camera Trigger: Send trigger signals to camera
- GPIO: Monitor the GPIs and set the GPOs
- Event test: Generate a sofware callback event
- Monitoring: FPGA Temperature, Power, PoCL, ... (See Chapter 13, '*Miscellaneous*')

The following diagram shows the functional blocks of the applet.

Figure 1.1. Block Diagram of the applet



Chapter 2. Test Procedure in microDisplay

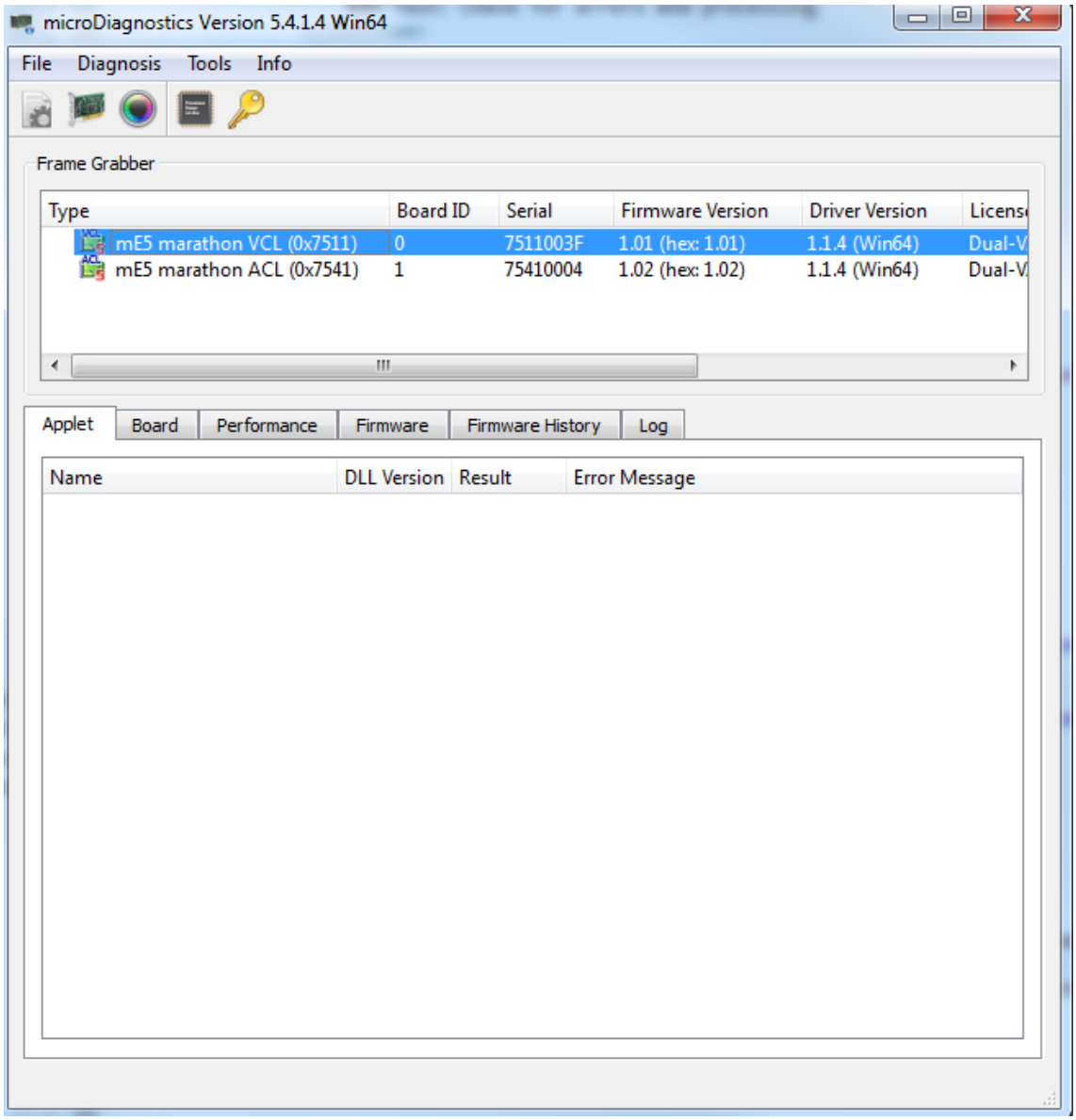
In the following, the steps to test the hardware with the applet FrameGrabberTest in microDisplay are explained. Of course, you can also integrate the tests in your own programs with the Silicon Software API and SDK.

Information: If you have connected your camera on frame grabber port B, C or D you have to press the button "Acquisition Start" in the GenICam explorer to start image acquisition with camera. In microDisplay you will see the information: "No camera detected", which you can ignore. For the connection of your camera to frame grabber port A no additional steps are necessary.

2.1. Load the Applet

First flash the applet "FrameGrabberTest.dll" to the frame grabber. Open the program 'microDiagnostics' and choose your frame grabber as displayed in Fig. 2.1. Click on 'Tools' and 'Flash Board(s)'. Select 'FrameGrabbertest.dll'. Having flashed the board follow the instructions in 'microDiagnostics' and close the program!

Figure 2.1. Flash the applet "FrameGrabberTest.dll in 'microDiagnostics'



To load the applet "FrameGrabberTest.dll" open the program 'microDisplay' and click on the button 'LoadApplet' (see Fig. 2.2). Choose "FrameGrabberTest.dll", click on the button in the middle and then on 'close' (see Fig. 2.3).

Figure 2.2. Load the applet in 'microDisplay'

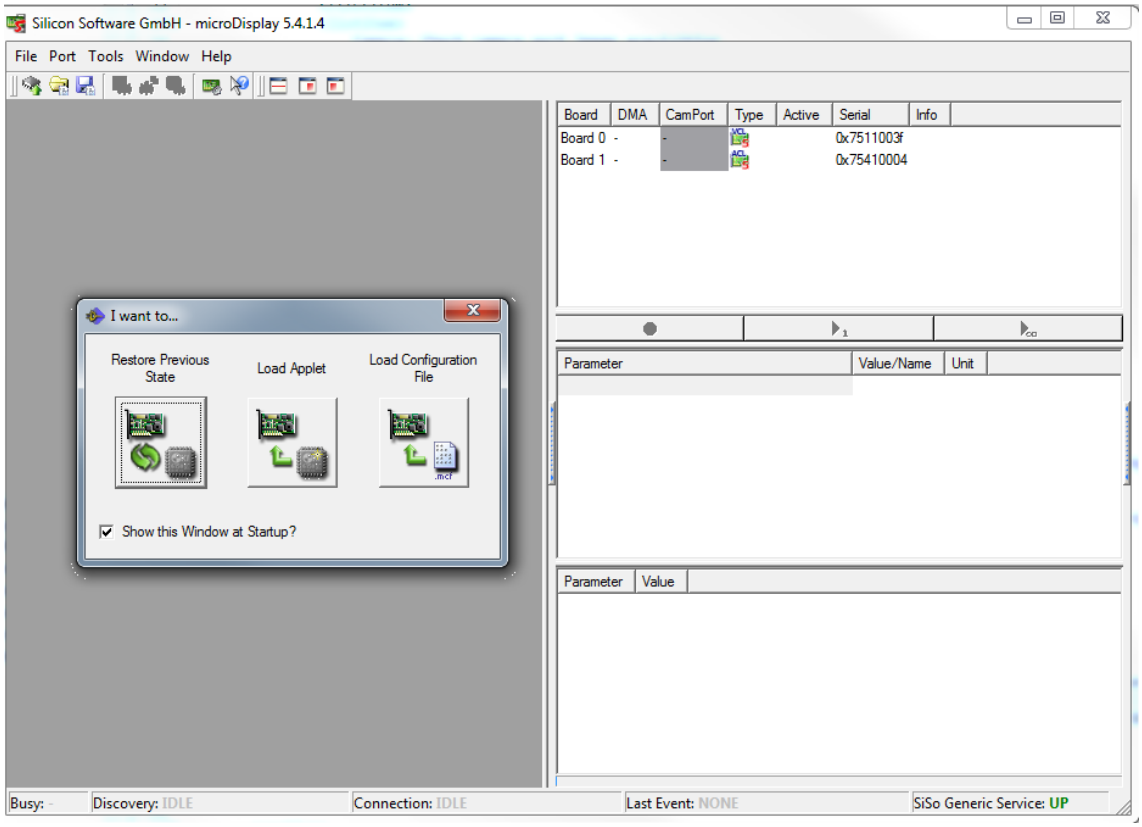
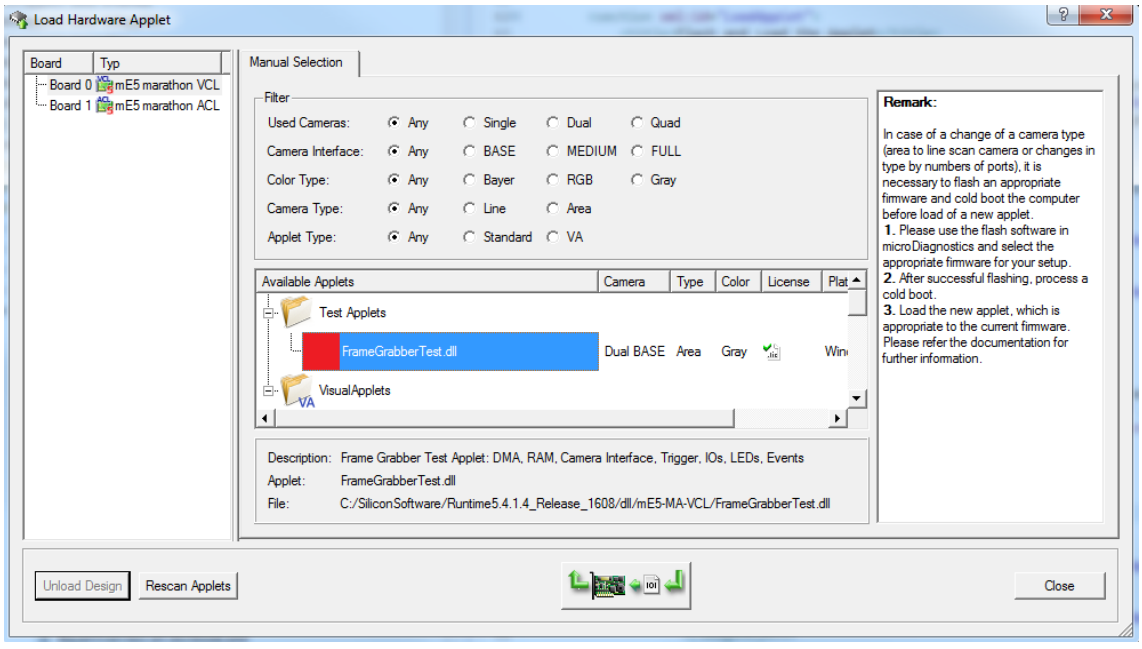


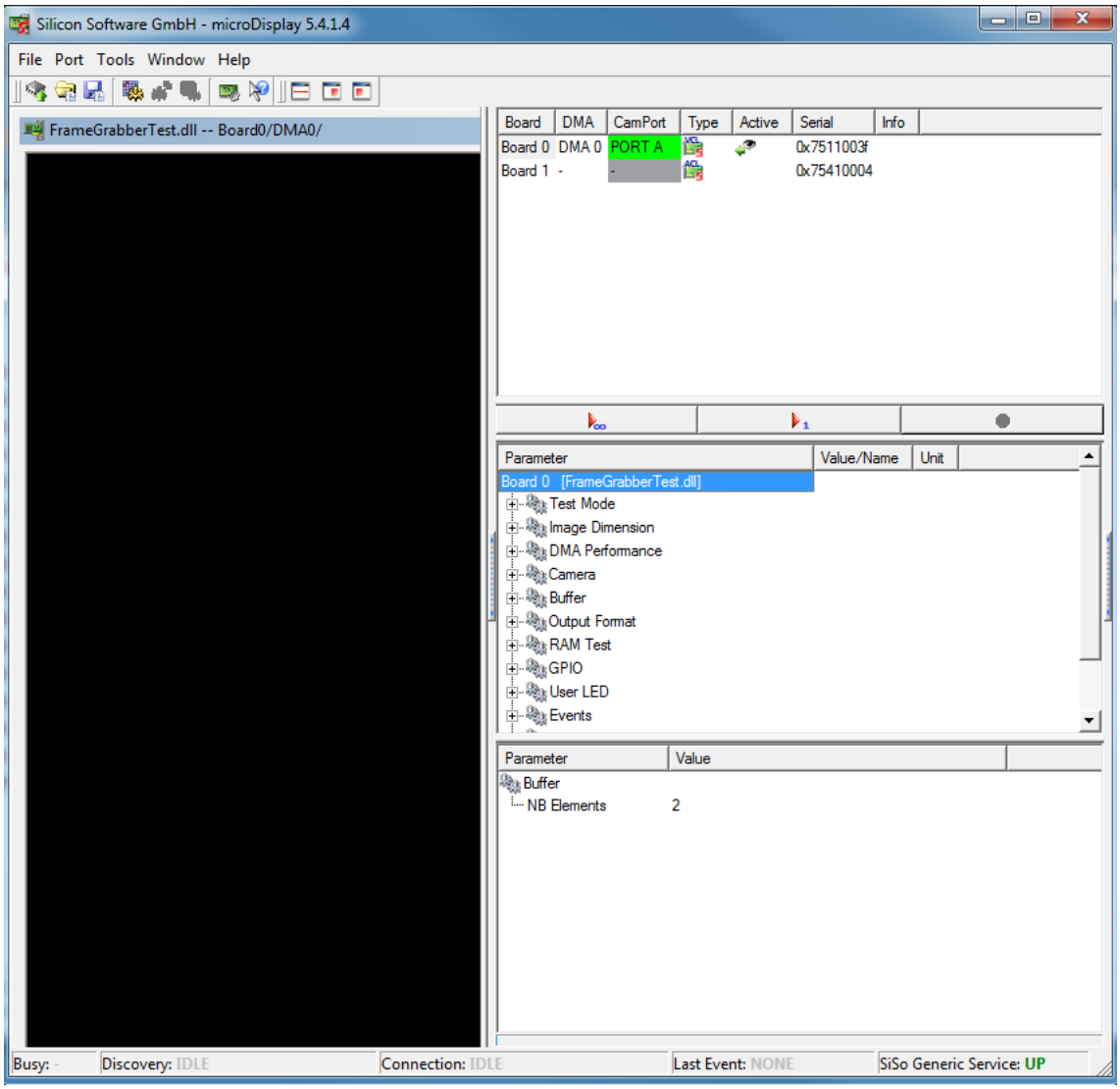
Figure 2.3. Load the applet "FrameGrabberTest.dll in 'microDisplay'



2.2. Choose Your Test Procedure

In the following we describe how you can choose the single test procedures, which are listed in the introduction text. In Fig. 2.4 you see a list of parameters ('TestMode' to 'Events'). To set the test procedures we use these parameters. In chapter 3 to 13 their functionality and settings are explained in detail.

Figure 2.4. Parameters of the applet 'FrameGrabbertest.dll' in 'microDisplay'



2.2.1. DMA Performance Test

To test the DMA performance set the parameter 'OutputSelect' under 'Test Mode' with Right-Mouse-Click to 'DMA Performance'. You can choose the image dimensions for the test with the parameters 'Width' and 'Height' (under 'ImageDimensions' (see Chapter 4)). With Right-Mouse-Click on 'DMA Performance Output Mode' under 'DMA Performance' (see Chapter 5) you can choose between maximum DMA performance ('DMA Performance Maximum') and user defined DMA framerate ('DMA Performance Custom Framerate'). For the latter set the frame rate with the parameter 'DMA Performance Framerate'. In addition you have the possibility to stop completely the DMA output in setting 'DMA Performance Output Mode' to 'DMA Performance Off'. You can monitor the current DMA framerate with the read only parameter 'FPS' under 'Output Format'.

2.2.2. RAM Test

To test the RAM performance of the RAM modules (0 to 1 or 3: depends on platform) set the parameter 'OutputSelect' under 'Test Mode' with Right-Mouse-Click to 'RAM Difference' or 'RAM Errors' for the corresponding RAM module. In 'RAM Difference' mode (difference between expected and read value from RAM) you can see RAM defects in output values, which are not zero. In 'RAM Errors' mode a white pixel indicates an error (see also chapter 3). Output image size is always 512 MiB. Suggested display width in 'RAM Difference' mode is to 4096 pixels (parameter 'Width' under 'ImageDimensions' (see Chapter 4)). You can choose display height with parameter'Height' under 'ImageDimensions' (see Chapter 4)). If display size

exceeds output image size the output images are split to several displayed images. With the parameters 'Enable RAM0' to 'Enable RAM3' you have the possibility to stop the data processing for the corresponding RAM module (see also section 9.4). You can detect RAM errors, when RAM data processing is enabled, but the read-only parameter 'Image Count' of the corresponding RAm module does not increase. Defects of RAM modules can also be observed with the read-only parameters 'Error Occurred', 'Error COUNT_RAM0' to 'Error COUNT_RAM3'.

2.2.3. Camera Test/Camera Trigger Test

To test the camera port image acquisition set the parameter 'OutputSelect' under 'Test Mode' with Right-Mouse-Click to 'Camera'. You can choose the image ROI dimensions for the test with the parameters 'Width' and 'Height' (under 'ImageDimensions' (see Chapter 4)). Select your camera port with the parameter 'Camera Port' (under 'Camera') and choose your 'Camera Input Format'(see also Chapter 6). The read-only parameters 'Buffer fill level' and 'Buffer overflow' indicate the fill level and overflow of the BRAM between camera and DMA output (see also Chapter 7). It helps to identify problems during image acquisition. You have the possibility to send trigger signals to the the camera on port 0 and port 1 setting the parameters 'FG_CCSEL0' to 'FG_CCSEL3' (under parameter 'Camera').

2.2.4. GPIO

You can monitor the digital inputs with the parameters 'GPI Status bitmask' and 'Front GPI Status bitmask' (under parameter 'GPIO'). Bit 0 to bit N represent digital inputs 0 to N. Find more information on these parameters in sections 10.1 and 10.2. You can set the digital outputs of the frame grabber with the parameters 'Output bitmask' and 'Front Output bitmask'. Values between 0 to 255 and 0 to 37 are possible. Here also bit 0 to bit N represent digital outputs 0 to N. You find further information on these parameters in sections 10.3 and 10.4.

2.2.5. Event Test

With the parameter 'Generate a Test Event' you can start a software callback event for test purposes. More information you find in Chapter 12.

2.2.6. Monitoring

You have the possibility to monitor several Applet and frame grabber parameters under 'Miscellaneous'. There you find e.g. information on the 'Applet version', 'Applet revision', 'Build time' and several more. Also current FPGA temperature, voltage and link speed information are located there.

Chapter 3. Test Mode

3.1. FG_OUTPUT_SELECT

The frame grabber test applet offerst several DMA output modes

- DMA Performance Output
- Camera Image Output
- RAM Test Output

The DMA performance output uses a pattern generator which is directly connected to the DMA and can support the full bandwidth. Use parameters *FG_WIDTH* and parameter *FG_HEIGHT* set the generator and DMA output size. In this mode data will always be output at the maximum possible datarate which is capable by the PCIe interface and PC.

If you select camera output, the camera images are forwarded to the output. Again use parameters *FG_WIDTH* and *FG_HEIGHT* to set the output size.

If you select the RAM test you need to note the following

- RAM Difference output:

  Will output the absolute difference between the expected and read value from RAM. This should always be 0. Otherwise there is a RAM defect.

- RAM Error output:

  Will output a white pixel for any error.

  In this mode, the RAM data width is used so that the output is not 8 bit pixel. Instead for each RAM data one pixel is output. For example if your RAM has a data width of 128 bit, 16 8 bit pixel are merged together.

- The output image size will always be the size of the RAM. For example 512MiB or 256MiB.

  Parameter *FG_WIDTH* will set a display width. The width is constant depending on difference or error output. In difference output the width should always be 4096.

  Parameter *FG_HEIGHT* will set a display height. If the actual image height exceeds the height of the RAM, the image is split into many several images.

Table 3.1. Parameter properties of FG_OUTPUT_SELECT

Property	Value
Name	**FG_OUTPUT_SELECT**
Display Name	**Output Select**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_DMA_PERFORMANCE** DMA Performance **FG_CAMERA** Camera **FG_RAM0_DIFFERENCE** RAM 0 Difference **FG_RAM0_ERRORS** RAM 0 Errors **FG_RAM1_DIFFERENCE** RAM 1 Difference **FG_RAM1_ERRORS** RAM 1 Errors **FG_RAM2_DIFFERENCE** RAM 2 Difference **FG_RAM2_ERRORS** RAM 2 Errors **FG_RAM3_DIFFERENCE** RAM 3 Difference **FG_RAM3_ERRORS** RAM 3 Errors
Default value	**FG_DMA_PERFORMANCE**

Example 3.1. Usage of FG_OUTPUT_SELECT

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

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

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

Chapter 4. Image Dimension

4.1. FG_WIDTH

Set the output width using this parameter. The width setting defines the size for DMA test and camera ROI.

The DMA output is defined using parameter .

Note that for RAM test output the width and height settings simply define the display size.

Table 4.1. Parameter properties of FG_WIDTH

Property	Value
Name	**FG_WIDTH**
Display Name	**Width**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Transient**
Allowed values	**Minimum 16** **Maximum 16384** **Stepsize 16**
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, FG_WIDTH, &value, 0, type)) < 0) {
    /* error handling */
}

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

4.2. FG_HEIGHT

Set the output height using this parameter. The height setting defines the size for DMA test and camera ROI.

The DMA output is defined using parameter .

Note that for RAM test output the width and height settings simply define the display size.

Table 4.2. Parameter properties of FG_HEIGHT

Property	Value
Name	**FG_HEIGHT**
Display Name	**Height**
Type	**Unsigned Integer**
Access policy	**Read/Write**
Storage policy	**Persistent**
Allowed values	**Minimum 1** **Maximum 65536** **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, FG_HEIGHT, &value, 0, type)) < 0) {
    /* error handling */
}

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

Chapter 5. DMA Performance

5.1. FG_DMA_PERFORMANCE_OUTPUT_MODE

The DMA Performance test can be used in several modes.

- Off: No data will be output
- Maximum: The image generator will run at maximum speed and data is output as fast as the DMA transfer allows. To obtain the maximum possible bandwidth of the DMA use this mode.
- Custom Framerate: Allows you to specify any framerate in the allowed range. Use parameter *FG_DMA_PERFORMANCE_FRAMERATE* to define the framerate.

Table 5.1. Parameter properties of FG_DMA_PERFORMANCE_OUTPUT_MODE

Property	Value
Name	**FG_DMA_PERFORMANCE_OUTPUT_MODE**
Display Name	**DMA Performance Output Mode**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_DMA_PERFORMANCE_OFF** DMA Performance Off **FG_DMA_PERFORMANCE_MAXIMUM** DMA Performance Maximum **FG_DMA_PERFORMANCE_CUSTOM_FRAMERATE** DMA Performance Custom Framerate
Default value	**FG_DMA_PERFORMANCE_MAXIMUM**

Example 5.1. Usage of FG_DMA_PERFORMANCE_OUTPUT_MODE

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

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

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

5.2. FG_DMA_PERFORMANCE_FRAMERATE

For the DMA test you can specify a custom framerate. Set parameter *FG_DMA_PERFORMANCE_OUTPUT_MODE* to **FG_DMA_PERFORMANCE_CUSTOM_FRAMERATE** so that this parameter is enabled.

You can use any framerate. However, if the defined framerate exceeds the maximum possible by the DMA, the framerate is decreased.

Table 5.2. Parameter properties of FG_DMA_PERFORMANCE_FRAMERATE

Property	Value
Name	**FG_DMA_PERFORMANCE_FRAMERATE**
Display Name	**DMA Performance Framerate**
Type	**Double**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0.931323** **Maximum 1.25E8** **Stepsize 8.0E-9**
Default value	**100.0**
Unit of measure	**fps**

Example 5.2. Usage of FG_DMA_PERFORMANCE_FRAMERATE

```
int result = 0;
double value = 100.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

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

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

Chapter 6. Camera

6.1. FG_CAMERA_PORT

Select the camera port index.

Table 6.1. Parameter properties of FG_CAMERA_PORT

Property	Value
Name	**FG_CAMERA_PORT**
Display Name	**Camera Port**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**Minimum** 0 **Maximum** 1 **Stepsize** 1
Default value	**0**

Example 6.1. Usage of FG_CAMERA_PORT

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

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

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

6.2. FG_TRIGGERCAMERA_OUT_SELECT

Table 6.2. Parameter properties of FG_TRIGGERCAMERA_OUT_SELECT

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

Example 6.2. Usage of FG_TRIGGERCAMERA_OUT_SELECT

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

if ((result = Fg_setParameterWithType(fg, FG_TRIGGERCAMERA_OUT_SELECT, &value, 0, type)) < 0) {
```

```
    /* error handling */
}

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

Chapter 7. Buffer

7.1. FG_FILLLEVEL

Indicates the buffer filllevel of the BRAM based buffer between the camera interface and DMA. Use this value if you output camera images to the DMA.

Table 7.1. Parameter properties of FG_FILLLEVEL

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

Example 7.1. Usage of FG_FILLLEVEL

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

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

7.2. FG_OVERFLOW

Indicates a buffer overflow. The parameter is automatically reset when read. Note that microDisplay continuously reads all parameters so that you might not see the occurrence of an overflow. Have a look at the event counter in this case.

The overflow shows buffer overflows of the BRAM based buffer between the camera interface and DMA.

You can also use the overflow events instead of the parameter.

Table 7.2. Parameter properties of FG_OVERFLOW

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

Example 7.2. Usage of FG_OVERFLOW

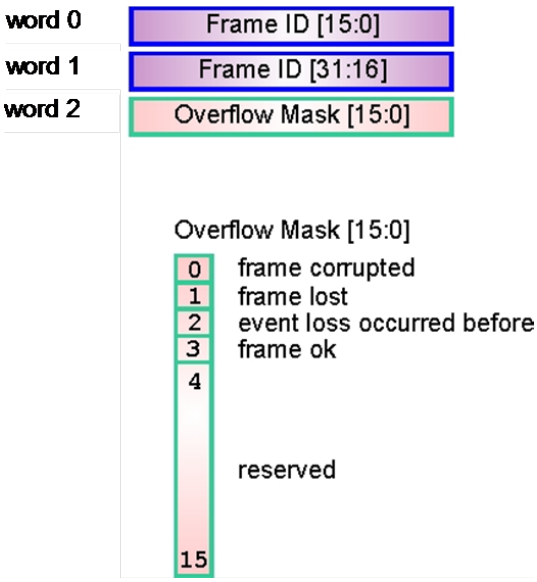
```
int result = 0;
unsigned int value = 0;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

7.3. FG_EVENT_OVERFLOW

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.

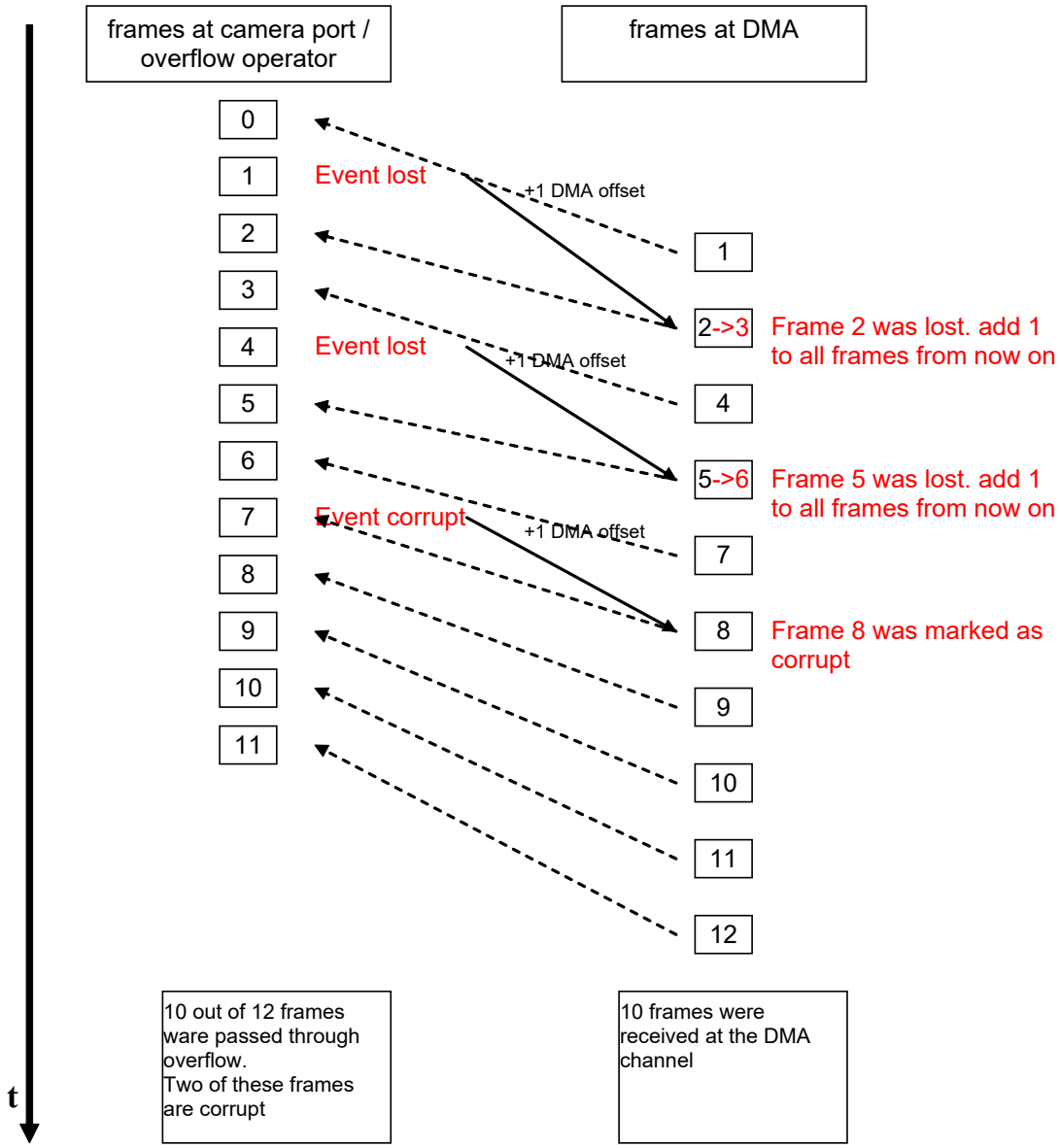
Figure 7.1. Illustration of Overflow Data Packet



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 7.2. 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 8. Output Format

8.1. FG_FORMAT

Table 8.1. Parameter properties of FG_FORMAT

Property	Value
Name	**FG_FORMAT**
Display Name	**Output Format**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**FG_GRAY** Gray 8bit
Default value	**FG_GRAY**

Example 8.1. Usage of FG_FORMAT

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

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

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

8.2. FG_FPS

This read only parameter shows the current DMA framerate. It measures the number of frames which are output in one second. Only integer values i.e. completed frames are considered.

Table 8.2. Parameter properties of FG_FPS

Property	Value
Name	**FG_FPS**
Display Name	**FPS**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum** 0 **Maximum** 125000000 **Stepsize** 1

Example 8.2. Usage of FG_FPS

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

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

Chapter 9. RAM Test

9.1. FG_NUMBER_OF_RAMS

Number of logic RAM modules the applet is using. The frame grabber might allow more but the applet might not use all of them.

Table 9.1. Parameter properties of FG_NUMBER_OF_RAMS

Property	Value
Name	**FG_NUMBER_OF_RAMS**
Display Name	**Number of RAMs**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 1** **Maximum 4** **Stepsize 1**
Unit of measure	

Example 9.1. Usage of FG_NUMBER_OF_RAMS

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

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

9.2. FG_RAM_SIZE

Size of one RAM module. Unit is Mebibyte i.e. Byte times 2^20.

Table 9.2. Parameter properties of FG_RAM_SIZE

Property	Value
Name	**FG_RAM_SIZE**
Display Name	**RAM Size**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 1** **Maximum 8192** **Stepsize 1**
Unit of measure	**MiB**

Example 9.2. Usage of FG_RAM_SIZE

```
int result = 0;
```

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

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

9.3. FG_ERROR_OCCURRED

Is set if an error in any of the RAM modules is detected. This value should always be at FG_NO.

Table 9.3. Parameter properties of FG_ERROR_OCCURRED

Property	Value
Name	**FG_ERROR_OCCURRED**
Display Name	**Erorr Occured**
Type	**Enumeration**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**FG_YES** Yes **FG_NO** No

Example 9.3. Usage of FG_ERROR_OCCURRED

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

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

9.4. FG_RAM_BANDWIDTH

Shows the throughput of the DRAM in MB/s. (10^6 byte). Ensure to not block the DRAM speed by the DMA. You can ensure this by setting the test output (parameter FG_OUTPUT_SELECT) mode to DMA performance or camera output.

Table 9.4. Parameter properties of FG_RAM_BANDWIDTH

Property	Value
Name	**FG_RAM_BANDWIDTH**
Display Name	**RAM Bandwidth MBs**
Type	**Double**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum** 0.0 **Maximum** 40000.0 **Stepsize** 1.0

Example 9.4. Usage of FG_RAM_BANDWIDTH

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

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

9.5. FG_ENABLE_RAM0 et al.

Note

This description applies also to the following parameters: FG_ENABLE_RAM1, FG_ENABLE_RAM2, FG_ENABLE_RAM3

You can stop the processing of data for each RAM module.

For frame grabbers with non shared memory this has no effect. However, for frame grabbers with shared memory, RAM modules can get more bandwidth if others are disabled.

Check the RAM image counter parameters *FG_IMAGE_COUNT_RAM0* to see if a RAM module processes data or not. If processing is enabled, but the counter value does not change, the RAM module might have a defect.

Table 9.5. Parameter properties of FG_ENABLE_RAM0

Property	Value
Name	**FG_ENABLE_RAM0**
Display Name	**Enable RAM 0**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_YES** Yes **FG_NO** No
Default value	**FG_YES**

Example 9.5. Usage of FG_ENABLE_RAM0

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

if ((result = Fg_setParameterWithType(fg, FG_ENABLE_RAM0, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_ENABLE_RAM0, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.6. FG_ERROR_COUNT_RAM0 et al.

Note

This description applies also to the following parameters: FG_ERROR_COUNT_RAM1, FG_ERROR_COUNT_RAM2, FG_ERROR_COUNT_RAM3

This parameter shows the number of errors detected for the respecive RAM module. One error indicates that in a RAM data cell at least one bit is not equal to the expected value. The RAM data cell size corresponds to the RAM data width and can be for example 128Bit or 256Bit.

Table 9.6. Parameter properties of FG_ERROR_COUNT_RAM0

Property	Value
Name	**FG_ERROR_COUNT_RAM0**
Display Name	**Error Count RAM 0**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 4294967295** **Stepsize 1**
Unit of measure	**pixel errors**

Example 9.6. Usage of FG_ERROR_COUNT_RAM0

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

if ((result = Fg_getParameterWithType(fg, FG_ERROR_COUNT_RAM0, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.7. FG_IMAGE_COUNT_RAM0 et al.

Note

This description applies also to the following parameters: FG_IMAGE_COUNT_RAM1, FG_IMAGE_COUNT_RAM2, FG_IMAGE_COUNT_RAM3

This value is incremented when the RAM module has been fully written and read. If this value does not increase it might show a defect in a RAM module.

Table 9.7. Parameter properties of FG_IMAGE_COUNT_RAM0

Property	Value
Name	**FG_IMAGE_COUNT_RAM0**
Display Name	**Image Count RAM 0**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 4294967295** **Stepsize 1**

Example 9.7. Usage of FG_IMAGE_COUNT_RAM0

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

if ((result = Fg_getParameterWithType(fg, FG_IMAGE_COUNT_RAM0, &value, 0, type)) < 0) {
    /* error handling */
}
```

9.8. FG_INJECT_ERRORS_RAM0 et al.

Note

This description applies also to the following parameters: FG_INJECT_ERRORS_RAM1, FG_INJECT_ERRORS_RAM2, FG_INJECT_ERRORS_RAM3

For self-test you can inject errors to the current processing.

Table 9.8. Parameter properties of FG_INJECT_ERRORS_RAM0

Property	Value
Name	**FG_INJECT_ERRORS_RAM0**
Display Name	**Inject Errors on RAM0**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_YES** Yes **FG_NO** No
Default value	**FG_NO**

Example 9.8. Usage of FG_INJECT_ERRORS_RAM0

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

if ((result = Fg_setParameterWithType(fg, FG_INJECT_ERRORS_RAM0, &value, 0, type)) < 0) {
    /* error handling */
}

if ((result = Fg_getParameterWithType(fg, FG_INJECT_ERRORS_RAM0, &value, 0, type)) < 0) {
    /* error handling */
}
```

Chapter 10. GPIO

10.1. FG_GPI

Parameter *FG_GPI* is used to monitor the digital inputs of the frame grabber.

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

Table 10.1. Parameter properties of FG_GPI

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

Example 10.1. Usage of FG_GPI

```
int result = 0;
unsigned int value = 255;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

10.2. FG_FRONT_GPI

Parameter *FG_FRONT_GPI* is used to monitor the digital inputs of the frame grabber.

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

Table 10.2. Parameter properties of FG_FRONT_GPI

Property	Value
Name	**FG_FRONT_GPI**
Display Name	**Front GPI Status bitmask**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 3** **Stepsize 1**

Example 10.2. Usage of FG_FRONT_GPI

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

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

10.3. FG_GPO

You can use this parameter to set the state of the digital outputs.

Bit 0 of the read value represents output 0, bit 1 represents output 1 and so on. For example, if you set the value to 37 or hexadecimal 0x25 the frame grabber will have high level on it's digital outputs 0, 2 and 5.

Table 10.3. Parameter properties of FG_GPO

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

Example 10.3. Usage of FG_GPO

```
int result = 0;
unsigned int value = 255;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

10.4. FG_FRONT_GPO

You can use this parameter to set the state of the front digital outputs.

Bit 0 of the read value represents output 0, bit 1 represents output 1 and so on. For example, if you set the value to 37 or hexadecimal 0x25 the frame grabber will have high level on it's digital outputs 0, 2 and 5.

Table 10.4. Parameter properties of FG_FRONT_GPO

Property	Value
Name	**FG_FRONT_GPO**
Display Name	**Front Output bitmask**
Type	**Unsigned Integer**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**Minimum 0** **Maximum 15** **Stepsize 1**
Default value	**15**

Example 10.4. Usage of FG_FRONT_GPO

```
int result = 0;
unsigned int value = 15;
const enum FgParamTypes type = FG_PARAM_TYPE_UINT32_T;

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

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

Chapter 11. User LED

11.1. FG_LED_MODE

The applet has several user LEDs. You can either define the state of this LEDs manual using parameter *FG_LED_PATTERN* or use an automatic pattern. Use this parameter to set the desired mode.

Table 11.1. Parameter properties of FG_LED_MODE

Property	Value
Name	**FG_LED_MODE**
Display Name	**LED Mode**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Persistent**
Allowed values	**FG_MANUAL** Manual **FG_COUNTER** Counter
Default value	**FG_MANUAL**

Example 11.1. Usage of FG_LED_MODE

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

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

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

11.2. FG_LED_PATTERN

The applet has several user LEDs. Set the state of the user LEDs using this parameter. Use a bitmask. For example, if you set the parameter to value 5, LEDs 0 and 2 will be switched on. Note that the number of user LEDs depends on the frame grabber used.

Table 11.2. Parameter properties of FG_LED_PATTERN

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

Example 11.2. Usage of FG_LED_PATTERN

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

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

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

Chapter 12. Events

12.1. FG_GENERATE_TEST_EVENT

With this parameter you can start a software callback event for test purposes. The event name is *FG_EVENT_TEST*. Check the runtime documentation on how to use events.

Table 12.1. Parameter properties of FG_GENERATE_TEST_EVENT

Property	Value
Name	**FG_GENERATE_TEST_EVENT**
Display Name	**Generate a Test Event**
Type	**Enumeration**
Access policy	**Read/Write/Change**
Storage policy	**Transient**
Allowed values	**FG_APPLY** Apply
Default value	**FG_APPLY**

Example 12.1. Usage of FG_GENERATE_TEST_EVENT

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

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

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

12.2. FG_EVENT_TEST

Chapter 13. 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.

13.1. FG_SYSTEMMONITOR_CHANNEL_CURRENT

Returns the channel current in Ampere.

Table 13.1. Parameter properties of FG_SYSTEMMONITOR_CHANNEL_CURRENT

Property	Value
Name	**FG_SYSTEMMONITOR_CHANNEL_CURRENT**
Display Name	**Channel Current**
Type	**Double Field**
Field Size	**0**
Access policy	**Read-Only**
Storage policy	**Transient**
Unit of measure	**A**

Example 13.1. Usage of FG_SYSTEMMONITOR_CHANNEL_CURRENT

```
int result = 0;

FieldParameterDouble access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMDOUBLE;

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

13.2. FG_SYSTEMMONITOR_CHANNEL_VOLTAGE

Returns the channel voltage.

Table 13.2. Parameter properties of FG_SYSTEMMONITOR_CHANNEL_VOLTAGE

Property	Value
Name	**FG_SYSTEMMONITOR_CHANNEL_VOLTAGE**
Display Name	**Channel Voltage**
Type	**Double Field**
Field Size	**0**
Access policy	**Read-Only**
Storage policy	**Transient**
Unit of measure	**V**

Example 13.2. Usage of FG_SYSTEMMONITOR_CHANNEL_VOLTAGE

```
int result = 0;

FieldParameterDouble access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMDOUBLE;

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

13.3. FG_TIMEOUT

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

Table 13.3. Parameter properties of FG_TIMEOUT

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

Example 13.3. Usage of FG_TIMEOUT

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

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

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

13.4. FG_APPLET_VERSION

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

Table 13.4. Parameter properties of FG_APPLET_VERSION

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

Example 13.4. Usage of FG_APPLET_VERSION

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

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

13.5. FG_APPLET_REVISION

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

Table 13.5. Parameter properties of FG_APPLET_REVISION

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

Example 13.5. Usage of FG_APPLET_REVISION

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

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

13.6. FG_APPLET_ID

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

Table 13.6. Parameter properties of FG_APPLET_ID

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

Example 13.6. Usage of FG_APPLET_ID

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

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

13.7. FG_APPLET_BUILD_TIME

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

Table 13.7. Parameter properties of FG_APPLET_BUILD_TIME

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

Example 13.7. Usage of FG_APPLET_BUILD_TIME

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

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

13.8. FG_HAP_FILE

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

Table 13.8. Parameter properties of FG_HAP_FILE

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

Example 13.8. Usage of FG_HAP_FILE

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

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

13.9. 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 13.9. Parameter properties of FG_DMASTATUS

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

Example 13.9. Usage of FG_DMASTATUS

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

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

13.10. FG_CAMSTATUS

For CoaXPress, this parameter is not used. Please use the SDK's GenICam functions to identify camera detection state. More details on this can be found in the Basler Framegrabber SDK documentation.

Table 13.10. Parameter properties of FG_CAMSTATUS

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

Example 13.10. Usage of FG_CAMSTATUS

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

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

13.11. 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, currently NOT supported by CoaXPress interface.
- 1 = CameraLval, currently NOT supported by CoaXPress interface.
- 2 = CameraFval, currently NOT supported by CoaXPress interface.
- 3 = Camera CC1 Signal, currently NOT supported by CoaXPress interface.
- 4 = ExTrg / external trigger, currently NOT supported by CoaXPress interface.
- 5 = BufferOverflow
- 6 = BufStatus, LSB
- 7 = BufStatus, MSB

Table 13.11. Parameter properties of FG_CAMSTATUS_EXTENDED

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

Example 13.11. Usage of FG_CAMSTATUS_EXTENDED

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

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

13.12. FG_SYSTEMMONITOR_FPGA_TEMPERATURE

Returns the current FGPA die temperature.

Table 13.12. Parameter properties of FG_SYSTEMMONITOR_FPGA_TEMPERATURE

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

Example 13.12. Usage of FG_SYSTEMMONITOR_FPGA_TEMPERATURE

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

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

13.13. FG_SYSTEMMONITOR_FPGA_VCC_INT

Returns the current FPGA internal voltage.

Table 13.13. Parameter properties of FG_SYSTEMMONITOR_FPGA_VCC_INT

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

Example 13.13. Usage of FG_SYSTEMMONITOR_FPGA_VCC_INT

```
int result = 0;
```

```
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

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

13.14. FG_SYSTEMMONITOR_FPGA_VCC_AUX

Returns the current FPGA Vcc auxiliary voltage.

Table 13.14. Parameter properties of FG_SYSTEMMONITOR_FPGA_VCC_AUX

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

Example 13.14. Usage of FG_SYSTEMMONITOR_FPGA_VCC_AUX

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

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

13.15. FG_SYSTEMMONITOR_FPGA_VCC_BRAM

Returns the current FPGA Vcc of the BlockRAM voltage.

Table 13.15. Parameter properties of FG_SYSTEMMONITOR_FPGA_VCC_BRAM

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

Example 13.15. Usage of FG_SYSTEMMONITOR_FPGA_VCC_BRAM

```
int result = 0;
```

```
double value = 0.0;
const enum FgParamTypes type = FG_PARAM_TYPE_DOUBLE;

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

13.16. FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH

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

Table 13.16. Parameter properties of FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH

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

Example 13.16. Usage of FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH

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

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

13.17. FG_SYSTEMMONITOR_CURRENT_LINK_SPEED

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

Table 13.17. Parameter properties of FG_SYSTEMMONITOR_CURRENT_LINK_SPEED

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

Example 13.17. Usage of FG_SYSTEMMONITOR_CURRENT_LINK_SPEED

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

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

13.18. FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE

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

Table 13.18. Parameter properties of FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE

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

Example 13.18. Usage of FG_SYSTEMMONITOR_PCIE_LINK_GEN2_CAPABLE

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

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

13.19. FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE

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

Table 13.19. Parameter properties of FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE

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

Example 13.19. Usage of FG_SYSTEMMONITOR_PCIE_LINK_PARTNER_GEN2_CAPABLE

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

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

```
}
```

13.20. FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE

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

Table 13.20. Parameter properties of FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE

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

Example 13.20. Usage of FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE

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

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

13.21. FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT

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

Table 13.21. Parameter properties of FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT

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

Example 13.21. Usage of FG_SYSTEMMONITOR_EXTENSION_CONNECTOR_PRESENT

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

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

13.22. FG_ALTERNATIVE_BOARD_DETECTION

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

This parameter is used for support purposes only.

Table 13.22. Parameter properties of FG_ALTERNATIVE_BOARD_DETECTION

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

Example 13.22. Usage of FG_ALTERNATIVE_BOARD_DETECTION

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

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

13.23. FG_SYSTEMMONITOR_FPGA_DNA

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

Table 13.23. Parameter properties of FG_SYSTEMMONITOR_FPGA_DNA

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

Example 13.23. Usage of FG_SYSTEMMONITOR_FPGA_DNA

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

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

13.24. FG_SYSTEMMONITOR_CHANNEL_STATE

Returns the Power over CXP link status. Value range is based on enumerator Fg_PoCXPState.

Table 13.24. Power over CXP state enumerator

Fg_PoCXPState	Description	Value
BOOTING	booting, not initialized	0x001
NOCABLE	no cable connected	0x002
NOPOCXP	no Power over CXP	0x004
POCXPOK	Power over CXP OK	0x008
MIN_CURR	minimum current	0x010
MAX_CURR	maximum current	0x020
LOW_VOLT	low voltage	0x040
OVER_VOLT	over voltage	0x080
ADC_Chip_Error	ADC Chip Error	0x100

The defines for the single values are named FG_POCXP_STATE_* corresponding to the enumerator names.

Table 13.25. Parameter properties of FG_SYSTEMMONITOR_CHANNEL_STATE

Property	Value
Name	**FG_SYSTEMMONITOR_CHANNEL_STATE**
Display Name	**Channel State**
Type	**Unsigned Integer Field**
Field Size	**0**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 13.24. Usage of FG_SYSTEMMONITOR_CHANNEL_STATE

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

Glossary

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

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

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

Index

G

I

M

O

R

T

U