imaFlex CXP-12 Penta

Test Applet User Documentation for

FrameGrabberTest

Functional Description
For Framegrabber SDK Usage

Document Number: AW001887
Part Number: 000 (English)
Document Version: 01
Release Date: 5 September 2024
Applet Version 1.0.1.0

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 imaFlex CXP-12 Penta frame grabbers.

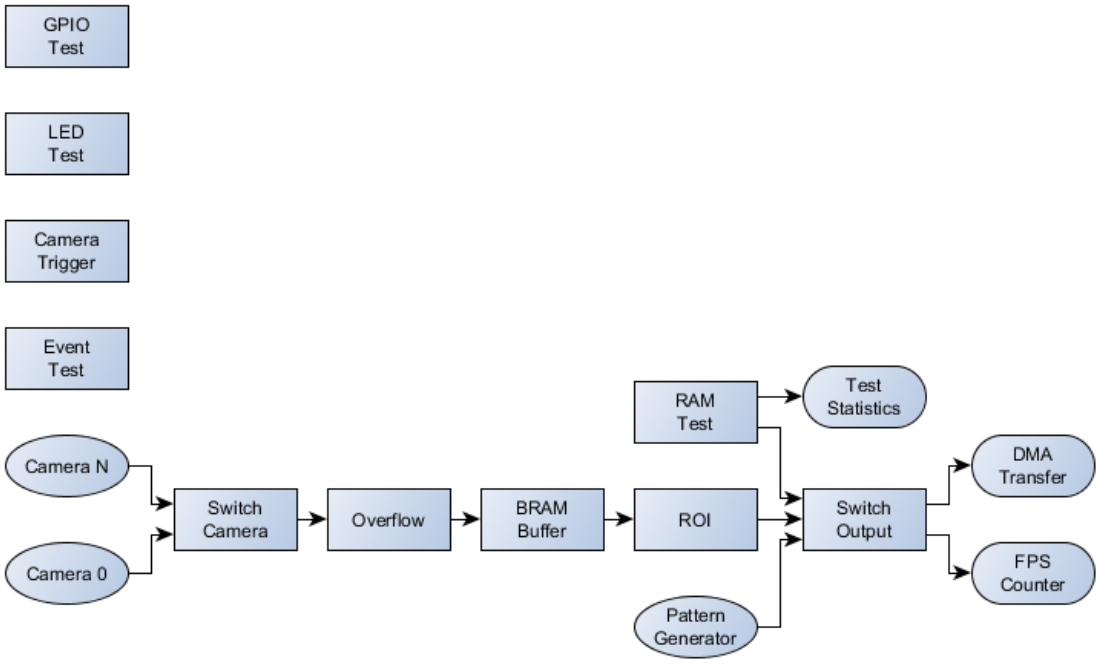
This applet is a frame grabber test applet. Its intention is to test the hardware. You shall not use this applet for your final image processing application. Use AcquisitionApplets or VA Applets instead.

The applet comprises the following functions:

- DMA Performance test: Different image dimensions for varying PC memory sizes and interrupt rates.
- RAM Test: Check for errors and bandwidth of the on board DRAM.
- 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*')
- DmaFromPC: Test the upload of data from the PC memory and back via direct memory access.

The following diagram shows the functional blocks of the applet.

Figure 1.1. Block Diagram of the applet



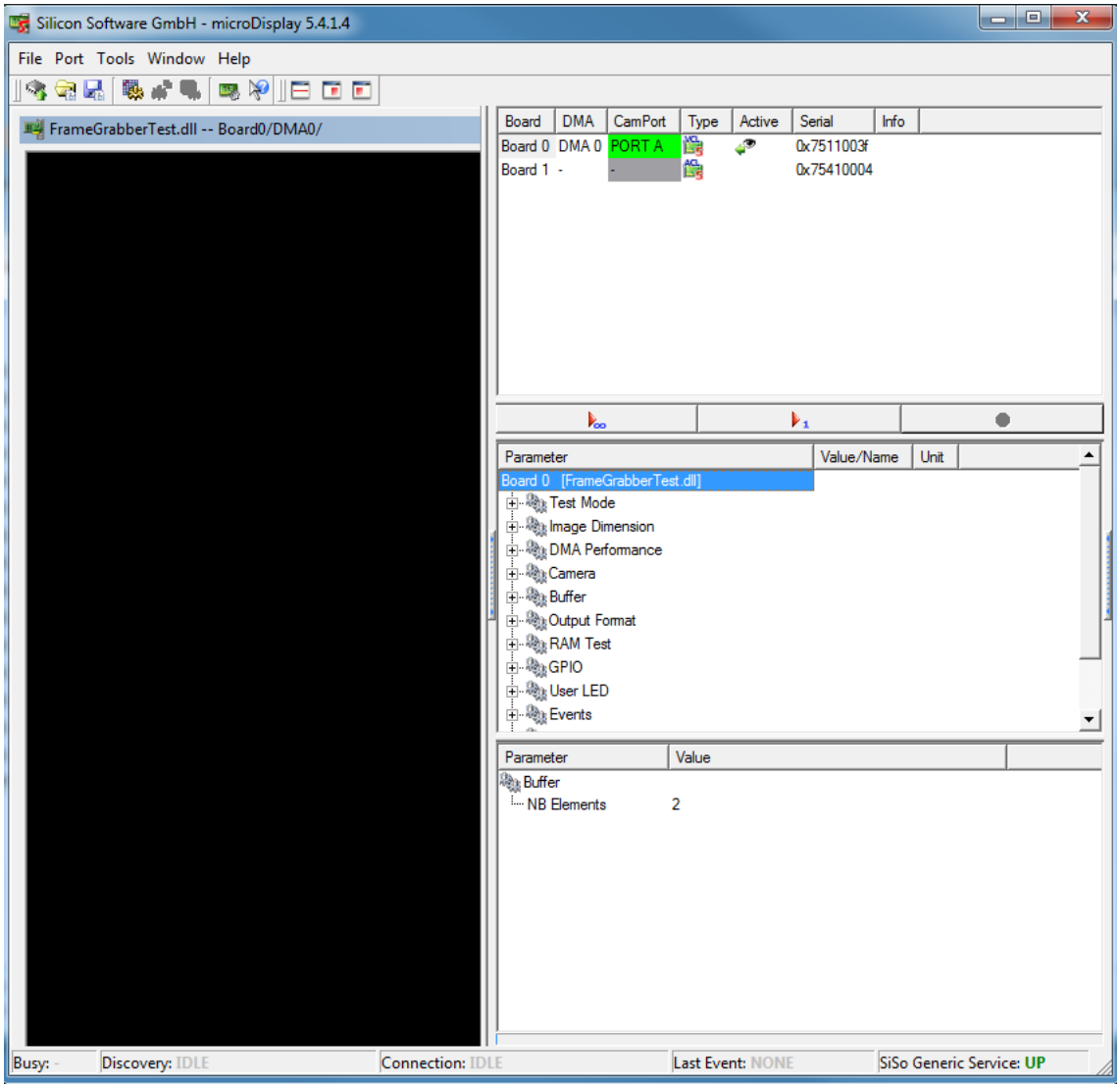
Chapter 2. Test Procedure in microDisplayX

In the following, the steps to test the hardware with the applet FrameGrabberTest in microDisplayX are explained. Of course, you can also integrate the tests in your own programs with the Framegrabber SDK, GenTL or pylon API.

2.1. 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. 2.1 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 2.1. Parameters of the applet 'FrameGrabbertest.dll' in 'microDisplay'



2.1.1. 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'.

2.1.2. RAM Test

To test the RAM performance of the RAM modules 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 does not exceed 1 MiB. Suggested display width in 'RAM Difference' mode is to 1024 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. 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'. You can also check the total RAM bandwidth using parameter *FG_RAM_BANDWIDTH*. You should set the output to **`RAM Differences`** or another test mode to esure that the DMA does not limit the DRAM.

2.1.3. 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 by setting the parameters 'FG_CCSEL0' to 'FG_CCSEL3' (under parameter 'Camera').

2.1.4. 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.

2.1.5. 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.

2.1.6. 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.

2.1.7. DmaFromPc Test

The applet allows a loopback test to send data from the PC memory to the frame grabber and back. Done via direct memory access. The test can be done to verify the functionality in software and test the PC PCIe and memory speed. Note that the test cannot be performed in microDisplayX. For further references on DmaFromPc check the VisualApplets user manual and FG SDK manual.

Chapter 3. CoaXPress

3.1. FG_SYSTEMMONITOR_POWER_OVER_CXP_STATE

Returns the power over CXP (PoCXP) state. Range: {BOOTING, POCXPOK, MAX_CURR, LOW_VOLT, OVER_VOLT, ADC_Chip_Error}. The first 5 states are defined by the CXP standard for the PoCXP state machine. The last state ADC_Chip_Error represents an error when the communication between the FPGA and the ADC chip is disrupted. The communication between the FPGA and ADC chip measures the voltage and current of the channel.

Table 3.1. Parameter properties of FG_SYSTEMMONITOR_POWER_OVER_CXP_STATE

Property	Value
Name	**FG_SYSTEMMONITOR_POWER_OVER_CXP_STATE**
Display Name	**Systemmonitor Power Over CXP State**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 3.1. Usage of FG_SYSTEMMONITOR_POWER_OVER_CXP_STATE

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

3.2. FG_SYSTEMMONITOR_POWER_OVER_CXP_CONTROLLER_ENABLED

Indicates whether the power over CXP (PoCXP) controller is enabled. Range: {NO, YES}. YES: The camera is powered via the CXP cable when connected. NO: The camera is not powered via the CXP cable. This parameter doesn't indicate whether the camera is sourced or not, instead it indicates whether powering the camera via the CXP cable is enabled or not.

Table 3.2. Parameter properties of FG_SYSTEMMONITOR_POWER_OVER_CXP_CONTROLLER_ENABLED

Property	Value
Name	**FG_SYSTEMMONITOR_POWER_OVER_CXP_CONTROLLER_ENABLED**
Display Name	**Systemmonitor Power Over CXP Controller Enabled**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 3.2. Usage of FG_SYSTEMMONITOR_POWER_OVER_CXP_CONTROLLER_ENABLED

```
int result = 0;
```

```
FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

3.3. FG_SYSTEMMONITOR_PORT_BIT_RATE

Returns the port bit rate of the CXP channel.

Table 3.3. Parameter properties of FG_SYSTEMMONITOR_PORT_BIT_RATE

Property	Value
Name	**FG_SYSTEMMONITOR_PORT_BIT_RATE**
Display Name	**Systemmonitor Port Bit Rate**
Type	**Double Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**
Unit of measure	**Gb/s**

Example 3.3. Usage of FG_SYSTEMMONITOR_PORT_BIT_RATE

```
int result = 0;

FieldParameterDouble access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMDOUBLE;

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

3.4. FG_SYSTEMMONITOR_STREAM_PACKET_SIZE

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

Table 3.4. Parameter properties of FG_SYSTEMMONITOR_STREAM_PACKET_SIZE

Property	Value
Name	**FG_SYSTEMMONITOR_STREAM_PACKET_SIZE**
Display Name	**Systemmonitor Stream Packet Size**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 3.4. Usage of FG_SYSTEMMONITOR_STREAM_PACKET_SIZE

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```
}
```

3.5. FG_SYSTEMMONITOR_CXP_STANDARD

Returns the version of the used CXP standard.

Table 3.5. CXP Standard Version

CXP Standard Version		
CXP_1_0		
CXP_1_1_1		
CXP_2_0		
Unknown		

Table 3.6. Parameter properties of FG_SYSTEMMONITOR_CXP_STANDARD

Property	Value
Name	**FG_SYSTEMMONITOR_CXP_STANDARD**
Display Name	**Systemmonitor CXP Standard**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 3.5. Usage of FG_SYSTEMMONITOR_CXP_STANDARD

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

3.6. FG_CXP_STREAM_PACKET_COUNT

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

Table 3.7. Parameter properties of FG_CXP_STREAM_PACKET_COUNT

Property	Value
Name	**FG_CXP_STREAM_PACKET_COUNT**
Display Name	**CXP Stream Packet Count**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 3.6. Usage of FG_CXP_STREAM_PACKET_COUNT

```
int result = 0;
```

```
FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

Chapter 4. Test Mode

4.1. FG_OUTPUT_SELECT

The frame grabber test applet offers several test modes

- DMA Performance Output
- Camera Image Output
- RAM Test Output
- DMA From PC loop Test

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.

The DMA From PC loop test can be used to test the upload of images to the frame grabber via a DMA channel. The uploaded images are directly forwarded to the download DMA channel and will be transfered back to the PC. So it is a simple loop.

To perform this test, a SDK program needs to be written. The test cannot be used in microDisplayX. Check the FG SDK user manual and DmaFromPc VisualApplets example code.

Table 4.1. 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_DMA_FROM_PC** DmaFromPC
Default value	**FG_DMA_PERFORMANCE**

Example 4.1. 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 5. Image Dimension

5.1. 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 *FG_OUTPUT_SELECT*.

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

Table 5.1. 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 32** **Maximum 16384** **Stepsize 32**
Default value	**1024**
Unit of measure	**pixel**

Example 5.1. 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 */
}
```

5.2. 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 *FG_OUTPUT_SELECT*.

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

Table 5.2. 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 5.2. 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 6. DMA Performance

6.1. 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 6.1. 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 6.1. 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 */
}
```

6.2. 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 6.2. 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 6.2. 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 7. Camera

7.1. FG_CAMERA_PORT

Select the camera port index.

Table 7.1. 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** 4 **Stepsize** 1
Default value	**0**

Example 7.1. 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 */
}
```

7.2. FG_TRIGGERCAMERA_OUT_SELECT

Table 7.2. 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 7.2. 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 8. Buffer

8.1. 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 8.1. 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 8.1. 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 */
}
```

8.2. 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.

Table 8.2. 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 8.2. 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 */
}
```

Chapter 9. Output Format

9.1. FG_FORMAT

Table 9.1. 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** Mono 8
Default value	**FG_GRAY**

Example 9.1. 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 */
}
```

9.2. 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 9.2. 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 9.2. 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 10. RAM Test

10.1. 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 10.1. 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 10.1. 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 */
}
```

10.2. FG_RAM_SIZE

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

Table 10.2. 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 10.2. 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 */
}
```

10.3. 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 10.3. 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 10.3. 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 */
}
```

10.4. 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 10.4. 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 10.4. 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 */
}
```

10.5. FG_ENABLE_RAM0

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 10.5. 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 10.5. 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 */
}
```

10.6. FG_ERROR_COUNT_RAM0

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 10.6. 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 10.6. 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 */
}
```

10.7. FG_IMAGE_COUNT_RAM0

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 10.7. 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 10.7. 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 */
}
```

10.8. FG_INJECT_ERRORS_RAM0

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

Table 10.8. 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 10.8. 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 11. GPIO

11.1. 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 11.1. 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 11.1. 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 */
}
```

11.2. 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 11.2. 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 11.2. 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 */
}
```

11.3. 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 11.3. 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 11.3. 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 */
}
```

11.4. 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 11.4. 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 11.4. 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 12. User LED

12.1. 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 12.1. 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 12.1. 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 */
}
```

12.2. 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 12.2. 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 12.2. 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 13. Miscellaneous

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

13.1. FG_TIMEOUT

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

Table 13.1. Parameter properties of FG_TIMEOUT

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

Example 13.1. Usage of FG_TIMEOUT

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

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

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

13.2. FG_APPLET_ID

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

Table 13.2. 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 13.2. 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 */
}
```

13.3. 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 13.3. 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 13.3. 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 */
}
```

13.4. 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 13.4. 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 13.4. 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 */
}
```

13.5. 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 13.5. 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 13.5. 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 */
}
```

13.6. 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 13.6. 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 13.6. 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 */
}
```

13.7. FG_SYSTEMMONITOR_FPGA_DNA_LOW

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

Table 13.7. Parameter properties of FG_SYSTEMMONITOR_FPGA_DNA_LOW

Property	Value
Name	**FG_SYSTEMMONITOR_FPGA_DNA_LOW**
Display Name	**FPGA DNA Low**
Type	**Unsigned Integer (64 Bit)**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 144115188075855872** **Stepsize 1**

Example 13.7. Usage of FG_SYSTEMMONITOR_FPGA_DNA_LOW

```
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_LOW, &value, 0, type)) < 0) {
    /* error handling */
}
```

13.8. FG_SYSTEMMONITOR_FPGA_DNA_HIGH

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

Table 13.8. Parameter properties of FG_SYSTEMMONITOR_FPGA_DNA_HIGH

Property	Value
Name	**FG_SYSTEMMONITOR_FPGA_DNA_HIGH**
Display Name	**FPGA DNA High**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 4294967295** **Stepsize 1**

Example 13.8. Usage of FG_SYSTEMMONITOR_FPGA_DNA_HIGH

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

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

13.9. GPIO Configuration

FIXME_CategoryDocumentation_Missing_Miscellaneous::GPIO Configuration

13.9.1. FG_EXTENSION_GPO_TYPE

Table 13.9. Parameter properties of FG_EXTENSION_GPO_TYPE

Property	Value	
Name	**FG_EXTENSION_GPO_TYPE**	
Display Name	**Extension GPO Type**	
Type	**Enumeration**	
Access policy	**Read/Write**	
Storage policy	**Transient**	
Allowed values	**FG_GPO_PUSH_PULL** **FG_GPO_OPEN_DRAIN**	Push/pull configuration Open drain configuration
Default value	**FG_GPO_OPEN_DRAIN**	

Example 13.9. Usage of FG_EXTENSION_GPO_TYPE

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

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

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

13.9.2. FG_FRONT_GPI_PULL_CONTROL

This parameter either activates the FPGA-internal pull-up or pull-down resistors for the front GPIs. In pull-up mode, the incoming signal must have been actively driven low, while in pull-down mode it must have been actively driven high.

Table 13.10. Parameter properties of FG_FRONT_GPI_PULL_CONTROL

Property	Value	
Name	**FG_FRONT_GPI_PULL_CONTROL**	
Display Name	**Front GPI Pull Control**	
Type	**Enumeration**	
Access policy	**Read/Write**	
Storage policy	**Transient**	
Allowed values	**FG_FRONT_GPI_PULL_DOWN** **FG_FRONT_GPI_PULL_UP**	Pull-down Pull-up
Default value	**FG_FRONT_GPI_PULL_UP**	

Example 13.10. Usage of FG_FRONT_GPI_PULL_CONTROL

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

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

```
    /* error handling */
}

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

13.9.3. FG_FRONT_GPI_TYPE

With this parameter, the front GPIs are configured either as single-ended signals or as differential signals.

Table 13.11. Parameter properties of FG_FRONT_GPI_TYPE

Property	Value
Name	**FG_FRONT_GPI_TYPE**
Display Name	**Front GPI Signal Type**
Type	**Enumeration**
Access policy	**Read/Write**
Storage policy	**Transient**
Allowed values	**FG_FRONT_GPI_SINGLE_ENDED** Single-ended **FG_FRONT_GPI_DIFFERENTIAL** Differential
Default value	**FG_FRONT_GPI_SINGLE_ENDED**

Example 13.11. Usage of FG_FRONT_GPI_TYPE

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

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

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

13.9.4. FG_FRONT_GPO_INVERSION

When enabled, the output of the front GPOs are inverted.

Table 13.12. Parameter properties of FG_FRONT_GPO_INVERSION

Property	Value
Name	**FG_FRONT_GPO_INVERSION**
Display Name	**Front GPO Inversion**
Type	**Enumeration**
Access policy	**Read/Write**
Storage policy	**Transient**
Allowed values	**FG_FRONT_GPO_INVERSION_OFF** Inversion off **FG_FRONT_GPO_INVERSION_ON** Inversion on
Default value	**FG_FRONT_GPO_INVERSION_OFF**

Example 13.12. Usage of FG_FRONT_GPO_INVERSION

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

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

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

13.10. Version

FIXME_CategoryDocumentation_Missing_Miscellaneous::Version

13.10.1. FG_APPLET_VERSION

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

Table 13.13. 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**
Allowed values	**Minimum 0** **Maximum 256** **Stepsize 1**

Example 13.13. 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 */
}
```

13.10.2. FG_APPLET_REVISION

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

Table 13.14. 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**
Allowed values	**Minimum 0** **Maximum 256** **Stepsize 1**

Example 13.14. 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 */
}
```

13.10.3. FG_VISUALAPPLETS_BUILD_VERSION

Returns the VisualApplets version used to build the applets.

Table 13.15. Parameter properties of FG_VISUALAPPLETS_BUILD_VERSION

Property	Value
Name	**FG_VISUALAPPLETS_BUILD_VERSION**
Display Name	**VisualApplets Build Version**
Type	**String**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 13.15. Usage of FG_VISUALAPPLETS_BUILD_VERSION

```
int result = 0;
char* value = n/a;
const enum FgParamTypes type = FG_PARAM_TYPE_CHAR_PTR;

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

Chapter 14. Boardstatus

14.1. FG_SYSTEMMONITOR_CHANNEL_CURRENT

Returns the power consumption of the CXP channel (PoCXP) in Ampere.

Table 14.1. Parameter properties of FG_SYSTEMMONITOR_CHANNEL_CURRENT

Property	Value
Name	**FG_SYSTEMMONITOR_CHANNEL_CURRENT**
Display Name	**Systemmonitor Channel Current**
Type	**Double Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**
Unit of measure	**A**

Example 14.1. 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 */
    }
}
```

14.2. FG_SYSTEMMONITOR_CHANNEL_VOLTAGE

Returns the voltage of the CXP channel (PoCXP).

Table 14.2. Parameter properties of FG_SYSTEMMONITOR_CHANNEL_VOLTAGE

Property	Value
Name	**FG_SYSTEMMONITOR_CHANNEL_VOLTAGE**
Display Name	**Systemmonitor Channel Voltage**
Type	**Double Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**
Unit of measure	**V**

Example 14.2. 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 */
    }
```

```
}
```

14.3. FG_SYSTEMMONITOR_MAPPED_TO_FG_PORT

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

Table 14.3. Parameter properties of FG_SYSTEMMONITOR_MAPPED_TO_FG_PORT

Property	Value
Name	**FG_SYSTEMMONITOR_MAPPED_TO_FG_PORT**
Display Name	**Systemmonitor Mapped to Fg Port**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 14.3. Usage of FG_SYSTEMMONITOR_MAPPED_TO_FG_PORT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

14.4. FG_DMASTATUS

Returns the status of the DMA transmission, i.e. the acquisition state. 0 = stopped DMA, 1 = started DMA.

Table 14.4. Parameter properties of FG_DMASTATUS

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

Example 14.4. 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 */
}
```

14.5. FG_SYSTEMMONITOR_FPGA_TEMPERATURE

Returns the current FGPA temperature.

Table 14.5. Parameter properties of FG_SYSTEMMONITOR_FPGA_TEMPERATURE

Property	Value
Name	**FG_SYSTEMMONITOR_FPGA_TEMPERATURE**
Display Name	**Systemmonitor 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 14.5. 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 */
}
```

14.6. FG_SYSTEMMONITOR_FPGA_VCC_INT

Returns the internal voltage of the FPGA.

Table 14.6. Parameter properties of FG_SYSTEMMONITOR_FPGA_VCC_INT

Property	Value
Name	**FG_SYSTEMMONITOR_FPGA_VCC_INT**
Display Name	**Systemmonitor 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 14.6. 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 */
}
```

14.7. FG_SYSTEMMONITOR_FPGA_VCC_AUX

Returns the VCC auxiliary voltage of the FPGA.

Table 14.7. 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 14.7. 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 */
}
```

14.8. FG_SYSTEMMONITOR_FPGA_VCC_BRAM

Returns the VCC of the BlockRAM voltage of the FPGA.

Table 14.8. Parameter properties of FG_SYSTEMMONITOR_FPGA_VCC_BRAM

Property	Value
Name	**FG_SYSTEMMONITOR_FPGA_VCC_BRAM**
Display Name	**Systemmonitor 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 14.8. 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 */
}
```

14.9. FG_SYSTEMMONITOR_CURRENT_LINK_WIDTH

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

Table 14.9. 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 14.9. 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 */
}
```

14.10. FG_SYSTEMMONITOR_CURRENT_LINK_SPEED

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

Table 14.10. Parameter properties of FG_SYSTEMMONITOR_CURRENT_LINK_SPEED

Property	Value
Name	**FG_SYSTEMMONITOR_CURRENT_LINK_SPEED**
Display Name	**Systemmonitor 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 14.10. 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 */
}
```

14.11. FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE

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

Table 14.11. Parameter properties of FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE

Property	Value
Name	**FG_SYSTEMMONITOR_PCIE_TRAINED_PAYLOAD_SIZE**
Display Name	**Systemmonitor 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 14.11. 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 */
}
```

14.12. FG_SYSTEMMONITOR_PCIE_TRAINED_REQUEST_SIZE

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

Table 14.12. Parameter properties of FG_SYSTEMMONITOR_PCIE_TRAINED_REQUEST_SIZE

Property	Value
Name	**FG_SYSTEMMONITOR_PCIE_TRAINED_REQUEST_SIZE**
Display Name	**Systemmonitor PCIe Trained Request Size**
Type	**Unsigned Integer**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**Minimum 0** **Maximum 4096** **Stepsize 0**
Unit of measure	**byte**

Example 14.12. Usage of FG_SYSTEMMONITOR_PCIE_TRAINED_REQUEST_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_REQUEST_SIZE, &value, 0, type)) < 0) {
    /* error handling */
}
```

14.13. FG_SYSTEMMONITOR_EXTERNAL_POWER

Indicates whether the external power connector is connected.

Table 14.13. Parameter properties of FG_SYSTEMMONITOR_EXTERNAL_POWER

Property	Value
Name	**FG_SYSTEMMONITOR_EXTERNAL_POWER**
Display Name	**Systemmonitor External Power**
Type	**Enumeration**
Access policy	**Read-Only**
Storage policy	**Transient**
Allowed values	**FG_GOOD** Good **FG_NO_POWER** No Power

Example 14.13. Usage of FG_SYSTEMMONITOR_EXTERNAL_POWER

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

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

14.14. FG_CXP_INPUT_MAPPED_FW_PORT_PORT

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

Table 14.14. Parameter properties of FG_CXP_INPUT_MAPPED_FW_PORT_PORT

Property	Value
Name	**FG_CXP_INPUT_MAPPED_FW_PORT_PORT**
Display Name	**CXP Input Mapped to Firmware Port Port**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 14.14. Usage of FG_CXP_INPUT_MAPPED_FW_PORT_PORT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

Chapter 15. Errors

15.1. FG_SYSTEMMONITOR_DECODER_8B_10B_ERROR

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

Table 15.1. Parameter properties of FG_SYSTEMMONITOR_DECODER_8B_10B_ERROR

Property	Value
Name	**FG_SYSTEMMONITOR_DECODER_8B_10B_ERROR**
Display Name	**Systemmonitor Decoder 8b10b Error**
Type	**Unsigned Integer Field (64 Bit)**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.1. Usage of FG_SYSTEMMONITOR_DECODER_8B_10B_ERROR

```
int result = 0;

FieldParameterAccess access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMACCESS;

    if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_DECODER_8B_10B_ERROR, &access, 0, type)) < 0) {
        /* error handling */
    }
}
```

15.2. FG_SYSTEMMONITOR_BYTE_ALIGNMENT_8B_10B_LOCKED

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

Table 15.2. Parameter properties of FG_SYSTEMMONITOR_BYTE_ALIGNMENT_8B_10B_LOCKED

Property	Value
Name	**FG_SYSTEMMONITOR_BYTE_ALIGNMENT_8B_10B_LOCKED**
Display Name	**Systemmonitor Byte Alignment 8B 10 B Locked**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.2. Usage of FG_SYSTEMMONITOR_BYTE_ALIGNMENT_8B_10B_LOCKED

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

    if ((result = Fg_getParameterWithType(fg, FG_SYSTEMMONITOR_BYTE_ALIGNMENT_8B_10B_LOCKED, &access, 0, type)) < 0) {
        /* error handling */
    }
```

```
}
```

15.3. FG_SYSTEMMONITOR_RX_STREAM_INCOMPLETE_COUNT

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

Table 15.3. Parameter properties of FG_SYSTEMMONITOR_RX_STREAM_INCOMPLETE_COUNT

Property	Value
Name	**FG_SYSTEMMONITOR_RX_STREAM_INCOMPLETE_COUNT**
Display Name	**Systemmonitor Rx Stream Incomplete Count**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.3. Usage of FG_SYSTEMMONITOR_RX_STREAM_INCOMPLETE_COUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.4. FG_SYSTEMMONITOR_RX_UNKNOWN_DATA_RECEIVED_COUNT

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

Table 15.4. Parameter properties of FG_SYSTEMMONITOR_RX_UNKNOWN_DATA_RECEIVED_COUNT

Property	Value
Name	**FG_SYSTEMMONITOR_RX_UNKNOWN_DATA_RECEIVED_COUNT**
Display Name	**Systemmonitor Rx Unknown Data Received Count**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.4. Usage of FG_SYSTEMMONITOR_RX_UNKNOWN_DATA_RECEIVED_COUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.5. FG_CXP_OVERTRIGGER_REQUEST_PULSECOUNT

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

Table 15.5. Parameter properties of FG_CXP_OVERTRIGGER_REQUEST_PULSECOUNT

Property	Value
Name	**FG_CXP_OVERTRIGGER_REQUEST_PULSECOUNT**
Display Name	**CXP Overtrigger Request Pulse Count**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.5. Usage of FG_CXP_OVERTRIGGER_REQUEST_PULSECOUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.6. FG_CXP_TRIGGER_ACK_MISSING_COUNT

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

Table 15.6. Parameter properties of FG_CXP_TRIGGER_ACK_MISSING_COUNT

Property	Value
Name	**FG_CXP_TRIGGER_ACK_MISSING_COUNT**
Display Name	**CXP Lost Trigger ACK Missing Count**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.6. Usage of FG_CXP_TRIGGER_ACK_MISSING_COUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```
}
```

15.7. FG_CXP_CONTROL_ACK_LOST_COUNT

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

Table 15.7. Parameter properties of FG_CXP_CONTROL_ACK_LOST_COUNT

Property	Value
Name	**FG_CXP_CONTROL_ACK_LOST_COUNT**
Display Name	**CXP Control ACK Lost Count**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.7. Usage of FG_CXP_CONTROL_ACK_LOST_COUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.8. FG_CXP_CONTROL_TAG_ERROR_COUNT

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

Table 15.8. Parameter properties of FG_CXP_CONTROL_TAG_ERROR_COUNT

Property	Value
Name	**FG_CXP_CONTROL_TAG_ERROR_COUNT**
Display Name	**CXP Control Tag Error Count**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.8. Usage of FG_CXP_CONTROL_TAG_ERROR_COUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

    if ((result = Fg_getParameterWithType(fg, FG_CXP_CONTROL_TAG_ERROR_COUNT, &access, 0, type)) < 0) {
```

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

15.9. FG_CXP_CONTROL_ACK_INCOMPLETE_COUNT

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

Table 15.9. Parameter properties of FG_CXP_CONTROL_ACK_INCOMPLETE_COUNT

Property	Value
Name	**FG_CXP_CONTROL_ACK_INCOMPLETE_COUNT**
Display Name	**CXP Control ACK Incomplete Count**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.9. Usage of FG_CXP_CONTROL_ACK_INCOMPLETE_COUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.10. FG_CXP_HEARTBEAT_INCOMPLETE_COUNT

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

Table 15.10. Parameter properties of FG_CXP_HEARTBEAT_INCOMPLETE_COUNT

Property	Value
Name	**FG_CXP_HEARTBEAT_INCOMPLETE_COUNT**
Display Name	**CXP Heartbeat Incomplete Count**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.10. Usage of FG_CXP_HEARTBEAT_INCOMPLETE_COUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

    if ((result = Fg_getParameterWithType(fg, FG_CXP_HEARTBEAT_INCOMPLETE_COUNT, &access, 0, type)) < 0) {
```

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

15.11. FG_CXP_HEARTBEAT_MAX_PERIOD_VIOLATION_COUNT

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

Table 15.11. Parameter properties of FG_CXP_HEARTBEAT_MAX_PERIOD_VIOLATION_COUNT

Property	Value
Name	**FG_CXP_HEARTBEAT_MAX_PERIOD_VIOLATION_COUNT**
Display Name	**CXP Hearbeat Max Period Violation Count**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.11. Usage of FG_CXP_HEARTBEAT_MAX_PERIOD_VIOLATION_COUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.12. CRC

FIXME_CategoryDocumentation_Missing_Errors::CRC

15.12.1. FG_SYSTEMMONITOR_RX_PACKET_CRC_ERROR_COUNT

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

Table 15.12. Parameter properties of FG_SYSTEMMONITOR_RX_PACKET_CRC_ERROR_COUNT

Property	Value
Name	**FG_SYSTEMMONITOR_RX_PACKET_CRC_ERROR_COUNT**
Display Name	**Systemmonitor Rx Packet CRC Error Count**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.12. Usage of FG_SYSTEMMONITOR_RX_PACKET_CRC_ERROR_COUNT

```
int result = 0;
```

```
FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.12.2. FG_CXP_STREAMPACKET_CRC_ERROR

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

Table 15.13. Parameter properties of FG_CXP_STREAMPACKET_CRC_ERROR

Property	Value
Name	**FG_CXP_STREAMPACKET_CRC_ERROR**
Display Name	**CXP Stream Packet CRC Error**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.13. Usage of FG_CXP_STREAMPACKET_CRC_ERROR

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.12.3. FG_CXP_CONTROL_ACK_PACKET_CRC_ERROR

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

Table 15.14. Parameter properties of FG_CXP_CONTROL_ACK_PACKET_CRC_ERROR

Property	Value
Name	**FG_CXP_CONTROL_ACK_PACKET_CRC_ERROR**
Display Name	**CXP Control ACK Packet CRC Error**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.14. Usage of FG_CXP_CONTROL_ACK_PACKET_CRC_ERROR

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;
```

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

15.13. LengthErrors

FIXME_CategoryDocumentation_Missing_Errors::LengthErrors

15.13.1. FG_SYSTEMMONITOR_RX_LENGTH_ERROR_COUNT

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

Table 15.15. Parameter properties of FG_SYSTEMMONITOR_RX_LENGTH_ERROR_COUNT

Property	Value
Name	**FG_SYSTEMMONITOR_RX_LENGTH_ERROR_COUNT**
Display Name	**Systemmonitor Rx Length Error Count**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.15. Usage of FG_SYSTEMMONITOR_RX_LENGTH_ERROR_COUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.13.2. FG_CXP_STREAMPACKET_LENGTH_ERROR

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

Table 15.16. Parameter properties of FG_CXP_STREAMPACKET_LENGTH_ERROR

Property	Value
Name	**FG_CXP_STREAMPACKET_LENGTH_ERROR**
Display Name	**CXP Stream Packet Length Error**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.16. Usage of FG_CXP_STREAMPACKET_LENGTH_ERROR

```
int result = 0;

FieldParameterInt access;
```

```
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.14. ReceivedPacketsCorrected

FIXME_CategoryDocumentation_Missing_Errors::ReceivedPacketsCorrected

15.14.1. FG_CXP_ERROR_CORRECTED

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

Table 15.17. Parameter properties of FG_CXP_ERROR_CORRECTED

Property	Value
Name	**FG_CXP_ERROR_CORRECTED**
Display Name	**CXP Error Corrected**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.17. Usage of FG_CXP_ERROR_CORRECTED

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.14.2. FG_CXP_ERROR_CORRECTED_TRIGGER

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

Table 15.18. Parameter properties of FG_CXP_ERROR_CORRECTED_TRIGGER

Property	Value
Name	**FG_CXP_ERROR_CORRECTED_TRIGGER**
Display Name	**CXP Error Corrected Trigger**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.18. Usage of FG_CXP_ERROR_CORRECTED_TRIGGER

```
int result = 0;
```

```
FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.14.3. FG_CXP_ERROR_CORRECTED_TRIGGER_ACK

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

Table 15.19. Parameter properties of FG_CXP_ERROR_CORRECTED_TRIGGER_ACK

Property	Value
Name	**FG_CXP_ERROR_CORRECTED_TRIGGER_ACK**
Display Name	**CXP Error Corrected Trigger ACK**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.19. Usage of FG_CXP_ERROR_CORRECTED_TRIGGER_ACK

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.14.4. FG_CXP_ERROR_CORRECTED_STREAM

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

Table 15.20. Parameter properties of FG_CXP_ERROR_CORRECTED_STREAM

Property	Value
Name	**FG_CXP_ERROR_CORRECTED_STREAM**
Display Name	**CXP Error Corrected Stream**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.20. Usage of FG_CXP_ERROR_CORRECTED_STREAM

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;
```

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

15.14.5. FG_CXP_ERROR_CORRECTED_CONTROL_ACK

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

Table 15.21. Parameter properties of FG_CXP_ERROR_CORRECTED_CONTROL_ACK

Property	Value
Name	**FG_CXP_ERROR_CORRECTED_CONTROL_ACK**
Display Name	**CXP Error Corrected Control ACK**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.21. Usage of FG_CXP_ERROR_CORRECTED_CONTROL_ACK

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.14.6. FG_CXP_ERROR_CORRECTED_LINKTEST

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

Table 15.22. Parameter properties of FG_CXP_ERROR_CORRECTED_LINKTEST

Property	Value
Name	**FG_CXP_ERROR_CORRECTED_LINKTEST**
Display Name	**CXP Error Corrected Link Test**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.22. Usage of FG_CXP_ERROR_CORRECTED_LINKTEST

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

```
    }
}
```

15.14.7. FG_CXP_ERROR_CORRECTED_HEARTBEAT

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

Table 15.23. Parameter properties of FG_CXP_ERROR_CORRECTED_HEARTBEAT

Property	Value
Name	**FG_CXP_ERROR_CORRECTED_HEARTBEAT**
Display Name	**CXP Error Corrected Heartbeat**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.23. Usage of FG_CXP_ERROR_CORRECTED_HEARTBEAT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.15. ReceivedPacketsUncorrected

FIXME_CategoryDocumentation_Missing_Errors::ReceivedPacketsUncorrected

15.15.1. FG_CXP_ERROR_UNCORRECTED

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

Table 15.24. Parameter properties of FG_CXP_ERROR_UNCORRECTED

Property	Value
Name	**FG_CXP_ERROR_UNCORRECTED**
Display Name	**CXP Error Uncorrected**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.24. Usage of FG_CXP_ERROR_UNCORRECTED

```
int result = 0;

FieldParameterInt access;
```

```
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.15.2. FG_CXP_ERROR_UNCORRECTED_TRIGGER

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

Table 15.25. Parameter properties of FG_CXP_ERROR_UNCORRECTED_TRIGGER

Property	Value
Name	**FG_CXP_ERROR_UNCORRECTED_TRIGGER**
Display Name	**CXP Error Uncorrected Trigger**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.25. Usage of FG_CXP_ERROR_UNCORRECTED_TRIGGER

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.15.3. FG_CXP_ERROR_UNCORRECTED_TRIGGER_ACK

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

Table 15.26. Parameter properties of FG_CXP_ERROR_UNCORRECTED_TRIGGER_ACK

Property	Value
Name	**FG_CXP_ERROR_UNCORRECTED_TRIGGER_ACK**
Display Name	**CXP Error Uncorrected Trigger ACK**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.26. Usage of FG_CXP_ERROR_UNCORRECTED_TRIGGER_ACK

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

    if ((result = Fg_getParameterWithType(fg, FG_CXP_ERROR_UNCORRECTED_TRIGGER_ACK, &access, 0, type)) < 0) {
```

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

15.15.4. FG_CXP_ERROR_UNCORRECTED_STREAM

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

Table 15.27. Parameter properties of FG_CXP_ERROR_UNCORRECTED_STREAM

Property	Value
Name	**FG_CXP_ERROR_UNCORRECTED_STREAM**
Display Name	**CXP Error Uncorrected Stream**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.27. Usage of FG_CXP_ERROR_UNCORRECTED_STREAM

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.15.5. FG_CXP_ERROR_UNCORRECTED_CONTROL_ACK

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

Table 15.28. Parameter properties of FG_CXP_ERROR_UNCORRECTED_CONTROL_ACK

Property	Value
Name	**FG_CXP_ERROR_UNCORRECTED_CONTROL_ACK**
Display Name	**CXP Error Uncorrected Control ACK**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.28. Usage of FG_CXP_ERROR_UNCORRECTED_CONTROL_ACK

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.15.6. FG_CXP_ERROR_UNCORRECTED_LINKTEST

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

Table 15.29. Parameter properties of FG_CXP_ERROR_UNCORRECTED_LINKTEST

Property	Value
Name	**FG_CXP_ERROR_UNCORRECTED_LINKTEST**
Display Name	**CXP Error Uncorrected Link Test**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.29. Usage of FG_CXP_ERROR_UNCORRECTED_LINKTEST

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.15.7. FG_CXP_ERROR_UNCORRECTED_HEARTBEAT

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

Table 15.30. Parameter properties of FG_CXP_ERROR_UNCORRECTED_HEARTBEAT

Property	Value
Name	**FG_CXP_ERROR_UNCORRECTED_HEARTBEAT**
Display Name	**CXP Error Uncorrected Heartbeat**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.30. Usage of FG_CXP_ERROR_UNCORRECTED_HEARTBEAT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.16. UnsupportedPackets

FIXME_CategoryDocumentation_Missing_Errors::UnsupportedPackets

15.16.1. FG_SYSTEMMONITOR_RX_UNSUPPORTED_PACKET_COUNT

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

Table 15.31. Parameter properties of FG_SYSTEMMONITOR_RX_UNSUPPORTED_PACKET_COUNT

Property	Value
Name	**FG_SYSTEMMONITOR_RX_UNSUPPORTED_PACKET_COUNT**
Display Name	**Systemmonitor Rx Unsupported Packet Unit**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.31. Usage of FG_SYSTEMMONITOR_RX_UNSUPPORTED_PACKET_COUNT

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.16.2. FG_CXP_UNSUPPORTED_GPIO_RECEIVED

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

Table 15.32. Parameter properties of FG_CXP_UNSUPPORTED_GPIO_RECEIVED

Property	Value
Name	**FG_CXP_UNSUPPORTED_GPIO_RECEIVED**
Display Name	**CXP Unsupported GPIO Received**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.32. Usage of FG_CXP_UNSUPPORTED_GPIO_RECEIVED

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.16.3. FG_CXP_UNSUPPORTED_EVENT_RECEIVED

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

Table 15.33. Parameter properties of FG_CXP_UNSUPPORTED_EVENT_RECEIVED

Property	Value
Name	**FG_CXP_UNSUPPORTED_EVENT_RECEIVED**
Display Name	**CXP Unsupported Event Received**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.33. Usage of FG_CXP_UNSUPPORTED_EVENT_RECEIVED

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.16.4. FG_CXP_UNSUPPORTED_HEARTBEAT_RECEIVED

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

Table 15.34. Parameter properties of FG_CXP_UNSUPPORTED_HEARTBEAT_RECEIVED

Property	Value
Name	**FG_CXP_UNSUPPORTED_HEARTBEAT_RECEIVED**
Display Name	**CXP Unsupported Hearbeat Received**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.34. Usage of FG_CXP_UNSUPPORTED_HEARTBEAT_RECEIVED

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.16.5. FG_CXP_UNSUPPORTED_GPIO_ACK_RECEIVED

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

Table 15.35. Parameter properties of FG_CXP_UNSUPPORTED_GPIO_ACK_RECEIVED

Property	Value
Name	**FG_CXP_UNSUPPORTED_GPIO_ACK_RECEIVED**
Display Name	**CXP Unsupported GPIO ACK Received**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.35. Usage of FG_CXP_UNSUPPORTED_GPIO_ACK_RECEIVED

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

15.16.6. FG_CXP_UNSUPPORTED_GPIO_REQUEST_RECEIVED

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

Table 15.36. Parameter properties of FG_CXP_UNSUPPORTED_GPIO_REQUEST_RECEIVED

Property	Value
Name	**FG_CXP_UNSUPPORTED_GPIO_REQUEST_RECEIVED**
Display Name	**CXP Unsupported GPIO Request Received**
Type	**Unsigned Integer Field**
Field Size	**5**
Access policy	**Read-Only**
Storage policy	**Transient**

Example 15.36. Usage of FG_CXP_UNSUPPORTED_GPIO_REQUEST_RECEIVED

```
int result = 0;

FieldParameterInt access;
const enum FgParamTypes type = FG_PARAM_TYPE_STRUCT_FIELDPARAMINT;

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

Glossary

Area of Interest (AOI)	See Region of Interest.
Board	A Basler hardware. Usually, a board is represented by a frame grabber. Boards might comprise multiple devices.
Board ID Number	An identification number of a Basler board in a PC system. The number is not fixed to a specific hardware but has to be unique in a PC system.
Camera Index	The index of a camera connected to a 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 Basler 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). Basler 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. Basler 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