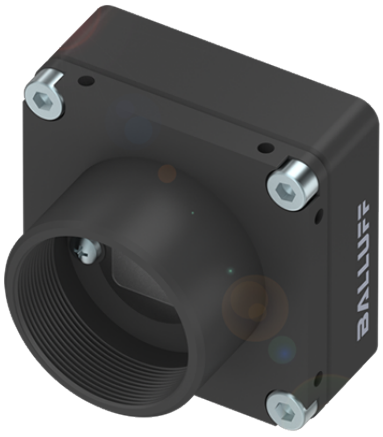
BALLUFF



BVS CA-MLC / BVS CA-IGC / mvBlueFOX

Technical Manual

English - Version 3.04

1 About this manual

1.1 Goal of the manual

This manual gives you an overview of the BVS CA-MLC / -IGC devices, Balluff's compact USB2 industrial camera family, its technical data and basic operation of the mvBlueFOX. Programming the device is detailed in a separate documentation, which will be available in an online format.

1.2 Contents of the manual

At the beginning of the manual, you will get an introduction to the possible usages of the camera. The following chapters contain general information about the camera including:

- Quickstart followed by
- Technical Data
- Sensor Overview
- Filters

The general information is followed by the description of the

- Software tools for the camera including the tools
- HRTC - Hardware Real-Time Controller shows how to use the FPGA built-in functionality called Hardware Real-Time Controller (short: HRTC).
- Developing applications using the Impact Acquire SDK
- DirectShow developers documents Balluff's Impact Acquire to DirectShow interface(DirectShow_acquire).
- Troubleshooting shows how to detect damages and other inconveniences.
- Use Cases describes solutions for general tasks and

- A Glossary explains abbreviations and technical terms.

- Appendix A. Specific Camera / Sensor Data contains all data of the sensors like timings, details of operation, etc.
  - A.1 CCD contains all data of the CCD sensors like timings, details of operation, etc.
  - A.2 CMOS contains all data of the other CMOS sensors like timings, details of operation, etc.

- Appendix C. Tested ARM platforms contains a list of ARM platforms tested with this product and information on how to setup these systems for achieving optimal results

2 Imprint

Headquarters	**DACH Service Center**	**Southern Europe Service Center**
Germany	**Germany**	**Italy**
Balluff GmbH Schurwaldstrasse 9 73765 Neuhausen a.d.F. Phone +49 7158 173-0 Fax +49 7158 5010 balluff@balluff.de	Balluff GmbH Schurwaldstrasse 9 73765 Neuhausen a.d.F. Phone +49 7158 173-370 service.de@balluff.de	Balluff Automation S.R.L. Corso Cuneo 15 10078 Venaria Reale (Torino) Phone +39 0113150711 service.it@balluff.it
Eastern Europe Service Center	**Americas Service Center**	**Asia Pacific Service Center**
Poland	**USA**	**Greater China**
Balluff Sp. z o.o. Ul. Graniczna 21A 54-516 Wrocław Phone +48 71 382 09 02 service.pl@balluff.pl	Balluff Inc. 8125 Holton Drive Florence, KY 41042 Toll-free +1 800 543 8390 Fax +1 859 727 4823 service.us@balluff.com	Balluff Automation (Shanghai) Co., Ltd. No. 800 Chengshan Rd, 8F, Building A, Yunding International Commercial Plaza 200125, Pudong, Shanghai Phone +86 400 820 0016 Fax +86 400 920 2622 service.cn@balluff.com.cn

Date

2023

This document assumes a general knowledge of PCs and programming.

Since the documentation is published electronically, an updated version may be available online. For this reason we recommend checking for updates on the Balluff website.

Balluff cannot guarantee that the data is free of errors or is accurate and complete and, therefore, assumes no liability for loss or damage of any kind incurred directly or indirectly through the use of the information of this document.

Balluff reserves the right to change technical data and design and specifications of the described products at any time without notice.

Copyright

- Windows® Vista, Windows® 7, 8, 10, 11 are trademarks of Microsoft, Corp.
- Linux® is a trademark of Linus Torvalds.
- Jetson is a registered trademark of NVIDIA Corporation.
- NVIDIA and Jetson are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries.
- Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

All other product and company names in this document may be the trademarks and tradenames of their respective owners and are hereby acknowledged.

3 Legal Notice

3.1 Firmware And Device Driver

The firmware running on Balluff/MATRIX VISION devices make use of a couple of third party software packages that come with various licenses. This section is meant to list all these packages and to give credit to those whose code helped in the creation of this software:

> **Note**
>
> If this section does not contain any additional information this means that for this particular product family no third party specific code was used.

3.1.1 cJSON

A slightly modified version of cJSON is used inside some of the modules that eventually build up the firmware.

```
Copyright (c) 2009 Dave Gamble

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

3.1.2 Unity

A slightly modified version of Unity (https://github.com/ThrowTheSwitch/Unity) is used for unit testing various modules that eventually build up the firmware.

3.2 Doxygen

Doxygen as well as the doxygen-awesome project have be used to generate the documentation you are just looking at. Details regarding licenses and versions can be found like described in the Impact Acquire SDK section.

3.3 Impact Acquire SDK

This SDK and its underlying libraries and drivers as well as some of the applications shipped with the Impact Acquire packages make use of a couple of third party software packages that come with various licenses. For up to date details on this topic please refer to the corresponding section in one of the SDK's API manuals. Direct links can be found here: Developing applications using the Impact Acquire SDK .

4 Revisions

Date	Rev.	Author	Description	Driver / Firmware version
07. September 2023	V3.04	LAN	Updated S-mount lensholder of BVS CA-MLC .	
01. August 2023	V3.03	LAN	Added glass filter wavelengths and relative quantum efficiency diagram : Glass filter .	
06. December 2022	V3.02	LAN	Updated main page .	
22. August 2022	V3.01	LAN	Removed "L" option of the digital I/Os (BVS CA-MLC) .	
03. May 2021	V3.00	LAN	Corrected Symbols and Conventions.	
03. February 2021	V2.01	LAN	Added BVS CA-[MLC|IGC]-0012V / mvBlueFOX-[MLC|IGC]202v (1.2 Mpix [1280 x 960])	
13. January 2021	V2.00	LAN	Separated GUI tools.	

5 Symbols and Conventions

Note

This symbol indicates general notes.

5.1 Explanation of the warnings

Always observe the warnings in these instructions and the measures described to avoid hazards. The warnings used here contain various signal words and are structured as follows:

Attention

SIGNAL WORD
"Type and source of the hazard"

Consequences if not complied with

→ Measures to avoid hazards.

The individual signal words mean:

Attention

Indicates a **danger** that can lead to **damage** or **destruction** of the product.

All due care and attention has been taken in preparing this manual. In view of our policy of continuous product improvement, however, we can accept no liability for completeness and correctness of the information contained in this manual. We make every effort to provide you with a flawless product.

In the context of the applicable statutory regulations, we shall accept no liability for direct damage, indirect damage or third-party damage resulting from the acquisition or operation of a Balluff/MATRIX VISION product. Our liability for intent and gross negligence is unaffected. In any case, the extend of our liability shall be limited to the purchase price.

6 Important Information

	We cannot and do not take any responsibility for the damage caused to you or to any other equipment connected to the device. Similarly, warranty will be void, if a damage is caused by not following the manual.
	Handle the device with care. Do not misuse the device. Avoid shaking, striking, etc. The device could be damaged by faulty handling or shortage.
	Use a soft cloth lightly moistened with a mild detergent solution when cleaning the camera.
	Never face the camera towards the sun. Whether the camera is in use or not, never aim at the sun or other extremely bright objects. Otherwise, blooming or smear may be caused.

	Please keep the camera closed or mount a lens on it to avoid the CCD or the CMOS from getting dusty.
	Clean the CCD/CMOS faceplate with care. Do not clean the CCD or the CMOS with strong or abrasive detergents. Use lens tissue or a cotton tipped applicator and ethanol.
	Never connect two USB cables to the device even if one is only connected to a PC.
	The camera is bus powered $<$ 2.5 W.
	The device meets IP40 standards.
	Using the single-board or board-level versions: • Handle with care and avoid damage of electrical components by electrostatic discharge (ESD): – Discharge body static (contact a grounded surface and maintain contact). – Avoid all plastic, vinyl, and styrofoam (except antistatic versions) around printed circuit boards. – Do not touch components on the printed circuit board with your hands or with conductive devices.

6.1 High-Speed USB design guidelines

If you want to make own High-Speed (HS) USB cables, please pay attention to following design guidelines:

- Route High-Speed (HS) USB signals with a minimum number of vias and sharp edges!
- Avoid stubs!
- Do not cut off power planes VCC or GND under the signal line.
- Do not route signals no closer than 20 $*$ h to the copper layer edge if possible (h means height over the copper layer).
- Route signal lines with 90 Ohm +- 15% differential impedance.
  - 7.5 mil printed circuit board track with 7.5 mil distance result in approx. 90 Ohm @ 110 um height over GND plane.
  - There are other rules when using double-ply printed circuit boards.
- Be sure that there is 20 mil minimum distance between High-Speed USB signal pair and other printed circuit board tracks (optimal signal quality).

6.2 European Union Declaration of Conformity statement

CE	The device complies with the provision of the following European Directives: • 2014/30/EU (EMC directive) • 2014/35/EU (LVD - low voltage directive) • For EN 61000-6-3:2007, BVS CA-IGC with digital I/O needs the Steward snap-on ferrite 28A0350-0B2 on I/O cable. • For EN 61000-6-3:2007, BVS CA-MLC with digital I/O needs the Würth Elektronik snap-on ferrite WE74271142 on I/O cable and copper foil on USB.
	Balluff corresponds to the EU guideline WEEE 2002/96/EG on waste electrical and electronic equipment and is registered under WEEE-Reg.-No. DE 25244305.

A brand of Balluff
MATRIX VISION

EG-Konformitätserklärung
Declaration of conformity

Der Hersteller
The Manufacturer

MATRIX VISION GmbH
Talstraße 16
71570 Oppenweiler
Germany

erklärt hiermit, dass sein Produkt
herewith declares, that his product

Typbezeichnung: **mvBlueFOX**
Type: ***mvBlueFOX***

mit den Bestimmungen folgender Europäischer Richtlinien übereinstimmt:
complies with the provisions of the following European Directives:

2014/30/EU EMC
2011/65/EU RoHS II

auf Grundlage folgender harmonisierter Normen:
based on the following harmonized standards:

Elektromagnetische Verträglichkeit (EMV) / *Electromagnetic compatibility (EMC)*
Störaussendung / *Interference emission*
EN 61000-6-3:2007 + A1:2011

Störfestigkeit / *Interference immunity*
EN 61000-6-2:2019

Beschränkung gefährlicher Stoffe / Restriction of certain Hazardous Substances
EN IEC 63000:2018

Oppenweiler, 10.06.2021

Uwe Hagmaier
Vice President R & D

Bitte beachten: Um die Einhaltung obiger Normen sicherzustellen, ist die Verwendung hochwertiger, vollständig geschirmter Anschlusskabel unbedingt erforderlich. *Please note: In order to fulfil the above standards, the use of high-quality, shielded connecting cables is required.*

A brand of Balluff

MATRIX VISION

EG-Konformitätserklärung
Declaration of conformity

Der Hersteller
The Manufacturer

MATRIX VISION GmbH
Talstraße 16
71570 Oppenweiler
Germany

erklärt hiermit, dass sein Produkt
herewith declares, that his product

Typbezeichnung: **mvBlueFOX-IGC**
Type: ***mvBlueFOX-IGC***

mit den Bestimmungen folgender Europäischer Richtlinien übereinstimmt:
complies with the provisions of the following European Directives:

2014/30/EU EMC
2011/65/EU RoHS II

auf Grundlage folgender harmonisierter Normen:
based on the following harmonized standards:

Elektromagnetische Verträglichkeit (EMV) / *Electromagnetic compatibility (EMC)*
Störaussendung / *Interference emission*
EN 61000-6-3:2007 + A1:2011

Störfestigkeit / *Interference immunity*
EN 61000-6-2:2019

Beschränkung gefährlicher Stoffe / Restriction of certain Hazardous Substances
EN IEC 63000:2018

Oppenweiler, 10.06.2021

Uwe Hagmaier
Vice President R & D

Bitte beachten: Um die Einhaltung obiger Normen sicherzustellen, ist die Verwendung hochwertiger, vollständig geschirmter Anschlusskabel unbedingt erforderlich. *Please note: In order to fulfil the above standards, the use of high-quality, shielded connecting cables is required.*

A brand of Balluff

MATRIX VISION

EG-Konformitätserklärung
Declaration of conformity

Der Hersteller
The Manufacturer

MATRIX VISION GmbH
Talstraße 16
71570 Oppenweiler
Germany

erklärt hiermit, dass sein Produkt
herewith declares, that his product

Typbezeichnung: **mvBlueFOX-MLC**
Type: ***mvBlueFOX-MLC***

mit den Bestimmungen folgender Europäischer Richtlinien übereinstimmt:
complies with the provisions of the following European Directives:

2014/30/EU EMC
2011/65/EU RoHS II

auf Grundlage folgender harmonisierter Normen:
based on the following harmonized standards:

Elektromagnetische Verträglichkeit (EMV) / *Electromagnetic compatibility (EMC)*
Störaussendung / *Interference emission*
EN 61000-6-3:2007 + A1:2011

Störfestigkeit / *Interference immunity*
EN 61000-6-2:2019

Beschränkung gefährlicher Stoffe / Restriction of certain Hazardous Substances
EN IEC 63000:2018

Oppenweiler, 10.06.2021

Uwe Hagmaier
Vice President R & D

Bitte beachten: Um die Einhaltung obiger Normen sicherzustellen, ist die Verwendung eines geeigneten EMV-Gehäuses sowie hochwertiger, vollständig geschirmter Anschlusskabel unbedingt erforderlich.
Please note: In order to fulfill the above standards, the use of a suitable EMC-case, as well as the use of a high-quality shielded connecting cables is required.

6.3 Legal notice

6.3.1 For customers in the U.S.A.

FC	Class B

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a residential environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. However there is no guarantee that interferences will not occur in a particular installation. If the equipment does cause harmful interference to radio or television reception, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the distance between the equipment and the receiver.
- Use a different line outlet for the receiver.
- Consult a radio or TV technician for help.

You are cautioned that any changes or modifications not expressly approved in this manual could void your authority to operate this equipment. The shielded interface cable recommended in this manual must be used with this equipment in order to comply with the limits for a computing device pursuant to Subpart B of Part 15 of FCC Rules.

- To be compliant to FCC Class B, mvBlueFOX-IGC requires an I/O cable with an retrofittable ferrite to be used such as
  - Company: Steward Type: 28A0350-0B2

6.3.2 For customers in Canada

This apparatus complies with the Class B limits for radio noise emissions set out in the Radio Interference Regulations.

6.3.3 Pour utilisateurs au Canada

Cet appareil est conforme aux normes classe B pour bruits radioélectriques, spécifiées dans le Règlement sur le brouillage radioélectrique.

7 Introduction

The BVS CA-MLC / BVS CA-IGC / mvBlueFOX is a compact industrial CCD & CMOS camera solution for any PC with a Hi-Speed USB (USB 2.0) port. A superior image quality makes it suited for most applications. Integrated preprocessing like binning reduces the PC load to a minimum. The standard Hi-Speed USB interface guarantees an easy integration without any additional interface board. Various example application are available.

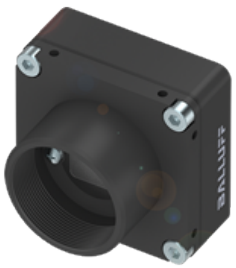


Figure 1: BVS CA-IGC

The camera is suitable for the following tasks:

- machine vision
- robotics
- surveillance
- microscopy
- medical imaging

With the name mvBlueFOX-M1xx, the industrial camera mvBlueFOX is also available as a single-board version.

7.1 Order code nomenclature

7.1.1 mvBlueFOX

The mvBlueFOX nomenclature scheme is as follows:

```
mvBlueFOX - A B - (1) (2) (3) (4)

  - A: Sensor model
       220: 0.3 Mpix, 640 x 480, 1/4", CCD
       220a: 0.3 Mpix, 640 x 480, 1/3", CCD
       200w: 0.4 Mpix, 752 x 480, 1/3", CMOS
       221: 0.8 Mpix, 1024 x 768, 1/3", CCD
       202a: 1.3 Mpix, 1280 x 1024, 1/2", CMOS
       223: 1.4 Mpix, 1360 x 1024, 1/2", CCD
       224: 1.9 Mpix, 1600 x 1200, 1/1.8", CCD
       205: 5.0 Mpix, 2592 x 1944, 1/2.5", CMOS

  - B: Sensor color
       G: Gray scale version
       C: Color version

  - (1): Lensholder
      1: C-mount with adjustable backfocus (standard)
      2: CS-mount with adjustable backfocus
      3: S-mount

  - (2): Filter
      1: IR-CUT (standard)
      2: Glass
      3: Daylight cut
      9: None
         More filters on request \n \n
  - (3): Case
      1: Color blue (standard)
      2: Color black, no logo, no label
      3: Color blue, no logo, no label
      9: None

  - (4): Misc
      1: None (standard)
```

7.1.2 mvBlueFOX-M

The mvBlueFOX-M nomenclature scheme is as follows:

```
mvBlueFOX-M A B - (1) (2) (3) (4)

  - A: Sensor model
       220: 0.3 Mpix, 640 x 480, 1/4", CCD
       220a: 0.3 Mpix, 640 x 480, 1/3", CCD
       200w: 0.4 Mpix, 752 x 480, 1/3", CMOS
       221: 0.8 Mpix, 1024 x 768, 1/3", CCD
       202a: 1.3 Mpix, 1280 x 1024, 1/2", CMOS
       223: 1.4 Mpix, 1360 x 1024, 1/2", CCD
       224: 1.9 Mpix, 1600 x 1200, 1/1.8", CCD
       205: 5.0 Mpix, 2592 x 1944, 1/2.5", CMOS

  - B: Sensor color
       G: Gray scale version
       C: Color version

  - (1): Lensholder
      1: No holder (standard)
      2: C-mount with adjustable backfocus
      3: CS-mount with adjustable backfocus
      4: S-mount #9031
      5: S-mount #9033

  - (2): Filter
      1: None (standard)
      2: IR-CUT
      3: Glass
      4: Daylight cut
         More filters on request \n \n
  - (3): Misc
      1: None (standard)

  - (4): Misc
      1: None (standard)
```

7.1.3 BVS CA-IGC

The BVS CA-IGC nomenclature scheme is as follows:

Sensor	Model name	MATRIX VISION model name
0.4 Mpix, 752 x 480, 1/3", CMOS	BVS CA-IGC-0004ZG/C	mvBlueFOX-IGC200wG/C
1.2 Mpix, 1280 x 960, 1/3", CMOS		mvBlueFOX-IGC202bG/C
1.2 Mpix, 1280 x 960, 1/3", CMOS		mvBlueFOX-IGC202dG/C
1.2 Mpix, 1280 x 960, 1/3", CMOS	BVS CA-IGC-0012VG/C	mvBlueFOX-IGC202vG/C
1.3 Mpix, 1280 x 1024, 1/2", CMOS		mvBlueFOX-IGC202aG/C
5.0 Mpix, 2592 x 1944, 1/2.5", CMOS	BVS CA-IGC-0050ZG/C	mvBlueFOX-IGC205G/C

	Series	Sensor	Color		Options		HW Variant		CUP/STD
Model name	BVS CA-↩ IGC-	xxxxx	G/C			-	see legend below	-	see legend below
MATRIX VISION model name	mvBlueFOX-IGC	xxx	G/C/GE	-	see legend below	-	see legend below	-	see legend below

Legend	Model name	MATRIX VISION model name

Options		
HW Variant	(1)(2)(3)(4)(5)(6) (1): Handling **1**: Standard handling (2): Lensholder **0**: None **2**: C-mount with adjustable backfocus **4**: CS-mount with adjustable backfocus **6**: CS-mount without adjustable backfocus (standard) **9**: C-mount without adjustable backfocus (CS-mount with add. 5 mm extension ring) (3): Filter **0**: None (standard) **1**: IR-CUT **2**: Glass **3**: Daylight cut More filters on request (4): Housing **1**: Color black (standard) (5): I/O **0**: None (standard) **2**: With I/O #08727 (6): Connector **0**: None (standard)	(1)(2)(3)(4) (1): Lensholder **1**: No holder **2**: C-mount with adjustable backfocus **3**: CS-mount with adjustable backfocus **4**: C-mount without adjustable backfocus (CS-mount with add. 5 mm extension ring) **5**: CS-mount without adjustable backfocus (standard) **6**: LENSHOLDER SH04F85 #16323 **7**: LENSHOLDER SH02M13V3 #10590 **8**: LENSHOLDER M12X0,5 22_16,2 #13759 (2): Filter **1**: IR-CUT **2**: Glass **3**: Daylight cut **9**: None (standard) More filters on request (3): Case **1**: Color black (standard) (4): I/O **1**: None (standard) **2**: With I/O #08727
CUP/STD	**001**	

7.1.4 BVS CA-MLC

The BVS CA-MLC nomenclature scheme is as follows:

Sensor	Model name	MATRIX VISION model name
0.4 Mpix, 752 x 480, 1/3", CMOS	BVS CA-MLC-0004ZG/C	mvBlueFOX-MLC200wG/C
1.2 Mpix, 1280 x 960, 1/3", CMOS		mvBlueFOX-MLC202bG/C
1.2 Mpix, 1280 x 960, 1/3", CMOS		mvBlueFOX-MLC202dG/C
1.2 Mpix, 1280 x 960, 1/3", CMOS	BVS CA-MLC-0012VG/C	mvBlueFOX-MLC202vG/C
1.3 Mpix, 1280 x 1024, 1/2", CMOS		mvBlueFOX-MLC202aG/C
5.0 Mpix, 2592 x 1944, 1/2.5", CMOS	BVS CA-MLC-0050ZG/C	mvBlueFOX-MLC205G/C

	Series	Sensor	Color		Options		HW Variant		CUP/STD
Model name	BVS CA-↩ MLC-	xxxxx	G/C			-	see legend below	-	see legend below
MATRIX VISION model name	mvBlueFOX-MLC	xxx	G/C/GE	-	see legend below	-	see legend below	-	see legend below

Legend	Model name	MATRIX VISION model name
Options		(C)(D)(E) (C): Mini USB **U:** with Mini USB (standard) **X:** without Mini USB (D): Digital I/Os **O:** 1x IN + 1x OUT opto-isolated (standard) **T:** 2x TTL IN + 2x TTL OUT (E): Connector **W:** board-to-wire (standard) **B:** board-to-board **A:** board-to-wire (angled connector)
HW Variant	(1)(2)(3)(4)(5)(6) (1): Handling **1**: Standard handling (2): Lensholder **0**: None (standard) **2**: C-mount with adjustable backfocus **4**: CS-mount with adjustable backfocus **6**: CS-mount without adjustable backfocus **9**: C-mount without adjustable backfocus (CS-mount with add. 5 mm extension ring) **Z:** LENSHOLDER SH04F85 #16323 **U:** LENSHOLDER SH02M13V3 #10590 **V:** LENSHOLDER M12X0,5 22_16,2 #13759 (3): Filter **0**: None (standard) **1**: IR-CUT **2**: Glass **3**: Daylight cut More filters on request (4): Housing **0**: None (5): I/O **O:** 1x IN + 1x OUT opto-isolated (standard) **T:** 2x TTL IN + 2x TTL OUT (6): Connector **U:** Mini USB & board-to-wire (standard) **V:** Mini USB & board-to-board **X:** without Mini USB & board-to-wire **Y:** without Mini USB & board-to-board	(1)(2)(3)(4) (1): Lensholder **1**: No holder (standard) **2**: C-mount with adjustable backfocus **3**: CS-mount with adjustable backfocus **4**: C-mount without adjustable backfocus (CS-mount with add. 5 mm extension ring) **5**: CS-mount without adjustable backfocus **6**: LENSHOLDER SH04F85 #16323 **7**: LENSHOLDER SH02M13V3 #10590 **8**: LENSHOLDER M12X0,5 22_16,2 #13759 (2): Filter **1**: None (standard) **2**: IR-CUT **3**: Glass **4**: Daylight cut More filters on request (3): Misc **1**: None (standard) (4): Misc **1**: None (standard)
CUP/STD	**001**	

7.1.5 Ordering code samples

Model name	MATRIX VISION model name	Description
BVS CA-MLC-0004ZC-1600OX-001	mvBlueFOX-MLC200wC-XOW-5111	52 x 480, CMOS 1/3", color, single-board, without Mini-USB, 1x IN + 1x OUT opto-isolated, board-to-wire, CS-mount (w/o backfocus adjustment)

7.2 What's inside and accessories

Due to the varying fields of application the mvBlueFOX is shipped without accessories. The package contents:

- **mvBlueFOX or BVS CA-MLC / BVS CA-IGC**
- **instruction leaflet**

For the first use of the camera we recommend the following accessories to get the camera up and running:

- **A USB 2.0 cable**

Attention

"Reduced image quality"

According to the customer and if the BVS CA-MLC is shipped without lensholder, the BVS CA-MLC will be shipped with a protective foil on the sensor which will affect the image quality.

→ Before usage, please remove this foil!

Accessories for the mvBlueFOX:

Part code	Description
ADAPTER CS-MOUNT	Lens fixing for mvBlueFOX to match with CS-mount lenses
KS-USB2-AA-EXT 05.0	USB 2.0 extension, activeUSB2 A plug to USB2 A jack, length 5m
KS-USB2-AB 03.0 TR	USB 2.0 cable A-B, transparent, Profi Line. Length 3m
KS-USB2-B4ST 02.0	USB 2.0 cable for mvBlueFOX, Binder 4pol to USB2-A. Length: 2m
KS-USB2-B4ST 03.0	USB 2.0 cable for mvBlueFOX, Binder 4pol to USB2-A. Length: 3m
KS-USB2-B4ST 05.0	USB 2.0 cable for mvBlueFOX, Binder 4pol to USB2-A. Length: 5m
KS-USB2-PHR4 01.5	USB connector cable for mvBlueFOX-M1xx. Length: 1.5m
KS-PHR12 500	Cable for mvBlueFOX-M1xx dig. I/O, 12-pin. Length: 500mm 1..4 brown 5..8 gray 9 red 10 black 11 yellow 12 black
KS-MLC-IO-TTL 00.5	BVS CA-MLC board-to-board TTL IO cable for master-slave synchronization (Molex plug to Molex plug), Length: 0.5m
KS-MLC-IO-W	BVS CA-MLC board-to-wire I/O data cable, Molex 0510211200 with crimp terminal 50058. Length: up to 1m

KS-MLC-USB2-IO-W	BVS CA-MLC board-to-wire I/O data and USB 2.0 cable, Molex 0510211200 with crimp terminal 50058 to USB2-A. Length: up to 1m
MV-Lensholder BFM-C	C-mount lensholder for mvBlueFOX-M,incl. IR-Cut filter
MV-Lensholder BFM-S 9031	S-mount lensholder M12 x 0,5 type MS-9031 for mvBlueFOX-M102
MV-Lensholder BFM-S 9033	S-mount lensholder M12 x 0,5 type: MS-9033 for mvBlueFOX-M102
MV-LENSHOLDER SH02M13	S-mount lensholder M12 x 0.5, height 13mm for BVS CA-MLC
MV-LENSHOLDER SH01F08	S-mount lensholder M12 x 0.5, height 8mm for BVS CA-MLC
ADAPTER S-C AD01S	Adapter for S-mount lens (M12x0,5) to C-mount, high penetration depth for BVS CA-IGC
ADAPTER S-C AD02F	Adapter for S-mount lens (M12x0,5) to CS-mount, penetration depth: 5.5mm, outside diameters: 31mm for BVS CA-IGC
MV-Tripod Adapter BF	Tripod adapter for mvBlueFOX

8 Quickstart

8.1 System Requirements

The device is an industrial camera which requires a reliable USB connection to the host system.

For this reason the following components are recommended:

Component	Recommendation
Processor	Preferably multi core Intel or ARM CPUs
RAM	4 GB in 32-bit OS; 8 GB in 64-bit OS
Mainboard	USB connectors

There is a huge variety of ARM based devices available on the market. Some suitable platforms have been tested by Balluff and a summary of the results of this test can be found here: Appendix C. Tested ARM platforms

Please ask your system vendor for further advice and consult our technical documentation.

8.1.1 Supported Operating Systems

8.1.1.1 Windows The following versions of Windows are supported officially:

- Microsoft Windows 7 (32-bit, 64-bit)
- Microsoft Windows 8.1 (32-bit, 64-bit)
- Microsoft Windows 10 (32-bit, 64-bit)
- Microsoft Windows 11

Other Windows versions might work as well but will not be tested on a regular basis.

8.1.1.2 Linux Please check the 'Support' section of the Balluff website for the availability of the latest Linux driver package.

See also

https://www.balluff.com/en-de/downloads/software

Currently supported Kernel versions are:

- Kernel 3.5.x or greater

8.2 Installing the Impact Acquire driver

8.2.1 Windows

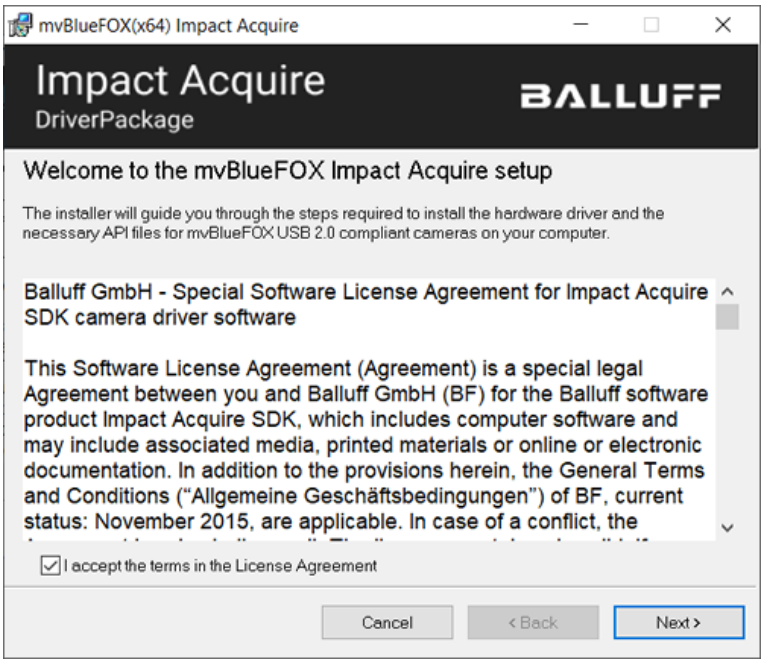
Note

Before connecting the mvBlueFOX, please install the software and driver first!

All necessary drivers are available from the Balluff website: https://www.balluff.com/en-de/downloads/softwar

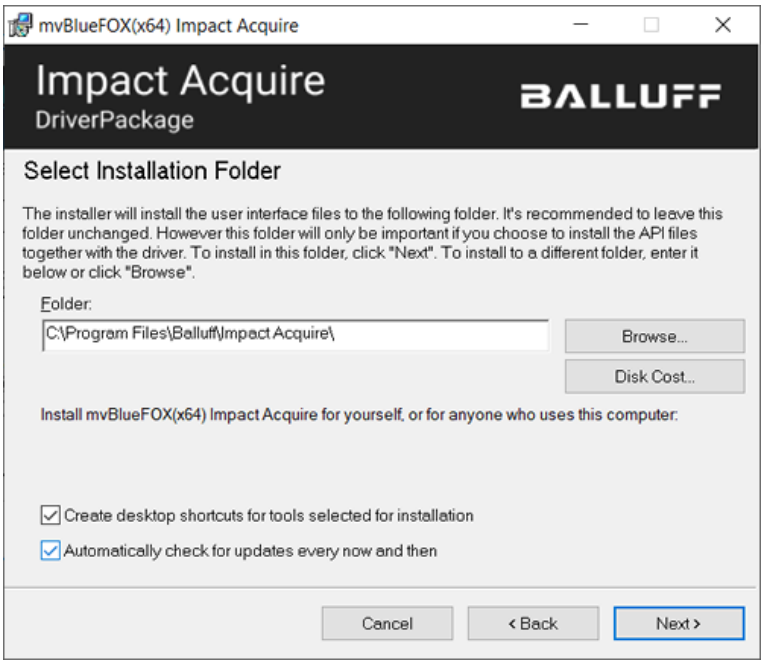
Starting the installer application

- **mvBlueFOX-x86-3.0.1.exe** (for 32-bit systems) or
- **mvBlueFOX-x86_64-3.0.1.exe** (for 64-bit systems): will display the following dialog:



mvBlueFOX installer - Start window

- Now, follow the instructions of the installation program and adjust the settings to your needs:

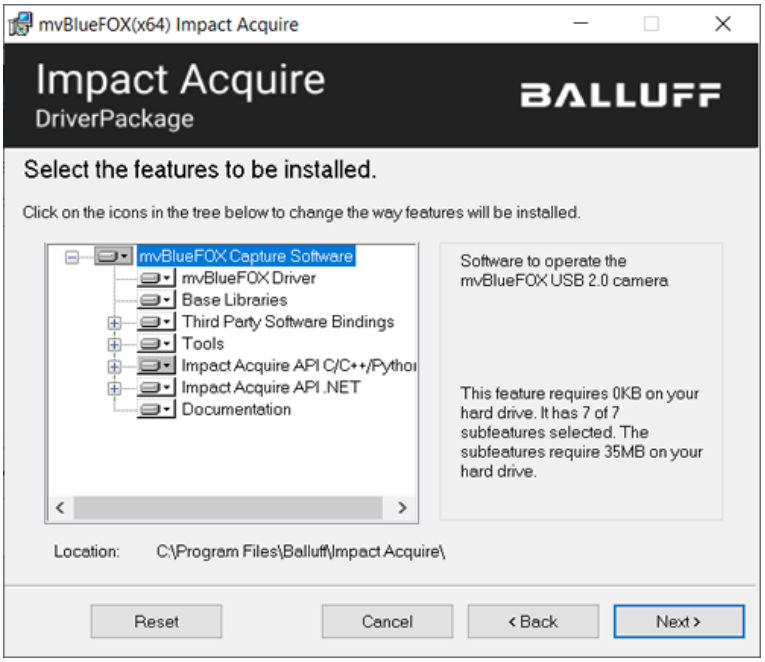


mvBlueFOX installer - Select folder

Since

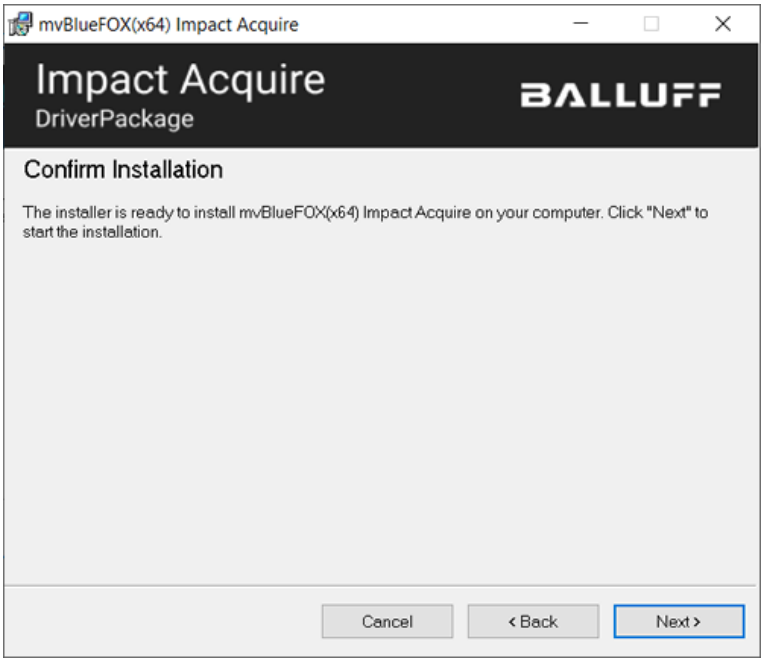
Version 2.25.0 of this driver package

ImpactControlCenter is able to check the availability of new driver versions weekly. Deactivate the check box if ImpactControlCenter should not check for updates. You can activate this again in ImpactControlCenter via the help menu.



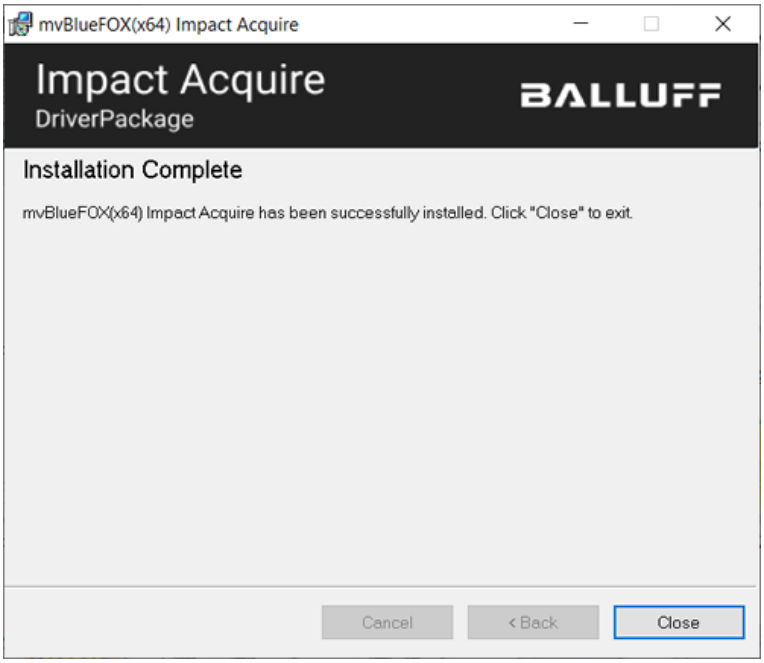
mvBlueFOX installer - Select features

- After confirmation, the installation will start and copy files and install device drivers.



mvBlueFOX installer - Confirm installation

- The installation is finished now you can close the window.



mvBlueFOX installer - Finished installation

You will find all tools like

- ImpactControlCenter and
- DeviceConfigure

either as shortcuts on the desktop or in the Windows start menu under `Balluff -> Impact Acquire`.

Afterwards, you can use DeviceConfigure to update the firmware if needed. The latest firmware image is available on the web - please check for updates. The current firmware version can be read out using ImpactControlCenter.

8.2.2 Linux

Additional packages will be needed to use all features of Impact Acquire.

Compiler etc. for building applications:

- build-essential (meta package)
- gcc 5.5.0 environment or newer

wxWidget release 3.0 or 3.2 packages, e.g.

- libwxbase3.0-0v5
- libwxbase3.0-dev
- libwxgtk3.0-gtk3-0v5
- libwxgtk3.0-gtk3-dev
- libwxgtk-webview3.0-gtk3-0v5
- libwxgtk-webview3.0-gtk3-dev
- wx3.0-headers
- libgtk2.0-dev

Note

The names of the packages mentioned above are Debian / Ubuntu specific. Other distributions (e.g. SuSE, Arch, Redhat, ...) will use different names.

The installation script will ask if the packages should be downloaded during the installation process. If some of the packages are not installed some features might not be available. If the e.g. wxWidgets related packages are missing on the target system then all GUI application coming as part of the Impact Acquire installation won't be available.

Note

If you are going to install the Impact Acquire package on an ARM device, please read this section first.

To use a (camera) device in Linux (capture images from it and change its settings), a driver is needed, consisting of several libraries and several configuration files. These files are required during runtime.

To develop applications that can use the device an API is needed, containing header files, makefiles, samples, and a few libraries.

Both file collections are distributed in a single package which is available in the `Support` section of the Balluff website. In addition to that an installation script is provided which can be downloaded from the same location. Using this script makes installing the driver package a lot easier.

Note

The following table shows the supported platforms and the corresponding package and installation script name:

Architecture	Package	Installation Script
ARM64	mvBlueFOX ARM64_gnu 3.0.1.tgz	install_mvBlueFOX_ARM
ARMhf	mvBlueFOX ARMhf_gnueabi 3.0.1.tgz	install_mvBlueFOX_ARM
x86_64	mvBlueFOX x86_64_ABI2 3.0.1.tgz	install_mvBlueFOX

The following example explains the installation process for the x86_64 package. **The installation process for other packages will work almost identical except different names as mentioned in the previous table.**

- Please start a console and change into the directory where the installation script and the installation package are located e.g. `/home/username/Downloads` :

```
cd /home/username/Downloads
```

> **Note**
>
> If root permissions are needed, the script will ask for the permissions. **There is no need to call it with root permissions.**

- You might need to enable the execute flag with:

```
chmod a+x install_mvGenTL_Acquire.sh
```

- Run the install script:

```
./install_mvGenTL_Acquire.sh
```

During installation the script will ask, if it should build all tools and samples.

> **Note**
>
> The installation scripts is developed for Ubuntu/Debian, SUSE Linux and Red Hat Linux based distributions. On other distributions some features of the installation script may or may not work. Get in touch with us if you encounter any problems!

The installation script checks for package dependencies described above and installs them with the respective standard package manager (e.g. **apt-get**) if necessary. So an Internet connection is recommended.

> **Note**
>
> The installation script (`install_mvBlueFOX.sh`) and the archive (`mvBlueFOX` x86_64_ABI2 3.↩ 0.1.tgz) must reside in the same directory. Nothing is written to this directory during script execution, so no write access to the directory is needed in order to execute the script.

The script supports various arguments, which allow to customize the installation, the desired functionalities and the installation process itself. All arguments are optional:

Argument	Function
`-h` or `--help`	Display the help.
`-p` or `--path`	Define a custom installation directory.
`-u` or `--unattended`	Unattended installation with default settings. By using this parameter you explicitly accept the EULA.
`-m` or `--minimal`	Minimal installation. No tools or samples will be built, and no automatic configuration and/or optimizations will be done. By using this parameter you explicitly accept the EULA.

The target directory name specifies where to place the driver. If the directory does not yet exist, it will be created.

The path can be either absolute or relative; i.e. the name may but need not start with / .

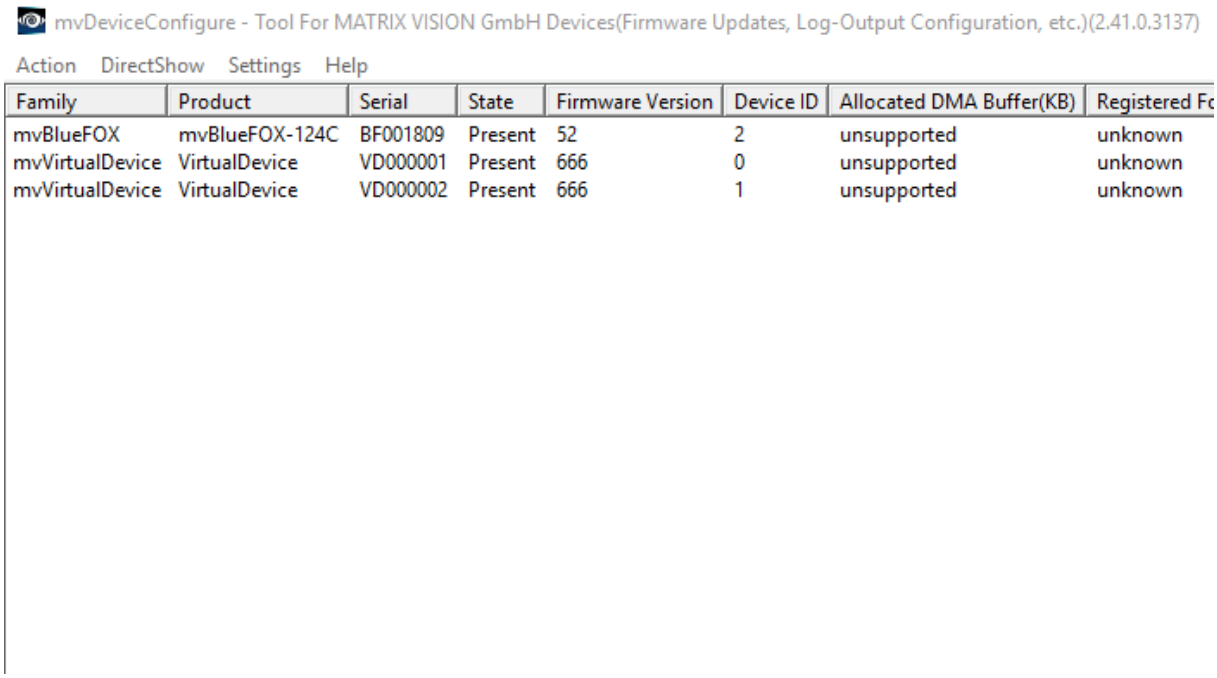
If no path is specified, the package will be installed to /**opt**/**Impact** Acquire.

8.3 Connecting The Camera

> **Note**
>
> Before connecting the camera, please install the software and driver first!

After the driver installation you have to connect the camera using a USB 2.0 cable.

You can check if the driver installation was successful by using DeviceConfigure. Supported device with an installed and running driver should be listed:



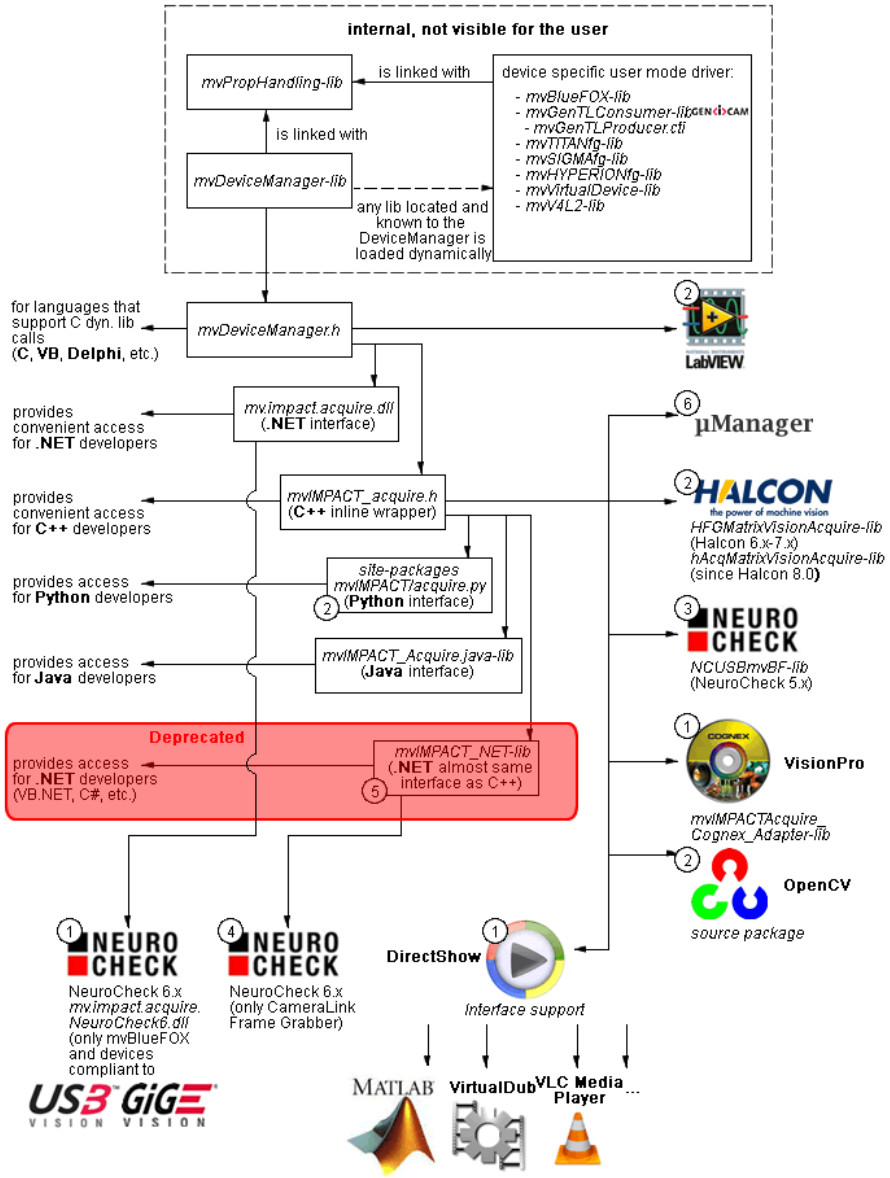
Connected camera

Afterwards, you can start ImpactControlCenter to configure the mvBlueFOX.

Since driver version 2.11.3, starting ImpactControlCenter the first time, the so called Quick Setup Wizard will be started. Read more about how to make optimal use of it in the Impact Acquire GUI manual: https://www.balluff.com/en-de/documentation-for-your-balluff-product/SDK_GUI_Tools/index.html

8.4 Driver concept

The driver supplied with the Balluff/MATRIX VISION product represents the port between the programmer and the hardware. The driver concept of Balluff provides a standardized programming interface to all image processing products made by Balluff GmbH.
The advantage of this concept for the programmer is that a developed application runs without the need for any major modifications to the various image processing products made by Balluff GmbH. You can also incorporate new driver versions, which are available for download free of charge on our website:
https://www.balluff.com.
The following diagram shows a schematic structure of the driver concept:



Driver concept

- **1** Part of any Impact Acquire driver installation package (Windows).
- **2** Separately available for 32 bit and 64 bit. Requires at least one installed driver package.
- **3** See 2, but requires an installed version of the mvBlueFOX driver.
- **4** Part of the NeuroCheck installer but requires at least one installed frame grabber driver.
- **5** Part of the mvIMPACT SDK installation. However, new designs should use the .NET libs that are now part of Impact Acquire (`mv.impact.acquire.dll`). The namespace `mv.impact.acquire` of `mv.impact.acquire.dll` provides a more natural and more efficient access to the same features as contained in the **deprecated** namespace `mvIMPACT_NET.acquire` of `mvIMPACT_NET.dll`, which is why the latter one should only be used for backward compatibility but **NOT** when developing a new application.
- **6** Part of Micro-Manager.

8.4.1 NeuroCheck Support

A couple of devices are supported by NeuroCheck. However between NeuroCheck 5.x and NeuroCheck 6.x there has been a breaking change in the internal interfaces. Therefore also the list of supported devices differs from one version to another and some additional libraries might be required.

For NeuroCheck 5.x the following devices are supported:

Device	Additional software needed
mvTITAN-G1	mvSDK driver for mvTITAN/mvGAMMA devices
mvTITAN-CL	mvSDK driver for mvTITAN/mvGAMMA devices
mvGAMMA-CL	mvSDK driver for mvTITAN/mvGAMMA devices
mvBlueFOX	Impact Acquire driver for mvBlueFOX devices, `NCUSBmvBF.dll`

For NeuroCheck 6.0 the following devices are supported:

Device	Additional software needed
mvTITAN-G1	Impact Acquire driver for mvTITAN/mvGAMMA devices
mvTITAN-CL	Impact Acquire driver for mvTITAN/mvGAMMA devices
mvGAMMA-CL	Impact Acquire driver for mvTITAN/mvGAMMA devices
mvHYPERION-CLb	Impact Acquire driver for mvHYPERION devices
Every other Impact Acquire compliant device	Impact Acquire driver for the corresponding device family, `mv.impact.acquire.NeuroCheck6.dll` (comes with the driver package, but the driver package must be installed **AFTER** installing NeuroCheck 6

For NeuroCheck 6.1 the following devices are supported:

Device	Additional software needed
mvTITAN-G1	Impact Acquire driver for mvTITAN/mvGAMMA devices
mvTITAN-CL	Impact Acquire driver for mvTITAN/mvGAMMA devices
mvGAMMA-CL	Impact Acquire driver for mvTITAN/mvGAMMA devices
mvHYPERION-CLb	Impact Acquire driver for mvHYPERION devices
Every other Impact Acquire compliant device	Impact Acquire driver for the corresponding device family, `mv.impact.acquire.NeuroCheck6_1.dll` (comes with the driver package, but the driver package must be installed **AFTER** installing NeuroCheck 6.1

8.4.2 VisionPro Support

Every *Impact Acquire* driver package on Windows comes with an adapter to VisionPro from Cognex. The installation order does not matter. After the driver package and VisionPro has been installed, the next time VisionPro is started it will allow selecting the *Impact Acquire* device. No additional steps are needed.

Balluff/MATRIX VISION devices that also comply with the GigE Vision or USB3 Vision standard don't need any software at all, but can also use VisionPro's built-in GigE Vision or USB3 Vision support.

8.4.3 HALCON Support

HALCON comes with built-in support for *Impact Acquire* compliant devices, so once a device driver has been installed for the *Impact Acquire* device, it can also be operated from a HALCON environment using the corresponding acquisition interface. No additional steps are needed.

Balluff/MATRIX VISION devices that also comply with the GigE Vision standard don't need any software at all, but can also use HALCON's built-in GigE Vision support.

As some *Impact Acquire* device driver packages also come with a GenTL compliant interface, these can also be operated through HALCON's built-in GenTL acquisition interface.

8.4.4 LabVIEW Support

Every *Impact Acquire* compliant device can be operated under LabVIEW through an additional set of VIs which is shipped by Balluff as a separate installation (`mvLabVIEW Acquire`).

Balluff/MATRIX VISION devices that also comply with the GigE Vision or USB3 Vision standard don't need any additional software at all, but can also be operated through LabVIEW's GigE Vision or USB3 Vision driver packages.

8.4.5 DirectShow Support

Every *Impact Acquire* compliant device driver package comes with an interface to DirectShow. In order to be usable from a DirectShow compliant application, devices must first be registered for DirectShow support. How to this is explained here.

8.4.6 Micro-Manager Support

Every *Impact Acquire* compliant device can be operated under `https://micro-manager.org` when using Impact Acquire and at least *Micro-Manager* 1.4.23 build **AFTER** 15.12.2016. The adapter needed is part of the *Micro-Manager* release. Additional information can be found here: `https://micro-manager.org/wiki/MatrixVision`.

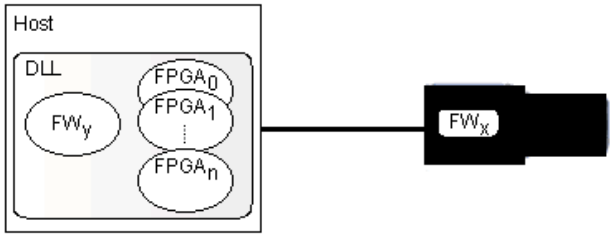
8.5 Relationship between driver, firmware and FPGA file

To operate the camera apart from the physical hardware itself 3 pieces of software are needed:

- a *firmware* running on the device (provides low-level functionality like allowing the device to act as a USB device, support for multiple power states etc.)
- an *FPGA file* loaded into the FPGA inside the device (provides access features to control the behaviour of the image sensor, the digital I/Os etc.)
- a *device driver* (this is the *mvBlueFOX.dll* on Windows® and the *libmvBlueFOX.so* on Linux) running on the host system (provides control over the device from an application running on the host system)

The physical camera has a firmware programmed into the device's non-volatile memory, thus allowing the device to act as a USB device by just connecting the device to a free USB port. So the firmware version that will be used when operating the device does **NOT** depend on the driver version that is used to communicate with the device.

On the contrary the FPGA file version that will be used will be downloaded in volatile memory (RAM) when accessing the device through the device driver thus the API. One or more FPGA files are a binary part of the device driver. This shall be illustrated by the following figure:



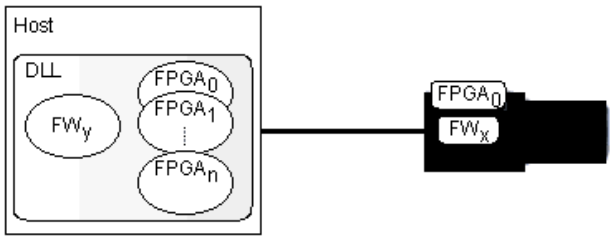
The firmware file is a binary part of the device driver

Note

As it can be seen in the image one or multiple firmware files are also a binary part of the device driver. However it is important to notice that this firmware file will **NOT** be used automatically but only when the user or an application explicitly updates the firmware on the device and will only become active after power-cycling the device. In Impact Acquire, every firmware starting from version 49 is available within a single driver library and can be selected for updating! DeviceConfigure however will always update the device firmware to the latest version. If you need to downgrade the firmware for any reason please get into contact with the Balluff support to get detailed instructions on how to do that.

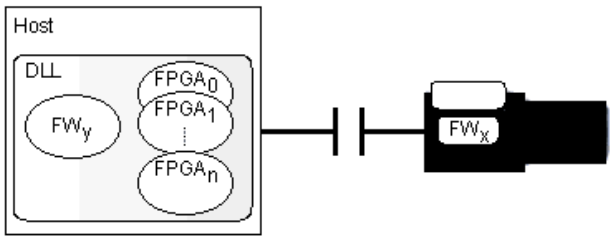
8.5.1 FPGA

Until the device gets initialized using the API no FPGA file is loaded in the FPGA on the device. Only by opening the device through the API the FPGA file gets downloaded and only then the device will be fully operational:



The FPGA file gets downloaded when the device will be opened through the API

As the FPGA file will be stored in RAM, disconnecting or closing the device will cause the FPGA file to be lost. The firmware however will remain:



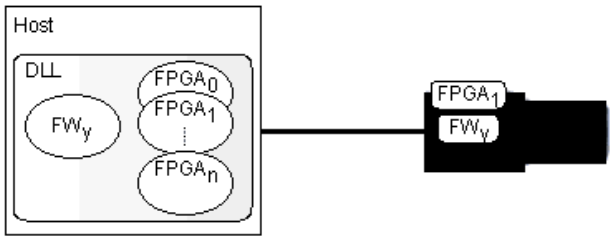
The FPGA file will be lost if the device is disconnected or closed

In case multiple FPGA files are available for a certain device the FPGA file that shall be downloaded can be selected by an application by changing the value of the property *Device/CustomFPGAFileSelector*. However the value of this property is only evaluated when the device is either initialized using the corresponding API function **OR** if a device has been unplugged or power-cycled while the driver connection remains open and the device is then plugged back in.

Note

There is just a limited set of devices that offer more than one FPGA file and these additional FPGA files serve very special purposes so in almost every situation the default FPGA file will be the one used by an application. Before using custom FPGA files, please check with Balluff about why and if this makes sense for your application.

So assuming the value of the property *Device/CustomFPGAFileSelector* has been modified while the device has been unplugged, a different FPGA file will be downloaded once the device is plugged back into the host system:



A different FPGA file can be downloaded

8.5.2 Firmware

Only during a firmware update the firmware file that is a binary part of the device driver will be downloaded permanently into the device's non-volatile memory.

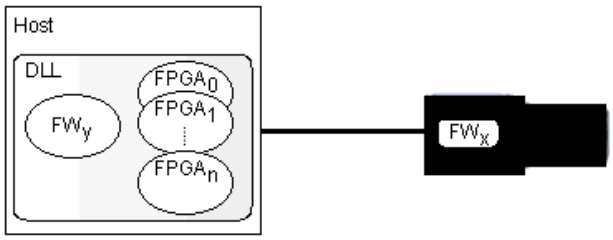
Attention

"Wrong firmware"

Until version 2.27.0 of the driver package for this device family, each device driver just contained one specific firmware version thus once a device's firmware has been updated using a specific device driver the only way to change the firmware version will be using another device driver version for upgrading/downgrading the firmware again. In Impact Acquire, every firmware starting from version 49 is available within a single driver library and can be selected for updating! DeviceConfigure however will always update the device firmware to the latest version. If you need to downgrade the firmware for any reason please get into contact with the Balluff support to get detailed instructions on how to do that.

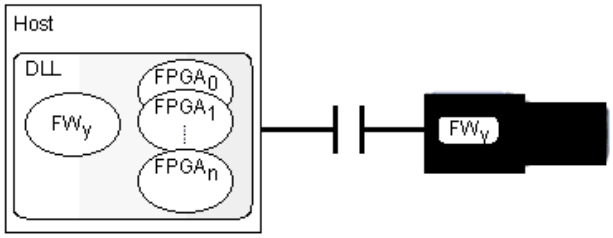
→ In order to select the appropriate firmware version for the device appropriate tools such as DeviceConfigure should be used.

So assume a device with a certain firmware version is connected to a host system:



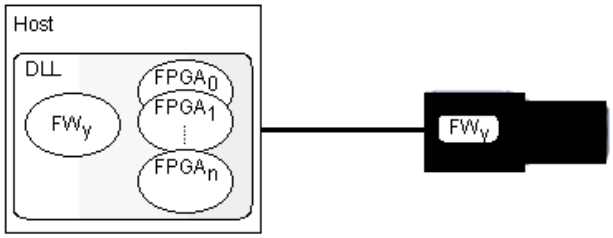
A certain firmware version is connected to a host system

During an explicit firmware update, the firmware file from inside the driver will be downloaded onto the device. In order to become active the device must be power-cycled:



Firmware file will be downloaded during an firmware update...

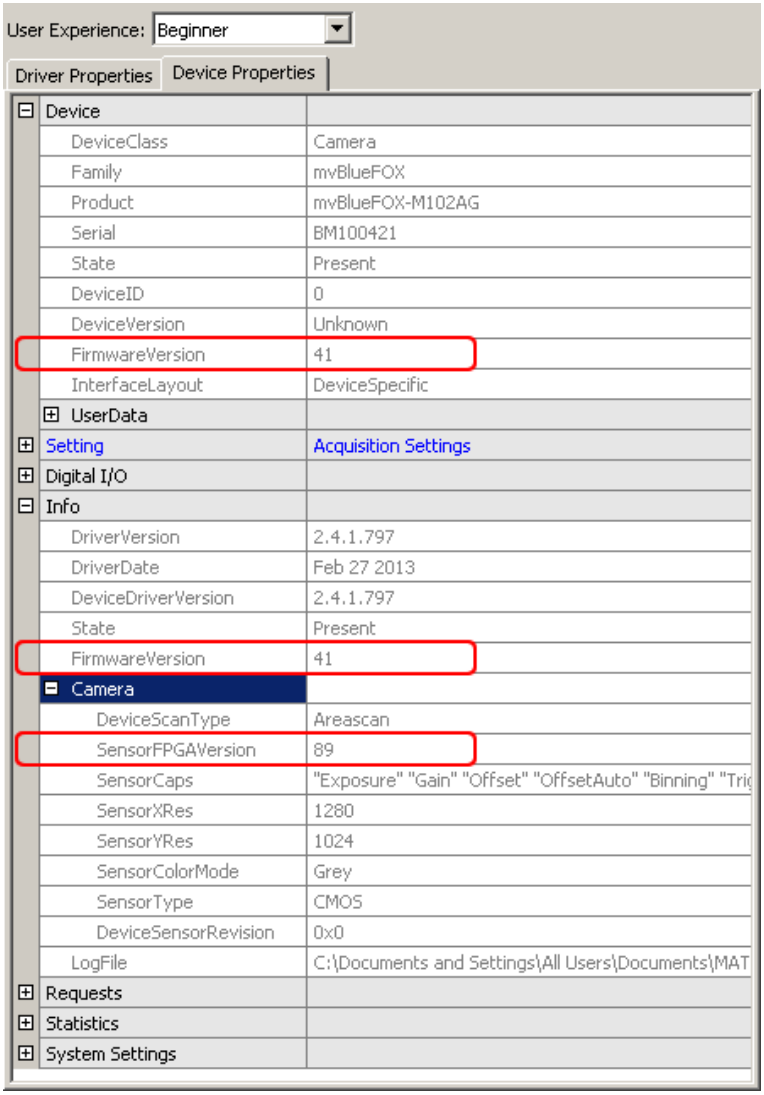
When then re-attaching the device to the host system, the new firmware version will become active:



... after power-cycling the device it will be active

- The current firmware version of the device can be obtained either by using one of the applications which are part of the SDK such as DeviceConfigure or by reading the value of the property *Device/FirmwareVersion* or *Info/FirmwareVersion* using the API
- The current FPGA file version used by the device can be obtained by reading the value of the property *Info/Camera/SensorFPGAVersion*

Using ImpactControlCenter the same information is available as indicated by the following figure:



ImpactControlCenter - FPGA and Firmware version numbers

8.6 About Settings

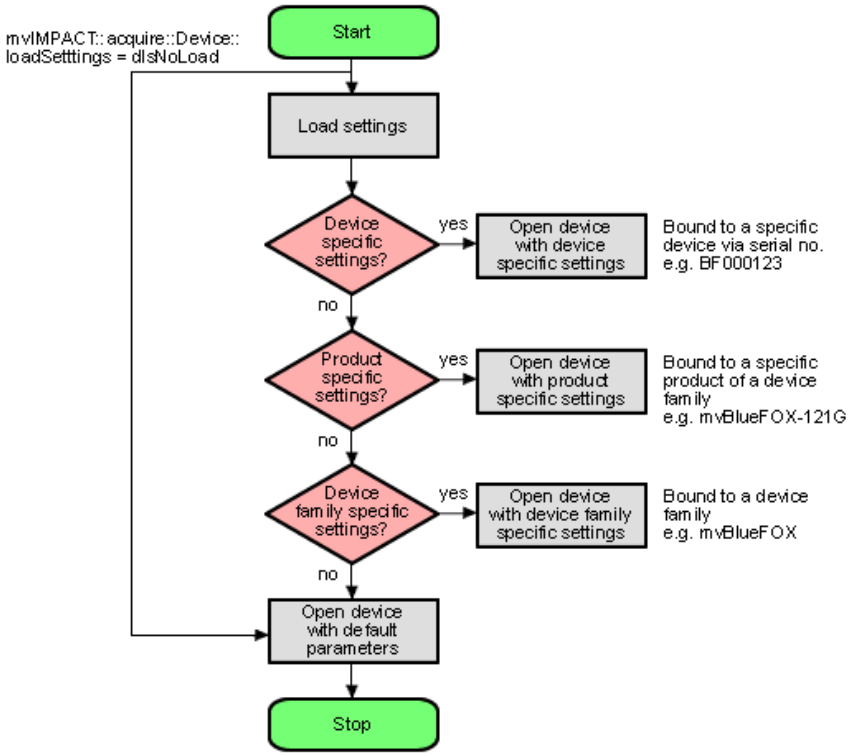
A setting contains all parameters that are needed to configure the device to a state it was in when the setting was created. Every image can be captured with a completely different set of parameters. In almost every case, these parameters are accessible via a property offered by the device driver. A setting e.g. might contain

- The gain to be applied to the analog to digital conversion process for analog video sources or
- The AOI to be captured from the incoming image data.

So for the user a setting is the one and only place where all the necessary modifications can be applied to achieve the desired data acquisition mode. There is however an important difference in behaviour between different interface layouts. See **"Impact Acquire SDK GUI Applications"** chapter **"ImpactControlCenter ->** **Device Configuration ->** **General Device Configuration ->** **Changing The Interface Layout To GenICam Or DeviceSpecific"** to find out how to modify the interface layout or check in the API documentation for the **interfaceLayout** property of the class **Device**.

- When working with the **DeviceSpecific** interface layout, each frame will be captured with the settings as present when requesting the image. Every parameter can be modified at any time. When requesting another image the settings valid at that moment will be used to fill this buffer with data
- For the **GenICam** interface layout all device properties modified during a continuous acquisition will be applied at once so might affect this or the next image transmitted by the device. Depending on various parameters (the number of buffer already captured but not collected by the application, the way the device internally operates(e.g. has already captured a couple of images that await transmission), etc.) this will have impact on that captured images somewhere in the near future thus when a precise moment to change settings is needed, continuous acquisition must be stopped and then restarted after modifying the features. Certain features (typically those affecting the buffer layout/size) cannot be changed while a continuous acquisition is running in GenICam interface layout anyway.

Now, whenever a device is opened, the driver will execute following procedure:



ImpactControlCenter - Device setting start procedure

- Please note that each setting location step in the figure from above internally contains two search steps. First the framework will try to locate a setting with user scope and if this can't be located, the same setting will be searched with global (system-wide) scope. On Windows this e.g. will access either the `HKEY_CURRENT_USER` or (in the second step) the `HKEY_LOCAL_MACHINE` branch in the Registry.
- Whenever storing a product specific setting, the device specific setting of the device used for storing will be deleted (if existing). E.g. you have a device "VD000001" which belongs to the product group "VirtualDevice" with a setting exclusively for "VD000001". As soon as you store a product specific setting **using THIS device**, the (device specific) setting for "VD000001" will be deleted. Otherwise a product specific setting would never be loaded as a device specific setting will always be found first. Storing a product specific setting with a different device belonging to the same family however will **NOT** delete device specific settings for other devices.
- The very same thing will also happen when opening a device from any other application! ImpactControlCenter does not behave in a special way but only acts as an arbitrary user application.

- Whenever storing a device family specific setting, the device specific or product specific setting of the device used for storing will be deleted (if existing). See above to find out why.
- On Windows the driver will **not** look for a matching XML file during start-up automatically as the native storage location for settings is the Windows Registry. This must be loaded explicitly by the user by using the appropriate API function offered by the SDK. However, under **Linux** XML files are the only setting formats understood by the driver framework thus here the driver will also look for them at start-up. The device specific setting will be an XML file with the serial number of the device as the file name, the product specific setting will be an XML file with the product string as the filename, the device family specific setting will be an XML file with the device family name as the file name. All other XML files containing settings will be ignored!
- Restoring of settings previously stored works in a similar way. After a device has been opened the settings will be loaded automatically as described above.
- A detailed description of the individual properties offered by a device will **not** be provided here but can be found in the C++ API reference, where descriptions for all properties relevant for the user (grouped together in classes sorted by topic) can be found. As ImpactControlCenter doesn't introduce new functionality but simply evaluates the list of features offered by the device driver and lists them any modification made using the GUI controls just calls the underlying function needed to write to the selected component. ImpactControlCenter also doesn't know about the type of component or e.g. the list of allowed values for a property. This again is information delivered by the driver and therefore can be queried by the user as well without the need to have special *inside information*. One version of the tool will always be delivered in source so it can be used as a reference to find out how to get the desired information from the device driver.

8.7 Optimizing USB Performance

Note

This section is only relevant for applications working with USB3 Vision™ or Balluff USB 2.0 devices!

8.7.1 Checklist for Windows

8.7.1.1 Host Controller Driver Also the USB host controller manufacturers provide driver updates for their cards/chips every now and then. Using the latest drivers is always recommended and might improve the overall performance of the system dramatically!

8.7.2 Checklist for Linux

8.7.2.1 udev rules Most Linux system nowadays use the **udev** device manager, which is responsible for dynamically managing the `/dev` tree. In order to be able to use the Balluff mvBlueFOX3 (BVS CA-SF) **"USB3 Vision"** camera as non-root user, a special set of rules has to be handed to the udev device manager.

On older systems this could be done by directly editing the contents of a "`/etc/udev/rules`" file, however nowadays a "`/etc/udev/rules.d`" directory exists, which may contain several different files, each defining the behavior of a system device.

In the specific case of BVS CA-SF device or any **"USB3 Vision"** device actually, if the camera has been installed through the respective installation script `install_mvGenTL_Acquire.sh` , a suitable set of rules has been installed automatically. However if for some reason these rules have to be created manually or must be changed at later time it should be done like this:

1. Create a file in the "`/etc/udev/rules.d`" directory with name `52-U3V.rules` if this doesn't exist already. The content of the file should be something like this:

```
SUBSYSTEM!="usb|usb_device|plugdev", GOTO="u3v_rules_end"
ACTION!="add", GOTO="u3v_rules_end"

ATTRS{bDeviceClass}=="ef", ATTRS{bDeviceSubClass}=="02", ATTRS{bDeviceProtocol}=="01",
        ENV{ID_USB_INTERFACES}=="*:ef0500:*", MODE="0664", GROUP="plugdev"

LABEL="u3v_rules_end"
```

2. OPTIONAL: Create another file in the "`/etc/udev/rules.d`" directory with name `52-mvbf3.rules` . This step is only necessary if a BVS CA-SF in the "mvbootloader" state should be recognised by the system. This might happen if for any reason a camera has no valid firmware running e.g. due to a power failure during a firmware update. The content of the file should be something like this:

```
SUBSYSTEM!="usb|usb_device|plugdev", GOTO="mvbf_rules_end"
ACTION!="add", GOTO="mvbf_rules_end"

ATTRS{idVendor}=="164c", ATTRS{idProduct}=="5531", MODE="0664", GROUP="plugdev"

LABEL="mvbf_rules_end"
```

Note

The above `52-U3V.rules` file provides the necessary access privileges not only for BVS CA-SF cameras, but also for any **"USB3 Vision"-compliant** device of other vendors.

As soon as this file is into place, each time the camera is plugged to the system it acquires the set of rights that allows the user to use it without having root privileges.

8.7.2.2 Disabling The Auto-Suspend Mode Usually the Linux kernel suspends USB devices when they are not in use for a certain time. In some cases this might cause unsuspected behaviour of USB devices. To avoid this kind of issues it is a good idea to disable the USB autosupend mode.

```
sudo sh -c 'echo -1 > /sys/module/usbcore/parameters/autosuspend'
```

8.8 Using USB2 Cameras In A Docker Container

When developing machine vision applications using Docker containers, it might be required to access the cameras inside the container. With the Impact Acquire driver stack this can be achieved fairly easily and this chapter will demonstrate how to build a basic Docker container where the cameras can be used.

8.8.1 Host Preparation

8.8.1.1 Linux

8.8.1.2 Windows

8.8.1.2.1 Host system requirements

- Windows 11 64-bit: Home or Pro version 21H2 or higher, or Enterprise or Education version 21H2 or higher (Build 22000 or later)
- Windows 10 64-bit: Home or Pro 21H1 (build 19043) or higher, or Enterprise or Education 20H2 (build 19042) or higher
- WSL2 backend (For installation please follow: `Docker Window Install`)
- Impact Acquire driver package >= 2.48.0 recommended

8.8.1.2.2 Attach the camera to the WSL2 Linux distro via USB/IP USB devices physically connected to the host system are not automatically accessible in the WSL2 Linux distro. They need to be first attached from the Windows host to the default Linux distro via USB/IP. Please follow `Connect USB devices WSL2` for implementation guidance.

8.8.1.2.3 Start udev manually udev is needed to identify attached USB devices and to access USB3 Vision™ devices as non-root users with the help of the udev-rules shipped by the Impact Acquire driver package. However, systemd, which starts udev automatically, is by default not supported in WSL2 distros. Besides, udev doesn't support containers. Since WSL2 distros themselves are technically containers, they are not supported by udev. In order for udev to work in WSL2 distros, the following lines need to be commented out in `/etc/init.d/udev` before manually starting udev, as shown below:

```
#if [ ! -w /sys ]; then
#    log_warning_msg "udev does not support containers, not started"
#    exit 0
#fi
```

Then start udev in the WSL2 default Linux distro:

```
$ sudo /etc/init.d/udev start
```

8.8.2 Building A Docker Image

The following demo Dockerfile builds a basic Docker image based on a slim version of Debian, where the Impact Acquire driver package for the cameras and its sample programs are installed. This Dockerfile can be used in many ways:

- Use it directly to test your device in a Docker container.
- Use it as a base image for your device applications.
- Use it as an inspiration for building your own Dockerfile.

Before building the Dockerfile, please download the required Impact Acquire driver installation files from Balluff website (https://www.balluff.com/en-de/downloads/software) (user login is required):

- The installation script: *install_mvBlueFOX.sh*
- The installation package: *mvBlueFOX-x86_64_ABI2-*.tgz* (* should be replaced by the version number)

Create a directory called *Impact* Acquire (as used in this demo Dockerfile) and move both installation files into this directory. In this example, both files are downloaded into the *Downloads* directory and the *Impact* Acquire directory is created inside the *Downloads:*

- `$ cd ~/Downloads`
- `$ mkdir Impact_Acquire`
- `$ mv install_mvBlueFOX.sh mvBlueFOX-x86_64_ABI2-*.tgz Impact_Acquire/`

Make the installation script *install_mvBlueFOX.sh* executable:

- `$ cd Impact_Acquire`
- `$ chmod a+x install_mvBlueFOX.sh`

Navigate back into the directory where *product_name* resides (e.g. *Downloads*) and create your Dockerfile:

- `$ cd ~/Downloads`
- `$ touch Dockerfile`

Create the content of your Dockerfile. Our demo Dockerfile looks as follows:

```
# start with slim version of actual Debian
FROM debian:9-slim

ENV LC_ALL C
ENV DEBIAN_FRONTEND noninteractive

# entrypoint of Docker
CMD ["/bin/bash"]

# set environment variables
ENV TERM linux
ENV MVIMPACT_ACQUIRE_DIR /opt/Impact_Acquire
ENV MVIMPACT_ACQUIRE_DATA_DIR /opt/Impact_Acquire/data
ENV container docker
```

```
# update packets and install minimal requirements
# after installation it will clean apt packet cache
RUN apt-get update && apt-get -y install build-essential && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

# move the directory Impact_Acquire with *.tgz and *.sh files to the container
COPY Impact_Acquire /var/lib/Impact_Acquire

# execute the setup script in an unattended mode
RUN cd /var/lib/Impact_Acquire && \
    ./install_mvBlueFOX.sh -u && \
    rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

At last, build a Docker image using this Dockerfile:

```
$ sudo docker build -t [image_name] .
```

Note

Please make sure to call *docker build* from within the directory where the Dockerfile resides. An Internet access is required for the *docker build*.

If built successfully, the newly built [image_name] will be listed when calling:

```
$ sudo docker images
```

8.8.3 Starting The Docker Container

Since the Docker container is isolated from the host system, it needs to be started with volume mount of */dev* and certain cgroup permissions for it to access the cameras. In order to avoid running the container in privileged mode, which is not secure, it can be started like this:

```
$ sudo docker run -ti -v /dev:/dev --device-cgroup-rule 'a 189:* rwm' [image_name] /bin/bash
```

Where:

- **-v /dev:/dev**: use volume mount to map the host */dev* directory to the container, so the container will be able to always detect devices also when they get unplugged and re-plugged at any time.
- **–device-cgroup-rule 'a 189:* rwm'**: with the *--device-cgroup-rule* flag, specific permission rules can be added to a device list that is allowed by the container's cgroup. Here in this example, *189* is the major number of the USB bus, * means all minor numbers, and *rwm* are respectively read, write, mknod accesses. By doing so, all USB devices will get read, write, mknod access. The camera can thus be enumerated successfully.

8.8.4 Validation

After starting the container, the correct operation of cameras can be validated by running one of the sample programs provided by the Impact Acquire (e.g. *SingleCapture*):

- `$ cd /opt/Impact_Acquire/apps/SingleCapture/x86_64`
- `$ ./SingleCapture` If the attached camera appears in the device list of the program's output, access to it in the container by using the Impact Acquire has been established. Now the camera can be used inside the Docker container for your machine vision applications.

9 Technical Data

9.1 Power supply

Symbol	Comment	Min	Typ	Max	Unit
$U_{USBPOWER\_IN}$	Power supply via USB	4.75	5	5.25	V
$I_{USBPOWER\_IN}$ (@ 5V / 40MHz)			280	500	mA
$I_{USBPOWER\_IN}$ (Power Off Mode - only with BVS CA-IGC / BVS CA-MLC)			66		mA

9.2 Standard version (mvBlueFOX-xxx)

9.2.1 Dimensions and connectors

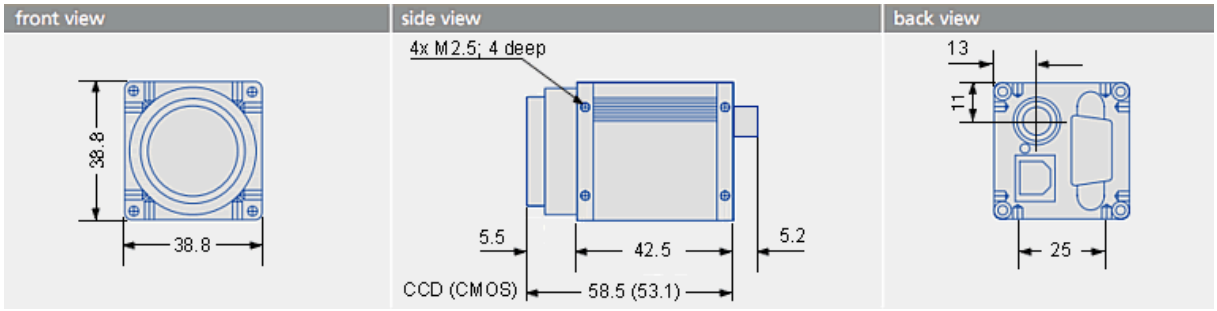


Figure 1: Connectors mvBlueFOX

mvBlueFOX	
Size without lens (w x h x l)	38.8 x 38.8 x 58.5 mm (CCD version) 38.8 x 38.8 x 53.1 mm (CMOS version)
General tolerance	DIN ISO 2768-1-m (middle)

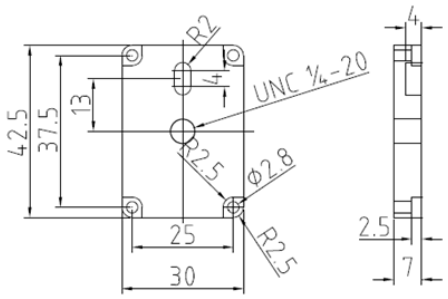


Figure 2: Dimensional drawing of tripod adapter

9.2.1.1 D-Sub 9-pin (male)

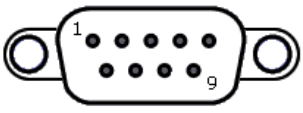


Figure 3: D-Sub 9-pin (male), digital I/O

Pin	Signal	Description
1	IN0-	Negative terminal of opto-isolated input [1]
2	OUT0-	Negative terminal of opto-isolated output (emitter of npn-phototransistor)
3	OUT1-	Negative terminal of opto-isolated output (emitter of npn-phototransistor)
4	IN1-	Negative terminal of opto-isolated input *
5	N/C	
6	IN0+	Positive terminal of opto-isolated input *
7	OUT0+	Positive terminal of opto-isolated output (collector of npn-phototransistor)
8	OUT1+	Positive terminal of opto-isolated output (collector of npn-phototransistor)
9	IN1+	Positive terminal of opto-isolated input *

[1] Voltage between + and - may be up to 26V, input current is 17mA.

9.2.1.1.1 Characteristics of the digital inputs

Open inputs will be read as a logic zero.

When the input voltage rises above the trigger level, the input will deliver a logic one.

Symbol	Comment	Min.	Std.	Max.	Unit
U_{IN_TTL}	High level input voltage TTL logic	3	5	6.5	V
	Low level input voltage TTL logic	-0.7		1	V
I_{IN_TTL}	Current TTL logic		8.↩5	12	mA
U_{IN_PLC}	High level input voltage PLC logic	12		24	V
	Low level input voltage PLC logic	-0.7		8	V
I_{IN_PLC}	Current PLC logic		17	25	mA

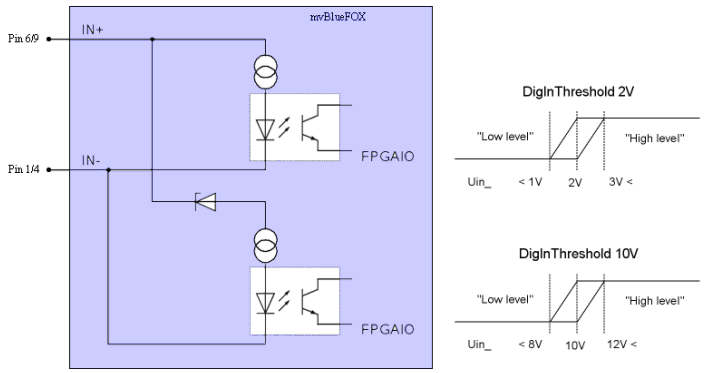


Figure 4: DigIn mvBlueFOX-xxx

In ImpactControlCenter you can change between

- **TTL** (`"DigitalInputThreshold = 2V"`) and
- **PLC** (`"DigitalInputThreshold = 10V"`)

input behavior of the digital inputs using the **DigitalInputThreshold** property in *"Digital I/O -> DigitalInput↩ Threshold"*:

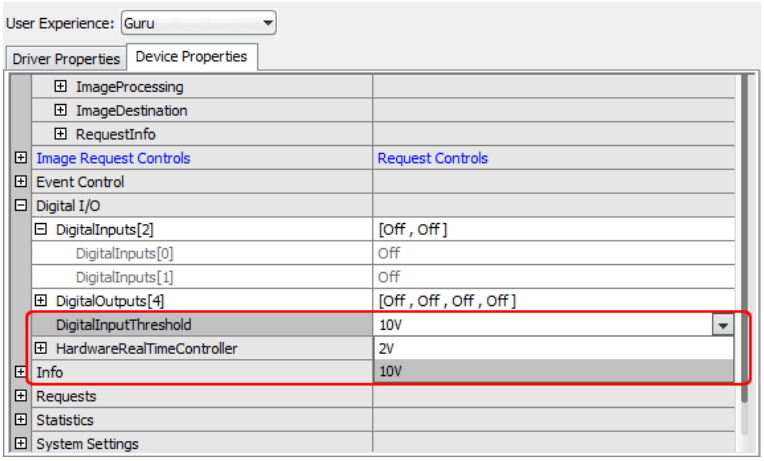


Figure 5: ImpactControlCenter - DigitalInputThreshold

	U_{min} [V]	U_{max} [V]	I_{min} [mA]	I_{max} [mA]
Output		30		100 (on state current)

9.2.1.1.2 Characteristics of the digital outputs

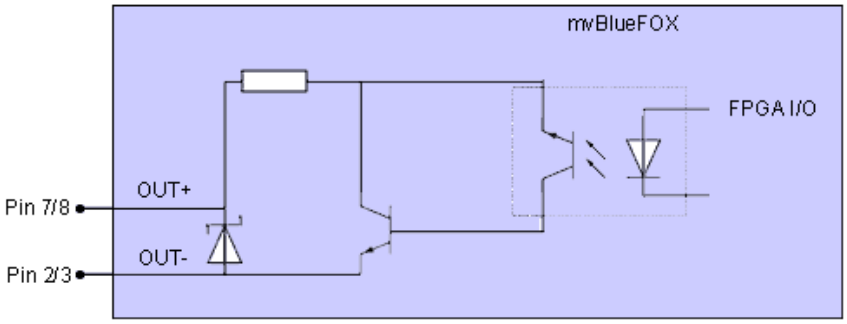


Figure 6: DigOut mvBlueFOX-xxx

9.2.1.1.3 Connecting flash to digital output You can connect a flash in series to the digital outputs as shown in the following figure, however, you should only use LEDs together with a current limiter:

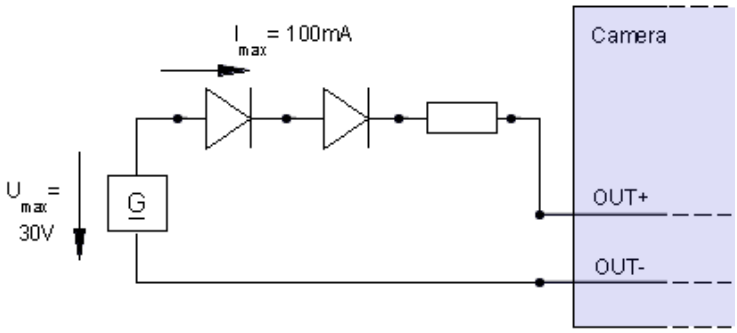


Figure 7: Connecting flash (LEDs) to DIG OUT

9.2.1.2 USB connector, type B (USB 2.0)

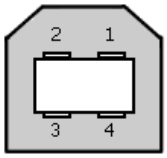


Figure 8: USB B connector (female)

Pin	Signal
1	USBPOWER_IN
2	D-
3	D+
4	GND
Shell	shield

Note

The mvBlueFOX is an USB device!

Attention

"Surge"

Using both USB ports at the same time can damage the device.

→ Do not connect both USB ports at the same time.

9.2.1.3 4-pin circular plug-in connector with lock (USB 2.0)

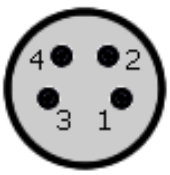


Figure 9: 4-pin circular plug-in connector (female)

Pin	Signal 'R' version	Signal 'U' version
1	USBPOWER_IN	Power out from USB
2	D+	not connected
3	GND	GND
4	D-	not connected

Manufacturer: Binder
Part number: 99-3390-282-04

Note

Differentiation between 'R' and 'U' version is obsolete. New mvBlueFOX versions have both connectors (circular connector and standard USB). The pin assignment corresponds to the description of 'R' version.

While mvBlueFOX is connected and powered via standard USB, it is possible to connect additional power via circular connector (**only power; the data lines must be disconnected!**). Only in this case, the power switch will change the power supply, if the current entry via standard USB is equal to or under the power supply of **"circular connector"**.

Attention

"Surge"

Using both USB ports at the same time can damage the device.

→ Do not connect both USB ports at the same time.

9.2.2 LED states

State	LED
Camera is not connected or defect	LED off
Camera is connected and active	Green light on

9.3 Board-level version (mvBlueFOX-Mxxx)

9.3.1 Dimensions and connectors

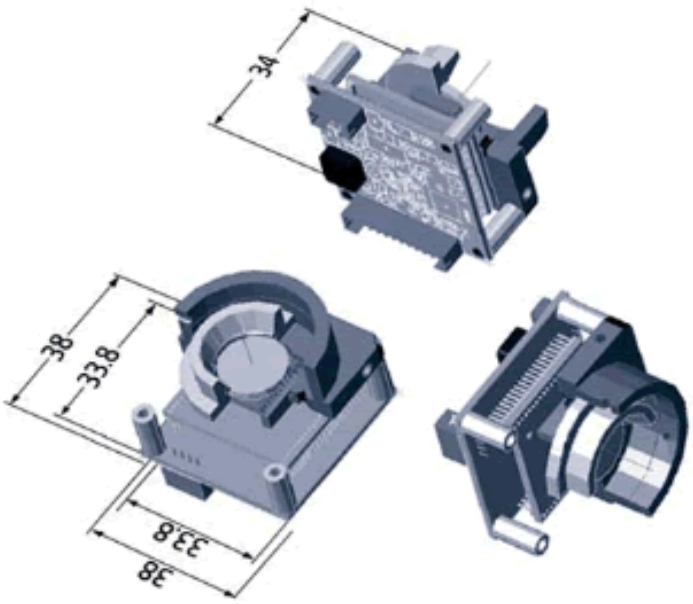


Figure 10: mvBlueFOX-M12x (CCD) with C-mount

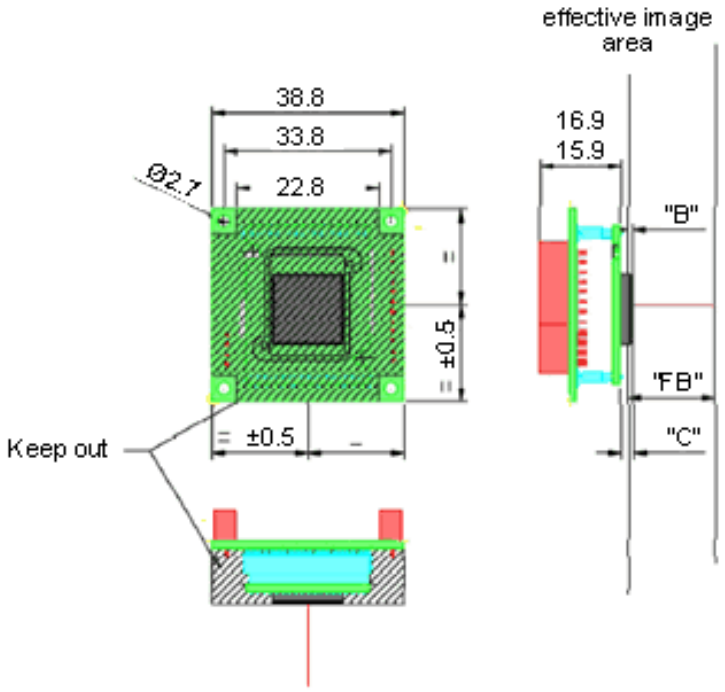


Figure 11: mvBlueFOX-M10x (CMOS)

Lens mount	
Type	"FB"
C-Mount	17.526
CS-Mount	12.526

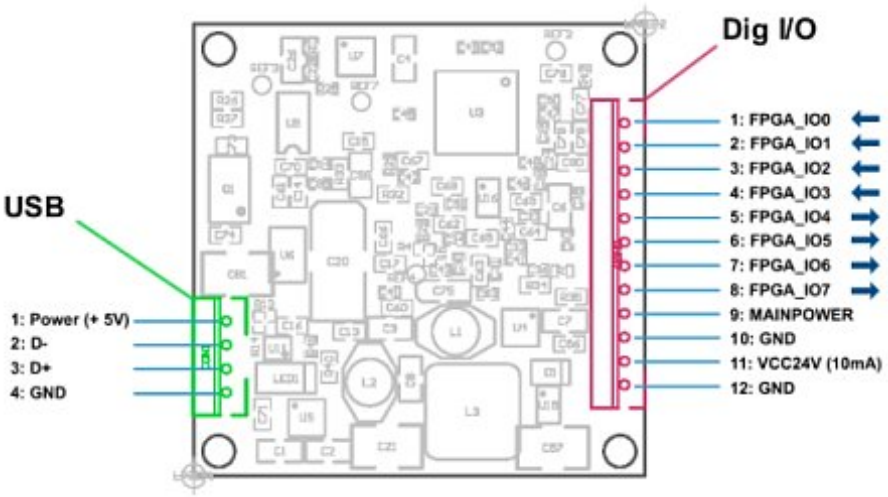


Figure 12: Backside view of the board

Note

The mvBlueFOX-M has a serial I²C bus EEPROM with 64 KBit of which 512 Bytes can be used to store custom arbitrary data.

See also

`UserDataEntry class description`

Pin	Signal	Comment	Cable
1	USBPOWER_IN	Supply voltage	red
2	USB_DATA-	Data	white
3	USB_DATA+	Data	green
4	GND	Ground	black

9.3.1.1 4-pin Wire-to-Board header (USB 2.0) Manufacturer: JST

Part number: B4B-PH-K

Pin	Signal	Comment
1	FPGA_IO0	Digital In 0
2	FPGA_IO1	Digital In 1
3	FPGA_IO2	Digital In 2
4	FPGA_IO3	Digital In 3
5	FPGA_IO4	Digital Out 0
6	FPGA_IO5	Digital Out 1
7	FPGA_IO6	Digital Out 2
8	FPGA_IO7	Digital Out 3
9	MAINPOWER	Current from the USB cable
10	GND	Ground
11	VCC24V	24 V output (10mA)
12	GND	Ground

9.3.1.2 12-pin Wire-to-Board header (Dig I/O) Manufacturer: JST

Part number: B12B-PH-K

Attention

"False tensions or short-circuits"

The digital I/O's are connected directly via a resistor to the FPGA pins and therefore they are not protected. If you connect the digital I/Os without providing a protection you will risk damaging the device.

→ - Provide a protection circuit to the digital I/O's of mvBlueFOX-M. - Afterwards connect the digital I/Os to the FPGA pins.

See also

High-Speed USB design guidelines

9.3.1.3 Contact

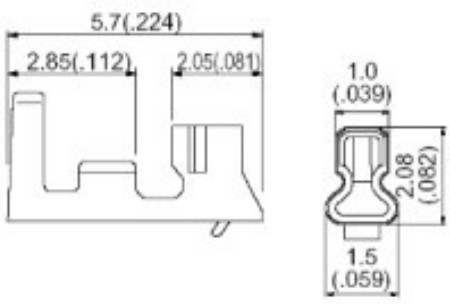


Figure 13: Contact, dimensions in mm (in.)

Application wire			Q'ty / reel
mm2	AWG #	Insulation O.D. mm (in.)	
0.05 to 0.22	30 to 24	0.9 to 1.5 (.035 to .059)	8.000

Material and finish: phosphor bronze, tin-plated
Manufacturer: JST
Part number: SPH-002T-P0.5S

9.3.1.4 Housing

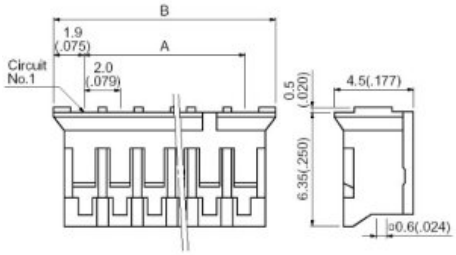


Figure 14: Housing, dimensions in mm (in.)

Circuits	Dimensions in mm (in.)		Q'ty / box
	A	B	
4	6.0 (.236)	9.8 (.386)	1.000
12	22.0 (.866)	25.8 (1.016)	1.000

Material and finish: nylon 66, UL94V-0, natural (white)
Manufacturer: JST
Part number: PHR-4 / PHR-12

See also

Suitable assembled cable accessories for mvBlueFOX-M: What's inside and accessories

9.3.1.5 Characteristics of the mvBlueFOX-Mxxx digital I/Os

Symbol	Comment	Min	Max	Unit
U_{DIG_IN}	Input voltage	-0.↩3	3.6	V

9.3.1.5.1 Dig I/O max. values

Symbol	Comment	Min	Nom	Max	Unit
$U_{DIG\_IN\_LOW}$	low level input voltage (I_{IN} = 1.67mA)	-0.3	0	0.9	V
$U_{DIG\_IN\_HIGH}$	high level input voltage (I_{IN} = 1.67mA)	2.2	3.3	3.6	V
I_{IN}	input current (@ 3.3V)	0.4		1.7	mA

9.3.1.5.2 Characteristics of the digital inputs

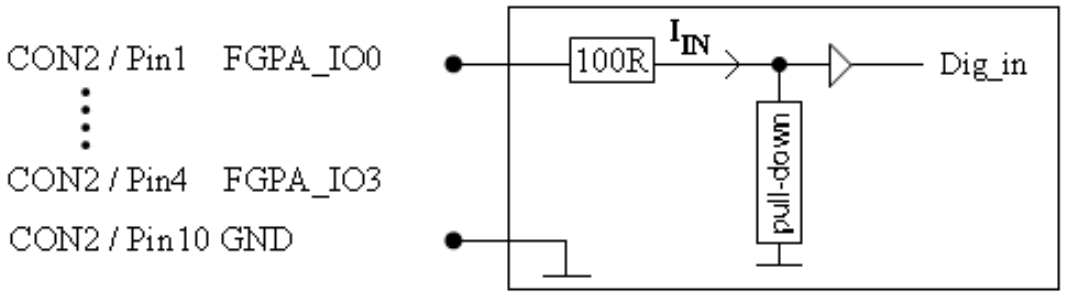


Figure 15: Digital input mvBlueFOX-Mxxx

Symbol	Comment	Min	Nom	Max	Unit
I_{DIG_OUT}	current at digital output	+-12		+-24	mA
$U_{DIG\_OUT\_HIGH}$	digital output (I_{OUT}=12mA)	1.6			V
	Digital output (I_{OUT}<2mA)	2.6		3.4	V
$U_{DIG\_OUT\_LOW}$	digital output (I_{OUT}=2mA)			0.2	V

9.3.1.5.3 Characteristics of the digital outputs $U_{DIG\_OUT\_HIGH\ min}$ = 2.8 - $I_{OUT} * 100$

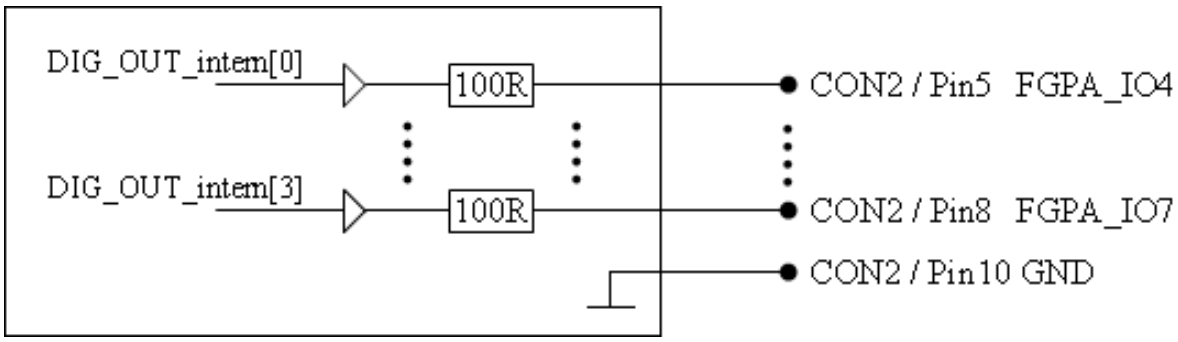


Figure 16: Digital output mvBlueFOX-Mxxx

Attention

"False tensions or short-circuits"

The digital I/O's are connected directly via a resistor to the FPGA pins and therefore they are not protected. If you connect the digital I/Os without providing a protection you will risk damaging the device.

→ - Provide a protection circuit to the digital I/O's of mvBlueFOX-M. - Afterwards connect the digital I/Os to the FPGA pins.

Note

The Dig I/O characteristics of the mvBlueFOX-M are not compatible to the Dig I/O of the mvBlueFOX standard version.

9.3.2 LED states

State	LED
Camera is not connected or defect	LED off
Camera is connected and active	Green light on

9.3.3 Accessories mvBlueFOX-Mxxx

9.3.3.1 mvBlueFOX-M-FC-S The **mvBF-M-FC-S** contains high capacity condensers with switching electronics for transferring stored energy of the condensers to external flash LEDs. It is possible to connect 2 pushbuttons/switches to the 8-pin header (CON3 - Control connector). Additionally, 2 LED interfaces are available. There are two version of **mvBF-M-FC-S**:

- Model 1 can be connected to mvBlueFOX-M with a cable via CON5.
- Model 2 can be mounted on the mvBlueFOX-M via CON1 directly.

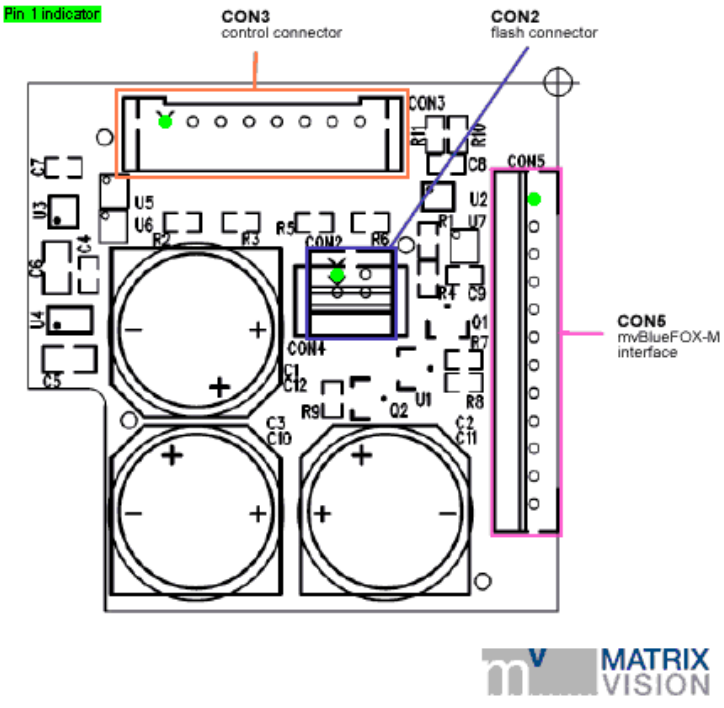


Figure 17: Model 1 with CON5 connector

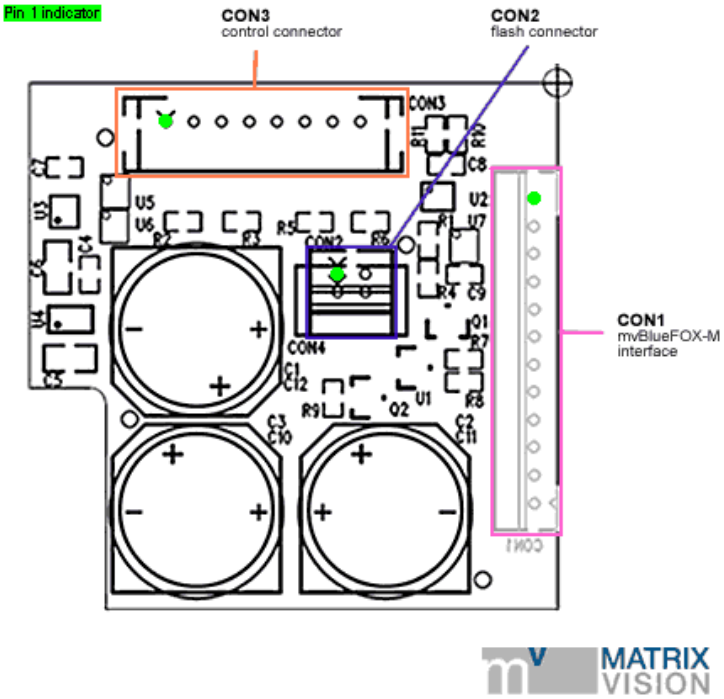


Figure 18: Model 2 with CON1 connector

Pin	Signal	Comment
1	Flash +	Flash power
2	Flash -	Switched to ground (low side switch)

9.3.3.1.1 CON2 - Flash connector Manufacturer: JST

Part number: B-2B-PH

Pin	Signal	Comment
1	GND	LED2 cathode connector / board ground
2	LED2 output	LED2 anode connector1
3	GND	LED1 cathode connector / board ground
4	LED1 output	LED1 anode connector
5	GND	Board ground
6	Input2	Switch to ground for setting Input2
7	GND	Board ground
8	Input1	Switch to ground for setting Input1

9.3.3.1.2 CON3 - Control connector Manufacturer: JST

Part number: B-8B-PH-SM4 TB

Signal	Parameter	Min	Typ	Max	Unit
GND	Board ground			0	V
LED 1/2 output (anode)	Output voltage [2]		5		V
	Internal series resistance	465.↩ 3	470	474.↩ 4	Ohm
	Forward current I_F at U_{LED} = 2V [1]		6		mA
Input 1/2 (internal 10k pull up to 3.3V)	Voltage (open contact)			3.3	V
	V_{IL} (low level input voltage)			0.9	V
	V_{IH} (high level input voltage)	2.5		5.5	V
Flash +	Voltage (open contact)	23	24	25	V
	Flash output capacitance	528	660	792	uF
	Internal capacitance storage energy		0.190		Ws
	Flash capacitance charge current / output DC current		20		mA
Flash [2]	I_{OUT}			-2	A
	On voltage at I_{OUT}MAX			0.15	V
	Off voltage	23	24	25	V

9.3.3.1.3 Electrical characteristic [1] Depends on mvBlueFOX-M power supply
[2] **Attention:** No over-current protection!

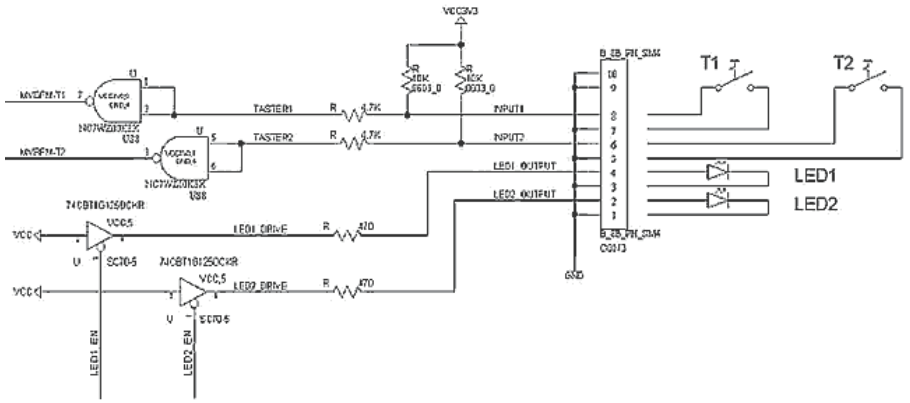


Figure 19: CON3 schematic

9.4 Single-board version (BVS CA-MLC)

9.4.1 Typical Power consumption @ 5V

Model	Power consumption (+/- 10%)	Unit
-200w	1.09	W
-202a	1.39	W
-202b	1.58	W
-202d	1.28	W
-202v	1.35	W
-205	1.37	W

9.4.2 Dimensions and connectors

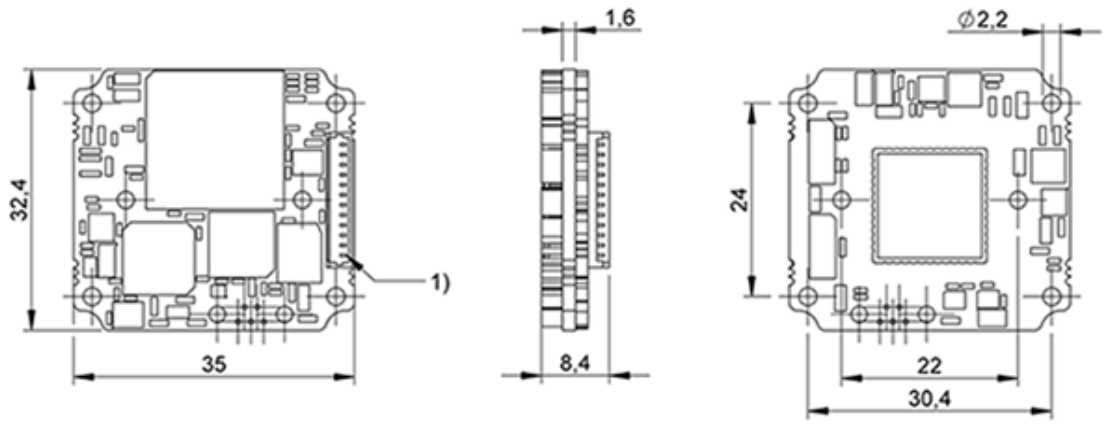


Figure 20: BVS CA-MLC-xxxxxx-1600OX-001

Note

The BVS CA-MLC has a serial I²C bus EEPROM with 16 KByte of which 8 KByte are reserved for the firmware and 8 KByte can be used to store custom arbitrary data.

See also

`UserDataEntry class description`

9.4.2.1 Sensor's optical midpoint and orientation The sensor's optical midpoint is in the center of the board (Figure 21: intersection point of the holes diagonals). The (0,0) coordinate of the sensor is located at the one bottom left corner of the sensor (please notice that Mini-B USB connector is located at the bottom at the back).

Note

Using a lens, the (0,0) coordinate will be mirrored and will be shown at the top left corner of the screen as usual!

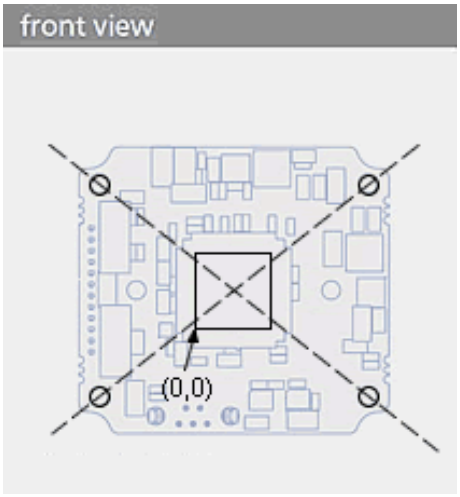


Figure 21: Sensor's optical midpoint and orientation

9.4.2.2 Mini-B USB (USB 2.0)

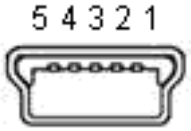


Figure 22: Mini-B USB

Pin	Signal	Comment
1	USBPOWER_IN	Supply voltage
2	USB_DATA-	Data
3	USB_DATA+	Data
4	ID	Not connected
5	GND	Ground

9.4.2.3 12-pin Wire-to-Board header (USB 2.0 / Dig I/O)

Note

If you have the BVS CA-MLC variant which uses the standard Mini-B USB connector, pin 2 and 3 (USB↩ _DATA+ / USB_DATA-) of the header won't be connected!

pin	Opto-isolated variant		TTL compliant variant		Cable KS-↩ MLC-USB2-↩ IO-W	Cable KS-↩ MLC-IO-W
	Signal	Comment	Signal	Comment		
1	GND	Ground	GND	Ground	GND	
2	USB_DATA+	Data	USB_DATA+	Data	USB_DATA+	
3	USB_DATA-	Data	USB_DATA-	Data	USB_DATA-	
4	USBPOWER↩ _IN	Supply voltage	USBPOWER↩ _IN	Supply voltage	USBPOWER↩ _IN	
5	I2C SDA	Serial data line (the I2C interface is master-only, which means that I2C slaves can only be connected externally)	I2C SDA	Serial data line		
6	I2C SCL	Serial clock line (the I2C interface is master-only, which means that I2C slaves can only be connected externally)	I2C SCL	Serial clock line		
7	USBPOWER↩ _IN	Supply voltage	USBPOWER↩ _IN	Supply voltage	red	
8	GND	Ground	GND	Ground	black	black

9	OUT0-	Opto-isolated digital output 0 (Negative voltage)	OUT1	TTL compliant digital output 1	blue	blue
10	OUT0+	Opto-isolated digital output 0 (Positive voltage)	OUT0	TTL compliant digital output 0	violet	violet
11	IN0-	Opto-isolated digital input 0 (Negative voltage)	IN1	TTL compliant digital input 1	gray	gray
12	IN0+	Opto-isolated digital input 0 (Positive voltage)	IN0	TTL compliant digital input 0	pink	pink

Note

I2C bus uses 3.3 Volts. Signals have a 2kOhm pull-up resistor. Access to the I2C bus from an application is possible for BVS CA-MLC devices using an mvBlueFOX driver with version 1.12.44 or newer.

Manufacturer (suitable board-to-wire connector): Molex
Part number: 0510211200 1.25mm Housing
Link: http://www.molex.com/molex/products/datasheet.jsp?part=active/0510211200_CRIMP_HOUSINGS.xml&channel=Products&Lang=en-US
Manufacturer (multi-pin connector for board-to-board connection): e.g. Garry
Link: http://www.mpe-connector.de/index.php?lang=de&menu=16&mating=1841&id_product=6591 (recommended variant: 659-1-012-O-F-RS0-xxxx; xxxx = length of the pins)

See also

Suitable assembled cable accessories for BVS CA-MLC: What's inside and accessories

High-Speed USB design guidelines

More information about the usage of retrofittable ferrite

9.4.2.3.1 Electrical characteristic *Digital inputs TTL*

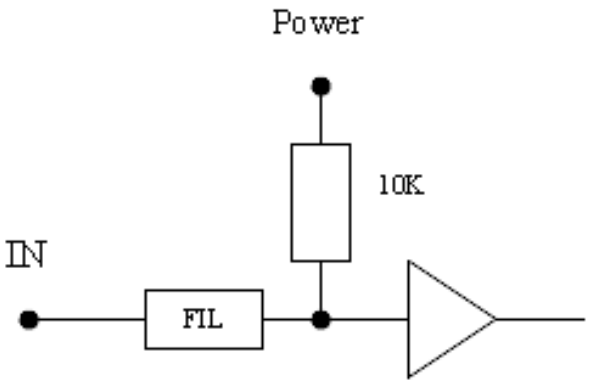


Figure 23: TTL digital inputs block diagram

Note

If the digital input is not connected, the state of the input will be "1" (as you can see in ImpactControlCenter).

TTL compliant variant					
	Comment	Min	Typ	Max	Unit
I_{IN}	I_{LOW} (INx)			-0.5	mA
U_{IN}	V_{IH}	3.6		5.5	V
	V_{IL}	-0.3		1.3	V
LVTTL compliant variant					
	Comment	Min	Typ	Max	Unit
I_{IN}	I_{LOW} (INx)			-0.5	mA
U_{IN}	V_{IH}	2		3.8	V
	V_{IL}	-0.3		0.8	V

TTL input low level / high level time: Typ. $<$ 210ns

Digital outputs TTL

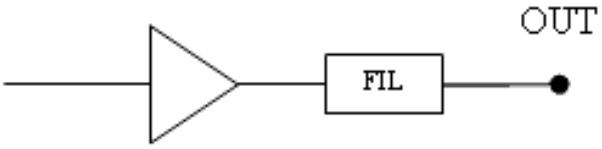


Figure 24: TTL digital outputs block diagram

	Comment	Min	Typ	Max	Unit
I_{OUT}	Dig_out power			+-32	mA
U_{OUT}	V_{OH} (I_{OUT}=32mA)	3.8			V
	V_{OH}			5.25	
	V_{OL} (I_{OUT}=32mA)			0.55	V
	V_{OL}	0.1			

TTL output low level / high level time: Typ. $<$ 40ns

Opto-isolated digital inputs

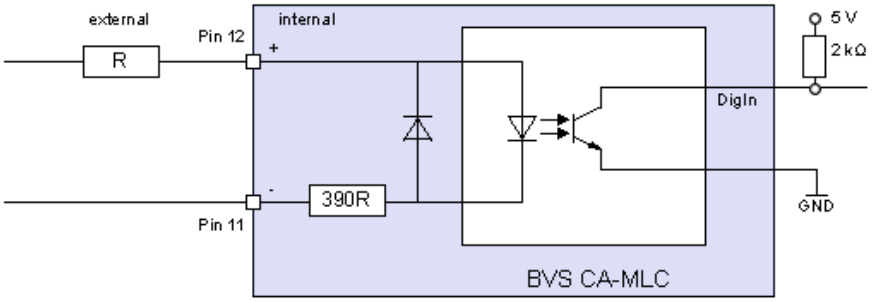


Figure 25: Opto-isolated digital inputs block diagram with example circuit

Delay

Characteristics	Symbol	Typ.	Unit
Turn-On time	t_{ON}	3	us

The inputs can be connected directly to +3.3 V and 5 V systems. If a higher voltage is used, an external resistor must be placed in series (Figure 25).

Used input voltage	External series resistor
3.3V .. 5V	none
12V	680 Ohm
24V	2 KOhm

	Comment	Min	Typ	Max	Unit
U_{IN}	V_{IH}	3		5.5	V
	V_{IL}	-5.↩5		0.8	V

Opto-isolated digital outputs

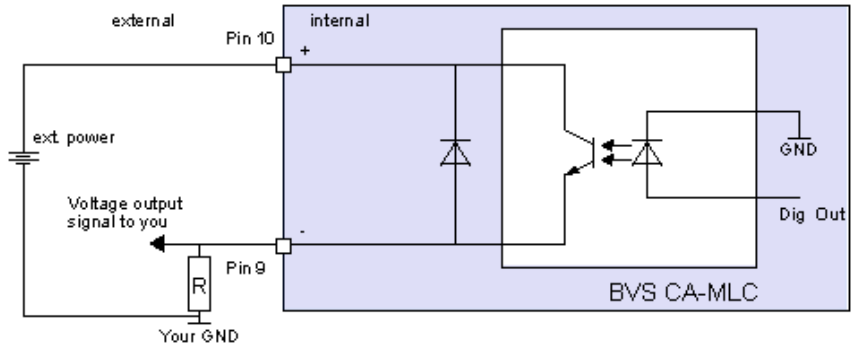


Figure 26: Opto-isolated digital outputs block diagram with example circuit

Delay

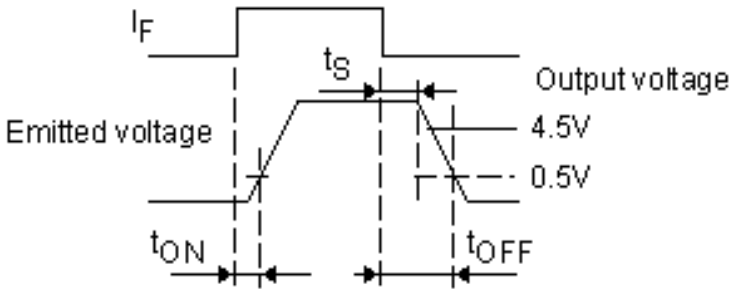


Figure 27: Output switching times

Characteristics	Symbol	Test conditions	Typ.	Unit
Turn-On time	t_{ON}	R_L = 1.9 kOhm, V_{CC} 5V, I_C = 16mA	2	us
Storage time	t_S		25	
Turn-Off time	t_{OFF}		40	

	Comment	Min	Typ	Max	Unit
I_{on}	load current			15	mA
I_{off}	leakage current			10	uA
V_{on} Sat. at 2.4 mA	V_{IH}	0	(0.↩ 2)	0.4	V
V_{off}				30	V

9.4.3 LED states

State	LED
Camera is not connected or defect	LED off
Camera is connected but not initialized or in "Power off" mode.	Orange light on
Camera is connected and active	Green light on

9.4.4 Assembly variants

The BVS CA-MLC is available with following differences:

- Mini-B USB connector and digital I/O pin header
  - 1/1 opto-isolated or 2/2 TTL compliant digital I/O
- USB via header without Mini-B USB connector
- female board connector instead of pin header (board-to-board connection)
- 3 different S-mount depths
- C(S)-mount compatibility using mvBlueCOUGAR-X flange
- ambient temperature operation: 5..55 deg C / 30..80 RH
- ambient temperature storage: -25..60 deg C / 20..90 RH

9.5 Single-board version with housing (BVS CA-IGC)

9.5.1 Dimensions and connectors

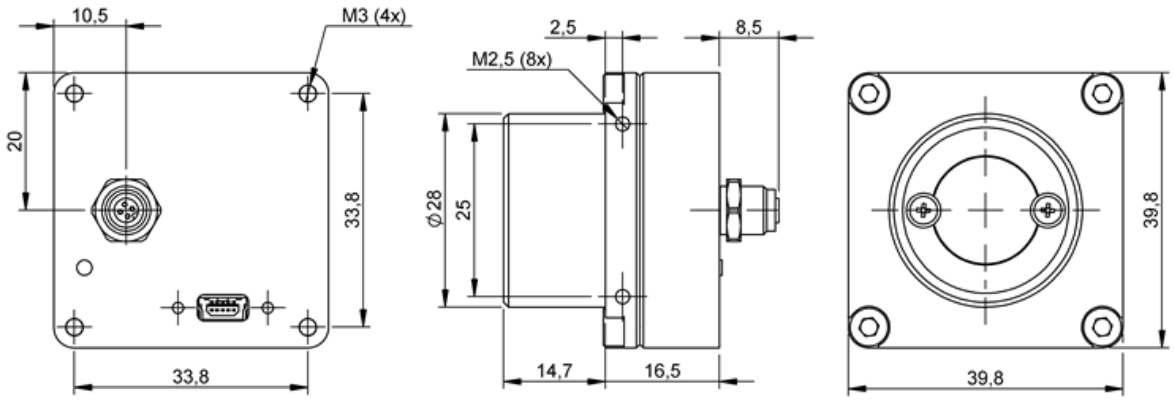


Figure 28: BVS CA-IGC-xxxxxx-16xxxx-001 with CS-mount without adjustable backfocus (standard)

Lens protrusion	C-Mount	CS-Mount
X	10 mm	5 mm

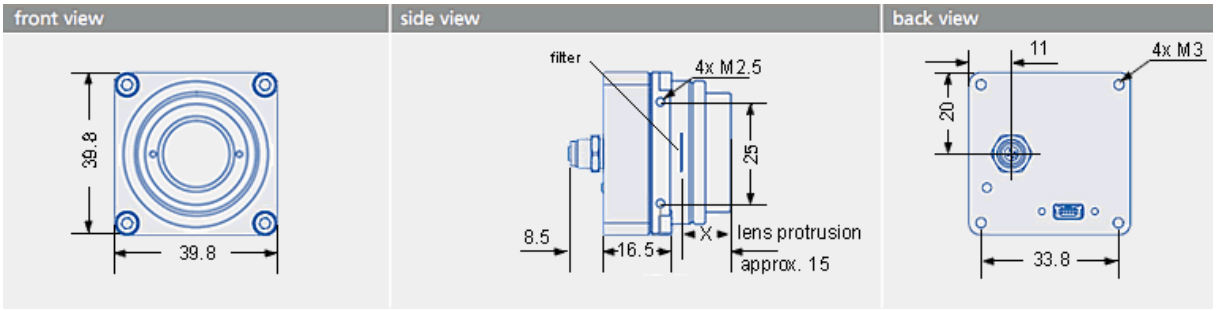


Figure 29: BVS CA-IGC-xxxxxx-19xxxx-001 with C-mount and adjustable backfocus

Lens protrusion	C-Mount
X	8 mm (9.5 mm with max. Ø 20 mm)

Note

The BVS CA-IGC has a serial I²C bus EEPROM with 16 KByte of which 8 KByte are reserved for the firmware and 8 KByte can be used to store custom arbitrary data.

See also

UserDataEntry class description

9.5.1.1 Mini-B USB (USB 2.0)

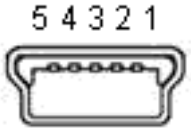


Figure 30: Mini-B USB

Pin	Signal	Comment
1	USBPOWER_IN	Supply voltage
2	USB_DATA-	Data
3	USB_DATA+	Data
4	ID	Not connected
5	GND	Ground

9.5.1.2 4-pin circular plug-in connector with lock (I/O)

Figure 31: 4-pin circular plug-in connector (female)

Pin	Signal	Comment	Color (of cable)
1	IN0 +	Opto-isolated digital input 0 (Positive voltage)	brown
2	IN0 -	Opto-isolated digital input 0 (Negative voltage)	white
3	OUT0 +	Opto-isolated digital output 0 (Positive voltage)	blue
4	OUT0 -	Opto-isolated digital output 0 (Negative voltage)	black

Manufacturer: Binder
Part number: 79-3107-52-04

9.5.1.2.1 Electrical characteristic Please have a look at the BVS CA-MLC digital I/O characteristics (opto-isolated model) of the 12-pin Wire-to-Board Header (USB / Dig I/O).

9.5.2 LED states

State	LED
Camera is not connected or defect	LED off
Camera is connected but not initialized or in "Power off" mode.	Orange light on
Camera is connected and active	Green light on

9.5.3 Positioning tolerances of sensor chip

The sensor's optical midpoint is in the center of the housing. However, several positioning tolerances in relation to the housing are possible because of:

- Tolerance of mounting holes of the printed circuit board in relation to the edge of the lens holder housing is not specified but produced according to general tolerance DIN ISO 2768 T1 fine.
- Tolerance of mounting holes on the printed circuit board because of the excess of the holes ± 0.1 mm (Figure 32; 2).
- Tolerance between conductive pattern and mounting holes on the printed circuit board.
  Because there is no defined tolerance between conductive pattern and mounting holes, the general defined tolerance of ± 0.1 mm is valid (Figure 32; 1 in the Y-direction ± 0.1 mm; 3 in the Z-direction ± 0.1 mm)

There are further sensor specific tolerances, e.g. for model BVS CA-MLC-0004ZG:

- Tolerance between sensor chip MT9V034 (die) and its package (connection pad)
  - Chip position in relation to the mechanical center of the package: 0.2 mm (± 0.1mm) in the X- and Y-direction (dimensions in the sensor data sheet according to ISO 1101)
- Tolerance between copper width of the sensor package and the pad width of the printed circuit board
  During the soldering the sensor can swim to the edge of the pad: width of the pad 0.4 mm (possible tolerance is not considered), width of pin at least 0.35 mm, max. offset: ± 0,025mm

Further specific tolerances of other models on request.

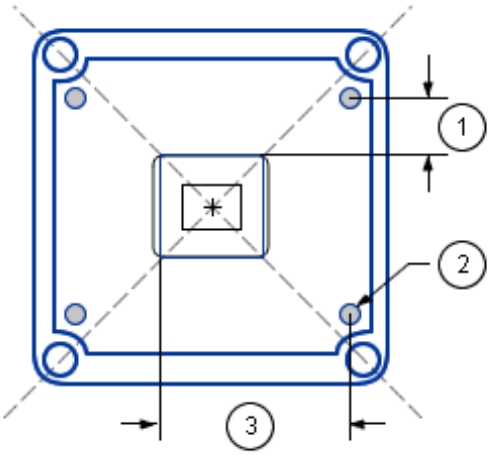


Figure 32: Positioning tolerances of sensor chip

Note

There are also tolerances in lens which could lead to optical offsets.

9.6 Summary of components

Features		Standard	-M	-MLC		
				"-xOx-"	"-xTx-"	"-xLx-"
		IP40				
Image Memory		8 Mpixels				
ADC resolution		CMOS: 10 bits (10/8 bit transfer) / CCD: 12 bits (10 bit transfer)				
Inputs		2	4	1	2	3
	Type	opto-isolated 5 V (TTL) / 24 V (PLC) switchable via software	LVTTL	opto-isolated	TTL	LVTTL
Outputs		2	4	1	2	-
	Type	opto-isolated	LVTTL	opto-isolated	TTL	-
USB 2.0		USB-B	4-pin Wire-to-Board header	Mini-B USB (-xxW-) / 12-pin Wire-to-Board header (-xxB-) / Mini-B U		
Optics						
	Lens Mount (Focal Distance)	C-mount (17.526 mm in air), CS-mount (12.5 mm in air)	C-mount (17.526 mm in air), CS-mount (12.5 mm in air), optional S-mount	C-mount (17.526 mm in air), CS-mount (12.5 mm in air), optional S-		
Environment						
	Ambient Temperature					
	Operation	0..45 deg C / 30 to 80% RH				
	Storage	-20..60 deg C / 20 to 90% RH				
	Protection class [1]	IP40				
Weight without lens		approx. 120 g	approx. 17 g	approx. 10 g		
Power supply (PWR↩_IN)						
	DC	4.75 to 5.25 V via USB				
	P_{max}	1.5 W				
	Peak current draw	0.5 A				

[1] not evaluated by UL

9.6.1 Summary of available digital I/O's

	mvBlueFOX	-M	BVS CA-MLC		BVS CA-IGC
order options			-xxxxOx	-xxxxTx	-xxxx2x
digital inputs	2 (opto-coupled) 5 V (TTL) / 24 V (PLC) switchable via software	4 (LVTTL)	1 (opto)	2 (TTL)	1 (opto)
digital outputs	2 (opto-coupled)	4 (LVTTL)	1 (opto)	2 (TTL)	1 (opto)

10 Sensor Overview

By default, the steps exposure and readout out of an image sensor are done one after the other. By design, CCD sensors support overlap capabilities also combined with trigger (see figure). In contrast, so-called pipelined CMOS sensors only support the overlapped mode. Even less CMOS sensors support the overlapped mode combined with trigger. Please check the sensor summary. In overlapping mode, the exposure starts the exposure time earlier during readout.

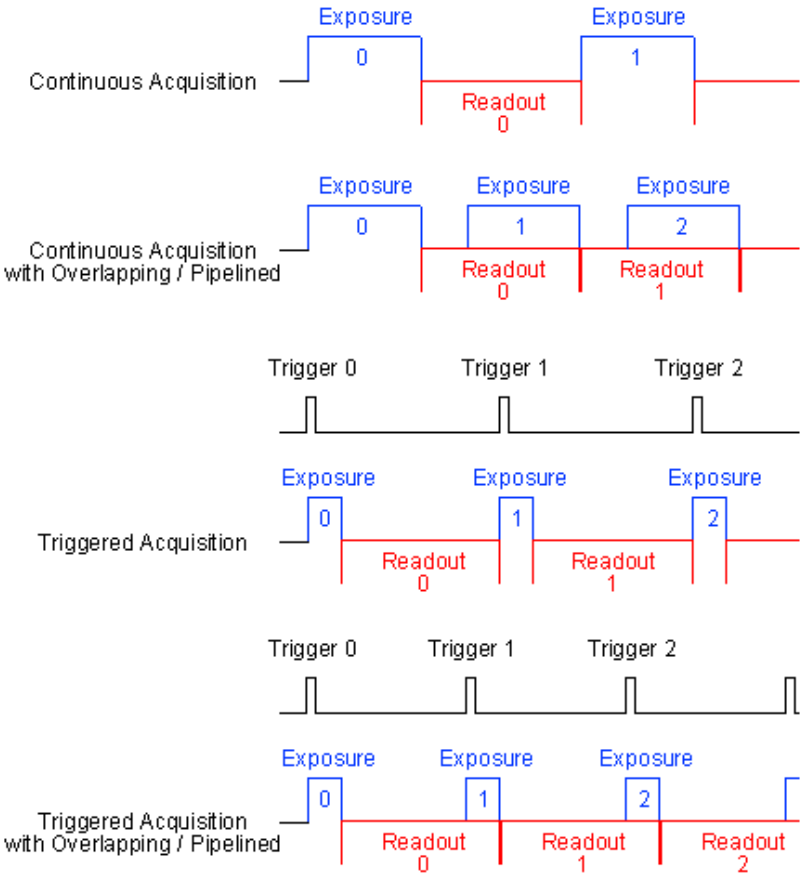


Figure 1: Overlapping / pipelined exposure and readout

10.1 CCD sensors

The CCD sensors are highly programmable imager modules which incorporate the following features:

Sensors	0.3 Mpixels resolution CCD sensor (-220)	0.3 Mpixels resolution CCD sensor (-220a)	0.8 Mpixels resolution CCD sensor (-221)	1.4 Mpixels resolution CCD sensor (-223)	1.9 Mpixels resolution CCD sensor (-224)
Sensor supplier	Sony	Sony	Sony	Sony	Sony
Sensor name	ICX098 AL/BL	ICX424 AL/AQ	ICX204 AL/AQ	ICX267 AL/AQ	ICX274 AL/AQ
Resolution	640 x 480 gray scale or RGB Bayer mosaic	640 x 480 gray scale or RGB Bayer mosaic	1024 x 768 gray scale or RGB Bayer mosaic	1360 x 1024 gray scale or RGB Bayer mosaic	1600 x 1200 gray scale or RGB Bayer mosaic
Sensor format	1/4"	1/3"	1/3"	1/2"	1/1.8"
Pixel clock	12 MHz / 24 MHz	20 MHz / 40 MHz	20 MHz / 40 MHz	tbd / 40 MHz	20 MHz / 40 MHz
Max. frames per second	60	100	39[1]	20	16
Binning	H+V	H+V	H+V	H+V	H+V
Exposure time	44 us - 10 s	26 us - 10 s	44 us - 10 s	33 us - 10 s	30 us - 10 s
ADC (on sensor board) resolution	12 bit (up to 10 bit transmission)	12 bit (up to 10 bit transmission)	12 bit (up to 10 bit transmission)	12 bit (up to 10 bit transmission)	12 bit (up to 10 bit transmission)
Programmable analog gain and offset	X				
Frame integrating progressive scan sensor (no interlaced problems!)	X				
High resolution	X				
High color reproductivity (for color version)	X				
High sensitivity, low dark current	X				
Continuous variable-speed shutter	X				
Pipelined in continuous / triggered mode	X / -				
Low smear	X				
Excellent antiblooming characteristics	X				
Programmable exposure time from usec to sec.	X				
Programmable readout timing with free capture windows and partial scan	X				
Trigger (Hardware / Software)	X / X				

Pipelined in continuous / triggered mode	X / -	X / -	X / -	X / -	X / -
Flash control output, synchronous to exposure period	X				
More specific data	mvBlueFOX-[Model] [illegible]	[illegible]	[illegible]	[illegible]	[illegible]

[1] With max. frame rate, image quality losings might be occur.

10.2 CMOS sensors

The CMOS sensor modules incorporate the following features:

Sensors:	0.4 Mpixels resolution CMOS sensor (-200w)	1.3 Mpixels resolution CMOS sensor (-202a)	1.2 Mpixels resolution CMOS sensor (-x02v)[1] only -MLC/-IGC	1.2 Mpixels resolution CMOS sensor (-x02b)[1] only -MLC/-IGC	1.2 Mpixels resolution CMOS sensor (-202d)[1] only -MLC/-IGC	5.0 Mpixels resolution CMOS sensor (-205)
Sensor supplier	Aptina	Aptina	Aptina	Aptina	Aptina	Aptina
Sensor name	MT9V034	MT9M001	AR0135	MT9M021	MT9M034	MT9P031
Resolution	752 x 480 gray scale or RGB Bayer mosaic	1280 x 1024 gray scale	1280 x 960 gray scale or RGB Bayer mosaic	1280 x 960 gray scale or RGB Bayer mosaic	1280 x 960 gray scale or RGB Bayer mosaic	2592 x 1944 gray scale or RGB Bayer mosaic
Indication of sensor category to be used	1/3"	1/2"	1/3"	1/3"	1/3"	1/2.5"
Pixel clock	40 MHz	40 MHz	40 MHz	40 MHz	40 MHz	40 MHz
Max. frames per second (in free-running full frame mode)	88.6[2]	24.6	24.6	24.6	24.4	5.8
Binning	H+V (frame rate 170 Hz)	H+V, AverageH+V (frame rate unchanged)	H+V, AverageH+V (frame rate unchanged)	H+V, AverageH+V (frame rate unchanged)	H+V, AverageH+V (frame rate unchanged)	H+V, 3H+3V, AverageH+V, Average3H+3V, DroppingH+V, Dropping3H+3V (frame rate 22.7 Hz)
Exposure time	10 us - 0.46 s	100 us - 10 s	10 us - 1 s	10 us - 1 s	10 us - 1 s	10 us - 10 s
ADC resolution	10 bit (10 / 8 bit transmission)	10 bit (10 / 8 bit transmission)	10 bit (10 / 8 bit transmission)	10 bit (10 / 8 bit transmission)	10 bit (10 / 8 bit transmission)	10 bit (10 / 8 bit transmission)

SNR		42 dB	40 dB	40 dB	< 43 dB	37.4 dB
DR (normal / HDR)	55 dB / > 110 dB	61 dB / -	> 61 dB /	> 61 dB /	> 61 dB / > 110 dB (with gray scale version)	65 dB /
Progressive scan sensor (no interlaced problems!)	X	X	X	X	X	X
Rolling shutter	-	X	-	-	X	X
Global shutter	X	-	X	X	-	X
Trigger (Hardware / Software)	X / X	X / -	X / -	X / -	X / -	X / X
Pipelined in continuous / triggered mode	X / -	X / -	X / -	X / -	X / -	X / - (reset only)
High color reproductivity (for color version)	X	no	no	no	no	no
Programmable readout timing with free capture windows and partial scan	X	X	X	X	X	X
Flash control output, synchronous to exposure period	X	no	no	no	no	no
More specific data	mvBlueFOX-[Mo					

[1] The operation in device specific AEC/AGC mode is limited in (non continuous) triggered modes. AEC/AGC only works while trigger signal is active. When the trigger signal is removed AEC/AGC stops and gain and exposure will be set to a static value. This is due to a limitation of the sensor chip.
[2] Frame rate increase with reduced AOI width, but only when width >= 560 pixels, below frame rate remains unchanged.

Note

For further information about rolling shutter, please have a look at the practical report about rolling shutter on our website: https://www.balluff.com/de-en/whitepapers/cmos-sensors-with-rolling-sh

For further information about image errors of image sensors, please have a look at

For further information about image errors of image sensors, please have a look at Correcting image errors of a sensor.

10.3 Output sequence of color sensors (RGB Bayer)

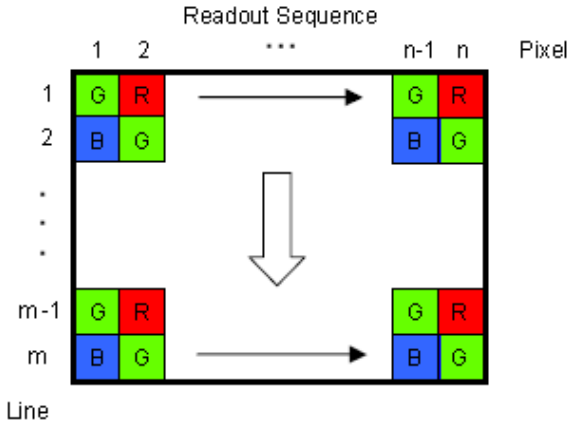


Figure 2: Output sequence of RAW data

10.4 Bilinear interpolation of color sensors (RGB Bayer)

For Bayer demosaicing in the camera, we use bilinear interpolation:

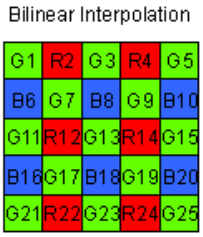


Figure 3: Bilinear interpolation

1. Interpolation of green pixels: the average of the upper, lower, left and right pixel values is assigned as the G value of the interpolated pixel.
   For example:

$$G8 = \frac{(G3+G7+G9+G13)}{4}$$

   For G7:

$$G7\_new = 0.5 * G7 + 0.5 * \frac{(G1+G3+G11+G13)}{4}$$

2. Interpolation of red/blue pixels:
   Interpolation of a red/blue pixel at a green position: the average of two adjacent pixel values in corresponding color is assigned to the interpolated pixel.
   For example:

$$B7 = \frac{(B6+B8)}{2} \; ; \; R7 = \frac{(R2+R12)}{2}$$

Interpolation of a red/blue pixel at a blue/red position: the average of four adjacent diagonal pixel values is assigned to the interpolated pixel.
For example:

$$R8 = \frac{(R2+R4+R12+R14)}{4} ; \quad B12 = \frac{(B6+B8+B16+B18)}{4}$$

Any colored edge which might appear is due to Bayer false color artifacts.

Note

There are more advanced and adaptive methods (like edge sensitive ones) available if the host is doing this debayering.

11 Filters

Several filters are available. The IR filter is part of the standard delivery condition.

11.1 Hot mirror filter

The hot mirror filter FILTER IR-CUT 15,5X1,75 FE has great transmission in the visible spectrum and blocks out a significant portion of the IR energy.

Technical data	
Diameter	15.5 mm
Thickness	1.75 mm
Material	Borofloat
Characteristics	T = 50% @ 650 +/- 10 nm
	T $>$ 92% 390-620 nm
	R_{avg} $>$ 95% 700-1150 nm
	AOI = 0 degrees
Surface quality	Polished on both sides
Surface irregularity	5/3x0.06 scratch/digs on both sides
	Edges cut without bezel

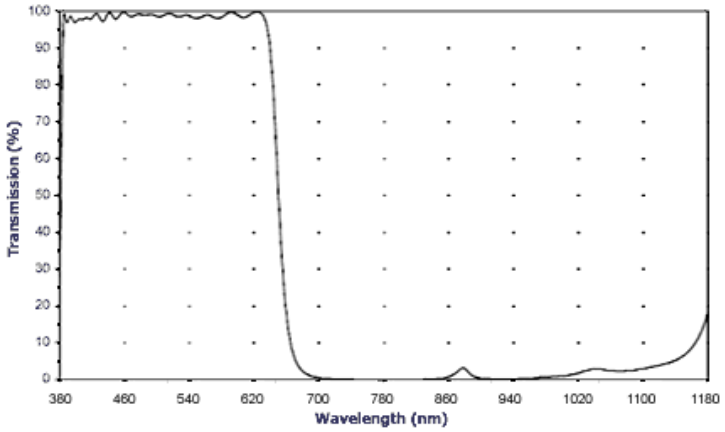


Figure 1: FILTER IR-CUT 15,5X1,75 FE wavelengths and transmission diagram

11.2 Cold mirror filter

The **FILTER** DL-CUT 15,5X1,5 is a high-quality day light cut filter and has optically polished surfaces. The polished surface allows the use of the filter directly in the path of rays in image processing applications. The filter is protected against scratches during the transport by a protection film that has to be removed before the installing the filter.

Technical data	
Diameter	15.5 mm
Thickness	1.5 +/- 0.2 mm
Material	Solaris S 306
Characteristics	$T_{avg} > 80\% > 780$ nm
	AOI = 0 degrees
	Protective foil on both sides
	Without antireflexion
	Without bezel

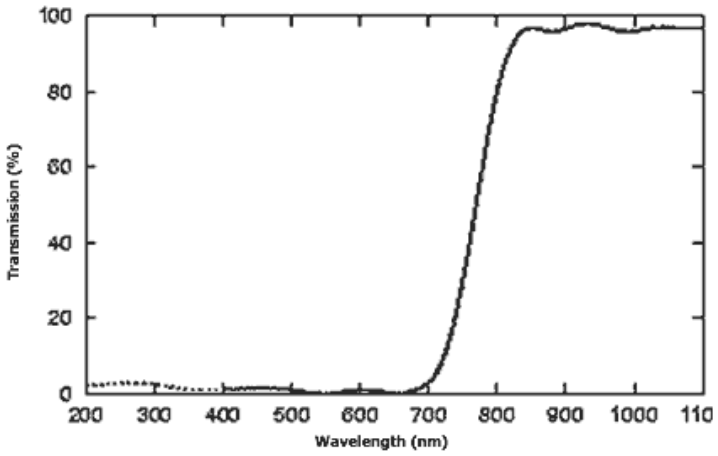


Figure 2: FILTER DL-CUT 15,5X1,5 wavelengths and transmission diagram

11.3 Glass filter

It is also possible to choose the glass filter **"FILTER GLASS 15,5X1,75"** with following characteristics:

Technical data	
Glass thickness	1.75 mm
Material	Borofloat without coating
	ground with protection chamfer
Surface quality	polished on both sides P4
Surface irregularity	5/3x0.06 on both sides

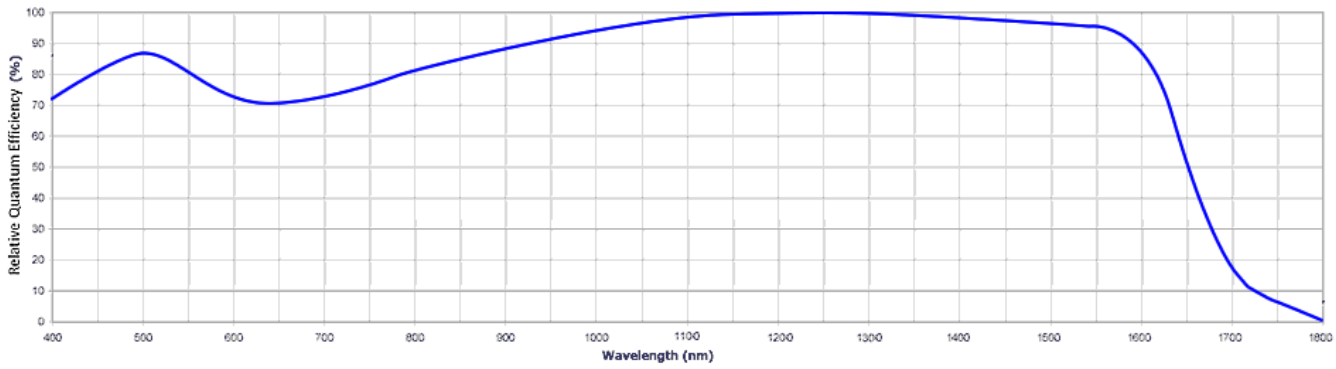


Figure 3: Glass filter wavelengths and relative quantum efficiency diagram

12 GUI tools

12.1 Introduction

Balluff provides several convenient tools with graphical user interface to set up and work with their devices. Please find a short list and description below:

12.2 ImpactControlCenter

With ImpactControlCenter it is possible

- to acquire images
- to configure the device, and
- to display and modify the device properties.

12.3 DeviceConfigure

DeviceConfigure can be used

- to set the device ID
- to update firmware
- to disable CPU sleep states(some version of Windows only)

12.4 IPConfigure

With IPConfigure it is possible

- to configure the network behavior of GigE Vision™ devices
- to determine and solve network issues.

12.5 GigEConfigure

With GigEConfigure it is possible

- to install, remove or configure the Balluff GigE Vision™ capture filter driver.

See also

For further information about the tools, please follow the link to the separate manual describing the GUI tools in great detail on our website: https://www.balluff.com/en-de/documentation-for-your-balluff-pro

13 HRTC - Hardware Real-Time Controller

13.1 Introduction

The *Hardware Real-Time Controller* (HRTC) is built into the FPGA. The user can define a sequence of operating steps to control the way how and when images are exposed and transmitted. Instead using an external PLC, the time critical acquisition control is directly build into the camera. This is a very unique and powerful feature.

13.1.1 Operating codes

The operating codes for each step can be one of the followings:

OpCode	Parameter	Description
Nop	-	No operation
SetDigout	Operation array on dig out	Set a digital output
WaitDigin	State definition array on dig in	Wait for a digital input
WaitClocks	Time in us	Wait a defined time
Jump	HRTC program address	Jump to any step of the program
TriggerSet	Frame ID	Set internal trigger signal to sensor controller
TriggerReset	-	Reset internal trigger signal to sensor controller
ExposeSet	-	Set internal expose signal to sensor controller
ExposeReset	-	Reset internal expose signal to sensor controller
FrameNrReset	-	Reset internal sensor frame counter

256 HRTC steps are possible.

The section How to use the HRTC should give the impression what everything can be done with the HRTC.

13.2 How to use the HRTC

To use the HRTC you have to set the trigger mode and the trigger source. With object orientated programming languages the corresponding camera would look like this (C++ syntax):

```
CameraSettings->triggerMode = ctmOnRisingEdge
CameraSettings->triggerSource = ctsRTCtrl
```

When working with ImpactControlCenter this are the properties to modify in order to activate the evaluation of the HRTC program:

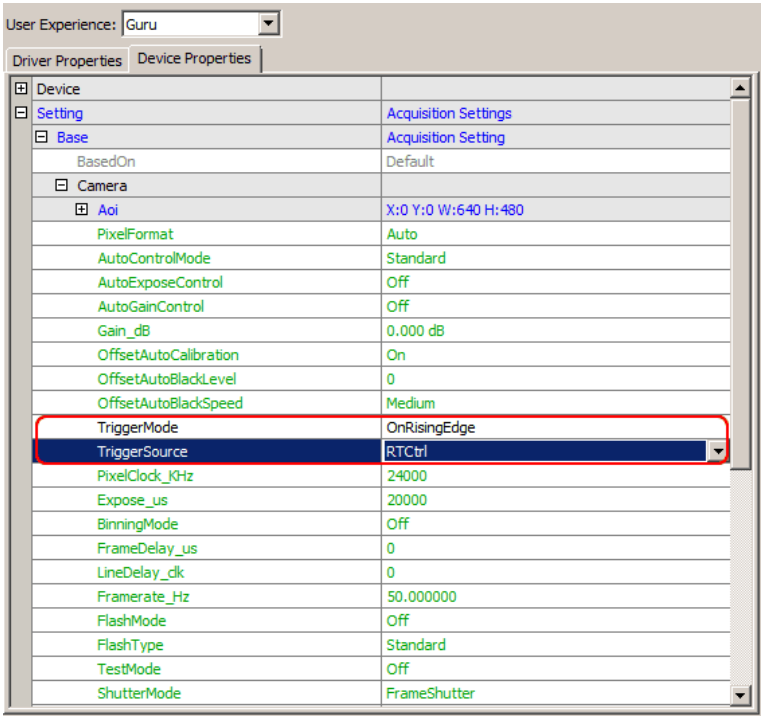


Figure 1: ImpactControlCenter - Setting up the HRTC usage

Following trigger modes can be used with HRTC:

- OnLowLevel
- OnHighLevel
- OnFallingEdge
- OnRisingEdge
- OnHighExpose

Further details about the mode are described in the API documentation:

See also

TCameraTriggerMode and TCameraTriggerSource in

- **"Enumerations (C developers)"**
- **"CameraSettingsBlueFOX (C++ developers)"**

In the Use Cases chapter there are the following HRTC sample:

- **"Using single camera"** :
  - Achieve a defined image frequency (HRTC)
  - Delay the external trigger signal (HRTC)
  - Creating double acquisitions (HRTC)
  - Take two images after one external trigger (HRTC)
  - Take two images with different expose times after an external trigger (HRTC)
- **"Using multiple cameras"** :
  - Delay the expose start of the following camera (HRTC)

14 Developing applications using the Impact Acquire SDK

The Impact Acquire SDK is a comprehensive software library that can be used to develop applications using the devices described in this manual. A wide variety of programming languages is supported.

For developers using C, C++, .NET, Python or Java, separate API descriptions can be found on the Balluff website:

- Impact Acquire C API
- Impact Acquire C++ API
- Impact Acquire Java API
- Impact Acquire .NET API
- Impact Acquire Python API

Compiled versions (CHM format) might already be installed on your system. These manuals contain chapters on

- how to link and build applications using Impact Acquire
- how the log output for devices is configured and how it works in general
- how to create your own installation packages for Windows and Linux
- a detailed API documentation
- etc.

15 DirectShow Interface

> **Note**
>
> **DirectShow can only be used in combination with the Microsoft Windows operating system**.
>
> Since Windows Vista, **Movie Maker** does not support capturing from a device registered for DirectShow anymore.

This is the documentation of the Balluff DirectShow_acquire interface. A Balluff specific property interface based on the IKsPropertySet has been added. All other features are related to standard DirectShow programming.

15.1 Supported Interfaces

- **IAMCameraControl**
- **IAMDroppedFrames**
- **IAMStreamConfig**
- **IAMVideoProcAmp**
- **ISpecifyPropertyPages**
- **IKsPropertySet** (for further information please refer to the Microsoft DirectX 9.0 Programmer's Reference). Supported property set GUID's:
  - AMPROPERTY_PIN_CATEGORY
  - DIRECT_SHOW_ACQUIRE_PROPERTYSET

15.1.1 C++ Example Code Using the IKsPropertySet Interface

This section provides a small C++ code snippet showing how the DirectShow interface could be used e.g. to query all properties from a device or how to set the value of a certain property

```
#include <DSImpactAcquire/Include/DirectShowAcquire.h>

//-----------------------------------------------------------------------------
// macro for writing the properties, feel free to replace the macros in the code
//-----------------------------------------------------------------------------
#define _WRITE_STRING_PROPERTY(iksprop,prop,str)\
    if( prop != 0 )\
    {\
        prop->value.s = str;\
        hr = iksprop->Set(DIRECT_SHOW_ACQUIRE_PROPERTYSET, MVPropIDWrite, NULL, 0, prop,
      sizeof(MVProperty));\
    }

//-----------------------------------------------------------------------------
#define _WRITE_FLOAT_PROPERTY(iksprop,prop,fval)\
    if( prop != 0 )\
    {\
        prop->value.f = fval;\
        hr = iksprop->Set(DIRECT_SHOW_ACQUIRE_PROPERTYSET, MVPropIDWrite, NULL, 0, prop,
      sizeof(MVProperty));\
    }

//-----------------------------------------------------------------------------
#define _DELETE_POINTER_ARRAY(prop_array,cnt)\
```

```
    for( unsigned long pr = 0; pr < cnt; pr++ )\
    {\
        if( prop_array[pr].stringArraySize > 0 )\
            delete [] prop_array[pr].stringArray;\
    }

//-----------------------------------------------------------------------------
// scanning the property list props for a property with name property_name
MVProperty* GetProperty( MVProperty* props, unsigned int property_count, char* property_name )
//-----------------------------------------------------------------------------
{
    for( unsigned int pr = 0; pr < property_count && props != 0; pr++ )
    {
        // please feel free to replace standard c string handling to Standard C++ Library strings
        if( strcmp( props[pr].propertyName, property_name ) == 0 )
        {
            return &props[pr];
        }
    }

    return 0;
}

//-----------------------------------------------------------------------------
// asking the ds-interface for the property list
unsigned long ScanAllProperties( IAMStreamConfig* pvsc, MVProperty* props, unsigned int property_count )
//-----------------------------------------------------------------------------
{
    IKsPropertySet* ksProp;
    unsigned long propCnt = 0;
    // try to get the ikspropertyset pointer
    HRESULT hr = pvsc->QueryInterface( IID_IKsPropertySet, ( void** )&ksProp );

    if( SUCCEEDED( hr ) )
    {
        DWORD cbReturned;
        ZeroMemory( props, property_count * sizeof( MVProperty ) );
        // asking for the property interface. If one could be found then fill the propertylist props
        hr = ksProp->Get( DIRECT_SHOW_ACQUIRE_PROPERTYSET, MVPropIDReadProperties, NULL, 0,
                          props, property_count * sizeof( MVProperty ), &cbReturned );

        //if successful returned then our list shows something like this
        //props[0x0]    {propertyName="IOSubSystem/DigitalInputThreshold" value={s= "2V" }...
        //props[0x1]    {propertyName="IOSubSystem/DigitalInputs" value={s= "Off" }...
        //props[0x2]    {propertyName="IOSubSystem/DigitalOutputs" value={s= "Off" }...
        //props[0x3]    {propertyName="IOSubSystem/HardwareRealTimeController/HRTCtrl_0/Filename" value={s=
      "default.rtp" }...
        //props[0x4]    {propertyName="IOSubSystem/HardwareRealTimeController/HRTCtrl_0/Mode" value={s=
      "Stop" } ...
        //props[0x5]    {propertyName="IOSubSystem/HardwareRealTimeController/HRTCtrl_0/ProgramSize"
      value={i=0x5 }...
        //...
        //props[0x40]   {propertyName="ImagingSubsystem/Setting/Base/Camera/AutoExposeControl" value={s=
      "Off" } ...
        //props[0x41]   {propertyName="ImagingSubsystem/Setting/Base/Camera/AutoGainControl" value={s= "Off"
      } ...
        //props[0x42]   {propertyName="ImagingSubsystem/Setting/Base/Camera/BinningMode" value={s=  "Off" }
      ...
        //props[0x43]   {propertyName="ImagingSubsystem/Setting/Base/Camera/ExposeMode" value={s= "Standard"
      } ...
        //props[0x44]   {propertyName="ImagingSubsystem/Setting/Base/Camera/Expose_us" value={i=0xea60 } ...
        //props[0x45]   {propertyName="ImagingSubsystem/Setting/Base/Camera/FlashMode" value={s= "Off" } ...
        //props[0x46]   {propertyName="ImagingSubsystem/Setting/Base/Camera/FrameDelay_us" value={i=0x0 }
      ...
        //props[0x47]   {propertyName="ImagingSubsystem/Setting/Base/Camera/Gain_dB" value={ f=0.0 } ...

        propCnt = cbReturned / sizeof( MVProperty );
        for( DWORD pr = 0; pr < propCnt; pr++ )
        {
            //asking for the actual values of all properties
            hr = ksProp->Get( DIRECT_SHOW_ACQUIRE_PROPERTYSET, MVPropIDRead, NULL, 0, &props[pr], sizeof(
      props[pr] ), &cbReturned );

            //if stringArraySize is greater than 0 this property is from type stringarray.
            if( props[pr].stringArraySize > 0 )
            {
                //allocate some buffer for the stringArray
                props[pr].stringArray = new const char* [props[pr].stringArraySize];

                //fill the allocate stringArray with references to the pointers from the internal
      properties,
                //do not change or free these pointers
                hr = ksProp->Get( DIRECT_SHOW_ACQUIRE_PROPERTYSET, MVPropIDReadStringArray, NULL, 0,
      &props[pr], sizeof( props[pr] ), &cbReturned );
            }
        }
        ksProp->Release();
```

```
    }
    return propCnt;
}

//-----------------------------------------------------------------------------
unsigned long functionXY( ..., bool bBayerOFF, double gain )
//-----------------------------------------------------------------------------
{

    // ...

    IKsPropertySet* ks_prop;
    hr = gcap.pVSC->QueryInterface( IID_IKsPropertySet, ( void** )&ks_prop );

    if( SUCCEEDED( hr ) )
    {
        const int max_prop = 500;
        MVProperty props[max_prop];
        unsigned int propCnt = ScanAllProperties( gcap.pVSC, props, max_prop );

        // if successful we get a list something like this otherwise propCnt = 0
        //...
        //props[0x40]   {propertyName="ImagingSubsystem/Setting/Base/Camera/AutoExposeControl" value={s=
      "Off" } ...
        //props[0x41]   {propertyName="ImagingSubsystem/Setting/Base/Camera/AutoGainControl" value={s= "Off"
      } ...
        //props[0x42]   {propertyName="ImagingSubsystem/Setting/Base/Camera/BinningMode" value={s=  "Off" }
      ...
        //props[0x43]   {propertyName="ImagingSubsystem/Setting/Base/Camera/ExposeMode" value={s= "Standard"
      } ...
        //props[0x44]   {propertyName="ImagingSubsystem/Setting/Base/Camera/Expose_us" value={i=0xea60 } ...
        //props[0x45]   {propertyName="ImagingSubsystem/Setting/Base/Camera/FlashMode" value={s= "Off" } ...
        //props[0x46]   {propertyName="ImagingSubsystem/Setting/Base/Camera/FrameDelay_us" value={i=0x0 }
      ...
        //props[0x47]   {propertyName="ImagingSubsystem/Setting/Base/Camera/Gain_dB" value={ f=0.0 } ...
        //...
        // it is strongly recommended to work only with the propertyNames and not with the index from the
      list.
        // The driver for our cameras and grabber will always return same propertyNames.
        MVProperty*  gain_prop = GetProperty( props, propCnt, "ImagingSubsystem/Setting/Base/Camera/Gain_dB"
      );
        _WRITE_FLOAT_PROPERTY( ks_prop, gain_prop, gain );

        MVProperty*  bayer_prop = GetProperty( props, propCnt,
      "ImagingSubsystem/Setting/Base/ImageProcessing/ColorProcessing" );
        // The property ColorProcessing is from type stringArray. Please have a look to the stringArray for
      the different strings.
        _WRITE_STRING_PROPERTY( ks_prop, bayer_prop, bBayerOFF ? "Raw" : "Auto" );

        // delete the stringarray
        _DELETE_POINTER_ARRAY( props, propCnt );
        ks_prop->Release();
    }

    // ...

}
```

15.2 Logging

The DirectShow_acquire logging procedure is equal to the logging of the Balluff products which uses Impact Acquire. The log output itself is based on **XML**.

If you want more information about the logging please have a look at the **Logging** chapter of the respective Impact Acquire API manual or read on how to configure the log-output using DeviceConfigure in the **"Impact Acquire GUI Applications"** manual.

15.3 Setting up Devices For DirectShow Usage

In order to be able to access a device through the Impact Acquire driver stack from an application through Direct↩ Show a registration procedure is needed. This can either be done using DeviceConfigure or by a command line tool that is part of the Windows operating system.

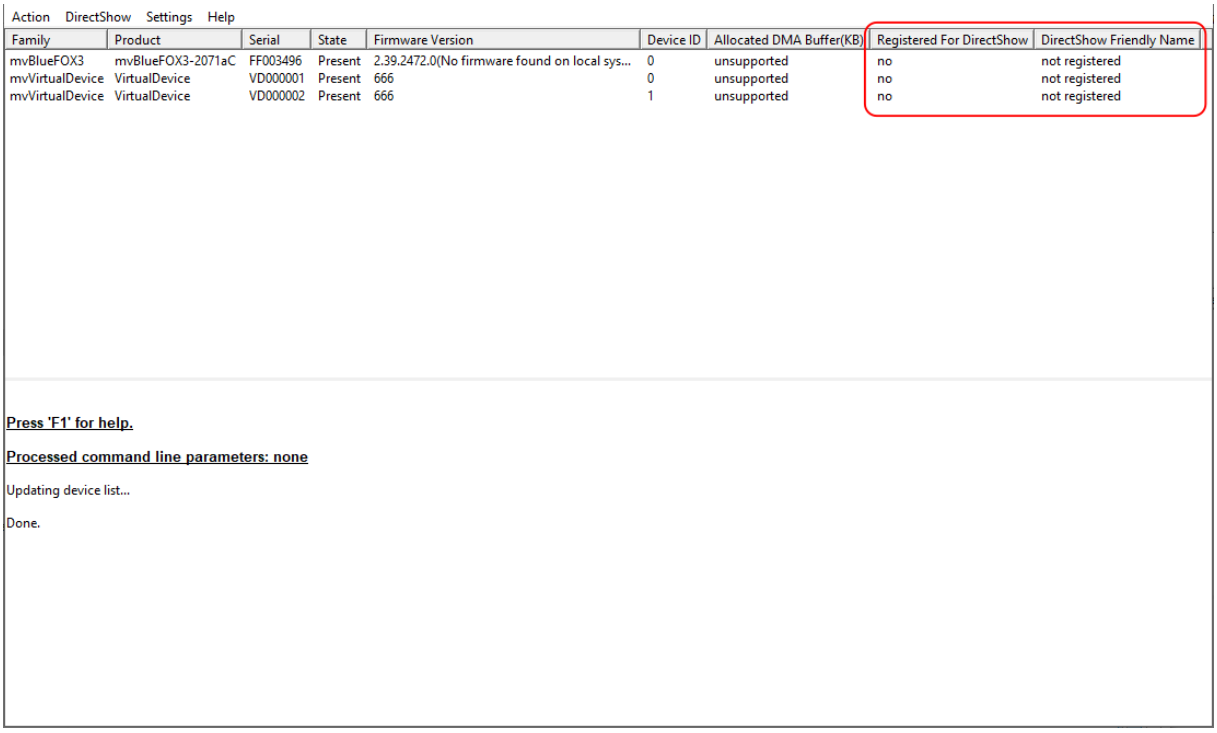
Note

Please be sure to register the device for DirectShow with a matching version of DeviceConfigure. I.e. if you have installed the 32-bit version of the VLC Media Player, Virtual Dub, etc., you have to register devices with the 32-bit version of DeviceConfigure (*"C:/Program Files/Balluff/Impact Acquire/bin"*, the 64-bit version resides in *"C:/Program Files/Balluff/Impact Acquire/bin/x64"*)!

15.3.1 Registering Devices

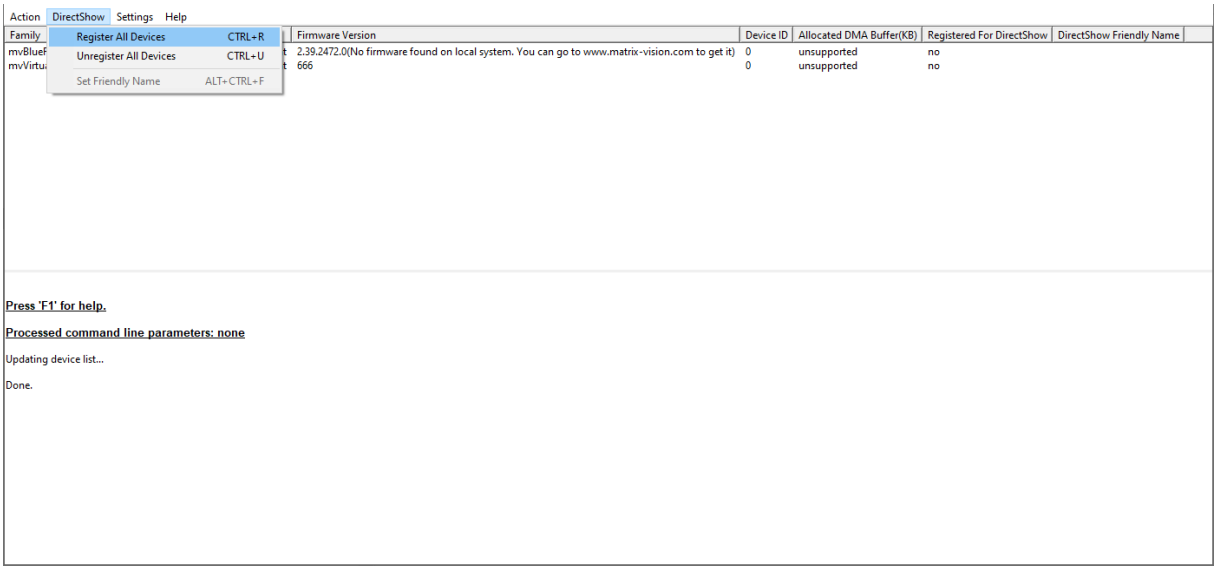
15.3.1.1 Using DeviceConfigure To register all devices currently recognized by the Impact Acquire driver stack for access with DirectShow the following registration procedure is needed:

1. DeviceConfigure needs to be started (with elevated rights).
   If no device has been registered the application will more or less (depending on the installed devices) look like this.



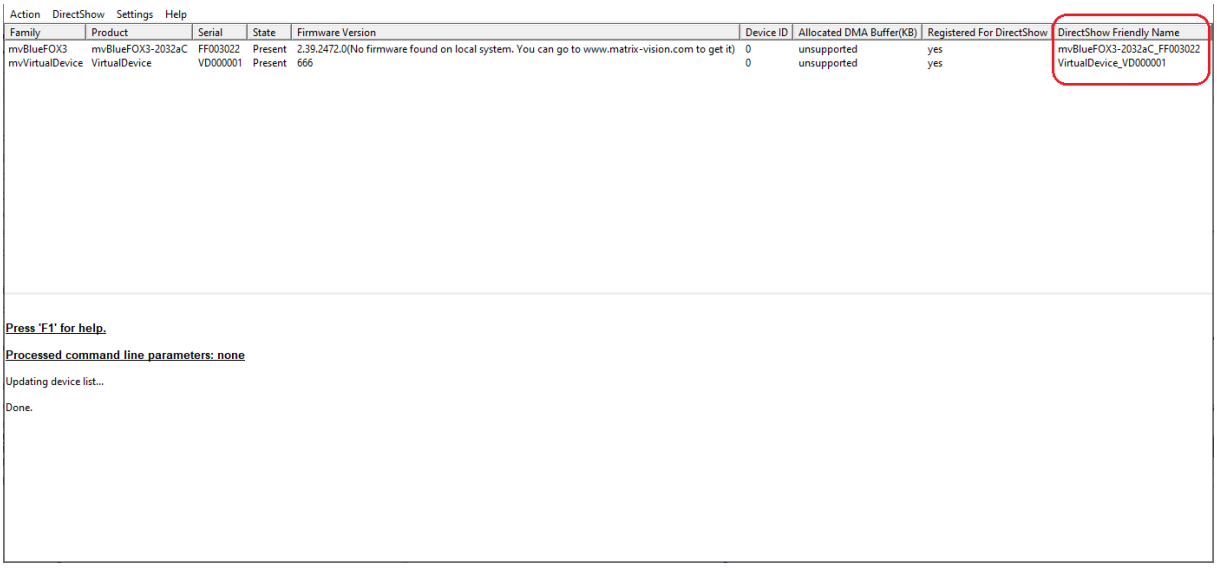
DeviceConfigure - After Start

2. To register every installed device for DirectShow access click on the menu item **"DirectShow"** → **"Register All Devices"**.



DeviceConfigure - Register All Devices

3. After a successful registration the column "Registered For DirectShow" will display **"yes"** for every device and the devices will be registered with a default DirectShow friendly name which is displayed in the **"DirectShow Friendly Name"** column.



DeviceConfigure - All Devices Registered For DirectShow Access

15.3.1.2 Using regsvr32 To register all devices currently recognized by the Impact Acquire driver stack with auto-assigned names, the Windows tool "`regsvr32`" can be used from an elevated command shell.

The following command line options are available and can be passed during the silent registration:

EXAMPLES:

Register ALL devices that are recognized by Impact Acquire (this will only register devices which have drivers installed) without any user interaction:

```
regsvr32 <path>\DirectShow_acquire.ax /s
```

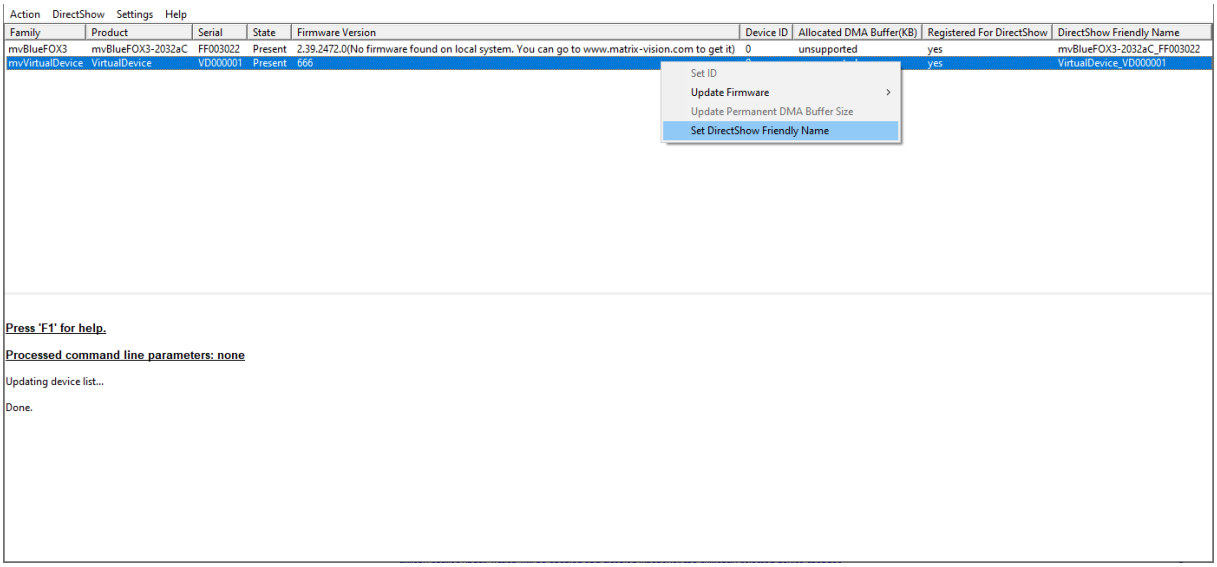
Unregister ALL devices that have been registered before without any user interaction:

```
regsvr32 <path>\DirectShow_acquire.ax /u /s
```

15.3.2 Renaming Devices

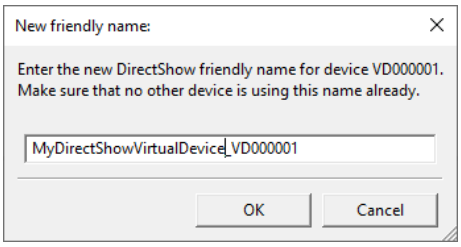
To modify the DirectShow friendly name of a device:

1. DeviceConfigure needs to be started (with elevated rights).
2. right-click on the device to rename and select **"Set DirectShow Friendly Name"**:



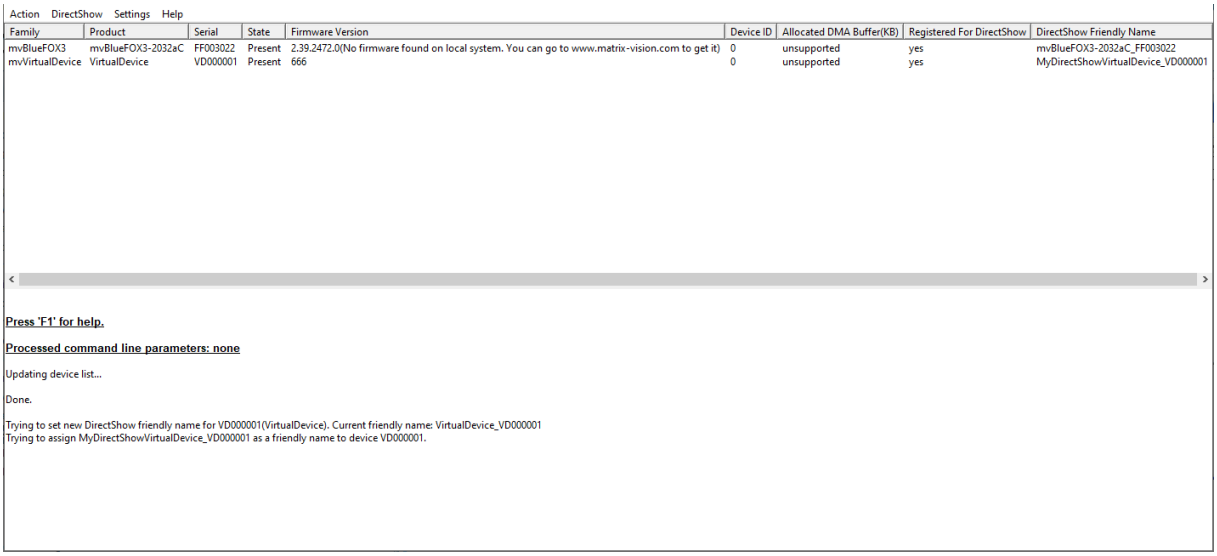
DeviceConfigure - Set DirectShow Friendly Name

3. Then, a dialog will appear. Please enter the new name and confirm it with **"OK"**.



DeviceConfigure - Dialog For New Name

4. Afterwards the column "DirectShow friendly name" will display the newly assigned friendly name.



DeviceConfigure - Renamed Device

Note

Please do not select the same friendly name for two different devices. In theory this is possible, however the DeviceConfigure GUI will not allow this to avoid confusion.

15.4 DirectShow-based Applications

Applications like the VLC Media Player or VirtualDub can be used to capture images via DirectShow. VirtualDub is a good choice if you want full control over the camera, while VLC Media Player decides some of the DirectShow configuration under the hood, which may lead to undesirable results. (See Wrong Colors in the VLC Media Player)

Note

Some Windows tools like Teams and Outlook also try to get hold of available DirectShow video capture devices. Keep this in mind if you experience strange behavior of any DirectShow device.

16 Troubleshooting

16.1 Accessing log files

If you need support using our products, you can shorten response times by sending us your log files. Accessing the log files is different in Windows and Linux:

16.1.1 Windows

Since

Version 2.11.9 of the host driver stack

You can access the log files in Windows using ImpactControlCenter. The way to do this is described in **"Accessing Log Files"** in the **"Impact Acquire SDK GUI Applications"** manual.

16.1.2 Linux

Since

Version 2.24.0 of the host driver stack

You can access the log files in Linux via `/opt/mvIMPACT_Acquire/data/logs`.

You can also extract the directory using the following command

```
env | grep MVIMPACT_ACQUIRE_DATA_DIR
```

or change the directory directly via

```
cd $MVIMPACT_ACQUIRE_DATA_DIR/logs
```

For older versions:

Like on Windows, log files will be generated, if the activation flag for logging called **mvDebugFlags.mvd** is available in the same folder as the application (however, using Windows log files will be generated automatically, because the applications are started from the same folder). By default, on Linux the **mvDebugFlags.mvd** will be installed in the installation's destination folder in the sub-folder `"apps"`. For example, if the destination folder was `"/home/workspace"`, you can locate the **mvDebugFlags.mvd** like the following way:

```
user@linux-desktop:~$
// <- Starting the console window, you will be in the home directory: /home/
user@linux-desktop:~$ cd workspace/apps/
// <- Change the directory
user@linux-desktop:/home/workspace/apps$ ls -l
// <- List the directory
insgesamt 144
drwxr-xr-x  9 user user  4096 Mai 21 15:08 Callback
drwxr-xr-x  8 user user  4096 Mai 21 15:08 Callback_C
drwxr-xr-x  9 user user  4096 Mai 21 15:08 CaptureToUserMemory_C
drwxr-xr-x  3 user user  4096 Mai 21 15:03 Common
drwxr-xr-x 11 user user  4096 Mai 21 15:09 ContinuousCapture
drwxr-xr-x  9 user user  4096 Mai 21 15:09 ContinuousCaptureAllDevices
drwxr-xr-x  6 user user  4096 Mai 21 15:09 ContinuousCaptureFLTK
drwxr-xr-x  9 user user  4096 Mai 21 15:09 ContinuousCapture_C
drwxr-xr-x 11 user user  4096 Mai 21 15:09 DigitalIOs
drwxr-xr-x 11 user user  4096 Mai 21 15:09 GenericInterfaceLayout
drwxr-xr-x 11 user user  4096 Mai 21 15:09 GenICamInterfaceLayout
-rw-r--r--  1 user user   854 Mai 21 15:03 Makefile
-rw-r--r--  1 user user  7365 Mai 21 15:03 Makefile.samp.inc
-rw-r--r--  1 user user 20713 Mai 21 15:03 mvDebugFlags.mvd
// <- Log activation flag
drwxr-xr-x  7 user user  4096 Mai 21 15:09 DeviceConfigure
drwxr-xr-x  6 user user  4096 Mai 21 15:10 IPConfigure
drwxr-xr-x  6 user user  4096 Mai 21 15:11 ImpactControlCenter
drwxr-xr-x  9 user user  4096 Mai 21 15:11 SingleCapture
drwxr-xr-x  9 user user  4096 Mai 21 15:11 SingleCaptureStorage
```

For log file generation you have to execute your app from the folder where **mvDebugFlags.mvd** is located. E.g. if you want to start ImpactControlCenter:

```
user@linux-desktop:/home/workspace/apps$ ./ImpactControlCenter/x86_64/ImpactControlCenter
// <- Start the executable from the folder, where \b mvDebugFlags.mvd is located.
```

Another possibility would be, to copy the **mvDebugFlags.mvd** file to the folder of the executable:

```
user@linux-desktop:/home/workspace/apps$ cp mvDebugFlags.mvd ./ImpactControlCenter/x86_64/ImpactControlCenter
// <- Copy the log activation flag
user@linux-desktop:/home/workspace/apps$ cd ./ImpactControlCenter/x86_64/
// <- Change the directory
user@linux-desktop:/home/workspace/apps/ImpactControlCenter/x86_64/$ ./ImpactControlCenter
// <- Start the executable
```

Afterwards, several log files are generated which are listed in `files.mvloglist`. The log files have the file extension .mvlog. Please send these files to our support team.

16.2 VLC Media Player Issues

16.2.1 Wrong Colors in the VLC Media Player

When using the **DirectShow_Acquire** capture source with VLC Media Player **3**, wrong colors can be seen in the camera images.

This is a known bug in VLC Media Player **3**, which will apparently be fixed in version 4.x, but probably not in version 3.x. The effect results from VLC interpreting the normal 24-bit video format delivered by *all* DirectShow Video Capture sources (=RV24, MEDIASUBTYPE_RGB24) as a different pixel layout and consequently swapping the red and the blue channels.

As a workaround, the image format of the **DirectShow_Acquire** capture source can be changed before starting streaming. To do this, follow the steps according to Changing Camera Properties and change the Image↩ Destination/PixelFormat from "RGB888Packed" to "BGR888Packed".

After clicking "OK" in the property dialog, choose your desired activity, and the camera will show correct colors.

> **Note**
>
> The PixelFormat used for streaming can only be changed manually **for a recording/playback session, but not permanently**.

16.2.1.1 Background Changing the device's property ImageDestination/PixelFormat from "RGB888Packed" to "BGR888Packed" might seem odd if you are used to the PixelFormatNamingConvention or have already delved into details about in-memory image format storage.

The 24-bit Windows RGB format defines pixels of blue, green and red color in this order, so logically, this is a "BGR" format. Due to historical reasons, the Impact Acquire image format "RGB888Packed" corresponds to this standard Windows bitmap format.

The PixelFormat property of the camera, on the other hand, conforms to the GenICam Pixel Format Naming Convention, and would be described there as "BGR8".

17 Error code list

Numerical Value	String Representation	Brief Description	Detailed Description
-2000	PROPHANDLING_NOT_↩ A_LIST	This component is not a list.	This component is not a list. A list operation for this component has been called but this component does not reference a list.
-2001	PROPHANDLING_NOT_↩ A_PROPERTY	This component is not a property.	This component is not a property. A property operation for this component has been called but this component does not reference a property.
-2002	PROPHANDLING_NOT_↩ A_METHOD	This component is not a method.	This component is not a method. A method operation for this component has been called but this component does not reference a method.
-2003	PROPHANDLING_NO_↩ READ_RIGHTS	The caller has no read rights for this component.	The caller has no read rights for this component. It has been tried to read data from this component, but the caller has no read rights for this component.
-2004	PROPHANDLING_NO_↩ WRITE_RIGHTS	The caller has no write rights for this component.	The caller has no write rights for this component. It has been tried to modify data of this component, but the caller has no write rights for this component.

-2005	PROPHANDLING_NO_↩ MODIFY_SIZE_RIGHTS	The caller can't modify the size of this component.	The caller can't modify the size of this component. It has been tried to modify the size of this list or the number of values stored by a property, but the caller doesn't have the required right to do this. This error will also be reported if the user tried to increase the number of values handled by a property above the maximum number of values it can handle. Therefore before resizing a property check if the new size might exceeds this maximum value by calling the appropriate function.
-2006	PROPHANDLING↩ _INCOMPATIBLE_↩ COMPONENTS	The two involved components are not compatible.	The two involved components are not compatible. An operation requiring two compatible components has been called with two components, which are not compatible.
-2008	PROPHANDLING↩ _UNSUPPORTED_↩ PARAMETER	One or more of the specified parameters are not supported by the function.	One or more of the specified parameters are not supported by the function. This error might also be generated if a certain feature is not available on the current platform.
-2009	PROPHANDLING_SIZE_↩ MISMATCH	Different sized value buffers have been passed.	Different sized value buffers have been passed. While trying to read value pairs the caller passed two different sized value buffers to a function while one is too small to hold all the information.
-2010	PROPHANDLING_↩ IMPLEMENTATION_↩ MISSING	A feature that is not implemented so far has been requested.	A feature that is not implemented so far has been requested. The caller requested a feature, that hasn't been implemented so far. This error code is only provided for compatibility and will be set in very rare cases only.

-2011	PROPHANDLING↩ _ACCESSTOKEN_↩ CREATION_FAILED	An access token object couldn't be created.	An access token object couldn't be created. This can either happen, because the caller has not the rights required to create an access token or because the system runs very low on memory. **Deprecated:** This error code currently is not used anywhere within this framework. It might be removed in a future version.
-2012	PROPHANDLING_↩ INVALID_PROP_VALUE	It has been tried to assign an invalid value to a property.	It has been tried to assign an invalid value to a property. This can either happen if the value lies above or below the min. or max. value for a property or when it has been tried to write a value to a property, which is not in the properties translation dictionary (if it defines one). To find out, which values are allowed for the property in question the user should • Check if the property defines a translation dictionary. • Check the allowed values within a translation dictionary if one is defined. • Check the min and max value for properties, that define limits.
-2013	PROPHANDLING_PROP↩ _TRANSLATION_TABLE↩ _CORRUPTED	The properties translation table has been corrupted.	The properties translation table has been corrupted. The properties translation table has been corrupted for an unknown reason and can't be used anymore.

-2014	PROPHANDLING_PROP↩ _VAL_ID_OUT_OF_↩ BOUNDS	Invalid value index.	Invalid value index. The caller tried to read a value from an invalid index from a property. Most properties store one value only, thus the only valid positive value index will be 0 (some negative index values are reserved for special values like e.g. the min/max value of a property). However some properties might store more than one value, thus the max. allowed index might be higher. The highest index allowed will always be the value count of a property minus one for properties with the **mvIMPACT↩ ::acquire::cfFixedSize** flag set. Other properties will automatically adjust the size once the user writes to an index out of bounds.
-2015	PROPHANDLING_PROP↩ _TRANSLATION_TABLE↩ _NOT_DEFINED	This property doesn't define a translation table.	This property doesn't define a translation table. The caller tried to modify a translation table, that hasn't been defined for this property.
-2016	PROPHANDLING_↩ INVALID_PROP_VALUE_↩ TYPE	An invalid value has been passed to the property.	An invalid value has been passed to the property. Although properties are quite tolerant regarding the allowed assignment for them some value types can't be used to write all properties. As an example assigning a float value to an integer property would result in this error. Another reason for this error might be when a user tried to access e.g. a float property with functions meant to be used for int properties.
-2017	PROPHANDLING_PROP↩ _VAL_TOO_LARGE	A too large value has been passed.	A too large value has been passed. One or more of the values the caller tried to write to the property are larger than the max. allowed value for this property.

-2018	PROPHANDLING_PROP↩ _VAL_TOO_SMALL	A too small value has been passed.	A too small value has been passed. One or more of the values the caller tried to write to the property are smaller than the min. allowed value for this property.
-2019	PROPHANDLING_↩ COMPONENT_NOT_↩ FOUND	The specified component could not be found.	The specified component could not be found.
-2020	PROPHANDLING_LIST_↩ ID_INVALID	An invalid list has been referenced.	An invalid list has been referenced.
-2021	PROPHANDLING_↩ COMPONENT_ID_INVALID	An invalid component within a list has been referenced.	An invalid component within a list has been referenced.
-2022	PROPHANDLING_LIST_↩ ENTRY_OCCUPIED	The specified list index is occupied.	The specified list index is occupied. During the creation of a new component the caller tried the insert the newly created component into a list at a position already used to store another component.
-2023	PROPHANDLING_↩ COMPONENT_HAS_↩ OWNER_ALREADY	The specified component already has an owner.	The specified component already has an owner. The caller tried to assign an owner to a component that already has an owner. An owner once defined can't be modified anymore.
-2024	PROPHANDLING↩ _COMPONENT_↩ ALREADY_REGISTERED	It has been tried to register the same component at twice in the same list.	It has been tried to register the same component at twice in the same list.
-2025	PROPHANDLING_LIST_↩ CANT_ACCESS_DATA	The desired data can't be accessed or found.	The desired data can't be accessed or found. During loading or saving data this error can occur e.g. if it has been tried to import a setting from a location where the desired setting couldn't be found. Another reason for this error might be that the current user is not allowed to perform a certain operation on the desired data (e.g. a user tries to delete a setting that is stored with global scope but does not have elevated access rights).
-2026	PROPHANDLING_↩ METHOD_PTR_INVALID	The function pointer of the referenced method object is invalid.	The function pointer of the referenced method object is invalid.
-2027	PROPHANDLING_↩ METHOD_INVALID_↩ PARAM_LIST	A method object has an invalid parameter list.	A method object has an invalid parameter list.

-2028	PROPHANDLING_SWIG↩ _ERROR	This indicates an internal error occurred within the SWIG generated wrapper code, when working under Python.	This indicates an internal error occurred within the SWIG generated wrapper code, when working under Python.
-2029	PROPHANDLING↩ _INVALID_INPUT_↩ PARAMETER	A invalid input parameter has been passed to a function of this module.	A invalid input parameter has been passed to a function of this module. In most cases this might be a unassigned pointer, where a valid pointer to a user defined storage location was expected.
-2030	PROPHANDLING_↩ COMPONENT_NO_↩ CALLBACK_REGISTERED	The user tried to modify a registered callback, but no callback has been registered for this component.	The user tried to modify a registered callback, but no callback has been registered for this component.
-2031	PROPHANDLING_INPUT↩ _BUFFER_TOO_SMALL	The user tried to read data into a user supplied storage location, but the buffer was too small to accommodate the result.	The user tried to read data into a user supplied storage location, but the buffer was too small to accommodate the result.
-2032	PROPHANDLING_↩ WRONG_PARAM_COUNT	The number of parameters is incorrect.	The number of parameters is incorrect. This error might occur if the user called a function with a variable number of input or output parameters and the number of parameters passed to the function does not match the number of required parameters.
-2033	PROPHANDLING↩ _UNSUPPORTED_↩ OPERATION	The user tried to execute an operation, which is not supported by the component he is referring to.	The user tried to execute an operation, which is not supported by the component he is referring to.
-2034	PROPHANDLING_CANT↩ _SERIALIZE_DATA	The user tried to save(serialize) a property list without having the right to do this.	The user tried to save(serialize) a property list without having the right to do this.
-2035	PROPHANDLING_↩ INVALID_FILE_CONTENT	The user tried to use a file to update or create a component list, that does not contain valid data for this operation.	The user tried to use a file to update or create a component list, that does not contain valid data for this operation. This e.g. might happen, if the file does not contain valid XML data or XML data that is not well formed.
-2036	PROPHANDLING_CANT↩ _ALLOCATE_LIST	This error will occur when the modules internal representation of the tree structure does not allow the allocation of a new list.	This error will occur when the modules internal representation of the tree structure does not allow the allocation of a new list. In this case either new list can't be allocated. The only way to solve this problem is to delete another list.

-2037	PROPHANDLING_↩ CANT_REGISTER_↩ COMPONENT	The referenced list has no space left to register this component at the desired position.	The referenced list has no space left to register this component at the desired position. There might however be an empty space within the list where this element could be registered, but no more components can be registered at the end of this list.
-2038	PROPHANDLING_PROP↩ _VALIDATION_FAILED	The user tried to assign a value to a property, that is invalid.	The user tried to assign a value to a property, that is invalid. This will result in a detailed error message in the log-file. This error might arise e.g. when a string property doesn't allow the string to contain numbers. In this case trying to set the properties value to 'blabla7bla' would cause this error.
-2099	PROPHANDLING_LAST↩ _VALID_ERROR_CODE	Defines the last valid error code value for the property module.	Defines the last valid error code value for the property module.
-2100	DMR_DEV_NOT_FOUND	The specified device can't be found.	The specified device can't be found. This error occurs either if an invalid device ID has been passed to the device manager or if the caller tried to close a device which currently isn't initialized.
-2101	DMR_INIT_FAILED	The device manager couldn't be initialized.	The device manager couldn't be initialized. This is an internal error.
-2102	DMR_DRV_ALREADY_IN↩ _USE	The device is already in use.	The device is already in use. This error e.g. will occur if this or another process has initialized this device already and an application tries to open the device once more or if a certain resource is available only once but shall be used twice.
-2103	DMR_DEV_CANNOT_↩ OPEN	The specified device couldn't be initialized.	The specified device couldn't be initialized.
-2104	DMR_NOT_INITIALIZED	The device manager or another module hasn't been initialized properly.	The device manager or another module hasn't been initialized properly. This error occurs if the user tries e.g. to close the device manager without having initialized it before or if a library used internally or a module or device associated with that library has not been initialized properly or if

-2105	DMR_DRV_CANNOT_↩ OPEN	A device could not be initialized.	A device could not be initialized. In this case the log-file will contain detailed information about the source of the problem.
-2106	DMR_DEV_REQUEST_↩ QUEUE_EMPTY	The devices request queue is empty.	The devices request queue is empty. This error e.g. occurs if the user waits for an image request to become available at a result queue without having send an image request to the device before. It might also arise when trying to trigger an image with a software trigger mechanism before the acquisition engine has been completely started. In this case a small delay and then again calling the software trigger function will succeed.
-2107	DMR_DEV_REQUEST_↩ CREATION_FAILED	A request object couldn't be created.	A request object couldn't be created. The creation of a request object failed. This might e.g. happen, if the system runs extremely low on memory.
-2108	DMR_INVALID_↩ PARAMETER	An invalid parameter has been passed to a function.	An invalid parameter has been passed to a function. This might e.g. happen if a function requiring a pointer to a structure has been passed an unassigned pointer or if a value has been passed, that is either too large or too small in that context.
-2109	DMR_EXPORTED_↩ SYMBOL_NOT_FOUND	One or more symbols needed in a detected driver library couldn't be resolved.	One or more symbols needed in a detected driver library couldn't be resolved. In most cases this is an error handled internally. So the user will not receive this error code as a result of a call to an API function. However when the user tries to get access to an **IMPACT** buffer type while the needed **IMPACT Base libraries** are not installed on the target system this error code also might be returned to the user.
-2110	DEV_UNKNOWN_ERROR	An unknown error occurred while processing a user called driver function.	An unknown error occurred while processing a user called driver function.

-2111	DEV_HANDLE_INVALID	A driver function has been called with an invalid device handle.	A driver function has been called with an invalid device handle.
-2112	DEV_INPUT_PARAM_↩ INVALID	A driver function has been called but one or more of the input parameters are invalid.	A driver function has been called but one or more of the input parameters are invalid. There are several possible reasons for this error: • an unassigned pointer has been passed to a function, that requires a valid pointer. • one or more of the passed parameters are of an incorrect type. • one or more parameters contain an invalid value (e.↩ g. a filename that points to a file that can't be found, a value, that is larger or smaller than the allowed values. • within the current setup one or more parameters impose restrictions on the requested operation that don't allow its execution.
-2113	DEV_WRONG_INPUT_↩ PARAM_COUNT	A function has been called with an invalid number of input parameters.	A function has been called with an invalid number of input parameters.
-2114	DEV_CREATE_SETTING↩ _FAILED	The creation of a setting failed.	The creation of a setting failed. This can either happen, when a setting with the same name as the one the user tried to create already exists or if the system can't allocate memory for the new setting.
-2115	DEV_REQUEST_CANT_↩ BE_UNLOCKED	The unlock for a **mv**↩ **IMPACT::acquire::Request** object failed.	The unlock for a **mv**↩ **IMPACT::acquire::Request** object failed. This might happen, if the **mvIMPACT**↩ **::acquire::Request** is not locked at the time of calling the unlock function. It either has been unlocked by the user already or this request has never been locked as the request so far has not been used to capture image data into its buffer. Another reason for this error might be that the user tries to unlock a request that is currently processed by the device driver.

-2116	DEV_INVALID_↩ REQUEST_NUMBER	The number for the **mv**↩ **IMPACT::acquire::Request** object is invalid.	The number for the **mv**↩ **IMPACT::acquire::Request** object is invalid. The max. number for a **mvIMPACT**↩ **::acquire::Request** object is the value of the property *RequestCount* in the **mvIMPACT::acquire**↩ **::SystemSettings** list - 1.
-2117	DEV_LOCKED_↩ REQUEST_IN_QUEUE	A Request that hasn't been unlocked has been passed back to the driver.	A Request that hasn't been unlocked has been passed back to the driver. This error might occur if the user requested an image from the driver but hasn't unlocked the **mvIMPACT::acquire**↩ **::Request** that will be used for this new image.
-2118	DEV_NO_FREE_↩ REQUEST_AVAILABLE	The user requested a new image, but no free **mv**↩ **IMPACT::acquire::Request** object is available to process this request.	The user requested a new image, but no free **mv**↩ **IMPACT::acquire::Request** object is available to process this request.
-2119	DEV_WAIT_FOR_↩ REQUEST_FAILED	The wait for a request failed.	The wait for a request failed. This might have several reasons: • The user waited for an image, but no image has been requested before. • The user waited for a requested image, but the image is still not ready(e.g. because of a short timeout and a long exposure time). • A triggered image has been requested but no trigger signal has been detected within the wait period. • A plug and play device(e.g. an USB device) has been unplugged and therefore can't deliver images anymore. In this case the *'state'* property should be checked to find out if the device is still present or not.
-2120	DEV_UNSUPPORTED_↩ PARAMETER	The user tried to get/set a parameter, which is not supported by this device.	The user tried to get/set a parameter, which is not supported by this device.
-2121	DEV_INVALID_RTC_↩ NUMBER	The requested real time controller is not available for this device.	The requested real time controller is not available for this device.

-2122	DMR_INTERNAL_ERROR	Some kind of internal error occurred.	Some kind of internal error occurred. More information can be found in the ∗.log-file or the debug output.
-2123	DMR_INPUT_BUFFER_↩ TOO_SMALL	The user allocated input buffer is too small to accommodate the result.	The user allocated input buffer is too small to accommodate the result.
-2124	DEV_INTERNAL_ERROR	Some kind of internal error occurred in the device driver.	Some kind of internal error occurred in the device driver. More information can be found in the ∗.log-file or the debug output.
-2125	DMR_LIBRARY_NOT_↩ FOUND	One or more needed libraries are not installed on the system.	One or more needed libraries are not installed on the system.
-2126	DMR_FUNCTION_NOT_↩ IMPLEMENTED	A called function or accessed feature is not available for this device.	A called function or accessed feature is not available for this device.
-2127	DMR_FEATURE_NOT_↩ AVAILABLE	The feature in question is (currently) not available for this device or driver.	The feature in question is (currently) not available for this device or driver. This might be because another feature currently blocks the one in question from being accessible. More information can be found in the ∗.log-file or the debug output.
-2128	DMR_EXECUTION_↩ PROHIBITED	The user is not permitted to perform the requested operation.	The user is not permitted to perform the requested operation. This e.g. might happen if the user tried to delete user data without specifying the required password.
-2129	DMR_FILE_NOT_FOUND	The specified file can't be found.	The specified file can't be found. This might e.g. happen if the current working directory doesn't contain the file specified.
-2130	DMR_INVALID_LICENCE	The licence doesn't match the device it has been assigned to.	The licence doesn't match the device it has been assigned to. When e.g. upgrading a device feature each licence file is bound to a certain device. If the device this file has been assigned to has a different serial number then the one used to create the licence this error will occur.
-2131	DEV_SENSOR_TYPE_↩ ERROR	There is no sensor found or the found sensor type is wrong or not supported.	There is no sensor found or the found sensor type is wrong or not supported.

-2132	DMR_CAMERA_↩ DESCRIPTION_INVALID	A function call was associated with a camera description, that is invalid.	A function call was associated with a camera description, that is invalid. One possible reason might be, that the camera description has been deleted(driver closed?). **Since** 1.5.0
-2133	DMR_NEWER_LIBRARY↩ _REQUIRED	A suitable driver library to work with the device manager has been detected, but it is too old to work with this version of the mvDevice↩ Manager library.	A suitable driver library to work with the device manager has been detected, but it is too old to work with this version of the mv↩ DeviceManager library. This might happen if two different drivers have been installed on the target system and one introduces a newer version of the device manager that is not compatible with the older driver installed on the system. In this case this error message will be written into the log-file together with the name of the library that is considered to be too old. The latest drivers will always be available online under **https://www.↩ balluff.com**. There will always be an updated version of the library considered to be too old for download from here. **Since** 1.6.6
-2134	DMR_TIMEOUT	A general timeout occurred.	A general timeout occurred. This is the typical result of functions that wait for some condition to be met with a timeout among their parameters. More information can be found in the ∗.log-file or the debug output. **Since** 1.7.2

-2135	DMR_WAIT_ABANDONED	A wait operation has been aborted.	A wait operation has been aborted. This e.g. might occur if the user waited for some message to be returned by the driver and the device driver has been closed within another thread. In order to inform the user that this waiting operation terminated in an unusual wait, **mvIMPACT::acquire::↩ DMR_WAIT_ABANDONED** will be returned then. **Since** 1.7.2
-2136	DMR_EXECUTION_↩ FAILED	The execution of a method object or reading/writing to a feature failed.	The execution of a method object or reading/writing to a feature failed. More information can be found in the log-file. **Since** 1.9.0
-2137	DEV_REQUEST_↩ ALREADY_IN_USE	This request is currently used by the driver	This request is currently used by the driver This error may occur if the user tries to send a certain request object to the driver by a call to the corresponding image request function. **Since** 1.10.31
-2138	DEV_REQUEST_↩ BUFFER_INVALID	A request has been configured to use a user supplied buffer, but the buffer pointer associated with the request is invalid.	A request has been configured to use a user supplied buffer, but the buffer pointer associated with the request is invalid. **Since** 1.10.31

-2139	DEV_REQUEST_↩ BUFFER_MISALIGNED	A request has been configured to use a user supplied buffer, but the buffer pointer associated with the request has an incorrect alignment.	A request has been configured to use a user supplied buffer, but the buffer pointer associated with the request has an incorrect alignment. Certain devices need aligned memory to perform efficiently thus when a user supplied buffer shall be used to capture data into this buffer must follow these alignment constraints **Since** 1.10.31
-2140	DEV_ACCESS_DENIED	The requested access to a device could not be granted.	The requested access to a device could not be granted.There are multiple reasons for this error code. Detailed information can be found in the ∗.log-file. **POSSIBLE CAUSES:** • an application tries to access a device exclusively that is already open in another process • a network device has already been opened with control access from another system and the current system also tries to establish control access to the device • an application tried to execute a function that is currently not available • an application tries to write to a read-only location. **Since** 1.10.39

-2141	DMR_PRELOAD_↩ CHECK_FAILED	A pre-load condition for loading a device driver failed.	A pre-load condition for loading a device driver failed. Certain device drivers may depend on certain changes applied to the system in order to operate correctly. E.g. a device driver might need a certain environment variable to exist. When the device manager tries to load a device driver it performs some basic checks to detect problems like this. When one of these checks fails the device manager will not try to load the device driver and an error message will be written to the selected log outputs. **Since** 1.10.52
-2142	DMR_CAMERA_↩ DESCRIPTION_INVALID↩ _PARAMETER	One or more of the camera descriptions parameters are invalid for the grabber it is used with.	One or more of the camera descriptions parameters are invalid for the grabber it is used with. There are multiple reasons for this error code. Detailed information can be found in the ∗.log-file. **POSSIBLE CAUSES↩ :** • The TapsXGeometry or TapsYGeometry parameter of the selected camera description cannot be used with a user defined AOI. • A scan standard has been selected, that is not supported by this device. • An invalid scan rate has been selected. • ... This error code will be returned by frame grabbers only. **Since** 1.10.57

-2143	DMR_FILE_ACCESS_↩ ERROR	A general error returned whenever there has been a problem with accessing a file.	A general error returned whenever there has been a problem with accessing a file. There can be multiple reasons for this error and a detailed error message will be sent to the log-output whenever this error code is returned. **POSSIBLE CAUSES↩ :** • The driver tried to modify a file, for which it has no write access. • The driver tried to read from a file, for which it has no read access. • ... **Since** 1.10.87
-2144	DMR_INVALID_QUEUE_↩ SELECTION	An error returned when the user application attempts to operate on an invalid queue.	An error returned when the user application attempts to operate on an invalid queue. **Since** 1.11.0
-2145	DMR_ACQUISITION_↩ ENGINE_BUSY	An error returned when the user application attempts to start the acquisition engine at a	An error returned when the user application attempts to start the acquisition engine at a **Since** 2.5.3
-2146	DMR_BUSY	An error returned when the user application attempts to perform any operation that currently for any reason cannot be started because something else already running.	An error returned when the user application attempts to perform any operation that currently for any reason cannot be started because something else already running. The log-output will provide additional information. **Since** 2.32.0

-2147	DMR_OUT_OF_MEMORY	An error returned when for any reason internal resources (memory, handles, ...) cannot be allocated.	An error returned when for any reason internal resources (memory, handles, ...) cannot be allocated. The log-output will provide additional information. **Since** 2.32.0
-2199	DMR_LAST_VALID_↩ ERROR_CODE	Defines the last valid error code value for device and device manager related errors.	Defines the last valid error code value for device and device manager related errors.

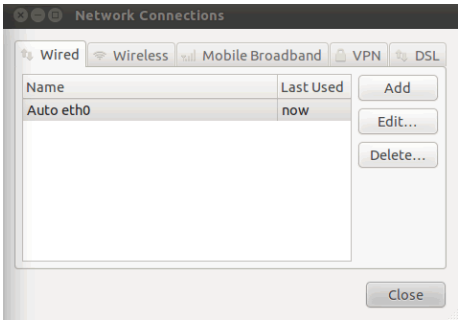
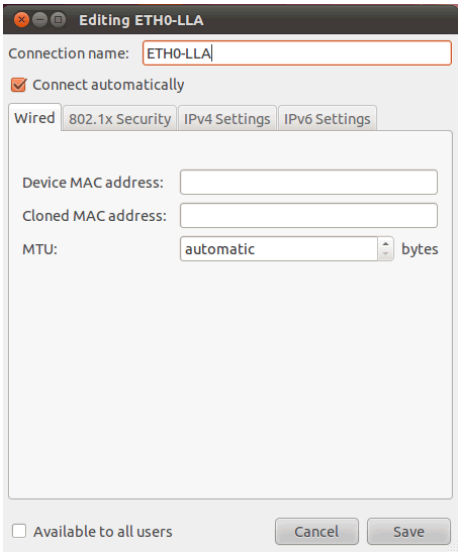
18 Glossary

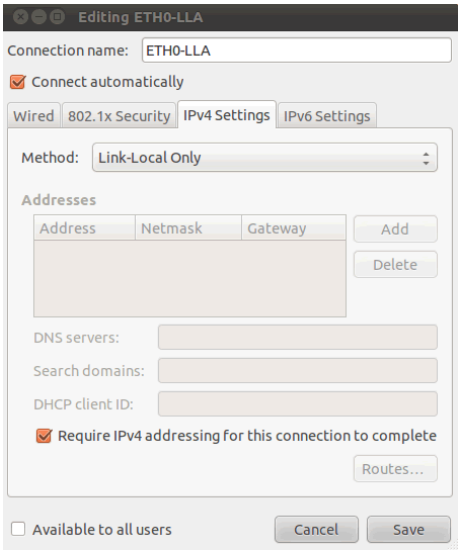
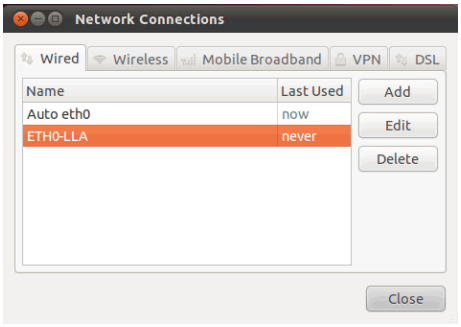
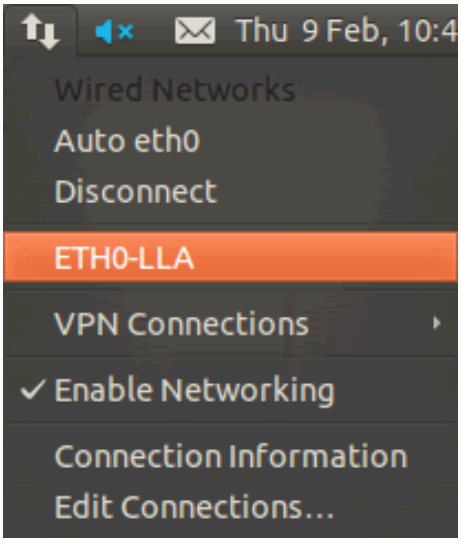
A/D reference	Upper threshold of video signal to be digitized. All values above this limit value are digitized to 255. Increasing the reference level results in contrast deterioration and vice versa.
ADC	Analog-to-digital converter (A/D converter).
API	Application programming interface (API). The standard API for Balluff/↩ MATRIX VISION products is called mvIMPACT_Acquire.
Base address	Starting address from which the memory or register are inserted.
Bpp	Bits per pixel
Bus	A group line via which the various parts of the computer communicate with one another.
BusyBox	Configurable monolithic application including shell and other useful command line tools - often called the "swiss army knife" for embedded systems. Even desktop distributions are sometimes relying on BusyBox due to its robustness. Please see http://www.busybox.net for details.
CCIR	Comité Consulatif International of the Radio Communications European video standard for 50 Hz gray scale.
CIFS	Common Internet file system (CIFS) replaced Samba in 2006. It gets rid of NetBIOS packets an introduced Unix features like soft/hard links and allows larger files.
Clamp signal	Clamp signal means, that a AC coupled video signal is clamped on the porch to get a signal transfer with less noise and independent from the d.c. voltage portion.
CPU	Central processing unit aka processor.
DAC	Digital to analog converter (D/A converter).
Defaults	Standard system settings.
DHCP	Dynamic Host Configuration Protocol (DHCP). DHCP is a protocol used by networked devices (clients) to obtain various parameters necessary for the clients to operate in an Internet Protocol (IP) network.
Digital I/O	Digital inputs and outputs.
External trigger	External event used to initiate image capture.
False colors	Colors are assigned to gray scale via a look-up table. This allows even small gray scale differences can be displayed clearly.
GDB	GDB, the GNU Project debugger.

GenICam	GenICam stands for **GEN**eric programming **I**nterface for **CAM**eras. It's a generic way to access and modify device parameters with a unified interface. A GenICam compliant device either directly provides a GenICam compliant description file (in internal memory) or the description file can be obtained from a local (hard disk etc.) or web location. A GenICam description file is something like a machine readable device manual. It provides a user readable name and value range for parameters that are offered by the device for reading and/or writing and instructions on what command must be sent to a device when the user modifies or reads a certain parameter from the device. These description files are written in XML. An excerpt from such a file can be seen in the figure below:  Excerpt of a GenICam description file (in XML) For further information on this topic please have a look at https://en.wikipedia.org/wiki/GenICam.
GenTL	GenTL is the transport layer interface for cameras, acquiring images from the camera, and moving them to the user application.
Gigabit Ethernet (GigE)	The term Gigabit Ethernet (defined by the IEEE 802.3-2008 standard) represents various technologies for transmitting Ethernet frames at a rate of a gigabit per second (1,000,000,000 bits per second).

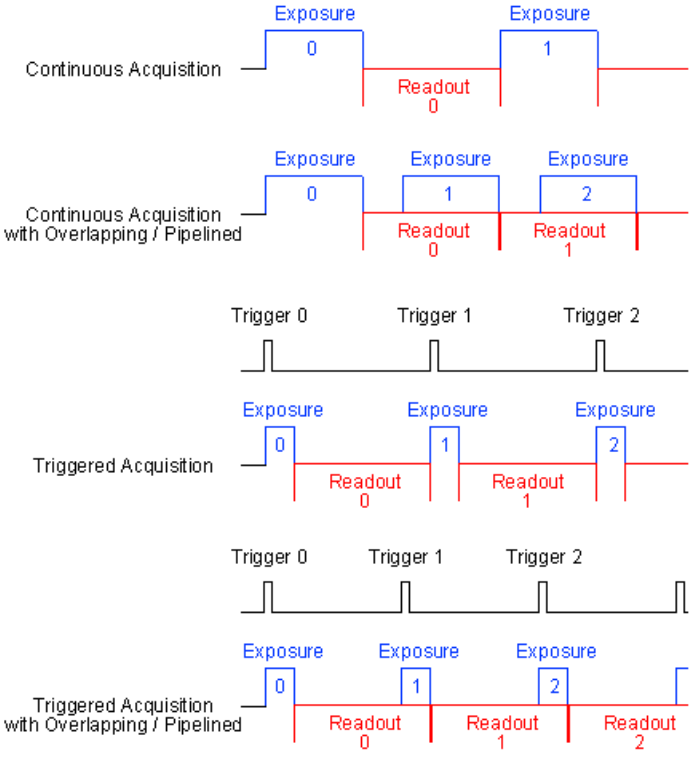
GigE Vision	GigE Vision is a network protocol designed for the communication between an imaging device and an application. This protocol completely describes: • device discovery • data transmission – image data – additional data • read/write of parameters. GigE Vision uses UDP for data transmission to reduce overhead introduced by TCP. **Note** UDP does not guarantee the order in which packets reach the client nor does it guarantee that packets arrive at the client at all. However, GigE Vision defines mechanisms that can detect lost packets. This allows capture driver manufacturers to implement algorithms that can reconstruct images and other data by requesting the device to resend lost data packets until the complete buffer has been assembled. For further information please have a look at https://en.wikipedia.org/wiki/GigE_↩ Vision The Balluff GigE Vision capture filter driver as well as the socket based acquisition driver and all Balluff GigE Vision compliant devices support resending thus lost data can be detected and in most cases reconstructed. This of course can not enhance the max. bandwidth of the transmission line thus if e.g. parts of the transmission line are overloaded for a longer period of time data will be lost anyway. Both capture drivers will allow to fine tune the resend algorithm used internally and both drivers will also provide information about the amount of data lost and the amount of data that was re-requested. This information/configuration will be part of the drivers SDK. More information about it can be found in the corresponding interface description. **Note** On Windows 2000 the filter driver does not support the **"↩ Resend"** mechanism.

High Dynamic Range (HDR)	The HDR (High Dynamic Range) mode increases the usable contrast range. This is achieved by dividing the integration time in two or three phases. The exposure time proportion of the three phases can be set independently. Furthermore, it can be set, how many signal of each phase is charged.
Horizontal sync	The portion of the analog signal which specifies the line end of the video signal.
Host	Here: the PC
HRTC	With a *Hardware Real-Time Controller* (HRTC) built inside the FPGA users can define a PLC like sequence of operating steps to control basic time critical functions like exposure time, image trigger and I/O ports. Timing is hard real-time with sub microsecond high resolution.
IDE	a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, a compiler and/or interpreter, build automation tools, and (usually) a debugger.
Image refresh rate	Number of transferred images per second. Normally specified in Hz (e.g. 70 Hz)
Interrupt	Interrupt signal sent to the processor. The program currently running is interrupted and a predefined function is executed.
IPKG	Itsy package management system originally designed for embedded systems. Please have a look at https://en.↩ wikipedia.org/wiki/Ipkg or a more sophisticated documentation at http://buffalo.nas-central.org/index.php "↩ Overview_of_the_ipkg_package_management_system" Balluff distributes all non-firmware, i.e. optional software as ipk packages.
IRQ	Interrupt request
ISR	Interrupt service routine
JFFS2	JFFS2 is a file system which supports wear leveling. **See also** Sources about the JFFS file system: • http://sources.redhat.com/jffs2/ • http://www.linux-mtd.infradead.↩ org/faq/jffs2.html
Link Aggregation (LAG)	Dual-GigE cameras need a network interface card with two network interfaces. However, both network interfaces have to work as a unit. Link aggregation (LAG) or bonding is the name of the game and has to be supported by the network interface card's driver. With it you can bond the two network interfaces so that the work as one interface.

LLA	Logical link address (LLA) is a type of mechanism to obtain a valid IP address without a DHCP server being present. Whether an IP address is available or not is resolved using address resolution protocol (ARP) packets. If no ARP response is received on a given address it is considered unused and will be assigned to the interface. LLA space is 169.254.x.y, i.e. 16bit netmask yielding 64K possible device addresses. With Linux you have to add LLA as an additional interface. By default, you can find one interface in *Connections:* (This description uses *"Gnome Network Manager"*, however using KDE should be similar)  In *Wired*, you can add interfaces via **Add:** 

	In the tab *"IPv4 Setting"* you have to set **"Link-Local Only"**:  After saving, you will find both connections in the summary:  Now, you can select the wished connection using the left mouse button in the *"Network Manager"* menu. In the LLA case it is just the new created connection: 
Look-up table	Table of assignments. Here, new gray scale or colors are normally assigned to gray scale. Look-up tables can, however, also be used for any other math functions.
LSB	Least significant bit

LUT	Look-up table
MAC address	Media Access Control address (MAC address) is a quasi-unique identifier attached to most network adapters (NICs) in computer networking.
MTU	Maximum transmission unit (MTU) refers to the size (in bytes) of the largest packet that a given layer of a communications protocol can pass onwards. The default MTU for Ethernet is 1500. The optimum for Gigabit Ethernet is 8000 - 12000. Different MTU settings in the same subnet can cause package losses, i.e. **never ever** change the MTU unless you know what you're doing. Changing the MTU to other values than 1500 when using file or web services from other computers are almost always a bug. It's save to increase MTU when working in peer-to-peer mode with both devices sharing the same MTU. Please not that few network cards support 16K, most Gigabit Ethernet cards are limited to 9k, some don't support Jumbo Frames (MTU $>$ 1500) at all.
Monochrome	A single-color (black and white) image
MSB	Most significant bit
Impact Acquire	This driver supplied with Balluff products represents the port between the programmer and the hardware. The driver concept of Balluff provides a standardized programming interface to all image processing products made by Balluff GmbH. The advantage of this concept for the programmer is that a developed application runs without the need for any major modifications to the various image processing products made by Balluff GmbH. You can also incorporate new driver versions, which are available for download free of charge on our website: https://www.balluff.com. The SDK documentation of the Impact Acquire can be found at the manuals section.
Netboot	With netboot you can boot a BVS CA-GX camera over network. This is especially useful when several devices share the same pieces of software, i.e. same root file system, which might be subject to change frequently.
NFS	Network File System (NFS) is a network file system protocol, allowing clients to access files over LAN. Given that you need a NFS server are uncommon on Windows, this protocol best fits for Linux-Linux connections.
NIC	Network interface card - synonym for network controller.

Overlapped / pipelined transfer	By default, the steps exposure and readout out of an image sensor are done one after the other. • By design, CCD sensors support overlap capabilities also combined with trigger (see figure). • In contrast, so-called pipelined CMOS sensors only support the overlapped mode. Even less CMOS sensors support the overlapped mode combined with trigger. Please check the sensor summary. In overlapping mode, the exposure starts the exposure time earlier during readout.  **Note** In overlapped trigger mode, you have to keep in mind the following formula `interval between two trigger events >= (readout time - exposure time)`
Pixels	Picture element
PSE	Power sourcing equipment. The network PoE element that inserts power onto an Ethernet cable.
Pseudo colors	Display of gray scale images in false colors. A corresponding color is assigned to a specific gray scale value.
Resolution	Number of pixels (horizontal x vertical)
ROI/AOI	Region/Area of interest.

SFNC	**S**tandard **F**eature **N**aming **C**onvention of GenICam. **See also** The latest GenICam properties list can be found here: http://www.emva.org/standards-technology/genicam/genicam-downloads/ The file is called "GenICam Standard Features Naming Convention (PDF)"
Shell	In computing, a shell is a piece of software that provides an interface for users. Command-line shells provide a command-line interface (CLI) to the operating system. The primary purpose of the shell is to invoke or "launch" another program; however, shells frequently have additional capabilities such as viewing the contents of directories.
Square pixels	Square-shaped pixels (height-width ratio 1:1)
True color	24-bit true color; 16.7 million colors
USB3 Vision	A closed source framework, defined and administered by the Automated Imaging Association (AIA), for transmitting video and related control data over USB 3. Sometimes U3V is used as an acronym.
UDP	The User Datagram Protocol (UDP) is an Internet protocol. It is used by applications to send messages to other hosts on an Internet Protocol (IP) network.
Vertical sync	Synchronization pulse in video signal for field end recognition.
Virtual Network Computing (VNC)	Virtual Network Computing (VNC) is a graphical desktop sharing system that uses the RFB protocol to remotely control another computer. Over a network, it transmits the mouse and keyboard events from one computer to another, relaying the graphical screen updates back in the other direction. To access the camera's desktop from a PC via VNC, • you have to know the IP address of the remote system. • Start a VNC viewer and • point it to the remote system. You won't need a password. Of course, you won't get a very fast live image display via the network with VNC but you should be able to start ImpactControlCenter and capture images.
wxWidgets	wxWidgets is a cross-platform GUI library. It can be used from languages such as C++, Python, Perl, and C#/.NET. **See also** http://www.wxwidgets.org
Zero signal	The zero signal was needed with the old frame grabbers, to calibrate the analog/digital converter (ADC) (signal and parameter aren't important anymore).

19 Use Cases

- Introducing acquisition / recording possibilities
- Improving the acquisition / image quality
- Saving data on the device
- Working with several cameras simultaneously
- Working with HDR (High Dynamic Range Control)
- Working with I2C devices
- Working with LUTs
- Working with triggers
- Working with 3rd party tools
- Working with the Hardware Real-Time Controller (HRTC)

19.1 Introducing acquisition / recording possibilities

There are several use cases concerning the acquisition / recording possibilities of the camera:

- Generating very long exposure times
- Using Video Stream Recording

19.1.1 Generating very long exposure times

Since

Version 1.10.65 of the mvBlueFOX driver package

Very long exposure times are possible with mvBlueFOX. For this purpose a special trigger/IO mode is used.

You can do this as follows (pseudo code):

```
TriggerMode = OnHighExpose
TriggerSource = DigOUT0 - DigOUT3
```

Attention

In the standard mvBlueFOX DigOUT2 and DigOUT3 are internal signals, however, they can be used for this intention.

Note

Make sure that you adjust the `ImageRequestTimeout_ms` either to 0 (infinite)(this is the default value) or to a reasonable value that is larger than the actual exposure time in order not to end up with timeouts resulting from the buffer timeout being smaller than the actual time needed for exposing, transferring and capturing the image:

```
ImageRequestTimeout_ms = 0 # or reasonable value
```

Now request a single image:

```
imageRequestSingle
```

Then the digital output is set and reset. Between these two instructions you can include source code to get the desired exposure time.

```
# The DigOUT which was chosen in TriggerSource
DigitalOutput* pOut = getOutput(digital output)
pOut->set();

# Wait as long as the exposure should continue.

pOut->reset();
```

Afterwards you will get the image.

If you change the state of corresponding output twice this will also work with ImpactControlCenter.

19.1.2 Using Video Stream Recording

With the FFmpeg libraries it is possible to record an Impact Acquire image stream into a compressed video stream.

Since

2.39.0

19.1.2.1 Requirements Since the Impact Acquire API internally uses FFmpeg to record video streams, the FFmpeg libraries need to be present on the target system as well. They can either be built **OR** installed into the systems default library search path **OR** installed somewhere and afterwards an environment variable **MVIMPACT_ACQUIRE_FFMPEG_DIR** can be defined that points to the folder containing the libraries.

> **Note**
>
> Please make sure that you fully understand the license coming with FFmpeg! Have a look at the corresponding `legal section` inside any of the SDK manuals.
>
> At least FFmpeg **4.x** is needed. Older versions of the FFmpeg API are NOT compatible!

19.1.2.1.1 Windows

1. Go to `https://ffmpeg.org/download.html` and download the dynamic libraries of FFmpeg (version >= 4.x) according to your operating system (*e.g. 'ffmpeg-20200809-6e951d0-win64-shared.zip'*)
2. Extract the *.zip file under *'${MVIMPACT_ACQUIRE_DIR}/Toolkits'*.
3. Rename the file to *'ffmpeg-4.2.2-win64-shared'*(64-bit)/*'ffmpeg-4.2.2-win32-shared'*(32-bit) **OR** set an environment variable *e.g. 'MVIMPACT_ACQUIRE_FFMPEG_DIR'* which points to the folder containing the libraries.

19.1.2.2 Recording in ImpactControlCenter In ImpactControlCenter, a video stream can be recorded by the *'Start'*, *'Pause'* and *'Stop'* buttons at the top right tool-bar. They are however inactive when the video stream recording mode is deactivated OR when the video stream is not correctly set up.

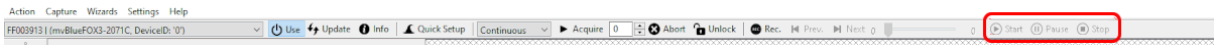


Figure 1: Video stream recording control buttons (inactive)

A video stream needs to be set up first to be able to get recorded. To do so:

1. Select the device to use and open it by clicking on the *'Use'* button.
2. Navigate to the *'Capture'* menu and click on the *'Video Stream Recording...'* option to start a setup dialog.

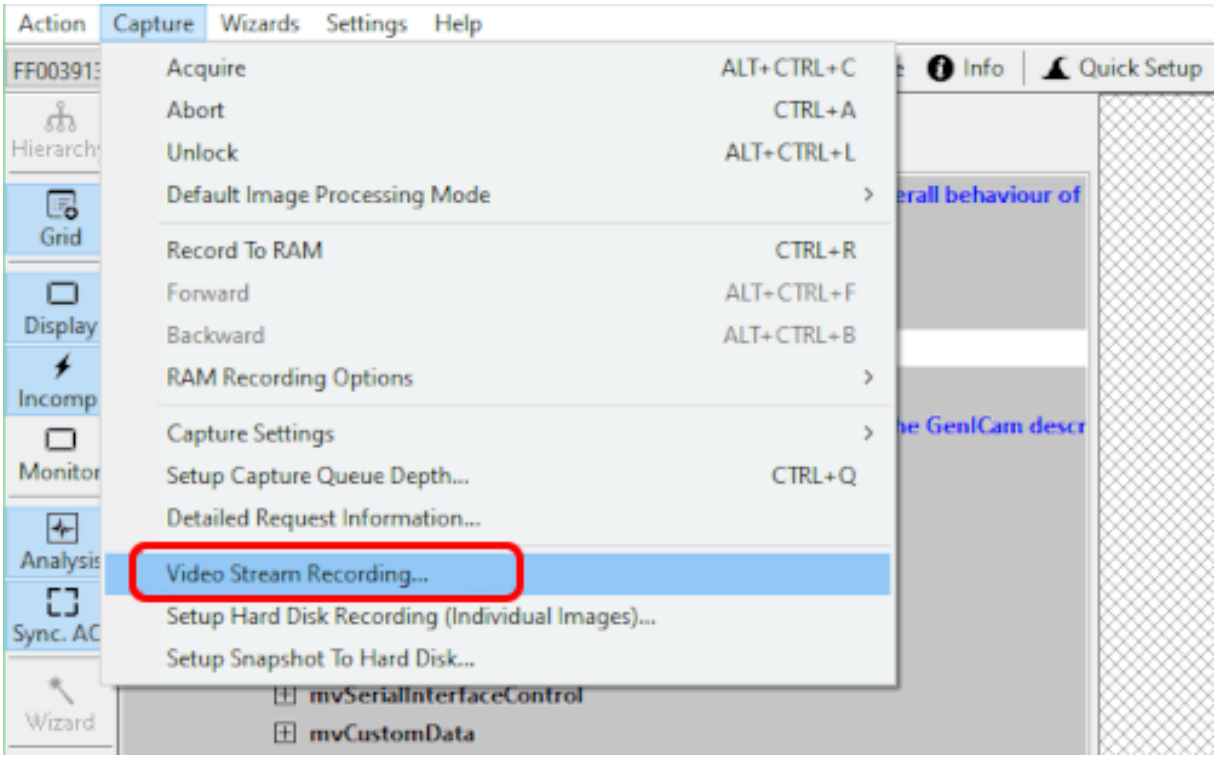


Figure 2: Click 'Video Stream Recording...'

3. A setup dialog will then be initialized as follows. Please read the setup hints in the text box for more information.

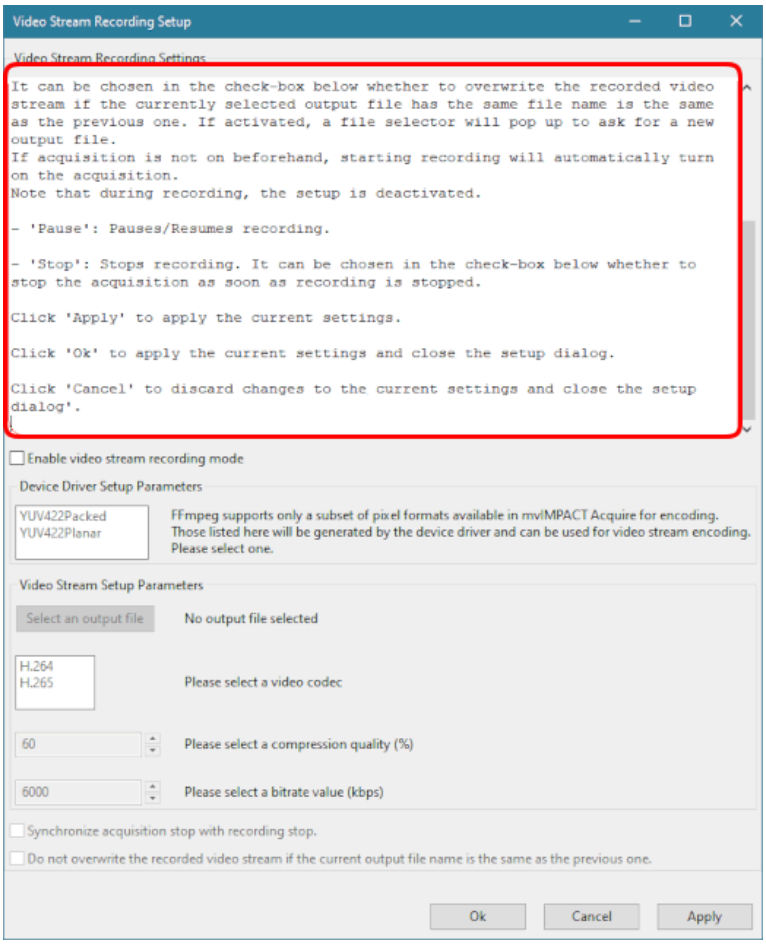


Figure 3: Video stream recording setup dialog

4. Enable the video stream recording mode. Choose a pixel format (e.g. *'YUV422Packed'* or *'YUV422Planar'*) that will be generated by the device driver and used by FFmpeg for video stream encoding. Then click on *'Select an output file'* to create/choose a file to hold the recorded video stream.

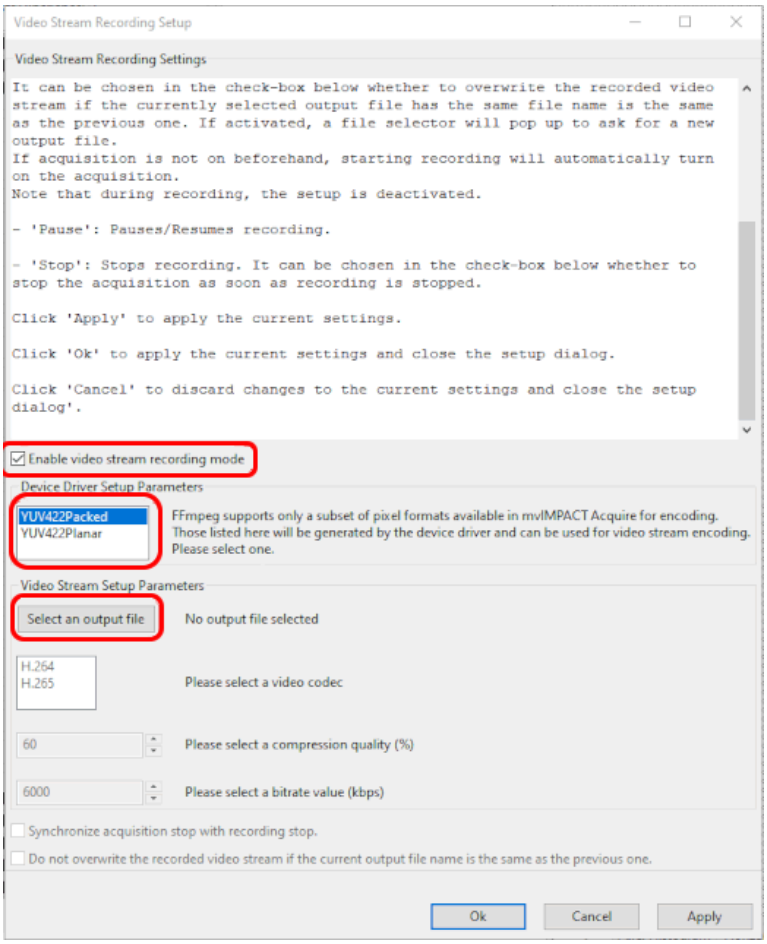


Figure 4: Enable the video stream recording mode and set up device driver related parameters

5. In the file selector, choose a file type (e.g. '∗.mp4' or '∗.m2v') and enter a file name.

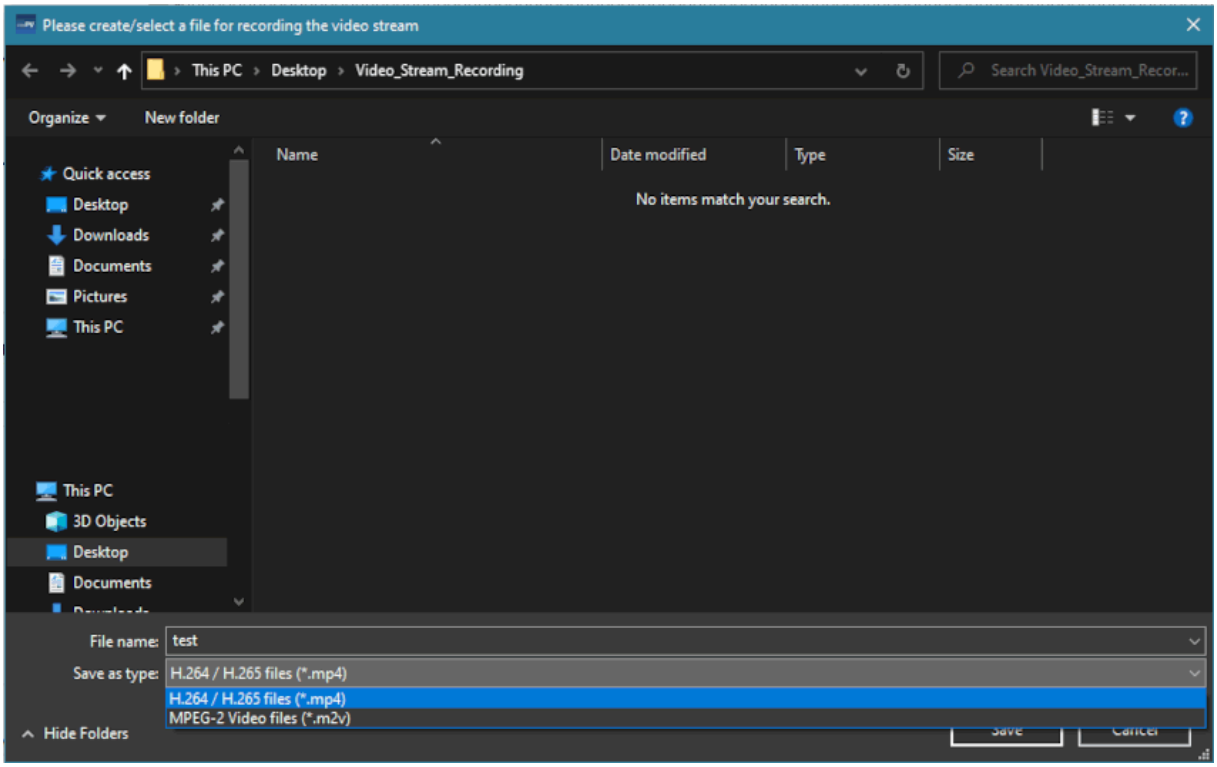


Figure 5: Select an output file

6. Set up video stream related parameters accordingly. In the check boxes below, users are allowed to choose whether to synchronize acquisition stop with recording stop and whether to overwrite the already recorded video stream if the currently selected output file has the same file name as the previous one.

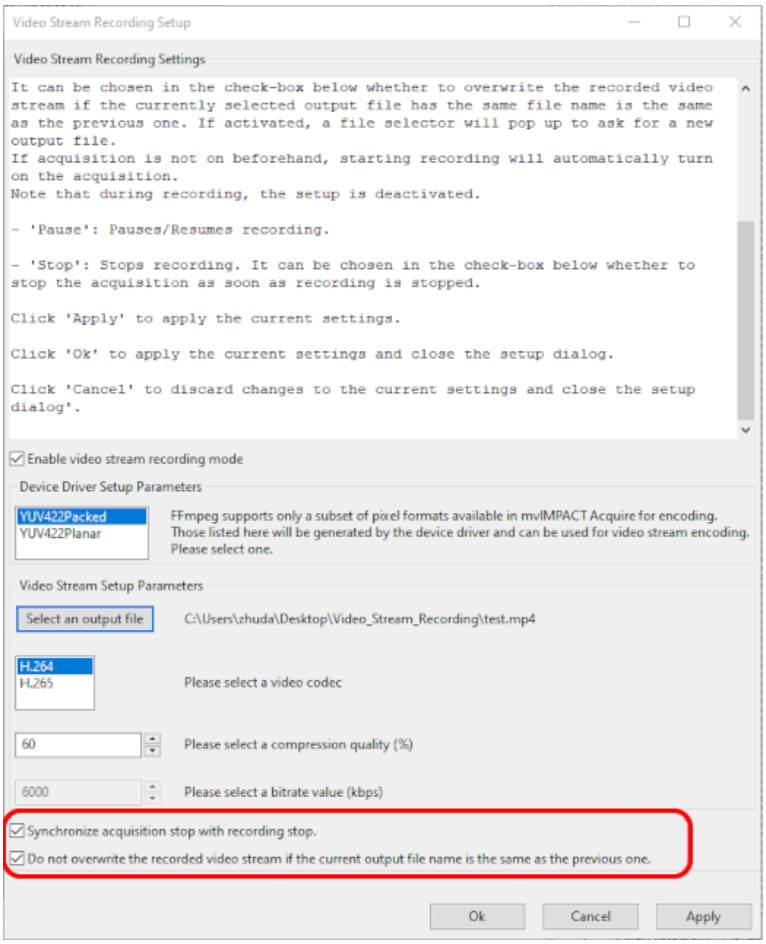


Figure 6: Set up video stream related parameters

7. Once the video stream recording has been set up, click *'Apply'* or *'Ok'* to apply the current settings. Afterwards, a log message in the analysis output will indicate whether the current settings have been applied successfully. If successful, the *'Start'* control button at the top right tool-bar will be enabled.

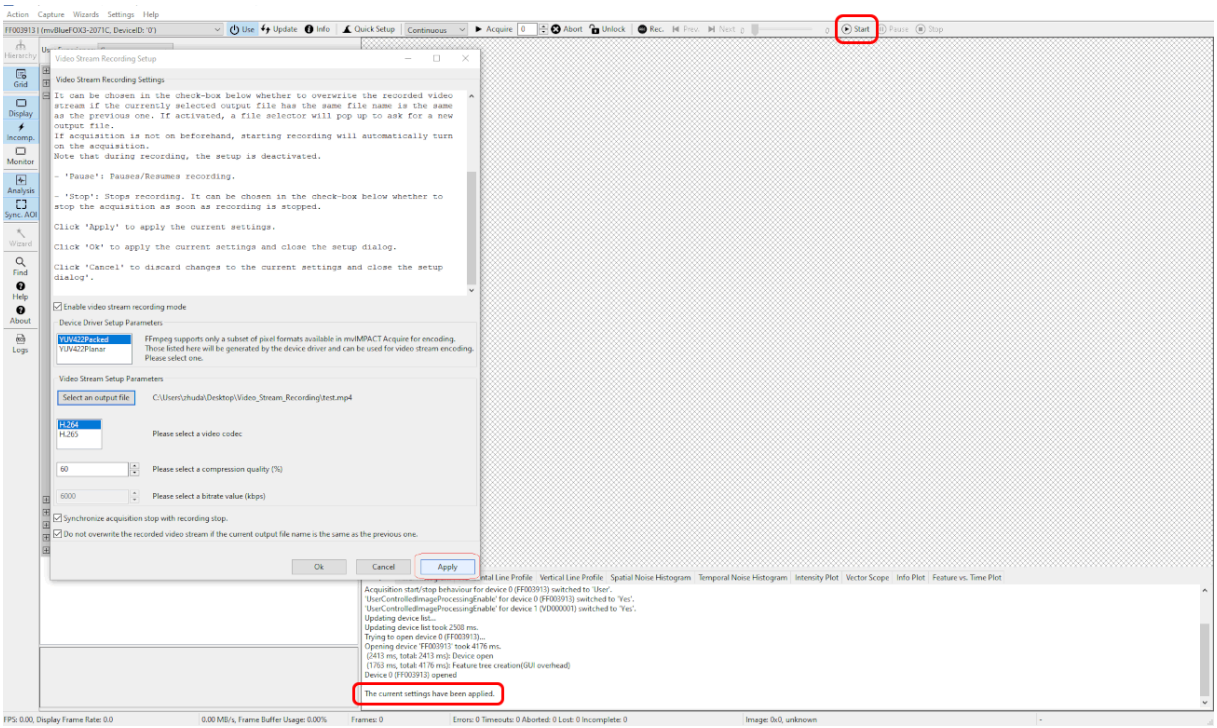


Figure 7: Apply the current settings

Note

When deactivating the video stream recording, uncheck the *'Enable video stream recording mode'* and then click *'Apply'* or *'Ok'* for the settings to take effect.

Once the settings have been applied, users can control the recording process via the *'Start'*, *'Pause'* and *'Stop'* buttons:

- **Start recording**: Click the *'Start'* control button to start recording the video stream. The current recording status and information will be displayed in the analysis output. During recording, the setup dialog as well as the *'Start'* button will be disabled. The *'Pause'* and *'Stop'* buttons will then be enabled.

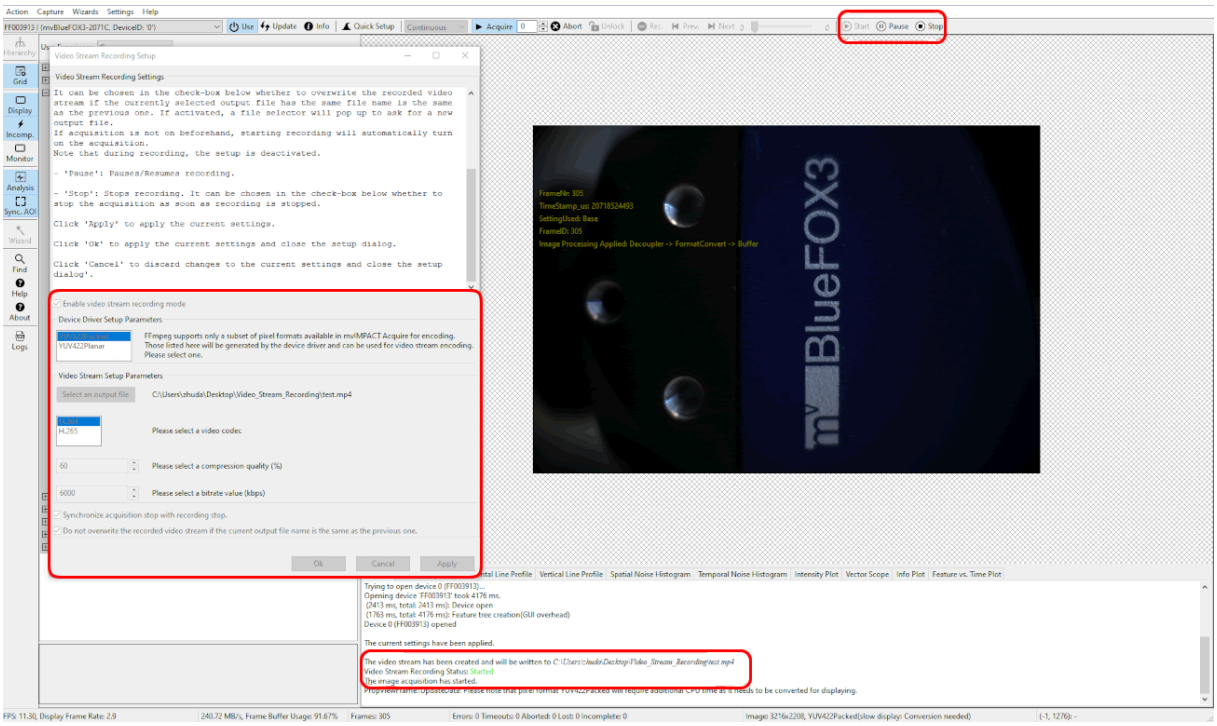


Figure 8: Start recording

- **Pause recording**: Click the *'Pause'* button to pause a running recording. The current recording status will be displayed in the analysis output.

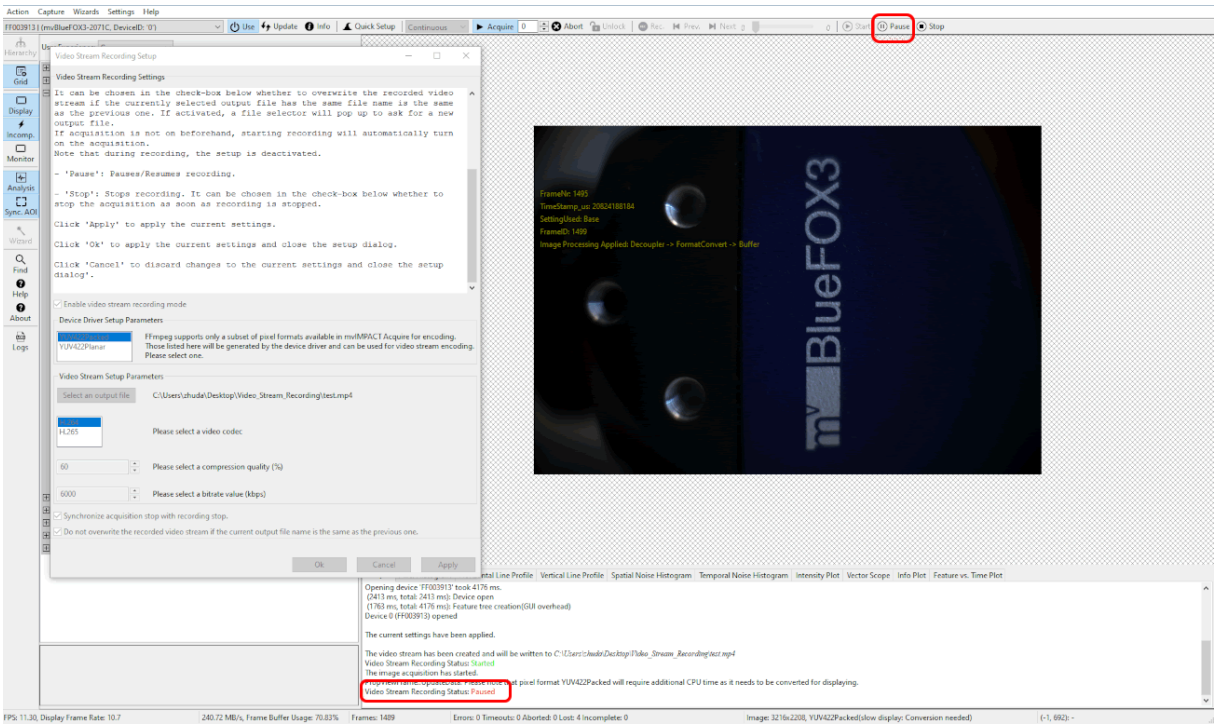


Figure 9: Pause recording

- **Resume recording**: Click the *'Pause'* button to resume a paused recording. The current recording status will be displayed in the analysis output.

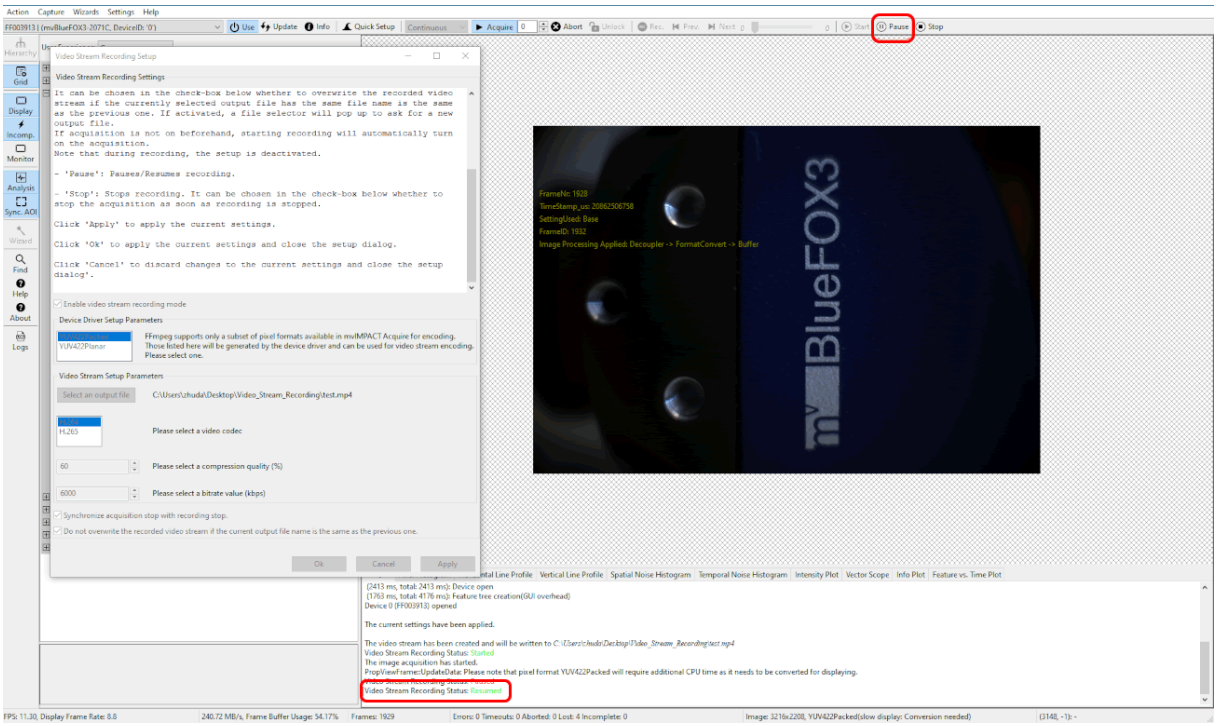


Figure 10: Resume recording

- **Stop recording**: Click the *'Stop'* button to stop recording the video stream. The current recording status and information will be displayed in the analysis output. Once the recording has been stopped, the setup dialog as well as the *'Start'* button will be enabled again. The *'Pause'* and *'Stop'* buttons will then be disabled.

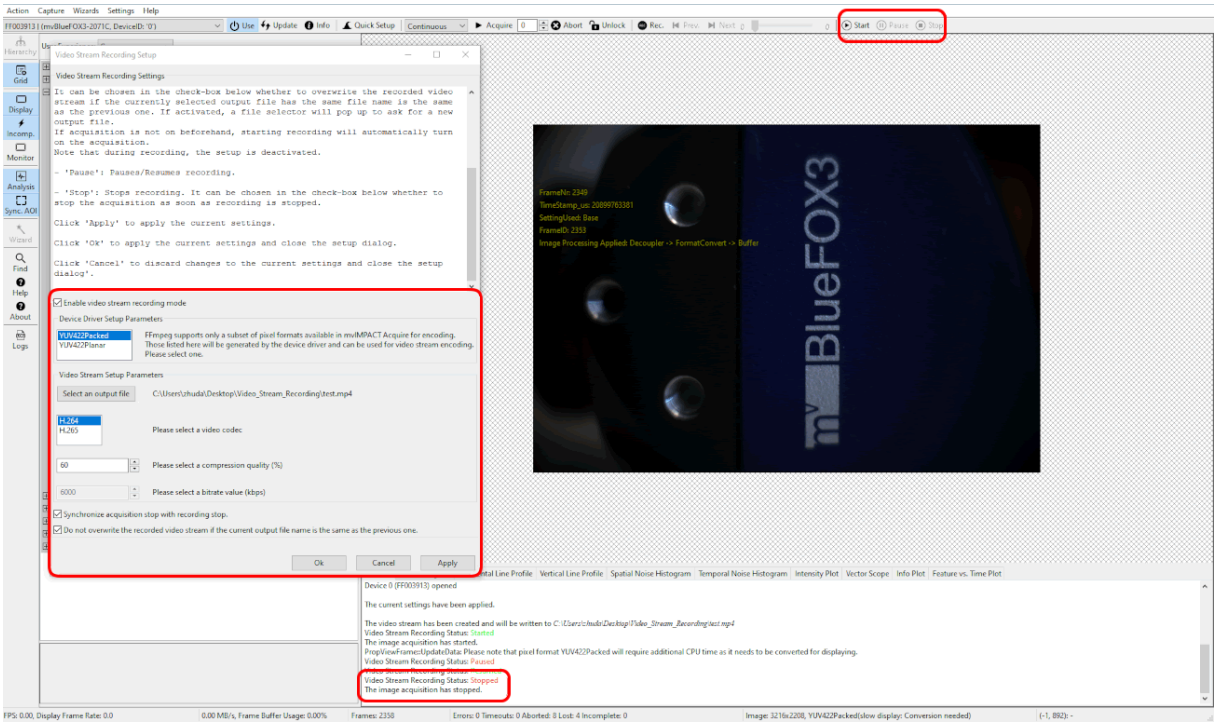


Figure 11: Stop recording

When recording to an output file which has the same file name as the previous one while overwriting the recorded content is not desired:

1. When clicking *'Start'*, a file selector will pop up to ask users to create a new output file with the same file type as the previous one. If a new set of parameters of the video stream recording is needed, please click *'Cancel'* in the file selector and re-configure parameters in the setup dialog.

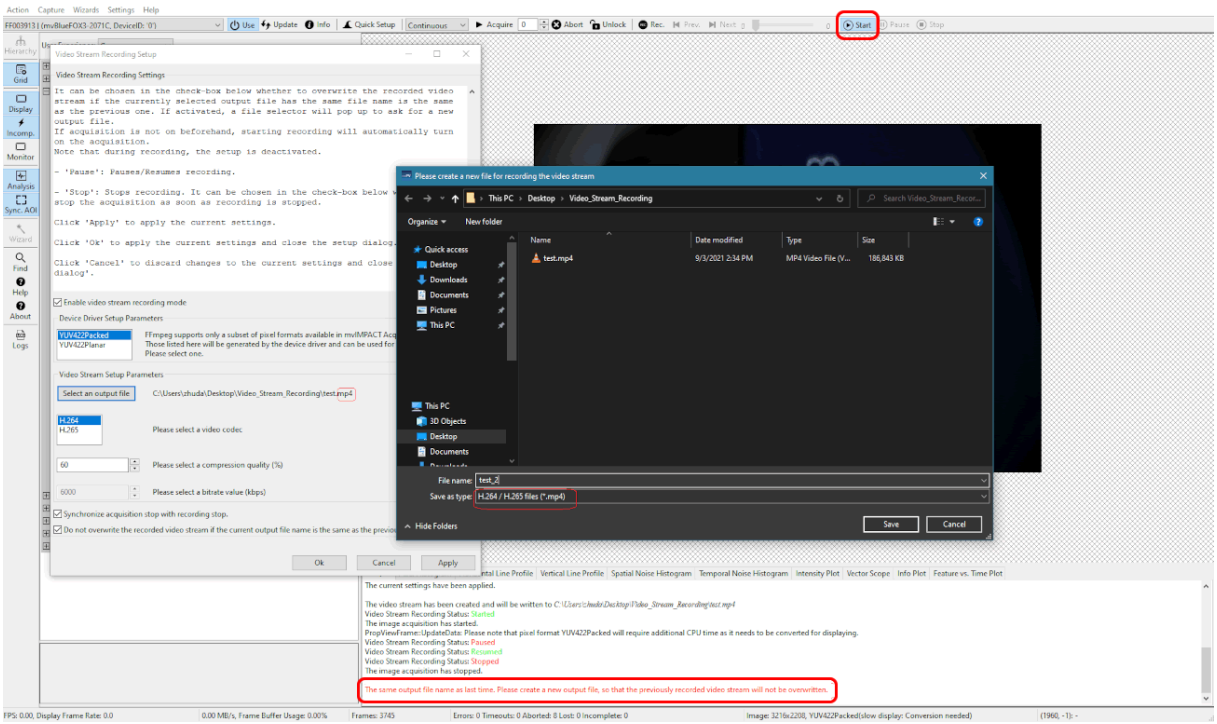


Figure 12: Select a new file when starting to record to an output file with the same file name as the previous one without overwriting

2. Once a new file has been created, the video stream will start to get recorded. The current recording status and information will be displayed in the analysis output. During recording, the setup dialog as well as the *'Start'* button will be disabled. The *'Pause'* and *'Stop'* buttons will then be enabled.

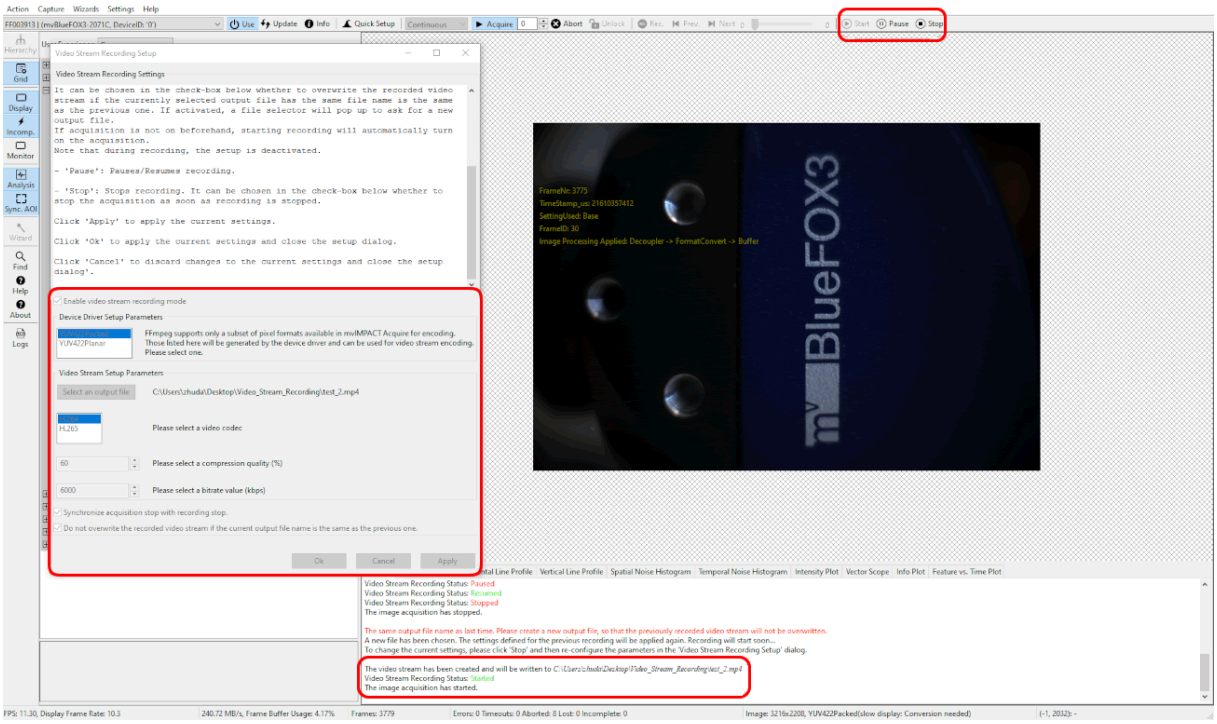


Figure 13: Start recording to an output file with the same file name as the previous one without overwriting

19.1.2.3 Recording Using The API

Please refer to the example on how to record a video stream using Impact Acquire C++ API: `Continuous↩CaptureFFmpeg.cpp` or have a look at the `VideoStream` class.

19.2 Improving the acquisition / image quality

There are several use cases concerning the acquisition / image quality of the camera:

- Correcting image errors of a sensor
- Optimizing the color/luminance fidelity of the camera
- Working With Gain And Black-Level Values Per Color Channel

19.2.1 Correcting image errors of a sensor

19.2.1.1 Defective Pixel Correction Due to random process deviations, technical limitations of the sensors, etc. there are different reasons that image sensors have image errors. Balluff provides several procedures to correct these errors, by default these are host-based calculations, however some device families support device-based corrections, which saves dozens of % CPU load and lowers latency.

Device Family	Algorithm-↩ Based detection and correction	List-Based correction	Storing facility for defective-pixel list	Flat-Field Correction (Host)	Flat-Field Correction (Device)
BVS CA-GX0	-	-	X	X	X
BVS CA-GX2	If binning/decimation is on -> no list is stored	X	X	X	
BVS CA-GT1	X	-	-	X	X
BVS CA-SF	If binning/decimation is on -> no list is stored	X	X	X	-

Generally, removing defect pixels requires two sub-tasks:

- Detection of defective pixels
- Correction of defective pixels

Both tasks can performed in different "locations":

- Detection and correction on the host using Impact Acquire
- Detection on the host using Impact Acquire, correction on the device using the device's `mvDefective↩ PixelCorrectionControl` in the list-based mode
- Detection and correction on the camera using `mvDefectivePixelCorrectionControl` in the algorithm-based mode.

If detection is not happening in real-time, meaning during the image acquisition itself, it is necessary to store the detected defects somewhere. This can be either on the device or the host or both.

19.2.1.1.1 Host-based defect pixel detection As mentioned, the defect pixel list can be generated using Impact Acquire. Since there are three types of defects, Impact Acquire offers three calibration methods for detection:

1. **leaky pixel** (in the dark)
   which indicates pixels that produce a higher read out code than the average
2. **hot pixel** (in standard light conditions)
   which indicates pixels that produce a higher non-proportional read out code when temperatures are rising
3. **cold pixel** (in standard light conditions)
   which indicates pixels that produce a lower read out code than average when the sensor is exposed (e.g. caused by dust particles on the sensor)

Note

Please use either an Mono or RAW Bayer image format when detecting defective pixel data in the image.

19.2.1.1.2 Detecting leaky pixels To detect leaky pixels the following steps are necessary:

1. Set gain ("`Setting -> Base -> Camera -> GenICam -> Analog Control -> Gain = 0 dB`") and exposure time "`Setting -> Base -> Camera -> GenICam -> Acquisition Control -> ExposureTime = 360 msec`" to the given operating conditions
   The total number of defective pixels found in the array depend on the gain and the exposure time.
2. Black out the lens completely
3. Set the (Filter-) "`Mode = Calibrate leaky pixel`"
4. Acquire an image (e.g. by pressing **Acquire** in ImpactControlCenter with "`Acquisition Mode = SingleFrame`")

The filter checks:

```
Pixel > LeakyPixelDeviation_ADCLimit // (default value: 50)
```

All pixels above this value are considered as leaky pixel.

19.2.1.1.3 Detecting hot or cold pixels

Note

With "Mode = Calibrate Hot And Cold Pixel" you can execute both detections at the same time.

To detect hot or cold pixels the following steps are necessary:

1. You will need a uniform sensor illumination approx. 50 - 70 % saturation (which means an average gray value between 128 and 180)
2. Set the (Filter-) "`Mode = Calibrate Hot Pixel`" or "`Mode = Calibrate Cold Pixel`" or "`Mode = Calibrate Hot And Cold Pixel`"
3. Acquire an image (e.g. by pressing **Acquire** in ImpactControlCenter with "`Acquisition Mode = SingleFrame`")

The filter checks:

```
Pixel > T[hot] // (default value: 15 %)

// T[hot] = deviation of the average gray value

Pixel < T[cold] // (default value: 15 %)

// T[cold] = deviation of the average gray value
```

Note

Repeating the defective pixel corrections will accumulate the correction data which leads to a higher value in "DefectivePixelsFound". If you want to reset the correction data or repeat the correction process you have to set the filter mode to "Reset Calibration Data". In order to limit the amount of defective pixels detected the "DefectivePixelsMaxDetectionCount" property can be used.

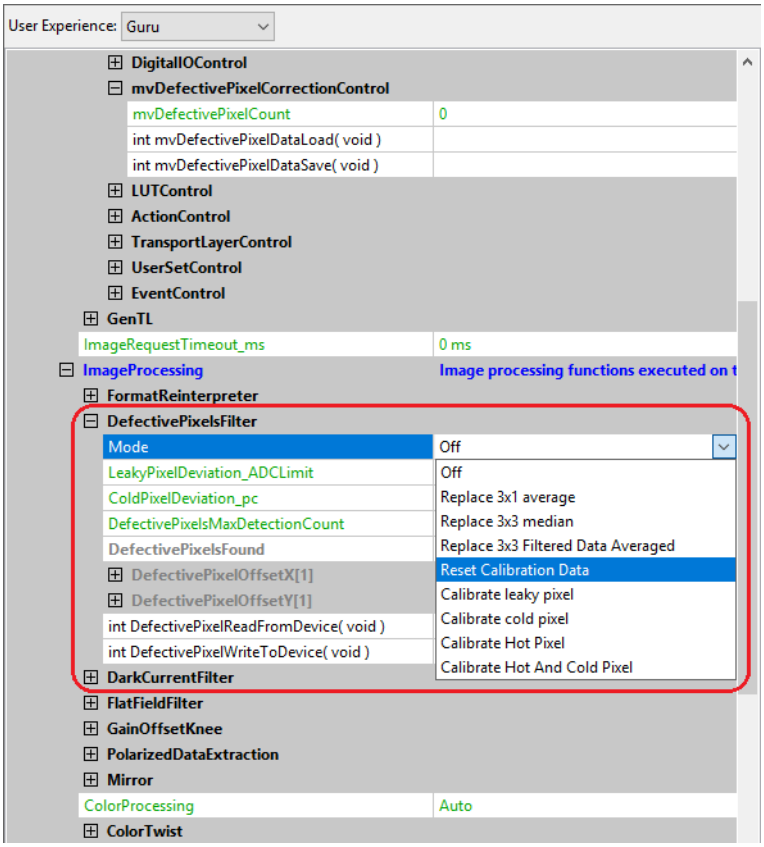


Figure 1: Image corrections: DefectivePixelsFilter

19.2.1.1.4 Storing defective pixel data on the device To save and load the defective pixel data, appropriate functions are available:

- **"int mvDefectivePixelDataLoad(void)"**
- **"int mvDefectivePixelDataSave(void)"**

The section *"Setting -> Base -> ImageProcessing -> DefectivePixelsFilter"* was also extended (see Figure 2). First, the "DefectivePixelsFound" indicates the number of found defective pixels. The coordinates are available through the properties "DefectivePixelOffsetX" and "DefectivePixelOffsetY" now. In addition to that it is possible to edit, add and delete these values manually (via right-click on the "DefectivePixel↩Offset" and select **"Append Value"** or **"Delete Last Value"**). Second, with the functions

- **"int mvDefectivePixelReadFromDevice(void)"**
- **"int mvDefectivePixelWriteToDevice(void)"**

you can exchange the data from the filter with the device and vice versa.

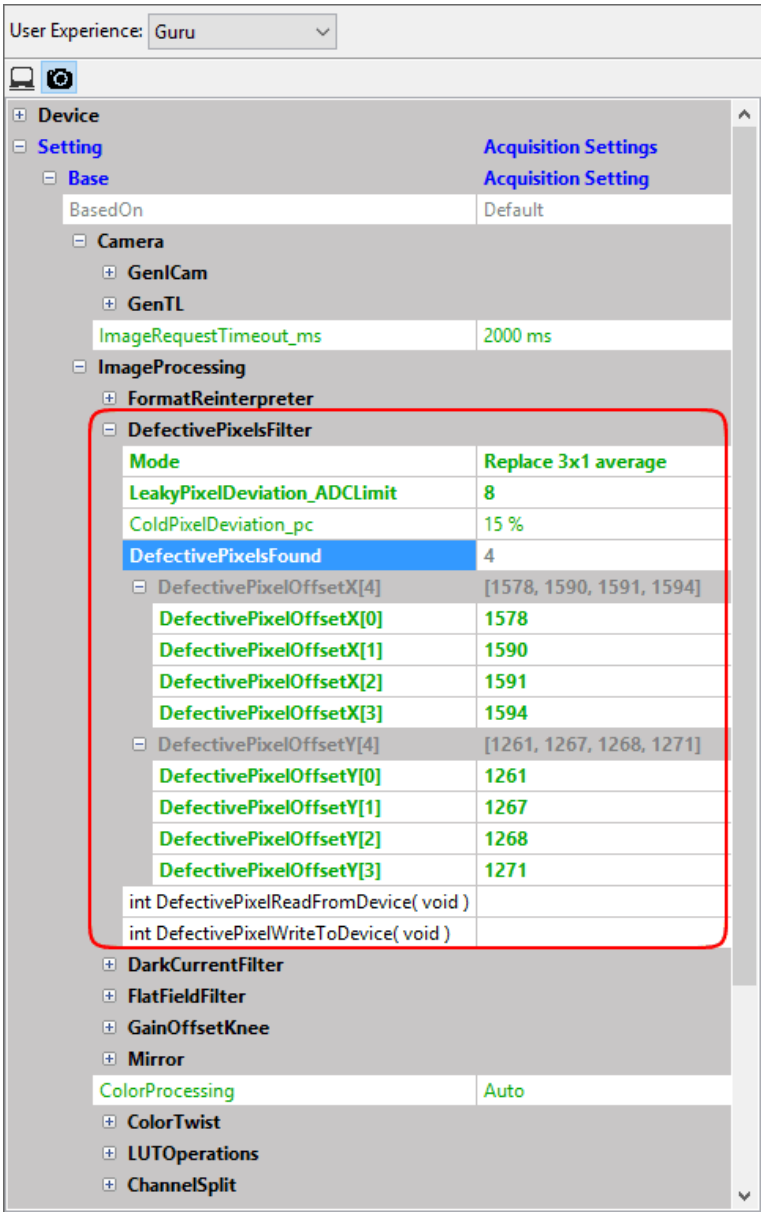


Figure 2: Image corrections: DefectivePixelsFilter (since driver version 2.17.1 and firmware version 2.12.406)

Just right-click on "`mvDefectivePixelWriteToDevice`" and click on **"Execute"** to write the data to the device (and hand over the data to the Storing pixel data on the device). To permanently store the data inside the device s non-volatile memory afterwards "`mvDefectivePixelDataSave`" must be called as well!

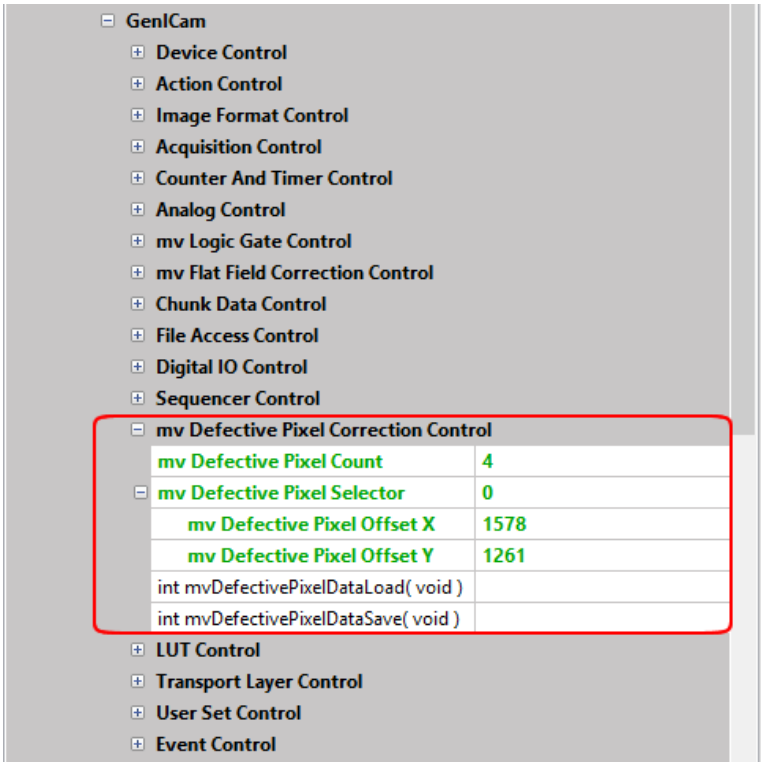


Figure 3: Defective pixel data are written to the device (since driver version 2.17.1 and firmware version 2.12.406)

While opening the device, the device will load the defective pixel data from the device. If there are pixels in the filter available (via calibration), nevertheless you can load the values from the device. In this case the values will be merged with the existing ones. I.e., new ones are added and duplicates are removed.

19.2.1.1.5 Host-based defect pixel correction After a defect-list is generated, a host-based correction can be performed using Impact Acquire.

To correct the defective pixels various substitution methods exist:

1. **"Replace 3x1 average"**
   which substitutes the detected defective pixels with the average value from the left and right neighboring pixel (3x1)
2. **"Replace 3x3 median"**
   which substitutes the detected defective pixels with the median value calculated from the nearest neighboring in a 3 by 3 region
3. **"Replace 3x3 Filtered Data Averaged"**
   which substitutes and treats the detected defective pixels as if they have been processed with a 3 by 3 filter algorithm before reaching this filter
   Only recommended for devices which do not offer a defective pixel compensation; packed RGB or packed YUV444 data is needed. See enumeration value `dpfmReplaceDefectivePixelAfter3x3Filter` in the corresponding API manual for additional details about this algorithm and when and why it is needed

19.2.1.1.6 List-based defect pixel correction on the device As described before, it is possible to upload lists of defect pixel onto the device. Different algorithms can be used to determine whether a pixel is defective or not, which is dependent of how much it is allowed a pixel to deviate, temperature, gain, and exposure time. As described before, the list-based correction is deterministic, meaning it is exactly known which pixels will be corrected.

Anyhow, the list-based correction has some disadvantages:

- A default list is stored in the device during production, but this might to fit to the target application because of much different temperature / exposure time setting
  → It is necessary to create the list using a detection algorithm (or Impact Acquire support)
- During time and sensor aging, new defects could/will appear
- It doesn't work in binning/decimation modes
- The memory for storing defective pixels is limited

19.2.1.1.7 Adaptive / algorithm-based correction on the device In this case, the device performs detection and correction on-the-fly without using any defect-list.

The adaptive correction addresses the above-mentioned disadvantages of the list-based method. While the correction itself (this is which pixels are used to correct an identified defect) is the same, no static information from a list is used, instead they are detected "on the fly".

To use reasonable thresholds, knowledge of the noise statistics of the sensor is used to detect the outliers. These will be corrected also on the fly. Because this is a dynamic approach, it also works in binning/decimation modes and would also detect new appearing defects.

Nevertheless, there are some disadvantages:

- It is non-deterministic
- Wrong positives can be detected, meaning non-defect pixels could be treated as defect
- If pixels are at the edge of the used thresholds, it could be corrected in one frame, but not in the next

On BVS CA-SF devices, the adaptive correction is always used if:

- There is no list stored on the device
- Binning or decimation is used

19.2.1.2 Flat-Field Correction

Each pixel of a sensor chip is a single detector with its own properties. Particularly, this pertains to the sensitivity as the case may be the spectral sensitivity. To solve this problem (including lens and illumination variations), a plain and equally "colored" calibration plate (e.g. white or gray) as a flat-field is snapped, which will be used to correct the original image. Between flat-field correction and the future application you must not change the optic. To reduce errors while doing the flat-field correction, a saturation between 50 % and 75 % of the flat-field in the histogram is convenient.

> **Note**
>
> Flat-field correction can also be used as a destructive watermark and works for all f-stops.

To make a **flat field correction** following steps are necessary:

1. You need a plain and equally "colored" calibration plate (e.g. white or gray)
2. No single pixel may be saturated - that's why we recommend to set the maximum gray level in the brightest area to max. 75% of the gray scale (i.e., to gray values below 190 when using 8-bit values)
3. Choose a `BayerXY` in *"Setting -> Base -> Camera -> GenICam -> Image Format Control -> PixelFormat"*.
4. Set the (Filter-) **"**`Mode = Calibrate`**"** (Figure 4)
5. Start a Live snap (**"Acquire"** with **"**`Acquisition Mode = Continuous`**"**)
6. Finally, you have to activate the correction: Set the (Filter-) **"**`Mode = On`**"**
7. Save the settings including the correction data via **"**`Action -> Capture Settings -> Save Active Device Settings`**"**
   (Settings can be saved in the Windows registry or in a file)

> **Note**
>
> After having re-started the device you have to reload the capture settings vice versa.

The filter snaps a number of images (according to the value of the `CalibrationImageCount`, e.g. `5`) and averages the flat-field images to one correction image.

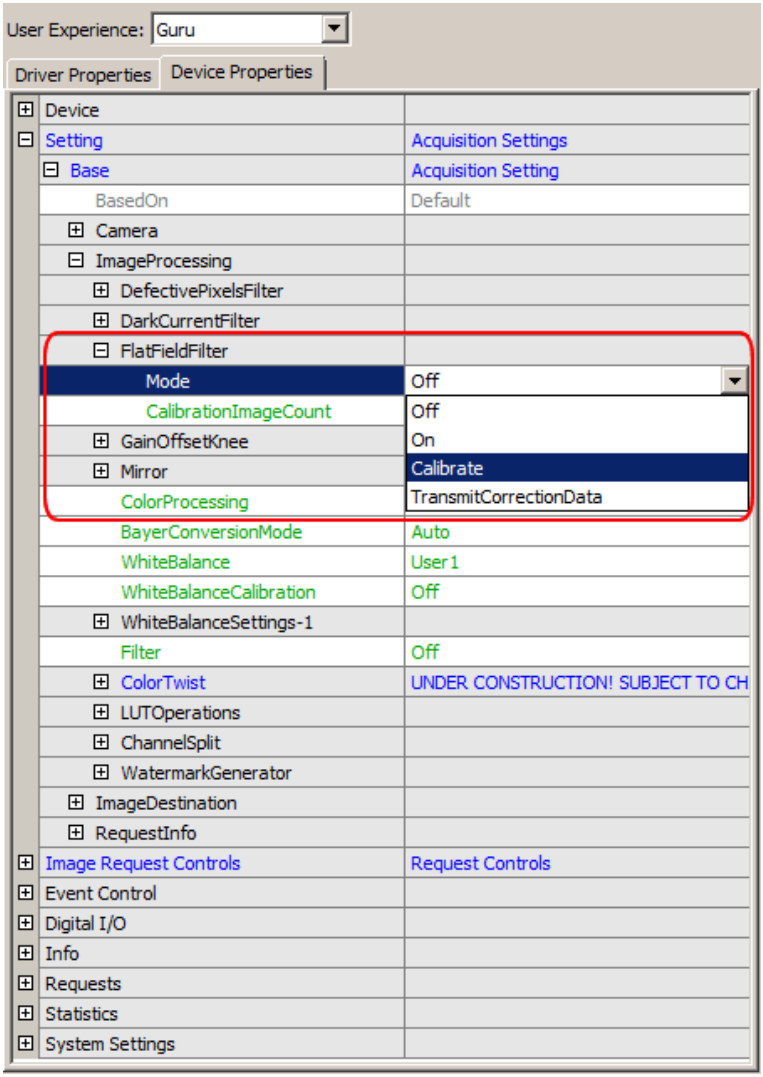


Figure 4: Image corrections: Host-based flat field correction

19.2.1.2.1 Host-based Flat-Field Correction With Calibration AOI

In some cases it might be necessary to use just a specific area within the device's field of view to calculate the correction values. In this case just a specific AOI will be used to calculate the correction factor.

You can set the **"host-based flat field correction"** in the following way:

1. All necessary setting can be found under **"ImageProcessing"->** **"FlatfieldFilter"**.
2. Stop **"Continuous"** acquisition mode.
3. Set **"CalibrationImageCount"** to, for example, 5.
4. Set **"Mode"** to "Calibrate".
5. Set **"CalibrationAoiMode"** to "UseAoi".
6. Set the properties (**"X, Y and W, H"**) appeared under **"CalibrationAOI"** to the desired AOI.
7. Start **"Continuous"** acquisition mode.
8. Finally, you have to activate the correction: Set the **"Mode"** to "On".

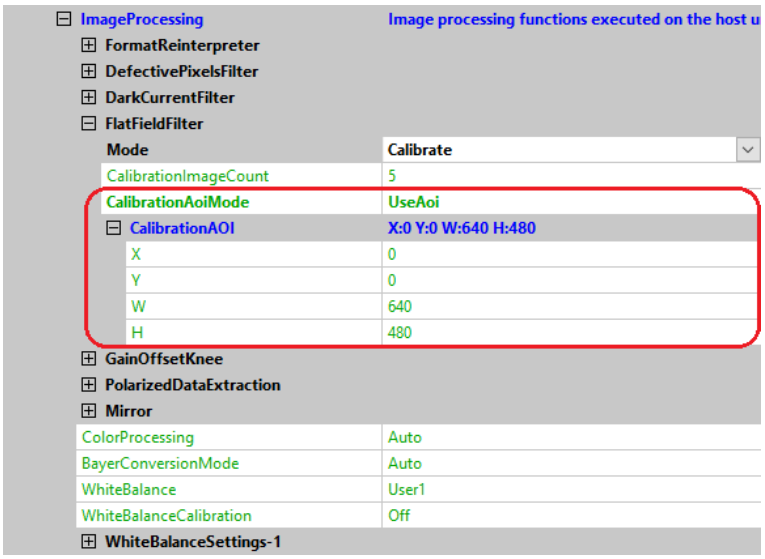


Figure 5: Image corrections: Host-based flat field correction with calibration AOI

19.2.1.2.2 Host-based Flat-Field Correction With Correction AOI

In some cases it might be necessary to correct just a specific area in the device's filed of view. In this case the correction values are only applied to a specific area. For the rest of the image, the correction factor will be just 1.0.

You can set the **"host-based flat field correction"** in the following way:

1. All necessary setting can be found under **"ImageProcessing"** -> **"FlatfieldFilter"**.
2. Stop **"Continuous"** acquisition mode.
3. Set **"CalibrationImageCount"** to, for example, 5.
4. Set **"Mode"** to "Calibrate".
5. Start **"Continuous"** acquisition mode.
6. Now, you have to activate the correction: Set the **"Mode"** to "On".
7. Set **"CorrectionAOIMode"** to "UseAoi".
8. Finally use the properties (**"X, Y and W, H"**) which appeared under **"CorrectionAOI"** to configure the desired AOI.

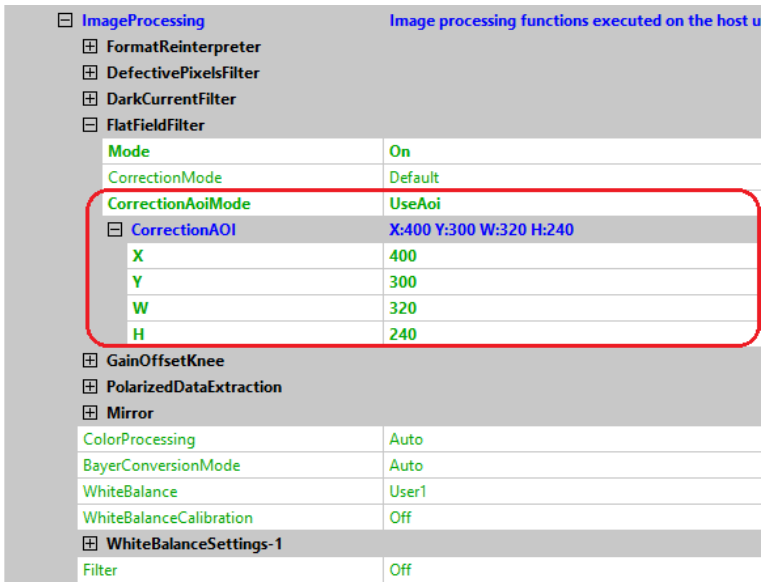


Figure 6: Image corrections: Host-based flat field correction with correction AOI

19.2.2 Optimizing the color/luminance fidelity of the camera

Purpose of this chapter is to optimize the color image of a camera, so that it looks as natural as possible on different displays and for human vision.

This implies some linear and nonlinear operations (e.g. display color space or Gamma viewing LUT) which are normally not necessary or recommended for machine vision algorithms. A standard monitor offers, for example, several display modes like *sRGB*, *"Adobe RGB"*, etc., which reproduce the very same color of a camera color differently.

It should also be noted that users can choose for either

- camera based settings and adjustments or
- host based settings and adjustments or
- a combination of both.

Camera based settings are advantageous to achieve highest calculating precision, independent of the transmission bit depth, lowest latency, because all calculations are performed in FPGA on the fly and low CPU load, because the host is not invoked with these tasks. These camera based settings are

- gamma correction
- negative gain / gain
- look-up table (LUT)
- white balance
- offset
- saturation and color correction

Host based settings save transmission bandwidth at the expense of accuracy or latency and CPU load. Especially performing gain, offset, and white balance in the camera while outputting RAW data to the host can be recommended.

Of course host based settings can be used with all families of cameras (e.g. also mvBlueFOX).

Host based settings are:

- look-up table (*LUTOperations*)
- color correction (*ColorTwist*)

To show the different color behaviors, we take a color chart as a starting point:

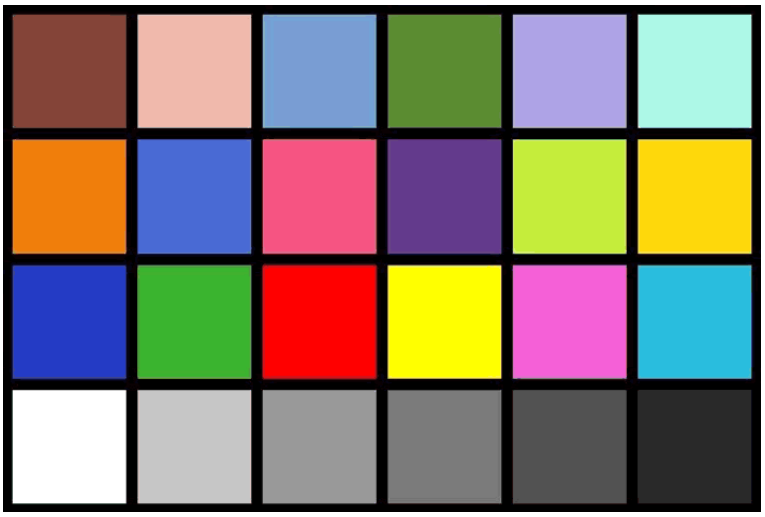


Figure 1: Color chart as a starting point

If we take a *SingleFrame* image without any color optimizations, an image can be like this:

Figure 2: SingleFrame snap without color optimization

Figure 3: Corresponding histogram of the horizontal white to black profile

As you can see,

- saturation is missing,
- white is more light gray,
- black is more dark gray,
- etc.

Note

You have to keep in mind that there are two types of images: the one generated in the camera and the other one displayed on the computer monitor. Up-to-date monitors offer different display modes with different color spaces (e.g. sRGB). According to the chosen color space, the display of the colors is different.

The following figure shows the way to a perfect colored image

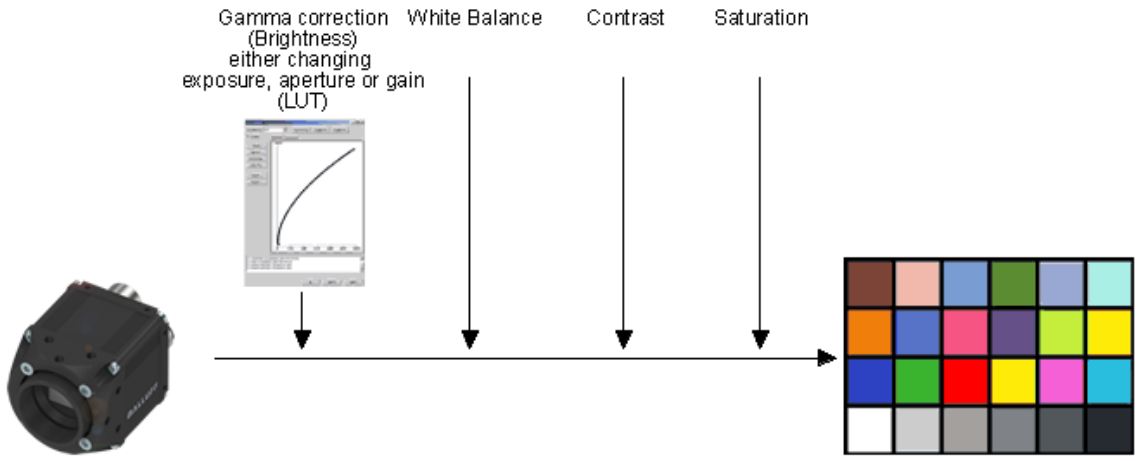


Figure 4: The way to a perfect colored image

including these process steps:

1. Do a Gamma correction (Luminance),
2. make a White balance and
3. Improve the Contrast.
4. Improve Saturation, and use a **"color correction matrix"** for both
   (a) the sensor and / or
   (b) the monitor.

The following sections will describe the single steps in detail.

19.2.2.1 Step 1: Gamma correction (Luminance)

First of all, a **Gamma correction** can be performed to change the image in a way how humans perceive light and color.

For this, you can change either

- the exposure time,
- the aperture or
- the gain.

You can change the gain via ImpactControlCenter like the following way:

1. Click on *"Setting -> Base -> Camera"*. There you can find
   (a) *"AutoGainControl"* and
   (b) *"AutoExposeControl"*.

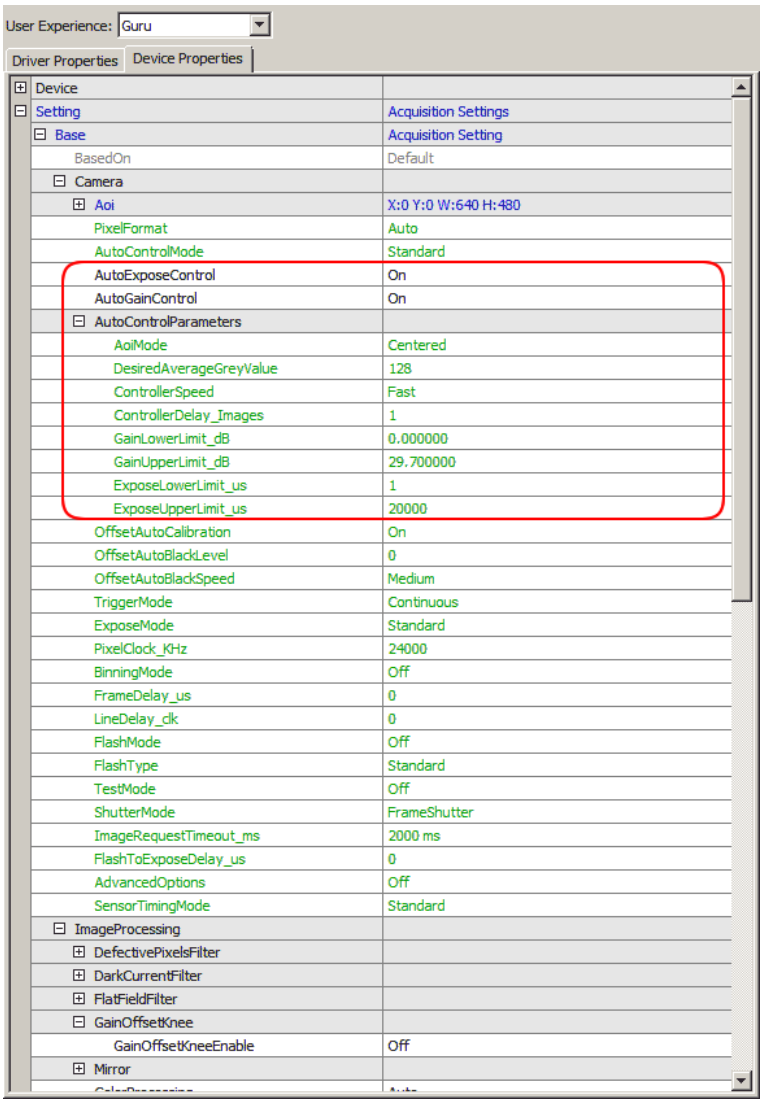


Figure 5: ImpactControlCenter: Setting -> Base -> Camera

You can turn them *"On"* or *"Off"*. Using the auto controls you can set limits of the auto control; without you can set the exact value.

After gamma correction, the image will look like this:

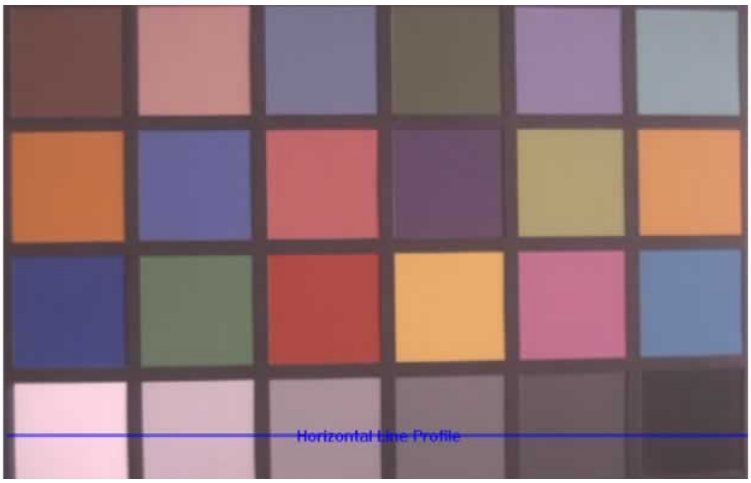


Figure 6: After gamma correction

Figure 7: Corresponding histogram after gamma correction

Note

As mentioned above, you can do a gamma correction via (*"Setting -> Base -> ImageProcessing -> LUTOperations"*):

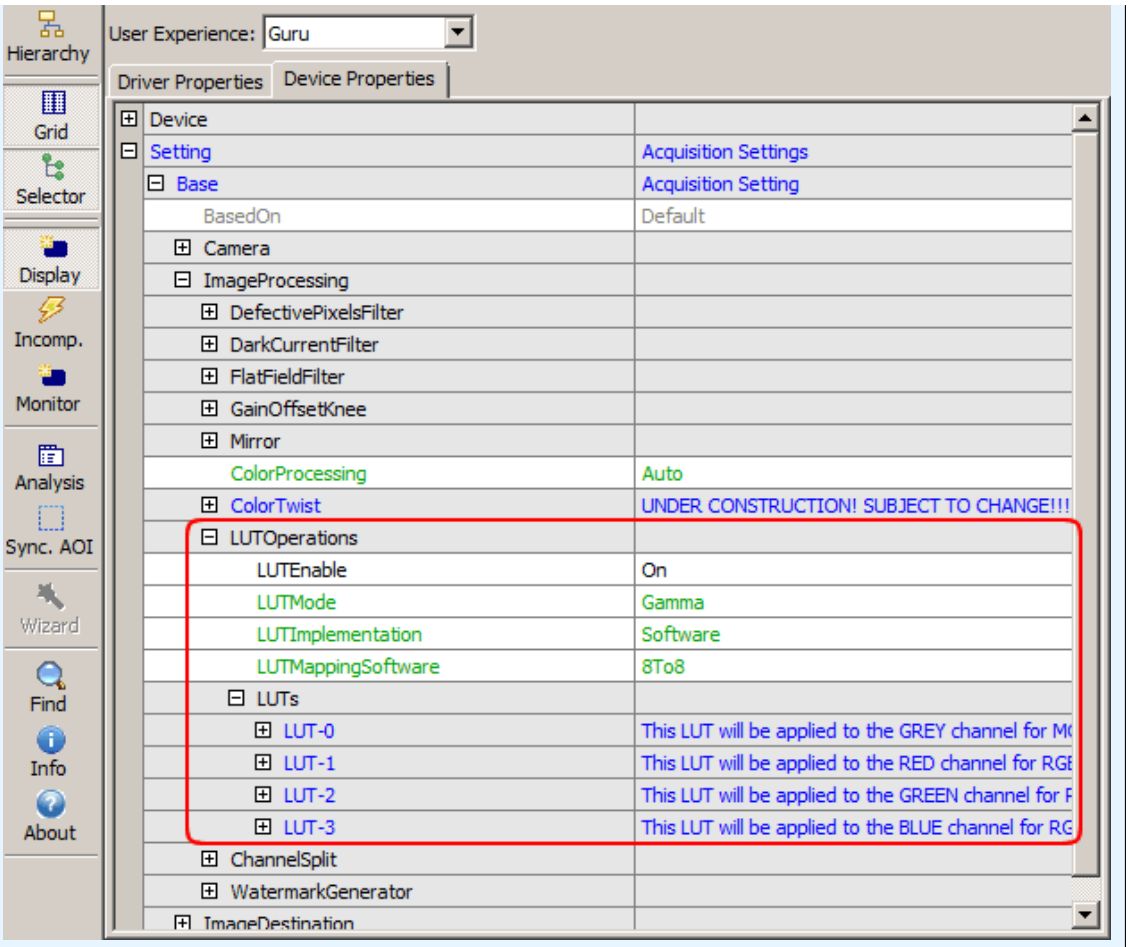


Figure 8: LUTOperations dialog

Just set *"LUTEnable"* to *"On"* and adapt the single LUTs like (LUT-0, LUT-1, etc.).

19.2.2.2 Step 2: White Balance

As you can see in the histogram, the colors red and blue are below green. Using green as a reference, we can optimize the white balance via *"Setting -> Base -> ImageProcessing"* (*"WhiteBalanceCalibration"*):

Please have a look at **"White Balancing A Color Camera"** in the **"Impact Acquire SDK GUI Applications"** manual for more information for an automatic white balance with ImpactControlCenter.

To adapt the single colors you can use the *"WhiteBalanceSettings-1"*.

After optimizing white balance, the image will look like this:

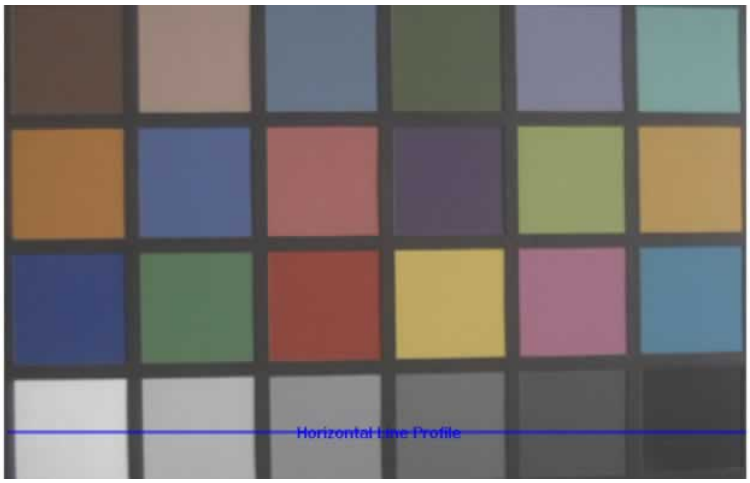


Figure 9: After white balance

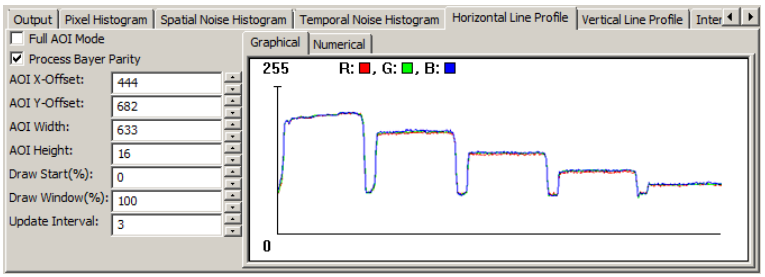


Figure 10: Corresponding histogram after white balance

19.2.2.3 Step 3: Contrast

Still, black is more a darker gray. To optimize the contrast you can use *"Setting -> Base -> ImageProcessing -> LUTControl"* as shown in Figure 8.

The image will look like this now:

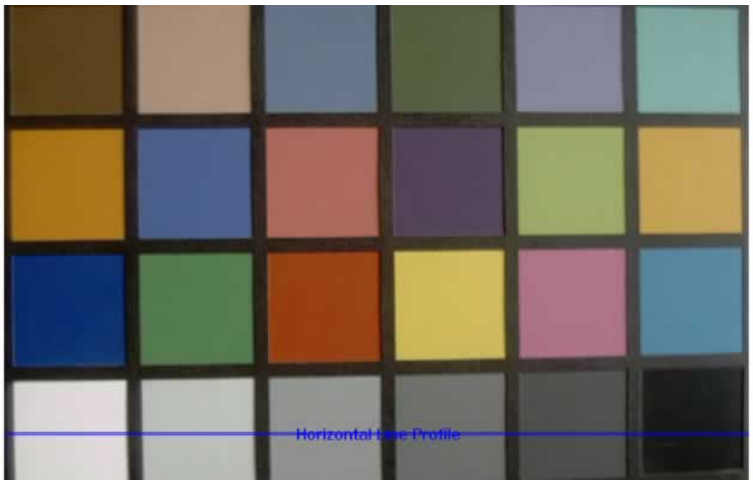


Figure 11: After adapting contrast

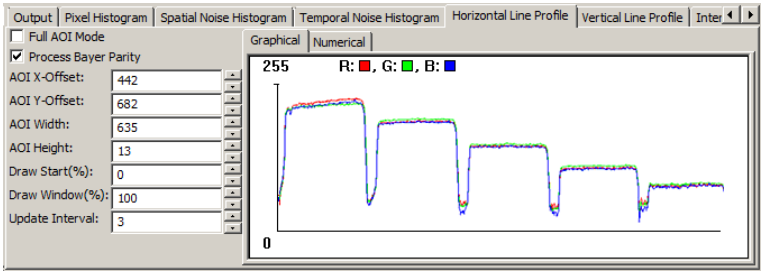


Figure 12: Corresponding histogram after adapting contrast

19.2.2.4 Step 4: Saturation and Color Correction Matrix (CCM)

Still saturation is missing. To change this, the **"Color Transformation Control"** can be used (*"Setting -> Base -> ImageProcessing -> ColorTwist"*):

1. Click on *"Color Twist Enable"* and
2. click on *"Wizard"* to start the saturation via *"Color Transformation Control"* wizard tool (since firmware version 1.4.57).

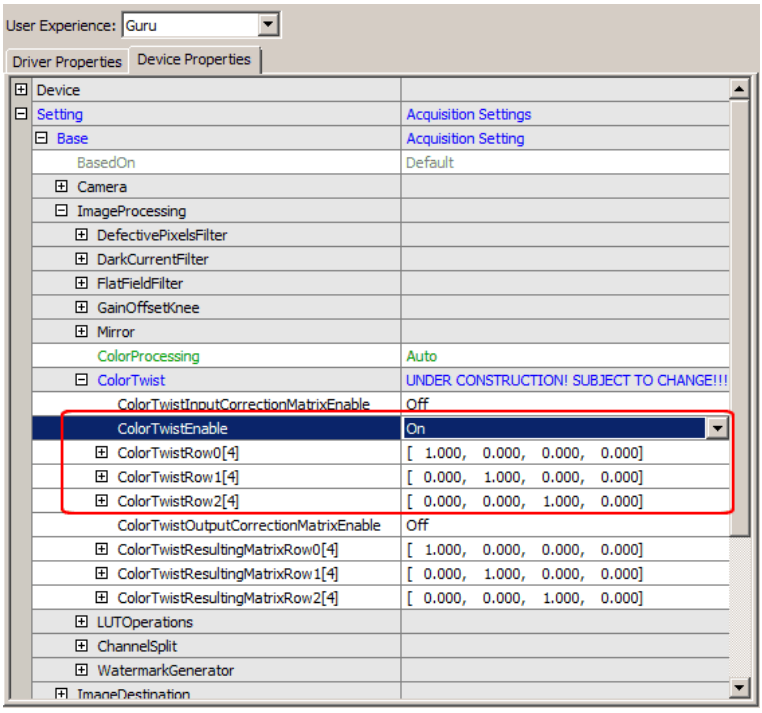


Figure 13: Selected Color Twist Enable and click on wizard will start wizard tool

3. Now, you can adjust the saturation e.g. *"1.1"*.

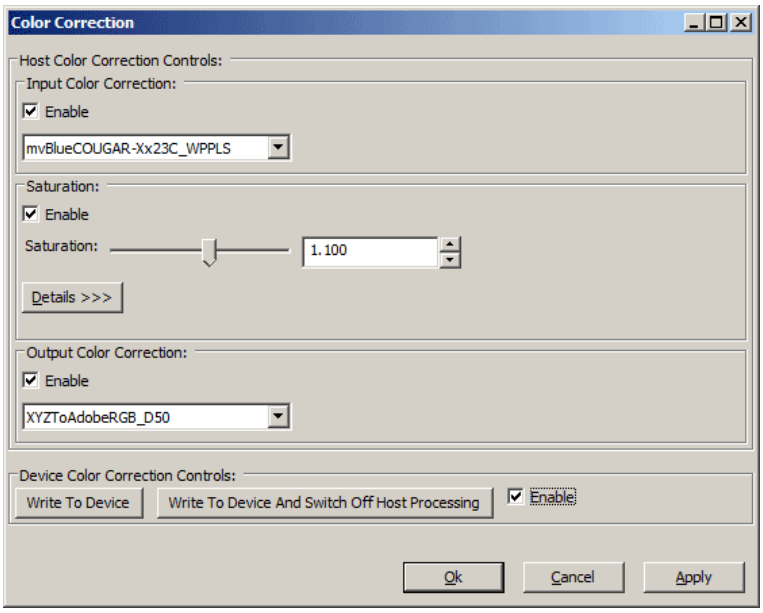


Figure 14: Saturation via Color Transformation Control dialog

4. Afterwards, click on *"Enable"*.
5. Since driver version 2.2.2, it is possible to set the special color correction matrices at
   (a) the input (sensor),
   (b) the output side (monitor) and
   (c) the saturation itself using this wizard.
6. Select the specific input and output matrix and
7. click on *"Enable"*.
8. As you can see, the correction is done by the host (*"Host Color Correction Controls"*).

   Note

   It is not possible to save the settings of the *"Host Color Correction Controls"* in the mvBlueFOX. Unlike in the case of Figure 14, the buttons to write the *"Device Color Correction Controls"* to the mvBlueFOX are not active.

9. Finally, click on *"Apply"*.

After the saturation, the image will look like this:

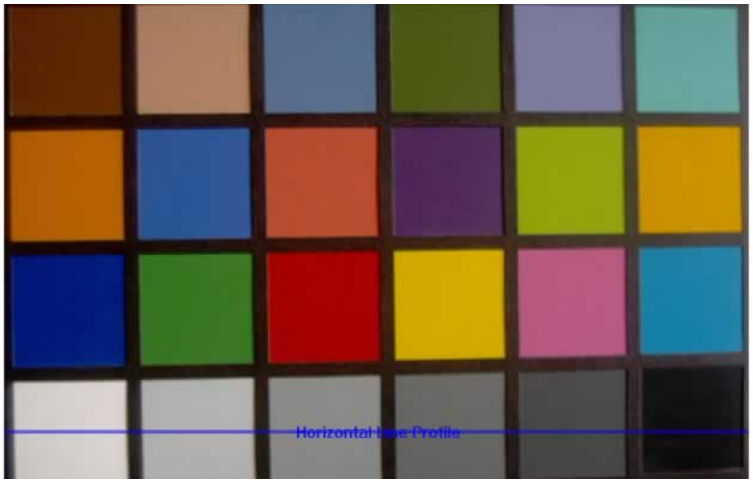


Figure 15: After adapting saturation

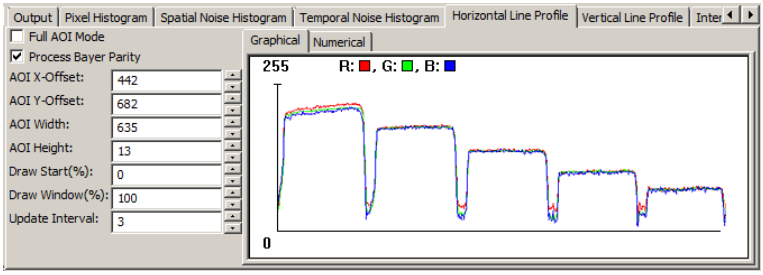


Figure 16: Corresponding histogram after adapting saturation

19.2.3 Working With Gain And Black-Level Values Per Color Channel

In many low-lighting applications the gain needs to be increased to enhance the brighness of the images. However, while the image brightness is increased, the black-level of the image is also increased, which in many cases should be avoided. With the help of the GainOffsetKnee filter it is possible to correct/adjust the overall black-level as well as the black-level per color channel, even when the gain is applied. *Figure 1* shows the working principle of the GainOffsetKnee filter.

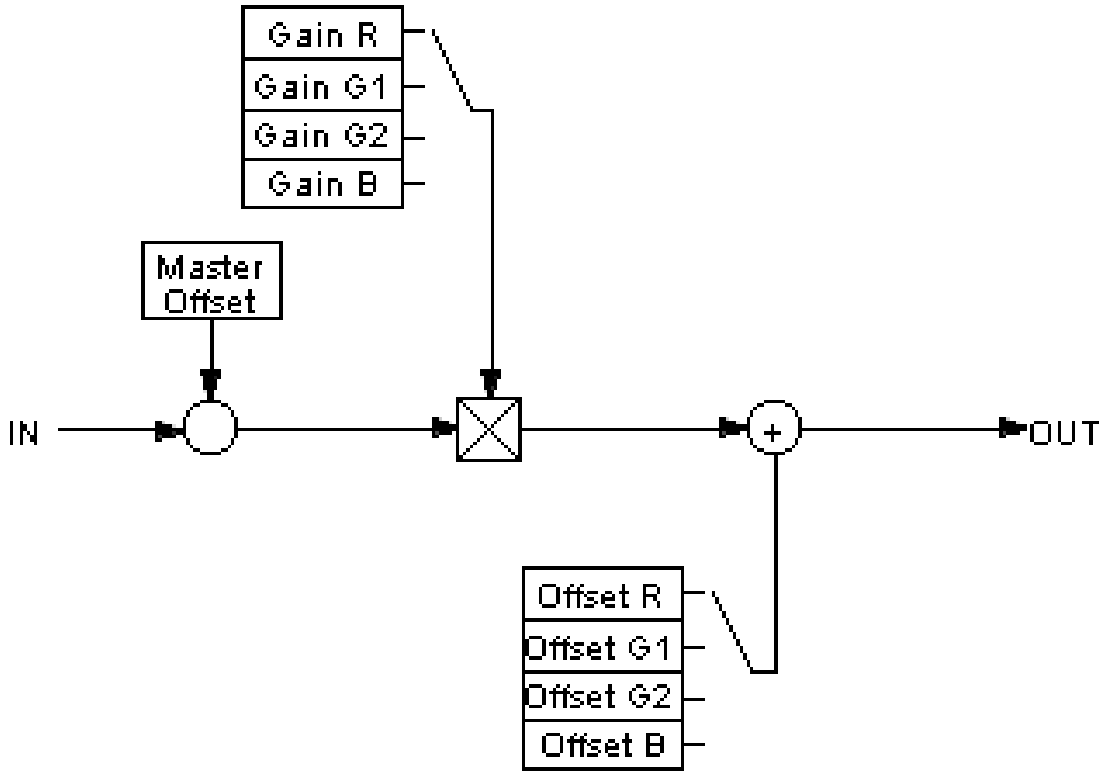


Figure 1: The GainOffsetKnee filter working principle

The GainOffsetKnee filter is one of the image processing methods performed on the host. It allows you to adjust:

- The overall offset (*i.e.* overall black-level) of an image.
- The individual gain per color channel.
- The individual offset (*i.e.* individual black-level) per color channel.

19.2.3.1 Configuration in ImpactControlCenter Here is how to configure the GainOffsetKnee filter in ImpactControlCenter and the impact the filter has on an image:

1. The GainOffsetKnee filter is located under *"Setting -> Base -> ImageProcessing"*.

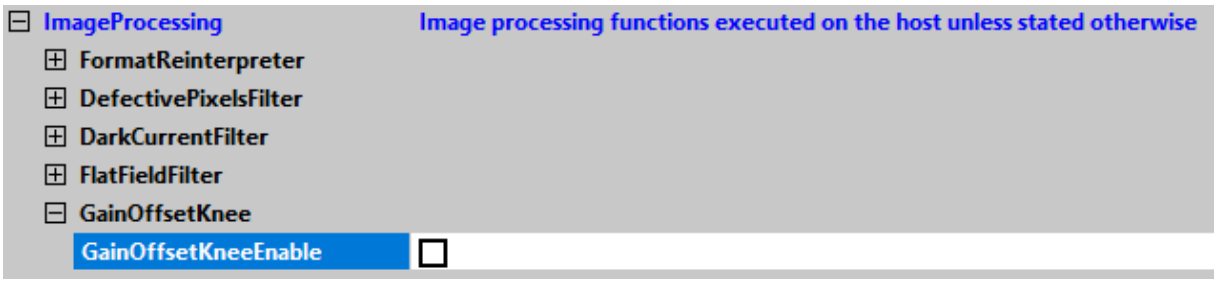


Figure 2: The GainOffsetKnee filter option in ImpactControlCenter

2. Once the GainOffsetKnee filter is activated, the configuration field will be displayed (see *Figure 3*). As an example, the current RGB image is shown in *Figure 4* and its histogram in *Figure 5*.

Figure 3: The configuration field for the GainOffsetKnee filter

Figure 4: An image without the GainOffsetKnee filter

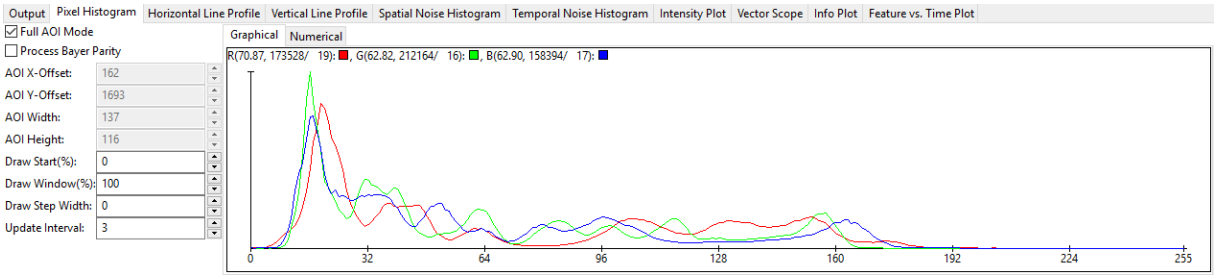


Figure 5: The histogram of *Figure 3*

3. The overall offset can be assigned using the *'GainOffsetKneeMasterOffset_pc'*. A positive offset increases the black-level of the image, whereas a negative offset reduces it. To visualize the effect, an offset of 5% is given as an example, which means that the overall black-level of the image will be increased by 5% of the max. pixel value (*i.e.* 255 in this example). As a result, the overall black-level in the current histogram (see *Figure 8*) has been increase by 12.75 (which is 5% x 255) comparing to the original histogram (see *Figure 5*).

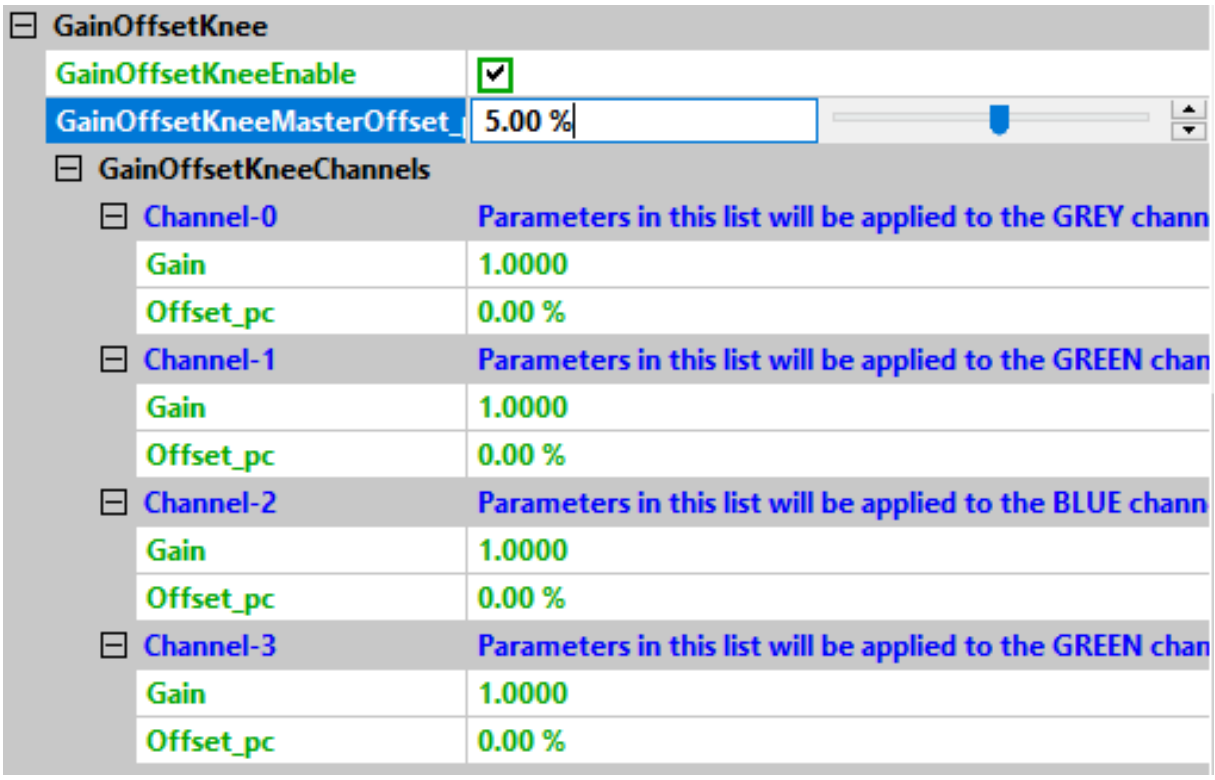


Figure 6: Assign overall/master offset to the image

Figure 7: The image with 5% overall offset

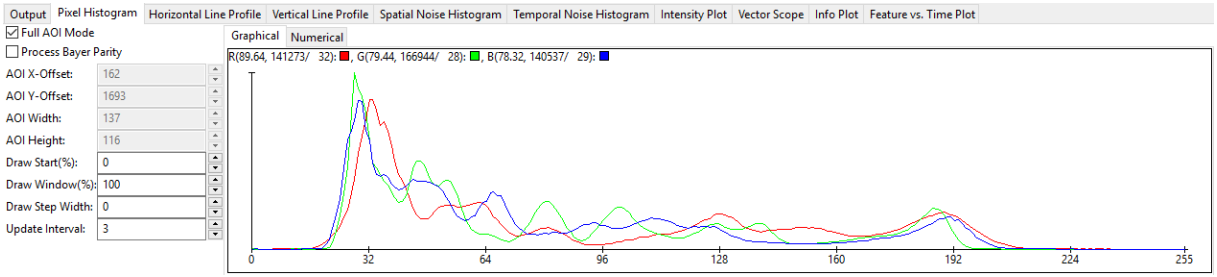


Figure 8: The histogram with 5% overall offset

4. Among the GainOffsetKneeChannels there are 4 channels. For mono images, only channel 0 is used. For RGB images, channel 0-2 are used: red channel, green channel and blue channel respectively. For Bayer images, channel 0-3 are used. For more description please refer to *Figure 3*. As an example, a gain of 1.0625dB is applied to the red channel. As shown in *Figure 10* and *Figure 11*, the grey-level of the red channel is increased while the other two channels remain the same.

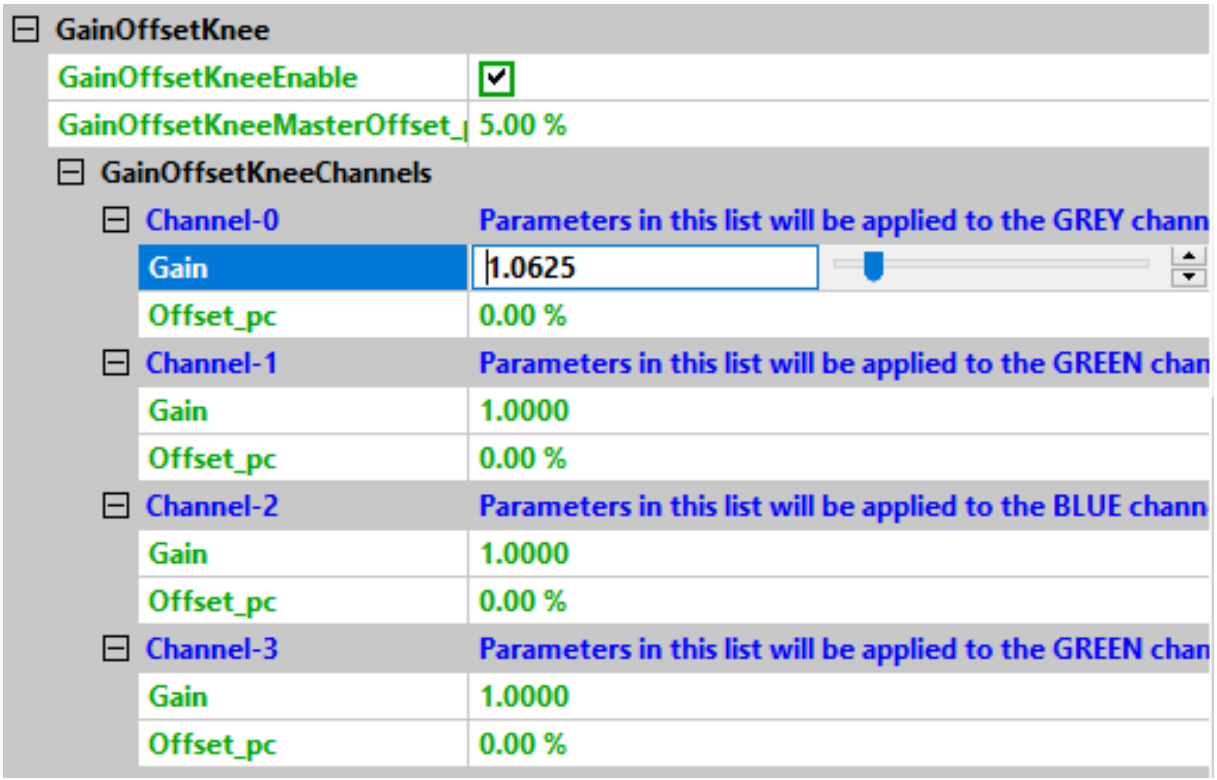


Figure 9: Assign individual gain to the red channel

Figure 10: The image with 1.0625dB gain in the red channel

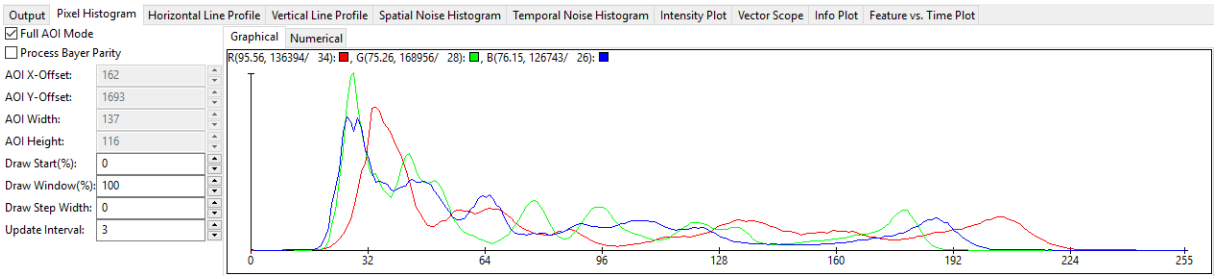


Figure 11: The histogram with 1.0625dB gain in the red channel

5. The individual black-level can be assigned using the channel specific *'Offset_pc'*. Analogous to 'GainOffset↩ KneeMasterOffset_pc', a positive offset increases the black-level of the channel, whereas a negative offset reduces it. To visualize the effect, an offset of 5% is given as an example in the red channel. The histogram (see *Figure 14*) shows therefore a 12.75 (which is 5% x 255) offset increase in the red channel.

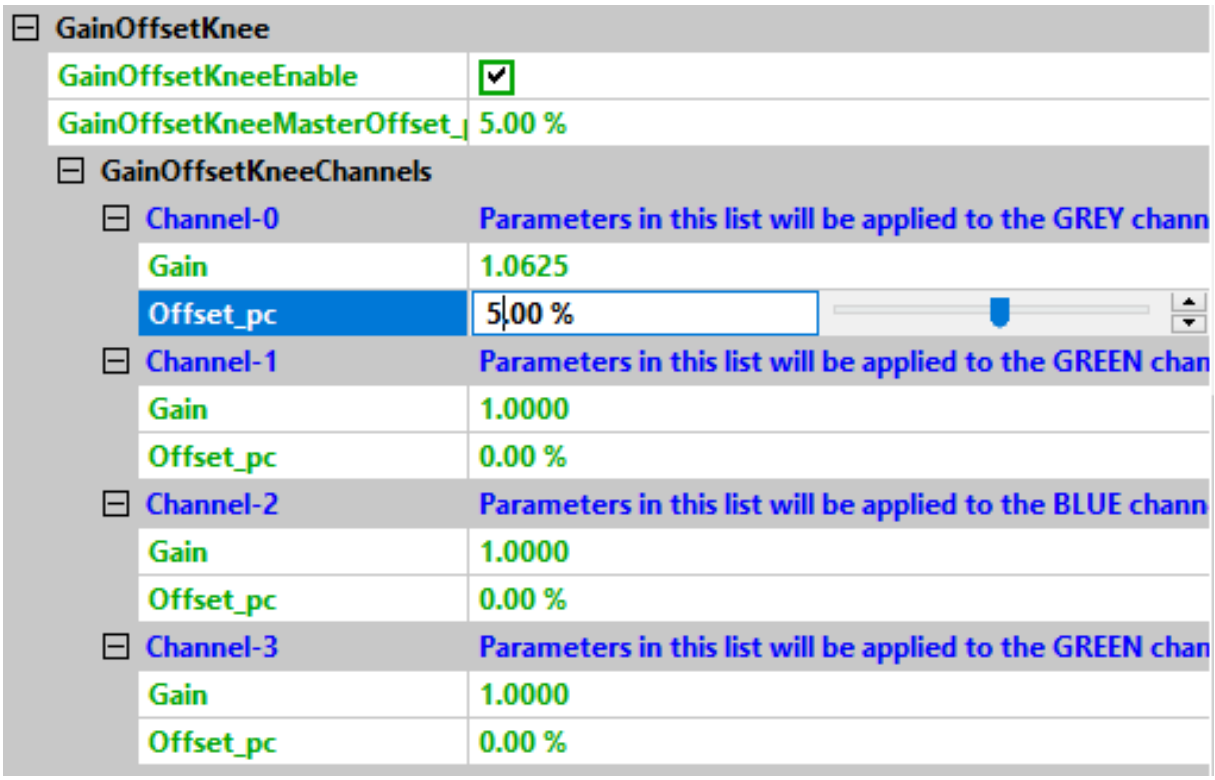


Figure 12: Assign individual offset to the red channel

Figure 13: The image with 5% offset in the red channel

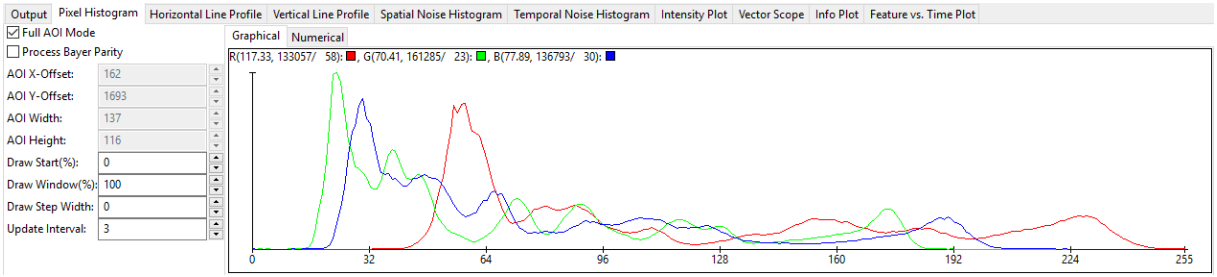


Figure 14: The histogram with 5% offset in the red channel

19.2.3.2 Configuration Using The API Depending on the programming language you are working with the names of classes, namespaces and properties might vary slightly. For C++ please refer to the `GainOffsetKneeChannelParameters` class and the `Image Processing` class for some guidance, for other languages when searching for the *offset* or *knee* properties similar things can be found.

19.3 Saving data on the device

Note

As described in **"Storing, Restoring And Managing Settings"** in the **"Impact Acquire SDK GUI Applications"** manual, it is also possible to save the settings as an XML file on the host system. You can find further information about for example the XML compatibilities of the different driver versions in the Impact Acquire SDK manuals and the according setting classes: https://www.balluff.com/en-de/documentation-for-your-balluff-product/SDK_CPP/classmvIMPACT_1_1acquire_1_1FunctionInterface.html (C++)

There are several use cases concerning device memory:

- Creating user data entries

19.3.1 Creating user data entries

19.3.1.1 Basics about user data

It is possible to save arbitrary user specific data on the hardware's non-volatile memory. The amount of possible entries depends on the length of the individual entries as well as the size of the devices non-volatile memory reserved for storing:

- mvBlueFOX,
- mvBlueFOX-M,
- BVS CA-MLC,
- BVS CA-SF,
- BVS CA-GX0,
- BVS CA-GX2,
- BVS CA-GT1, and
- BVS CA-BN

currently offer 512 bytes of user accessible non-volatile memory of which 12 bytes are needed to store header information leaving 500 bytes for user specific data.

```
One entry will currently consume:
1 + <length_of_name (up to 255 chars)> + 2 + <length_of_data (up to 65535 bytes)> + 1 (access mode) bytes

as well as an optional:
1 + <length_of_password> bytes per entry if a password has been defined for this particular entry
```

It is possible to save either **String** or **Binary** data in the *data* property of each entry. When storing binary data please note, that this data internally will be stored in `Base64` format thus the amount of memory required is 4/3 time the binary data size.

The UserData can be accessed and created using ImpactControlCenter (the device has to be closed). In the section `"UserData"` you will find the entries and following methods:

- `"CreateUserDataEntry"`

- "DeleteUserDataEntry"
- "WriteDataToHardware"

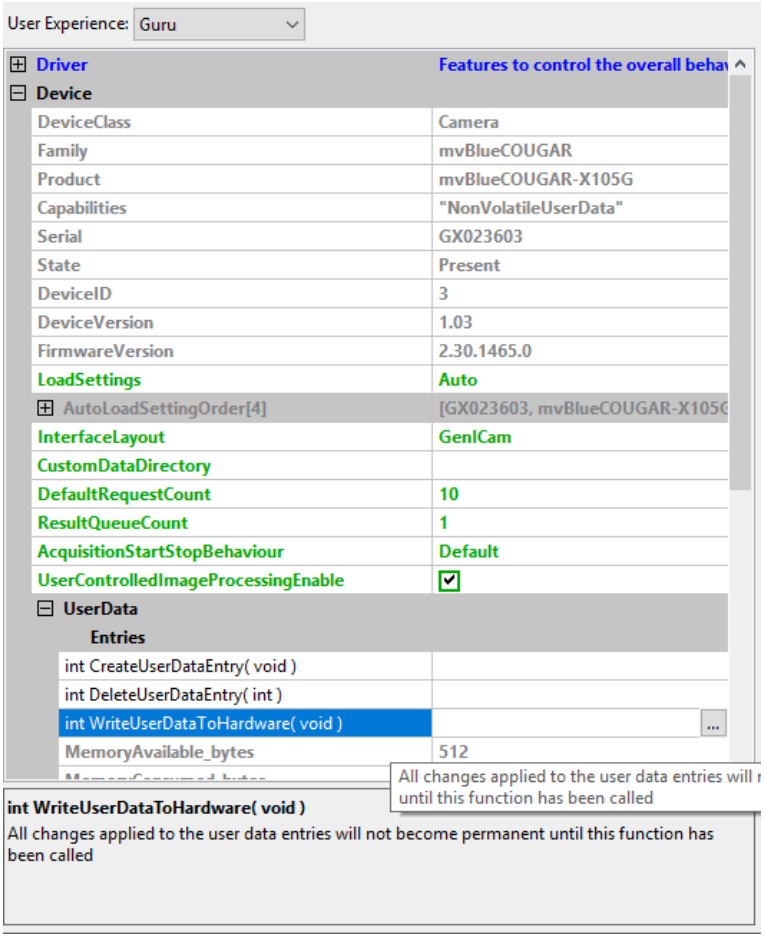


Figure 1: ImpactControlCenter - section "UserData -> Entries"

To create a user data entry, you have to

- Right click on "CreateUserDataEntry"
- Select **"Execute"** from the popup menu.
  An entry will be created.
- In "Entries" click on the entry you want to adjust and modify the data fields.
  To permanently commit a modification made with the keyboard the **ENTER** key **must be pressed**.
- To save the data on the device, you have to execute "WriteDataToHardware". Please have a look at the "Output" tab in the lower right section of the screen as shown in Figure 2, to see if the write process returned with no errors. If an error occurs a message box will pop up.

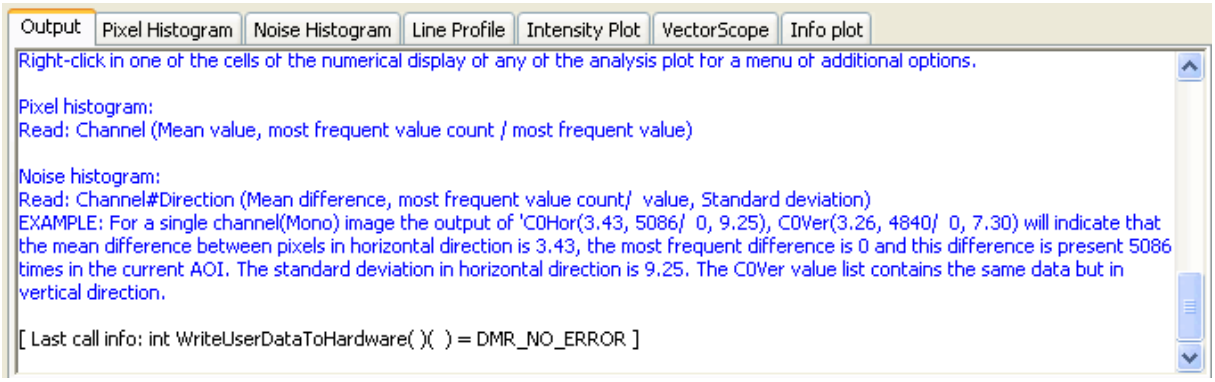


Figure 2: ImpactControlCenter - analysis tool "Output"

19.3.1.2 Coding sample

If you e.g. want to use the UserData as dongle mechanism (with binary data), it is not suggested to use ImpactControlCenter. In this case you have to program the handling of the user data.

See also

mvIMPACT::acquire::UserDataEntry in the manual for the corresponding programming language at the manuals section.

19.4 Working with several cameras simultaneously

There are several use cases concerning multiple cameras:

- Using 2 BVS CA-MLC cameras in Master-Slave mode
- Synchronize the cameras to expose at the same time

19.4.1 Using 2 BVS CA-MLC cameras in Master-Slave mode

19.4.1.1 Scenario If you want to have a synchronized stereo camera array (e.g. BVS CA-MLC-202dG) with a rolling shutter master camera (e.g. BVS CA-MLC-202dC), you can solve this task as follows:

1. Please check, if all cameras are using firmware version 1.12.16 or newer.
2. Now, open ImpactControlCenter and set the master camera:

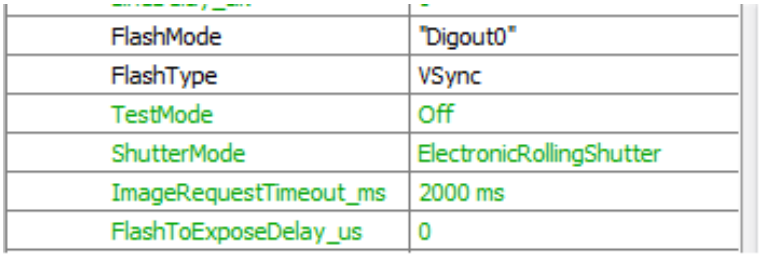


Figure 1: ImpactControlCenter - Master camera outputs at DigOut 0 a frame synchronous V-Sync pulse

Note

Alternatively, it is also possible to use HRTC - Hardware Real-Time Controller HRTC to set the master camera. The following sample shows the HRTC - Hardware Real-Time Controller HRTC program which sets the trigger signal and the digital output.
The sample will lead to a constant frame rate of 16 fps (50000 us + 10000 us = 60000 us for one cycle. 1 / 60000 us $*$ 1000000 = 16.67 Hz).

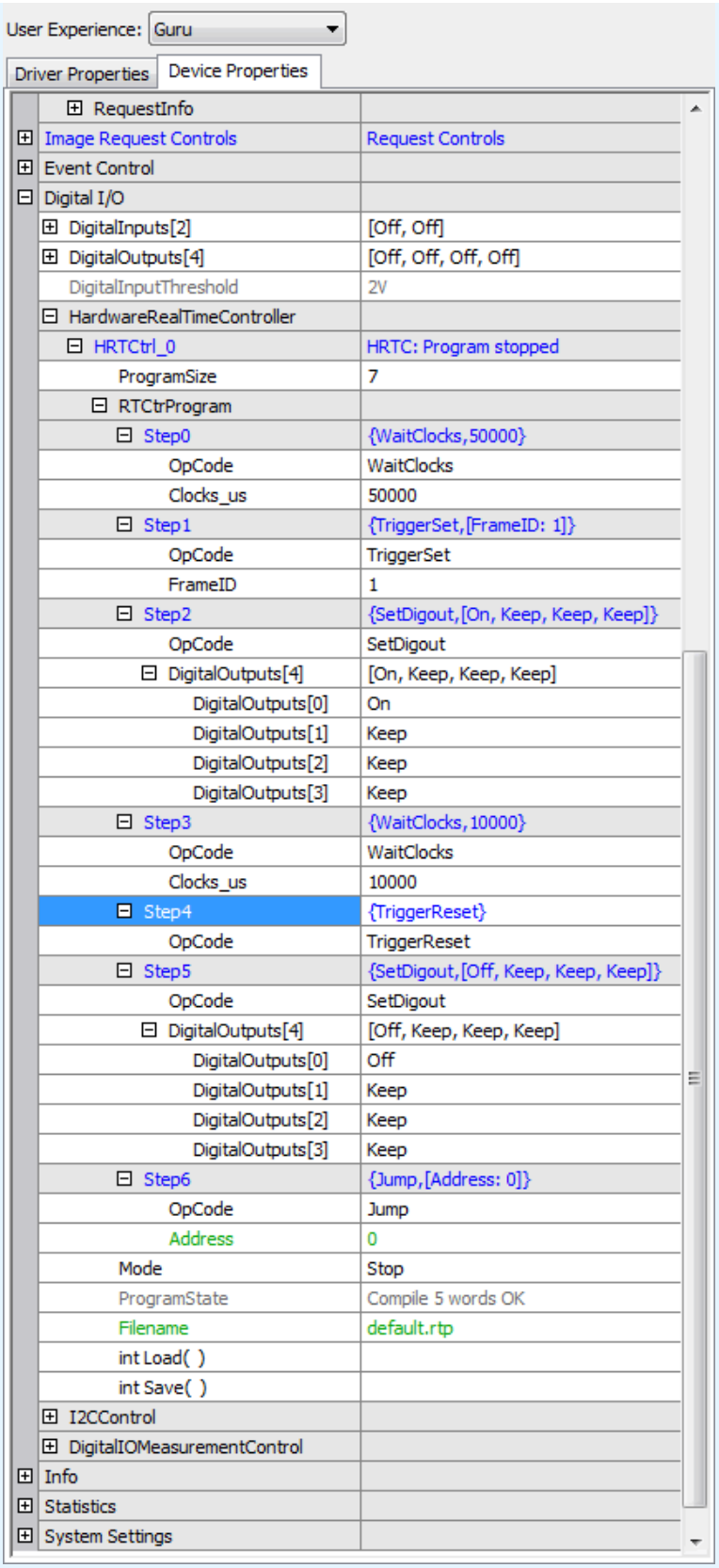


Figure 2: ImpactControlCenter - HRTC program sets the trigger signal and the digital output

Do not forget to set HRTC as the trigger source for the master camera.

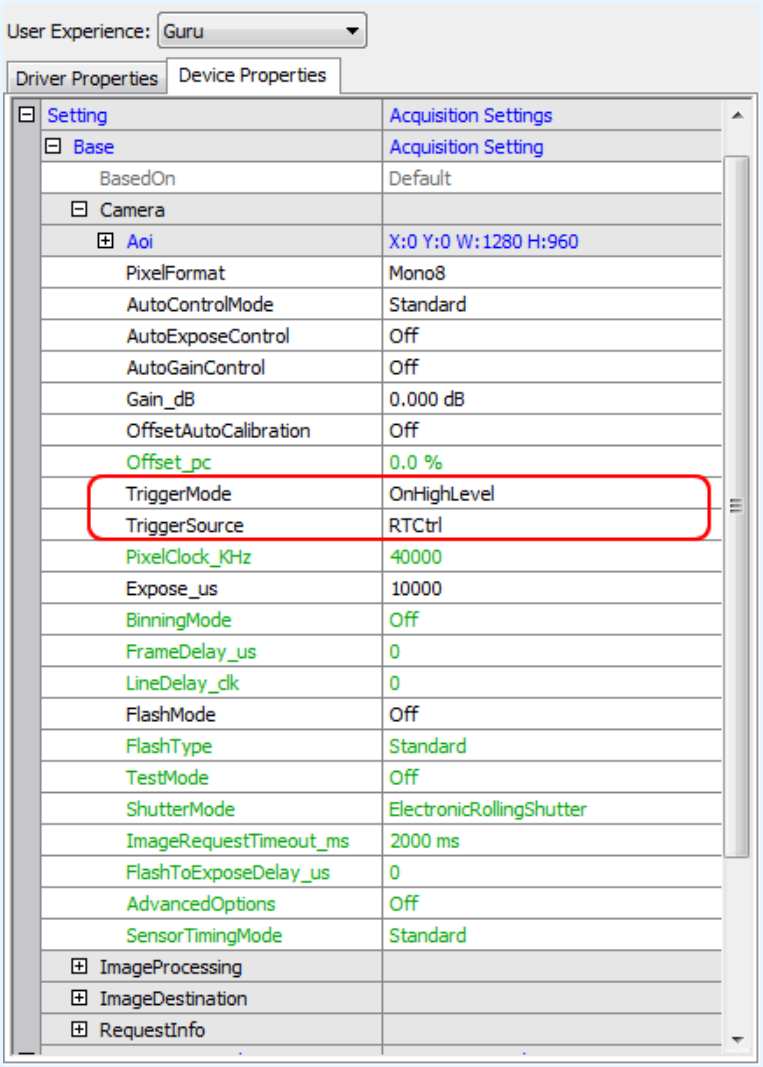


Figure 3: ImpactControlCenter - HRTC is the trigger source for the master camera

3. Then, set the slave with ImpactControlCenter :

Figure 4: ImpactControlCenter - Slave camera with TriggerMode "OnHighLevel" at DigIn 0

19.4.1.1.1 Connection using -UOW versions (opto-isolated inputs and outputs) The devices should be connected like this:

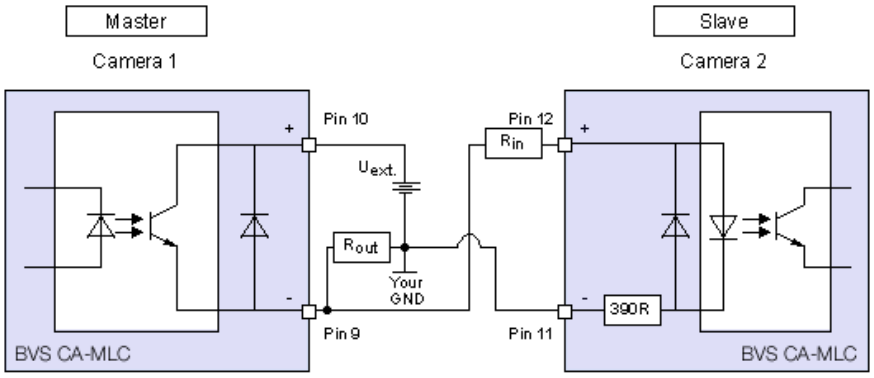


Figure 5: Connection with opto-isolated digital inputs and outputs

Symbol	Comment	Input voltage	Min	Typ	Max	Unit
$U_{ext.}$	External power		3.↩ 3		30	V
R_{out}	Resistor digital output			2		kOhm
R_{in}	Resistor digital input	3.3 V .. 5 V		0		kOhm
		12 V		0.68		kOhm
		24 V		2		kOhm

You can add further slaves.

19.4.1.1.2 Connection using -UTW versions (TTL inputs and outputs) The devices should be connected like this:

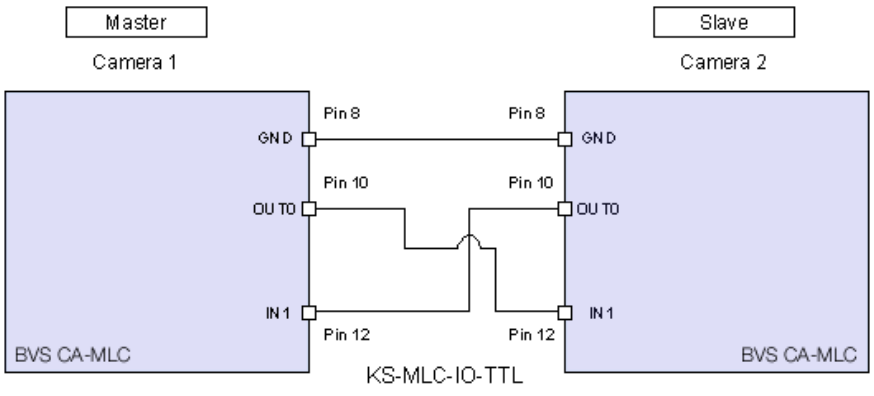


Figure 6: Connection with TTL digital inputs and outputs

For this case we offer a synchronization cable called **"KS-MLC-IO-TTL 00.5"**.

Note

There a no further slaves possible.

See also

- Dimensions and connectors Figure 18 pin reference.
- Dimensions and connectors Table of connector pin out of "12-pin through-hole type shrouded header (USB / Dig I/O)".
- Dimensions and connectors Electrical drawing "opto-isolated digital inputs" and "opto-isolated digital outputs".
- A predefined frame rate is also possible using HRTC.

19.4.2 Synchronize the cameras to expose at the same time

This can be achieved by connecting the same external trigger signal to one of the digital inputs of each camera like it's shown in the following figure:

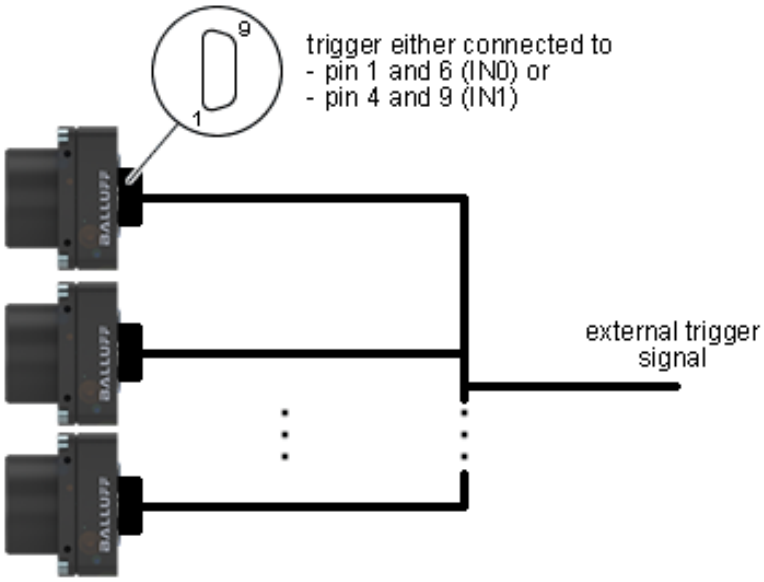


Figure 1: Electrical setup for sync. cameras

Each camera then has to be configured for external trigger somehow like in the image below:

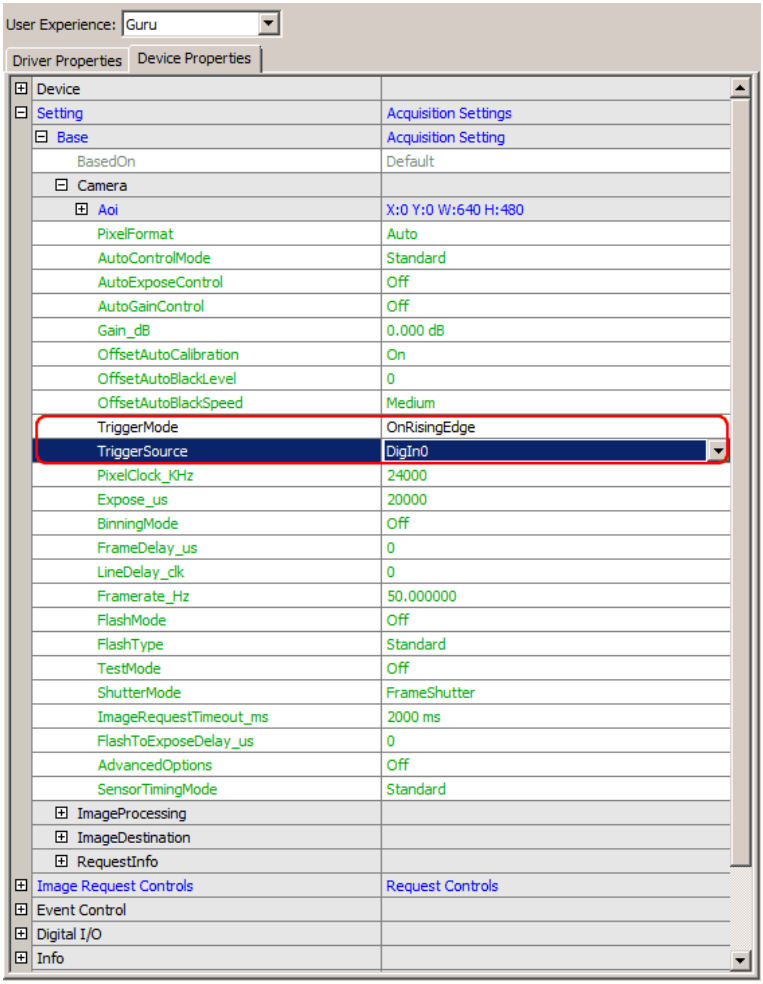


Figure 2: ImpactControlCenter - Setup for sync. cameras

This assumes that the image acquisition shall start with the rising edge of the trigger signal. Every camera must be configured like this. Each rising edge of the external trigger signal then will start the exposure of a new image at the same time on each camera. Every trigger signal that will occur during the exposure of an image will be silently discarded.

19.5 Working with HDR (High Dynamic Range Control)

There are several use cases concerning High Dynamic Range Control:

- Adjusting sensor of camera models with onsemi MT9V034
- Adjusting sensor of camera models with onsemi MT9M034

19.5.1 Adjusting sensor of camera models with onsemi MT9V034

19.5.1.1 Introduction

The HDR (High Dynamic Range) mode available for cameras built with the onsemi MT9V034 chip increases the usable contrast range. This is achieved by dividing the integration time in two or three phases. The exposure time proportion of the three phases can be set independently. Furthermore, it can be set, how much signal of each phase is charged.

19.5.1.2 Functionality

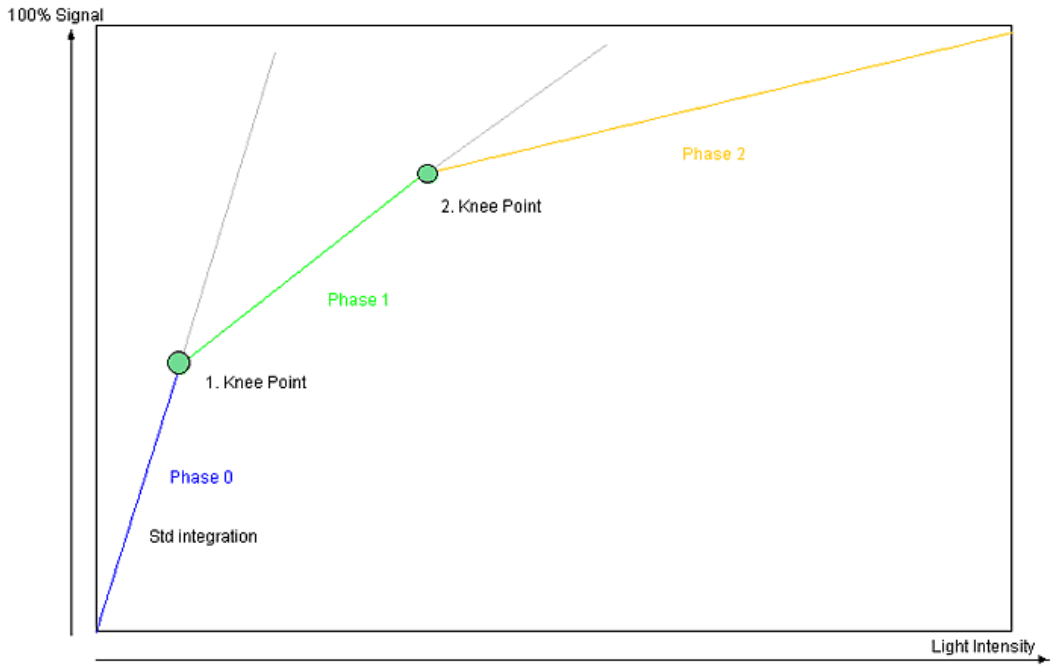


Figure 1: Diagram of the HDR mode

19.5.1.2.1 Description

- **"Phase 0"**
  - During T1 all pixels are integrated until they reach the defined signal level of Knee Point 1.
  - If one pixel reaches the level, the integration will be stopped.
  - During T1 no pixel can reached a level higher than P1.
- **"Phase 1"**
  - During T2 all pixels are integrated until they reach the defined signal level of Knee Point 2.
  - T2 is always smaller than T1 so that the percentage compared to the total exposure time is lower.
  - Thus, the signal increase during T2 is lower as during T1.
  - The max. signal level of Knee Point 2 is higher than of Knee Point 1.
- **"Phase 2"**
  - During T2 all pixels are integrated until the possible saturation.
  - T3 is always smaller than T2, so that the percentage compared to the total exposure time is again lower here.
  - Thus, the signal increase during T3 is lower as during T2.

For this reason, darker pixels can be integrated during the complete integration time and the sensor reaches its full sensitivity. Pixels, which are limited at each Knee Points, lose a part of their integration time - even more, if they are brighter.

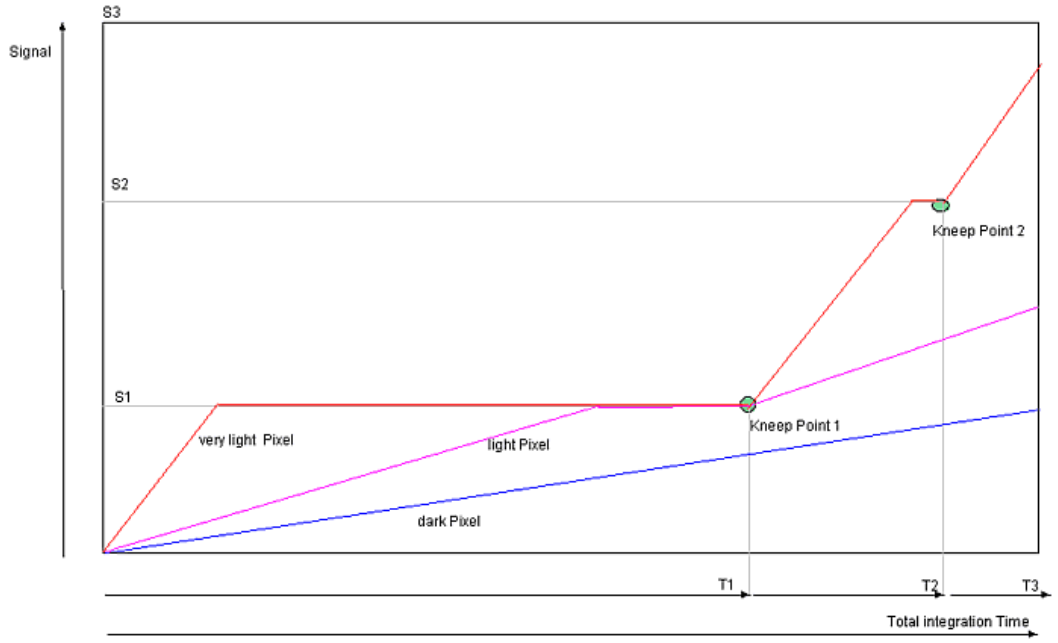


Figure 2: Integration time of different bright pixels

In the diagram you can see the signal line of three different bright pixels. The slope depends of the light intensity , thus it is per pixel the same here (granted that the light intensity is temporally constant). Given that the very light pixel is limited soon at the signal levels S1 and S2, the whole integration time is lower compared to the dark pixel. In practice, the parts of the integration time are very different. T1, for example, is 95% of T_{total}, T2 only 4% and T3 only 1%. Thus, a high decrease of the very light pixels can be achieved. However, if you want to divide the integration thresholds into three parts that is S2 = 2 x S1 and S3 = 3 x S1, a hundredfold brightness of one pixel's step from S2 to S3, compared to the step from 0 and S1 is needed.

19.5.1.3 Using HDR

Figure 3 is showing the usage of the HDR mode. Here, an image sequence was created with the integration time between 10us and 100ms. You can see three slopes of the HDR mode. The "waves" result from the rounding during the three exposure phases. They can only be partly adjusted during one line period of the sensor.

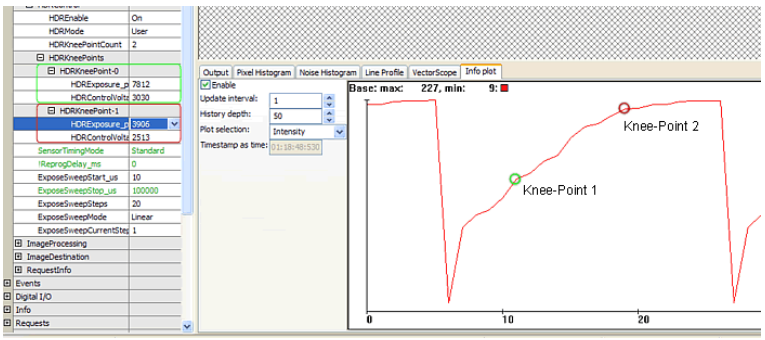


Figure 3: ImpactControlCenter HDR screenshot

19.5.1.3.1 Notes about the usage of the HDR mode

- In the HDR mode, the basic amplification is reduced by approx. 0.7, to utilize a huge, dynamic area of the sensor.
- If the manual gain is raised, this effect will be reversed.
- Exposure times, which are too low, make no sense. During the third phase, if the exposure time reaches a possible minimum (one line period), a sensible lower limit is reached.

19.5.1.3.2 Possible settings

Possible settings of HDR mode are:

"HDREnable":

- **"Off"**: Standard mode
- **"On"**: HDR mode on, reduced amplification:
- **"HDRMode"**:
  - **"Fixed"**: Fixed setting with 2 Knee Points. modulation Phase 0 .. 33% / 1 .. 66% / 2 .. 100%
  - **"Fixed0"**: Phase 1 exposure 12.5% , Phase 2 31.25% of total exposure
  - **"Fixed1"**: Phase 1 exposure 6.25% , Phase 2 1.56% of total exposure
  - **"Fixed2"**: Phase 1 exposure 3.12% , Phase 2 0.78% of total exposure
  - **"Fixed3"**: Phase 1 exposure 1.56% , Phase 2 0.39% of total exposure
  - **"Fixed4"**: Phase 1 exposure 0.78% , Phase 2 0.195% of total exposure
  - **"Fixed5"**: Phase 1 exposure 0.39% , Phase 2 0.049% of total exposure
- **"User"**: Variable setting of the Knee Point (1..2), threshold and exposure time proportion
  - **"HDRKneePointCount"**: Number of Knee Points (1..2)
  - **"HDRKneePoints"**
    - **"HDRKneePoint-0"**
      - **"HDRExposure_ppm"**: Proportion of Phase 0 compared to total exposure in parts per million (ppm)
      - **"HDRControlVoltage_mV"**: Control voltage for exposure threshold of first Knee Point (3030mV is equivalent to approx. 33%)
    - **"HDRKneePoint-1"**
      - **"HDRExposure_ppm"**: Proportion of Phase 1 compared to total exposure in parts per million (ppm)
      - **"HDRControlVoltage_mV"**: Control voltage for exposure threshold of first Knee Point (2630mV is equivalent to approx. 66%)

19.5.2 Adjusting sensor of camera models with onsemi MT9M034

19.5.2.1 Introduction

The HDR (High Dynamic Range) mode of the Aptina sensor increases the usable contrast range. This is achieved by dividing the integration time in three phases. The exposure time proportion of the three phases can be set independently.

19.5.2.2 Functionality

To exceed the typical dynamic range, images are captured at 3 exposure times with given ratios for different exposure times. The figure shows a multiple exposure capture using 3 different exposure times.

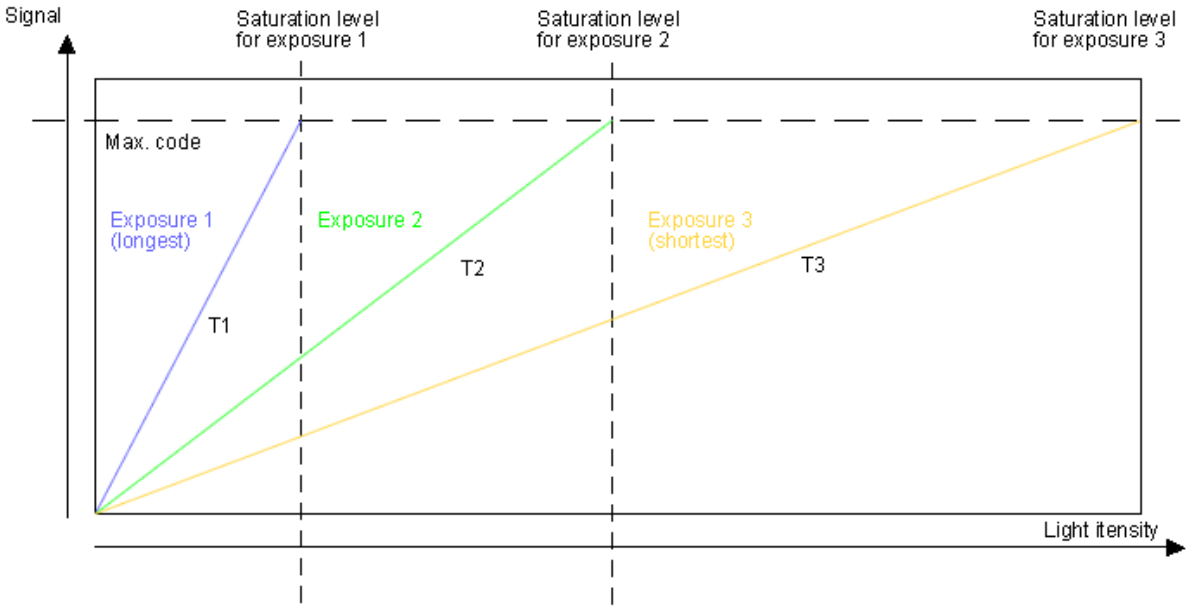


Figure 1: Multiple exposure capture using 3 different exposure times

> **Note**
>
> The longest exposure time (T1) represents the *Exposure_us* parameter you can set in ImpactControl↩ Center.

Afterwards, the signal is fully linearized before going through a compander to be output as a piece-wise linear signal. the next figure shows this.

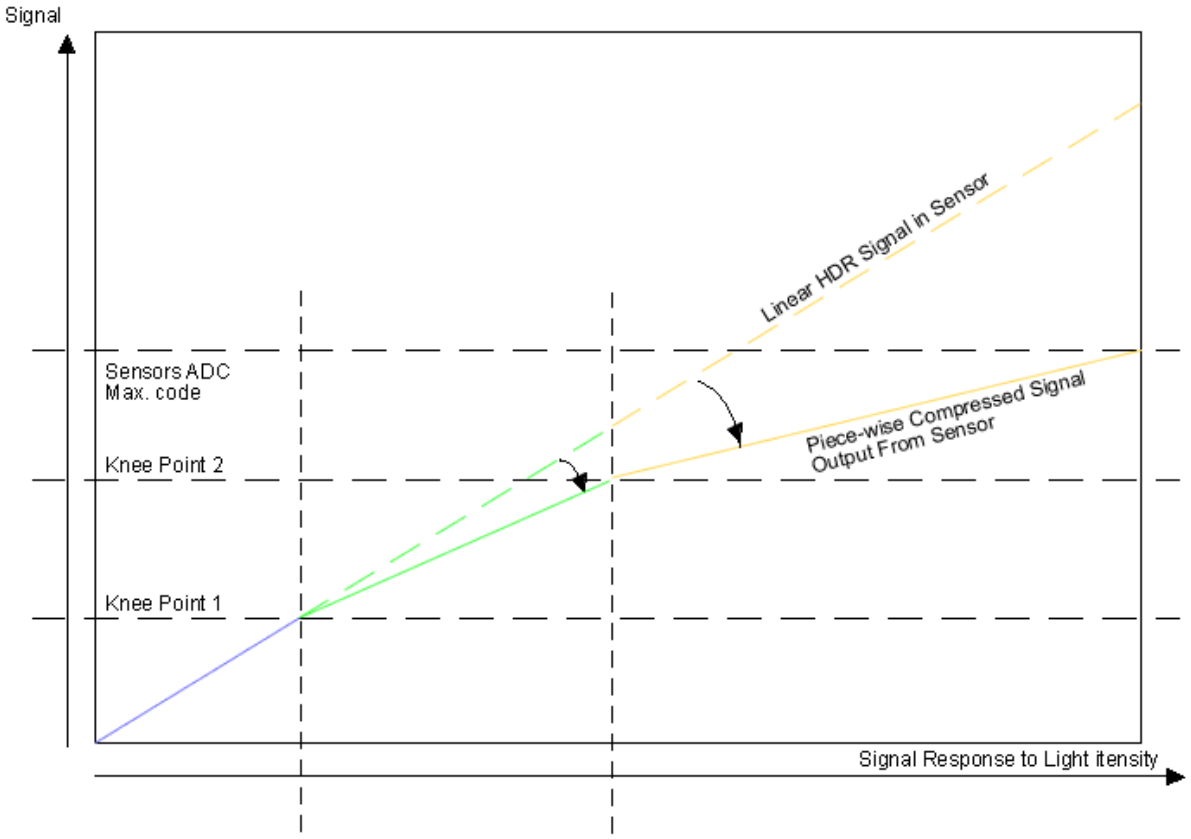


Figure 2: Piece-wise linear signal

19.5.2.2.1 Description

Exposure ratios can be controlled by the program. Two rations are used: R1 = T1/T2 and R2 = T2/T3.

Increasing R1 and R2 will increase the dynamic range of the sensor at the cost of lower signal-to-noise ratio (and vice versa).

19.5.2.2.2 Possible settings

Possible settings in HDR mode are:

- **"HDREnable"**:
  - **"Off"**: Standard mode
  - **"On"**: HDR mode on, reduced amplification
    - **"HDRMode"**:
      - **"Fixed"**: Fixed setting with exposure-time-ratios: T1 -> T2 ratio / T2 -> T3 ratio
      - **"Fixed0"**: 8 / 4
      - **"Fixed1"**: 4 / 8
      - **"Fixed2"**: 8 / 8
      - **"Fixed3"**: 8 / 16
      - **"Fixed4"**: 16 / 16
      - **"Fixed5"**: 16 / 32

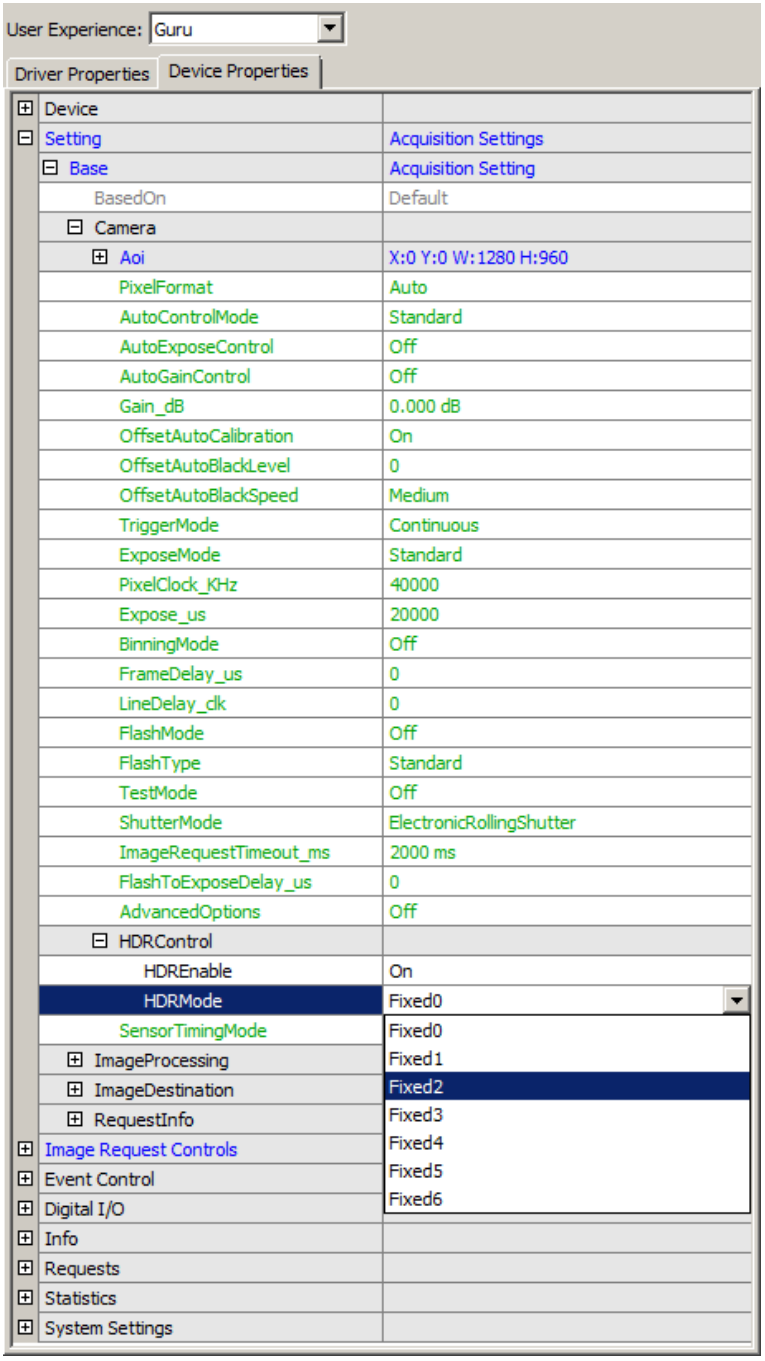


Figure 3: ImpactControlCenter - Working with the HDR mode

19.6 Working with I2C devices

Note

Please find a detailed description of the I2C interface class **I2CControl** in the **"Impact Acquire SDK "** manuals.

- Working with the I2C interface (I2C Control)
- Using BVS CA-MLC with motorized lenses (MotorFocusControl)

19.6.1 Working with the I2C interface (I2C Control)

19.6.1.1 Introduction As mentioned in the **Device specific interface layout** section of the Impact Acquire API manuals, the **"I2CControl"** is a feature which allows the mvBlueFOX device to communicate with custom-specific peripherals via the I2C interface.

19.6.1.2 Setting up the device The following steps will explain how to connect the mvBlueFOX with an I2C device. In this use case, a LM75 I2C Digital Temperature Sensor is used:

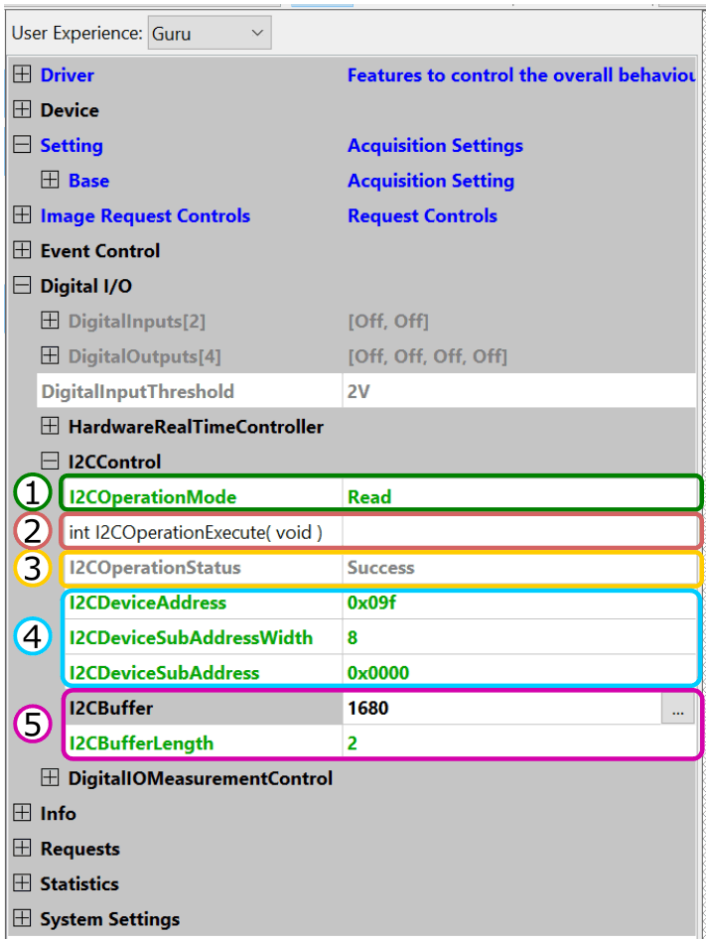


Figure 1: ImpactControlCenter - I2C Interface Control

1. Start ImpactControlCenter
2. Initialize the mvBlueFOX device
3. Navigate to *"Digital I/O -> I2CControl"*
4. Enter the connection settings to address the I2C device memory (4).
   E.g. to read the temperature of the example sensor, set **I2CDeviceAddress** to **"0x09F"**.
5. Set **I2CDeviceSubAddress** to **"0x00"**, because reading the temperature of this sensor does not need a sub address.
6. Set **I2CBufferLength** (5) to **2**, because this sensor stores the temperature in a register of two bytes length.
7. Set the **I2COperationMode** (1) to **Read**. The least significant bit (LSB) of the device address will automatically be overwritten by **0** for writing and by **1** for reading.

8. Navigate to the **int I2COperationExecute(void)** function (2). Click on the 3 dots next to the function name or right-click on the command and then select *Execute* from the context menu to send the current message to the I2C device.
9. The **I2COperationStatus** property (3) will display if the operation was successful.
10. On success, the data is now displayed in the **I2CBuffer** property (5). This value is a hexadecimal number and needs to be interpreted depending on the I2C device.

Note

The following I2C addresses will be blocked for access from an application:

i2c address range	affected devices
0x20-0x3F	all mvBlueFOX devices
0x66-0x67	all mvBlueFOX devices
0x90-0x91	mvBlueFOX-200w only
0xA0-0xA3	all mvBlueFOX devices
0xA6-0xA7	all mvBlueFOX devices
0xBA-0xBB	mvBlueFOX-202a and mvBlueFOX-205 only

19.6.1.3 Programming the I2C interface The following lines of source code are meant to give an overview of the possibilities of the I2CControl class.

```
I2CControl i2cc( pBF ); // assuming 'pBF' to be a valid 'Device*' instance to an mvBlueFOX device
if( i2cc.I2COperationMode.isValid() )
{
  // direct property access
  i2cc.I2CBufferLength.write( 0 );
  i2cc.I2COperationMode.write( I2ComRead );
  assert( ( i2cc.I2COperationExecute.call() == DMR_INVALID_PARAMETER ) && "Unexpected driver behaviour" );
  assert( ( i2cc.I2COperationStatus.read() == I2CosNotEnoughData ) && "Unexpected driver behaviour" );
  i2cc.I2CBufferLength.write( 1 );
  // assuming we write to an invalid address
  assert( ( i2cc.I2COperationExecute.call() == DMR_EXECUTION_FAILED ) && "Unexpected driver behaviour" );
  assert( ( i2cc.I2COperationStatus.read() == I2CosFailure ) && "Unexpected driver behaviour" );
  i2cc.I2COperationMode.write( I2ComWrite );
  i2cc.I2CBuffer.writeBinary( string() );
  assert( ( i2cc.I2COperationExecute.call() == DMR_INVALID_PARAMETER ) && "Unexpected driver behaviour" );
  assert( ( i2cc.I2COperationStatus.read() == I2CosNotEnoughData ) && "Unexpected driver behaviour" );
  {
    char binData[2] = { 'A', 'B' };
    i2cc.I2CBuffer.writeBinary( string(binData, sizeof(binData)) );
  }
  // assuming we write to an invalid address
  assert( ( i2cc.I2COperationExecute.call() == DMR_EXECUTION_FAILED ) && "Unexpected driver behaviour" );
  assert( ( i2cc.I2COperationStatus.read() == I2CosFailure ) && "Unexpected driver behaviour" );
  // Write some data. This will only work if several conditions are met:
  // - there is a device that can be written to at address 0xA6
  // - the sub-address 0x04 is valid
  // - the device is designed to work with 8 bit sub-addresses
  // - the device can deal with 9 bytes in a single command
  i2cc.I2CDeviceAddress.write( 0xA6 );
  i2cc.I2CDeviceSubAddress.write( 0x04 );
  i2cc.I2CDeviceSubAddressWidth.write( 8 );
  {
    char binData[9] = { 'D', 'E', 'A', 'D', ' ', 'B', 'E', 'E', 'F' };
    i2cc.I2CBuffer.writeBinary( string( binData, sizeof( binData ) ) );
  }
  i2cc.I2COperationMode.write( I2ComWrite );
  int I2COperationExecuteResult = i2cc.I2COperationExecute.call();
  if( I2COperationExecuteResult != DMR_NO_ERROR )
  {
```

```
    printf( "'I2COperationExecute' write failed. Return value: %s(%d).\n", ImpactAcquireException::getErrorCod
  }
  printf( "'I2COperationStatus' after write: %s.\n", i2cc.I2COperationStatus.readS().c_str() );
  // Read some data. Similar condition as for write apply
  const int bytesToRead = 4;
  i2cc.I2CDeviceAddress.write( 0xA8 );
  i2cc.I2CDeviceSubAddress.write( 0x00 );
  i2cc.I2CDeviceSubAddressWidth.write( 8 );
  i2cc.I2CBufferLength.write( bytesToRead ); // read 'bytesToRead' bytes
  i2cc.I2COperationMode.write( I2ComRead );
  i2cc.I2COperationExecute.call();
  I2COperationExecuteResult = i2cc.I2COperationExecute.call();
  if( I2COperationExecuteResult != DMR_NO_ERROR )
  {
    printf( "'I2COperationExecute' read failed. Return value: %s(%d).\n", ImpactAcquireException::getErrorCode
  }
  printf( "'I2COperationStatus' after read: %s.\n", i2cc.I2COperationStatus.readS().c_str() );
  if( i2cc.I2CBuffer.binaryDataBufferSize() != bytesToRead )
  {
    printf( "'I2CBuffer' reports %d bytes of data while %d bytes where expected.\n", i2cc.I2CBuffer.binaryData
  }
  // usage of the convenience functions
  i2cc.I2CWrite( 0xA4, 0x00, 8, string("TEST") );
  const string i2cReadBuffer = i2cc.I2CRead( 0xA4, 0x00, 8, 4 );
}
else
{
  printf( "I2CControl not available.\n" );
}
```

19.6.2 Using BVS CA-MLC with motorized lenses (MotorFocusControl)

19.6.2.1 Introduction It is possible to use the BVS CA-MLC with a motorized lens mount without optical filter. To control this lens mount you use either ImpactControlCenter or the class **MotorFocusControl** from Impact Acquire for your own applications.

This camera option consists of a single-board camera module plus additional interface board. The camera is equipped with a board-to-wire connector on the interface board for opto-coupled I/O connection and a Mini USB. Other versions with TTL coupled inputs and outputs and without Mini USB connector are available on demand. There is no optical filter in the light path. For color applications Balluff supplies S-mount lenses with IR absorbing coating.

Important technical data of the motorized lens mount: (Consult Balluff for more detailed data, application notes and the full lens command reference manual)

Lens Type	(Lens not included), accepts M12x0.5mm, smaller lenses to M8x0.35 with adapter from your lens supplier
Lens Weight*	< 5 grams
Travel Range	Up to 1.5 mm
Housing Dimension	20 x 22 x 16 mm
Max Image Sensor Area (image sensor not included)	17 x 17 x 1.25 mm (including 1/2" and 1/1.8" formats)
Speed	> 5 mm/s
Resolution	0.5 um
Hysteresis	None
Repeatability Uni-directional	+/- 5 um
Bi-directional	+/- 20 um
Linear Accuracy	+/- 30 um
Angular alignment (Static tip/tilt)	> +/- 1 degree
Angular movement (Dynamic tip/tilt)	> +/- 0.15 degree
Static Concentricity	> +/- 0.25 mm
Dynamic Concentricity	> +/- 0.02 mm
Input Voltage	3.1 to 3.6 Volts
Input Power**	< 0.5 Watts (5mm/s with 5g mass) < 0.13 Watts quiescent
Temperature / RH***	5 ° to 70 ℃ (lower possible) < 95% RH non-condensing
Mean Time Before Failure	> 2M Cycles (fixed orientation) 500K Cycles (random orientation)
Weight of module (without lens)	5.8 grams
Compliance	CE / RoHS

* Fixed orientation will allow for heavier lens operation. Consult Balluff if your lens does not fit or cannot be focused.
** Power depends on input voltage, speed & load. *** Consult Balluff if you have lower temperature requirements.

Note

By default, the motorized lens mount is shipped in "closed loop mode". This means that the motor will move to and keep its absolute position in a permanent loop. Therefore you will hear a high frequent sound if you touch the lens or if you adjust the focus manually while the module is operating. For this reason, we recommend to use the "open loop mode" while adjusting the lens.
In fixed focus applications you might also consider switching to open loop mode. This will also extend the life time of the motor and reduce operation noise. Please note that the lens will remain at its position also in open loop mode due to mechanical friction.
The open loop mode can be set via MotorFocusSendBuffer (value: "<20 0>") and then call MotorFocusSend either in ImpactControlCenter or by programming.

19.6.2.2 Controlling motorized lens mount with ImpactControlCenter

Figure 1 shows a live snap of a scene with different focus planes.

Figure 1: ImpactControlCenter - Preview with blurred background and focused label

To control the motorized lens mount via ImpactControlCenter you have to change the **"UserExperience"** either to "Expert" or "Guru". Afterwards, there is a special section in "Digital I/O" called "MotorFocus↩Control" in the **"Device Properties"** tab:

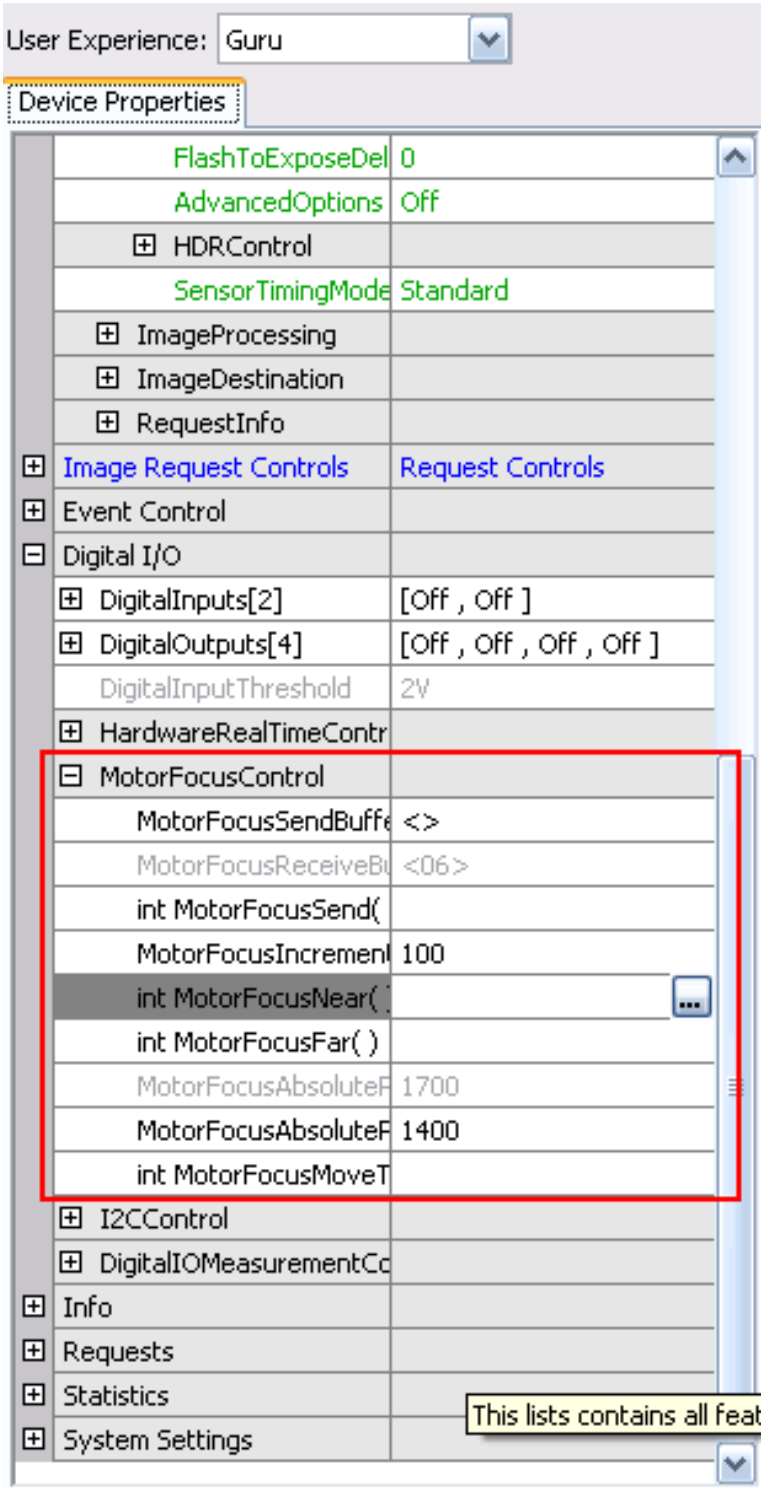


Figure 2: ImpactControlCenter - MotorFocusControl in section Digital I/O

This is a wrapper of the **MotorFocusControl** class which makes the following methods and properties available in the GUI:

- Properties:
  - `MotorFocusAbsolutePositionCurrent`: An integer property (read-only) storing the current absolute position (in encoder counts).

  - `MotorFocusAbsolutePositionDesired`: An integer property storing an absolute position (in encoder counts) that will be used by subsequent calls to the **MotorFocusMoveToAbsolutePosition↩ Desired** command.
  - `MotorFocusIncrement`: An integer property storing an increment (in encoder counts) that will be used by subsequent calls to **MotorFocusNear** and **MotorFocusFar** commands.
  - `MotorFocusReceiveBuffer`: A string property (read-only) that will contain answers sent by the motor focus controller.
  - `MotorFocusSendBuffer`: A string property storing a command to be sent to the motor focus.
- Methods:
  - `MotorFocusFar`: Calling this function will cause the motor focus to move backward by **Motor↩ FocusIncrement** encoder units.
  - `MotorFocusMoveToAbsolutePositionDesired`: Calling this function will cause the motor focus to move to the position defined by the value of **MotorFocusAbsolutePositionDesired**.
  - `MotorFocusNear`: Calling this function will cause the motor focus to move forward by **Motor↩ FocusIncrement** encoder units.
  - `MotorFocusSend`: Calling this function will send the value of **MotorFocusSendBuffer** to the hardware.

By clicking on the icon with the three dots (e.g. "`MotorFocusNear()`" in Figure 2), this method will be called using the set properties (e.g. "`100`" as "`MotorFocusIncrement`" value as shown in Figure 2).

The following explains the typical adjustment procedure:

1. Move the lens mount to the farthest position by applying `MotorFocusFar` command as often as needed.
2. Switch off closed loop mode by using the command described above.
3. Screw in lens and focus at infinity.
4. Move to the nearest position and check MOD.
5. Move to your working distance.

If your application does not require focusing at infinity you might set the lens mount to the middle position (1500 steps) and focus the lens for your working distance. This gives you equal focusing headroom in both directions. In this case please check that for `MotorFocusNear` the lens does not block the movement until Zero is reached.

19.6.2.3 Programming the motorized lens mount

If you want to program own application using the motorized lens mount, the Impact Acquire API offers the class **MotorFocusControl** which is described in *MotorFocusControl* class reference in the manual for the corresponding programming language at the `manuals section`.

> **Note**
>
> The following code snippets are C# pseudo code.

To use the **MotorFocusControl** class, you have to create an instance of the class:

```
// Initializing the device
DeviceManager deviceManager = new DeviceManager();
Device device = deviceManager.getDeviceByFamily("mvBlueFOX");

MotorFocusControl motorFocusControl = new MotorFocusControl(ref device);
```

Note

You can check, if the camera has a motorized lens mount at all via "`motorFocusControl.motor↩FocusIncrement.valid == false`".

Afterwards, you can

- "`write()`" (integer) / "`writeS()`" (string) and
- "`read()`" (integer) / "`readS()`" (string) the properties or
- "`call()`" the methods like

```
// setting the increment value to 100
motorFocusControl.motorFocusIncrement.write(100);
```

or

```
// moving the motor focus backwards by motorFocusIncrement
int status = motorFocusControl.motorFocusFar.call();
//getting the current position of the motor
int position = motorFocusControl.motorFocusAbsolutePositionCurrent.read();
```

Note

The call functions can be back while the motor is still moving. For this reason, it is necessary to create a method which will check if the motor still moves.

To get information from the motor you can use **"motorFocusSend"**. The following example shows how you can get status and position of the motor. Furthermore, it shows how you can check, if the motor is still running:

```
...
int motorRunning = 0x4;
motorFocusControl.motorFocusSendBuffer.writeS("<10>");
motorFocusControl.motorFocusSend.call();
String s = motorFocusControl.motorFocusReceiveBuffer.readS();
if(s.Length != 29) return false;
// string is in hex
int status = Convert.ToInt32(s.Substring(4, 6), 16);

String positionStr = s.Substring(11, 8);
// the position as integer
int pos = Convert.ToInt32(positionStr, 16);

return (status & motorRunning) == motorRunning;
...
```

Sending the command "`<10>`" will return the status and position with following format:

```
<10 SSSSSS PPPPPPPP EEEEEEEE>
```

- `SSSSSS` is the motor status (6-digit hex format, 24 bit unsigned integer),
- `PPPPPPPP` is the absolute position in encoder counts (8-digit hex format, 32-bit signed integer) and
- `EEEEEEEE` is the position error in encoder counts (8-digit hex format, 32-bit signed integer)

"Motor status values"

Bit	Description	Values
0	Reserved	N/A
1	Motor direction	0 = Reverse 1 = Forward
2	Running	1 = Motor is running
3	Motor interlock	1 = Motor is disconnected
4	Numbered burst mode	1 = Fixed number of bursts in progress
5	Timed run	1 = Timed free run in progress
6	Multiplexed axis	1 = Multiplexed axis (e.g., SQ-2306, 2206)
7	Controller status	1 = Under computer control (analog servo control, if supported, is not available)
8	Reserved	
9	Forward limit	1 = Forward travel limit reached
10	Reverse limit	1 = Reverse travel limit reached
11	Motor burst or amplitude mode	1 = Amplitude mode (always used in closed-loop mode). 1 = Burst mode (200 Hz)
12 - 15	Reserved	N/A
16	Encoder count error	1 = An error was detected in the encoder quadrature signal. Cleared by sending command <07>.
17	Zero reference enabled	1 = Encoder zero reference mark detection is enabled
18	Motor on target	1 = Encoder position error is zero
19	Motor moving toward target	1 = Motor is moving toward a target position; appears after command <08> or move step command <06>. Once the target is reached, bit 19 is set to zero.
20	Maintenance mode enabled	1 = Controller will actively hold the last target position **Note** If bits 20 and 21 are set 1 and bit 18 is set to 0, the controller is in the process of moving back toward the last targeted position.
21	Closed loop enabled	1 = Motion commands use the encoder for feedback
22	Motor accelerating	1 = The motor is accelerating to the desired velocity (set at the start of closed-loop motion) 0 = Required motor speed is reached, motor is decelerating, or motor is stopped
23	Stalled	1 = The position error exceeds the stall detection threshold

Following two examples will show, how the motor status value will look like (using the code snippet before):

19.6.2.3.1 Example 1: Motor doesn't move (return = false)

```
Status:
   Hex: 340080
   Dec: 3408000
Binary:  0011  0100  0000  0000  1000  0000
        23-20 19-16 15-12 11-08 07-04 03-00 => bit 2 is 0

motorRunning would be 4 = 0000 0000 0000 0000 0000 0100
(status & motorRunning) = 0 => return false
```

19.6.2.3.2 Example 2: Motor does move (return = true)

```
Status:
   Hex: 380086
   Dec: 3670150
Binary:  0011  1000  0000  0000  1000  0110
        23-20 19-16 15-12 11-08 07-04 03-00 => bit 2 is 1

(status & motorRunning) = 4 => return true
```

19.6.2.3.3 Example 3: Something a little more complex

This shows a more complex piece of code of how the motor focus can be used.

```
#ifdef _MSC_VER
#   include <windows.h>
#else
#   include <time.h>
#endif
#include <apps/Common/exampleHelper.h>
#include <iostream>
#include <iomanip>
#include <mvIMPACT_CPP/mvIMPACT_acquire.h>
#include <typeinfo>

using namespace std;

// This whole sample shows the low level access to the focus motor. For some
// of the functions presented here, there are also convenience functions in
// 'mvIMPACT::acquire::MotorFocusControl'

//-----------------------------------------------------------------------------
static void millisleep( long millisec )
//-----------------------------------------------------------------------------
{
#ifdef _MSC_VER
    Sleep( millisec );
#else
    timespec requested;
    requested.tv_sec = millisec / 1000;
    requested.tv_nsec = ( millisec % 1000 ) * 1000000L;
    nanosleep( &requested, NULL );
#endif
}

//-----------------------------------------------------------------------------
class MotorControl
//-----------------------------------------------------------------------------
{
private:
    const MotorFocusControl mfc;
    const string version;

    // private, so as to hide the command strings from the outside world
    TDMR_ERROR write( const string& command ) const
    {
        mfc.motorFocusSendBuffer.writeS( command );
        const TDMR_ERROR retval = static_cast<TDMR_ERROR>( mfc.motorFocusSend.call() );
#if _DEBUG
        if( retval != DMR_NO_ERROR )
        {
            cerr << "Call to command '" << command << "' failed with error '" << DMR_ErrorCodeToString( retval )
      << "'" << endl;
        }
#endif
        return retval;
    }

    void waitForCommandReceipt( const string& expectedReply ) const
    {
        const size_t length = expectedReply.length();
        string reply;
        while( ( ( reply = getReply() ).length() < length ) ||
               ( reply.length() == length && reply != expectedReply ) ||
               ( reply.length() > length && reply.substr( 0, length ) != expectedReply ) )
        {
            cout << "Waiting for confirmation of command " << expectedReply.substr( 1, length - 1 ) << " (" <<
      reply << ")" << endl;
        }
    }

    TDMR_ERROR writeConfirmed( const string& command ) const
    {
```

```
        const TDMR_ERROR retval = write( command );
        if( retval == DMR_NO_ERROR )
        {
            waitForCommandReceipt( command.substr( 0, 3 ) );
        }
        return retval;
    }

    string initializeAndGetVersion( void ) const
    {
        write( "<01>" ); // must be the first command written
        millisleep( 1500 ); // initialization takes time
        return getReply();
    }

    unsigned int replyToInt( const string& reply, int bitNr /* < 32, or it won't fit in an unsigned int */ )
      const
    {
        unsigned int nrOfCharactersNeeded = ( bitNr » 2 ) + 1; // four bits in one character
        if( reply.length() < nrOfCharactersNeeded + 4 ) // first four characters contain no status bits
        {
            throw runtime_error( "Reply string too short" );
        }
        unsigned int uistatus;
        istringstream iss( reply.substr( 4, nrOfCharactersNeeded ) ); // skip the first four characters,
      that contain no status bits
        iss » hex » uistatus; // string returned is in hex
        return uistatus;
    }

    bool bitIsSet( unsigned int uistatus, int bitNr ) const
    {
        return ( uistatus & ( 0x1 « bitNr ) ) != 0;
    }
    bool bitIsNotSet( unsigned int uistatus, int bitNr ) const
    {
        return ( uistatus & ( 0x1 « bitNr ) ) == 0;
    }
    bool bitIsSet( const string& reply, int bitNr ) const
    {
        return bitIsSet( replyToInt( reply, bitNr ), bitNr );
    }
    bool bitIsNotSet( const string& reply, int bitNr ) const
    {
        return bitIsNotSet( replyToInt( reply, bitNr ), bitNr );
    }

public:
    explicit MotorControl( Device* device ) : mfc( MotorFocusControl( device ) ), version(
      initializeAndGetVersion() )
    {
        if( bitIsNotSet( getStatus(), 21 ) )
        {
            writeConfirmed( "<20 1>" ); // only in closed-loop drive mode will command moveToPosition work
        }
    }

    ~MotorControl()
    {
        write( "<03>" ); // halt the motor (do not wait for confirmation, as the dtor will also be called
      when the USB plug is pulled, and then one would obviously wait forever)
        write( "<02>" ); // release computer control
    }

    string getVersion( void ) const
    {
        return version.length() > 8 ? version.substr( 6, version.length() - 7 ) : version;
    }

    string getReply( void ) const
    {
        millisleep( 40 );
        return mfc.motorFocusReceiveBuffer.read(); // mfc is private
    }

    string getStatus( void ) const
    {
        write( "<10>" ); // request status
        return getReply();
    }

    // only first 16 status bits of "<10>"
    string getShortStatus( void ) const
    {
        write( "<19>" ); // request motor status
        return getReply();
```

```
    }

    bool isRunning( void ) const
    {
        return bitIsSet( getShortStatus(), 2 );    // would also work with getStatus()
    }

    string getPositionStr( void ) const
    {
        string status = getStatus();
        if( status.empty() )
        {
            throw runtime_error( string( "Could not retrieve status" ) );
        }
        return status.substr( 11, 8 );
    }

    int getPosition( void ) const
    {
        int ipos;
        istringstream iss( getPositionStr() );
        iss » std::hex » ipos; // string returned is in hex
        return ipos;
    }

    void moveToPosition( int pos ) const
    {
        ostringstream oss;
        oss « string( "<08 " );
        oss « std::hex « setw( 8 ) « setfill( '0' ) « pos;
        oss « string( ">" ) « ends;
        writeConfirmed( oss.str() );
    }

    void waitForEndOfMove( void ) const
    {
        while( isRunning() )
        {
            millisleep( 40 );    // wait until the position has been reached
        }
    }

    // required after plugging FOX out and back in again
    void reestablishMotorControl( void ) const
    {
        while( initializeAndGetVersion().empty() ) {};
    }
};

//-----------------------------------------------------------------------------
bool isDeviceSupportedBySample( const Device* const pDev )
//-----------------------------------------------------------------------------
{
    if( pDev->family.read() != "mvBlueFOX" )
    {
        return false;
    }

    return true;
}

//-----------------------------------------------------------------------------
int main( void )
//-----------------------------------------------------------------------------
{
    DeviceManager devMgr;
    Device* pDev = getDeviceFromUserInput( devMgr, isDeviceSupportedBySample );
    if( !pDev )
    {
        cout « "Unable to continue! Press [ENTER] to end the application" « endl;
        cin.get();
        return -1;
    }

    try
    {
        pDev->open();
    }
    catch( const EDeviceManager& e )
    {
        cerr « "Could not open device " « pDev->serial.read() « " (" « e.what() « ", " «
      e.getErrorCodeAsString() « ")" « endl;
        return -1;
    }

    try
    {
```

```
        MotorControl mc( pDev );
        cout « "Found " « mc.getVersion() « endl;
        try
        {
            int i = 4;
            while( i-- )
            {
                mc.moveToPosition( 1500 );
                mc.waitForEndOfMove();
                cout « mc.getPosition() « endl;

                mc.moveToPosition( 2500 );
                mc.waitForEndOfMove();
                cout « mc.getPosition() « endl;
            }
        }
        catch( const exception& e )
        {
            cerr « "Exception of " « typeid( e ).name() « " '" « e.what() « "'" « std::endl;
            return -1;
        }
    }
    catch( const exception& e )
    {
        cerr « "No M3-F found on this device (" « e.what() « ")" « endl;
        return -1;
    }
    pDev->close();

    return 0;
}
```

19.7 Working with LUTs

There are several use cases concerning LUTs (Look-Up-Tables):

- Introducing LUTs

19.7.1 Introducing LUTs

19.7.1.1 Introduction

Look-Up-Tables (LUT) are used to transform input data into a desirable output format. For example, if you want to invert an 8 bit image, a Look-Up-Table will look like the following:

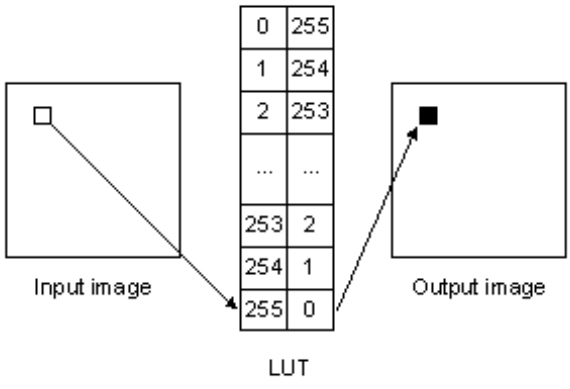


Figure 1: Look-Up-Table which inverts a pixel of an 8 bit mono image

I.e., a pixel which is white in the input image (value 255) will become black (value 0) in the output image.

All Balluff/MATRIX VISION devices use a hardware based LUT which means that

- no host CPU load is needed and
- the LUT operations are independent of the transmission bit depth.

19.7.1.2 Setting the hardware based LUTs via LUT Control

> **Note**
>
> The mvBlueFOX cameras also feature a hardware based LUT. Although, you have to set the LUT via Setting -> Base -> ImageProcessing -> LUTOperations, you can set where the processing takes place. For this reason, there is the parameter *LUTImplementation*. Just select either *"Software"* or *"Hardware"*.

19.7.1.3 Setting the Host based LUTs via LUTOperations

Host based LUTs are also available via *"Setting -> Base -> ImageProcessing -> LUTOperations"*). Here, the changes will affect the 8 bit image data and the processing needs the CPU of the host system.

The mvBlueFOX cameras also feature a hardware based LUT. Although, you have to set the LUT via *"Setting -> Base -> ImageProcessing -> LUTOperations"*, you can set where the processing takes place. For this reason, there is the parameter *LUTImplementation*. Just select either *"Software"* or *"Hardware"*.

Three *"LUTMode"s* are available:

- **"Gamma"**
  You can use "Gamma" to lift darker image areas and to flatten the brighter ones. This compensates the contrast of the object. `The calculation is described here`. It makes sense to set the *"GammaStartThreshold"* higher than 0 to avoid a too extreme lift or noise in the darker areas.
- **"Interpolated"**
  With "Interpolated" you can set the key points of a characteristic line. You can defined the number of key points. The following figure shows the behavior of all 3 LUTInterpolationModes with 3 key points:

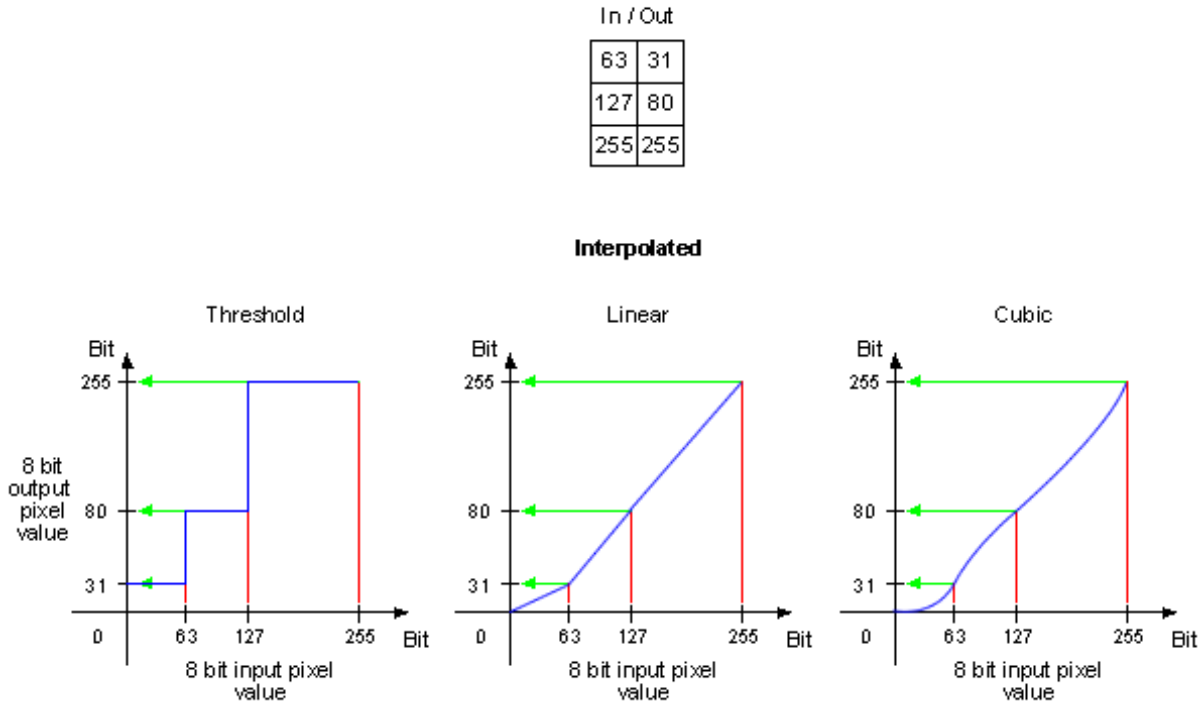


Figure 2: LUTMode "Interpolated" -> LUTInterpolationMode

- **"Direct"**
  With "Direct" you can set the LUT values directly.

19.7.1.3.1 Example 1: Inverting an Image

To get an inverted 8 bit mono image like shown in Figure 1, you can set the LUT using ImpactControlCenter. After starting ImpactControlCenter and using the device,

1. Set *"LUTEnable"* to **"On"** in *"Setting -> Base -> ImageProcessing -> LUTOperations"*.
2. Afterwards, set *"LUTMode"* to **"Direct"**.
3. Right-click on *"LUTs -> LUT-0 -> DirectValues[256]"* and select **"Set Multiple Elements... -> Via A User Defined Value Range"**.
   This is one way to get an inverted result. It is also possible to use the *"LUTMode"* - *"Interpolated"*.
4. Now you can set the range from 0 to 255 and the values from 255 to 0 as shown in Figure 2.

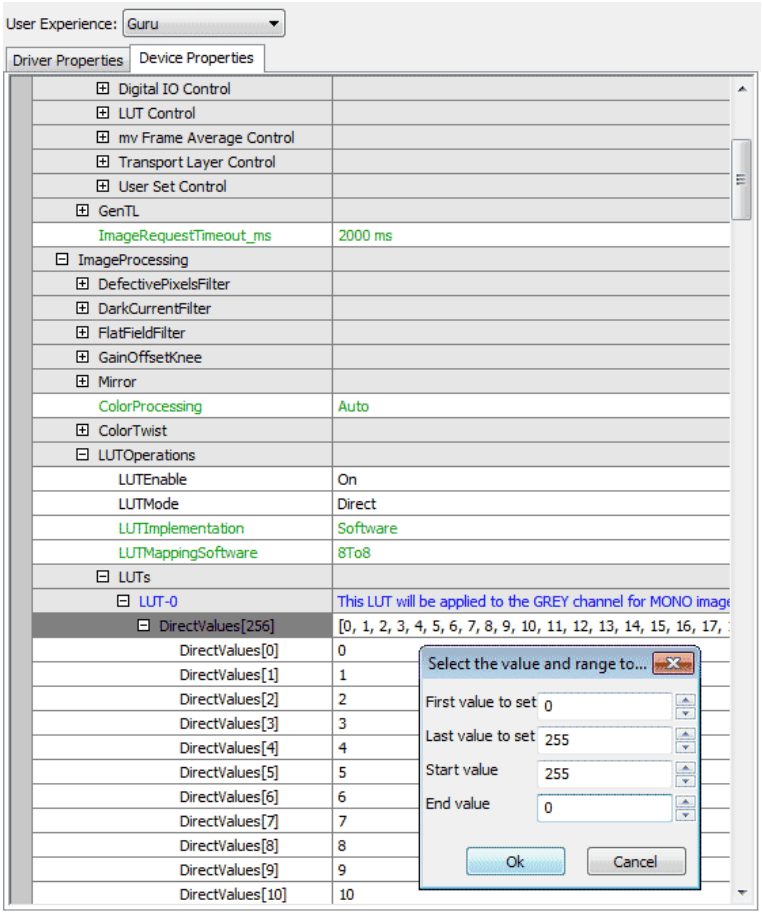


Figure 3: Inverting an image using ImpactControlCenter with LUTMode "Direct"

19.8 Working with triggers

There are several use cases concerning trigger:

- Using external trigger with CMOS sensors

19.8.1 Using external trigger with CMOS sensors

19.8.1.1 Scenario The CMOS sensors used in mvBlueFOX cameras support the following trigger modes:

- `Continuous`
- `OnDemand` (software trigger)
- `OnLowLevel`
- `OnHighLevel`
- `OnHighExpose` (only with mvBlueFOX-[Model]205 (5.0 Mpix [2592 x 1944]))

If an external trigger signal occurs (e.g. high), the sensor will start to expose and readout one image. Now, if the trigger signal is still high, the sensor will start to expose and readout the next image (see Figure 1, upper part). This will lead to an acquisition just like using continuous trigger.

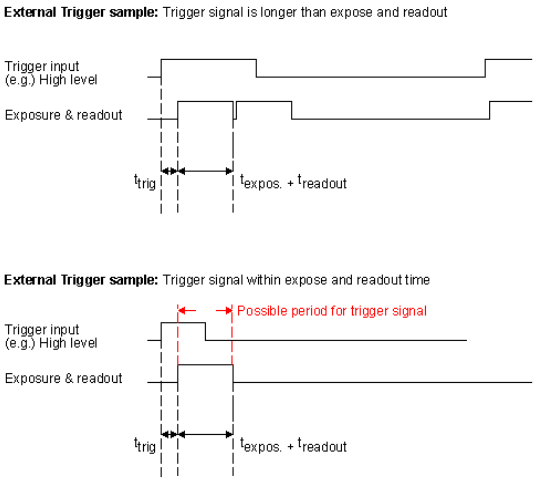


Figure 1: External Trigger with CMOS sensors

- t_{trig} = Time from trigger (internal or external) to integration start.

If you want to avoid this effect, you have to adjust the trigger signal. As you can see in Figure 1 (lower part), the possible period has to be smaller than the time an image will need ($t_{expose} + t_{readout}$).

19.8.1.2 Example

19.8.1.2.1 External synchronized image acquisition (high active)

Note

Using BVS CA-MLC or BVS CA-IGC, you have to select *DigIn0* as the trigger source, because the camera has only one opto-coupled input. Only the TTL model of the BVS CA-MLC has two I/O's.

- **Trigger** modes
  - `OnHighLevel`:
    The high level of the trigger has to be shorter than the frame time. In this case, the sensor will make one image exactly. If the high time is longer, there will be images with the possible frequency of the sensor as long as the high level takes. The first image will start with the low-high edge of the signal. The integration time of the exposure register will be used.
  - `OnLowLevel`:
    The first image will start with the high-low edge of the signal.
  - `OnHighExpose`
    This mode is like `OnHighLevel`, however, the exposure time is used like the high time of the signal.

See also

Block diagrams with example circuits of the opto-isolated digital inputs and outputs can be found in Dimensions and connectors.

19.9 Working with 3rd party tools

- Using VLC Media Player
- Using USB2 Cameras In A Docker Container

19.9.1 Using VLC Media Player

With the DirectShow Interface Impact Acquire compliant devices become video capture devices for the VLC Media Player.

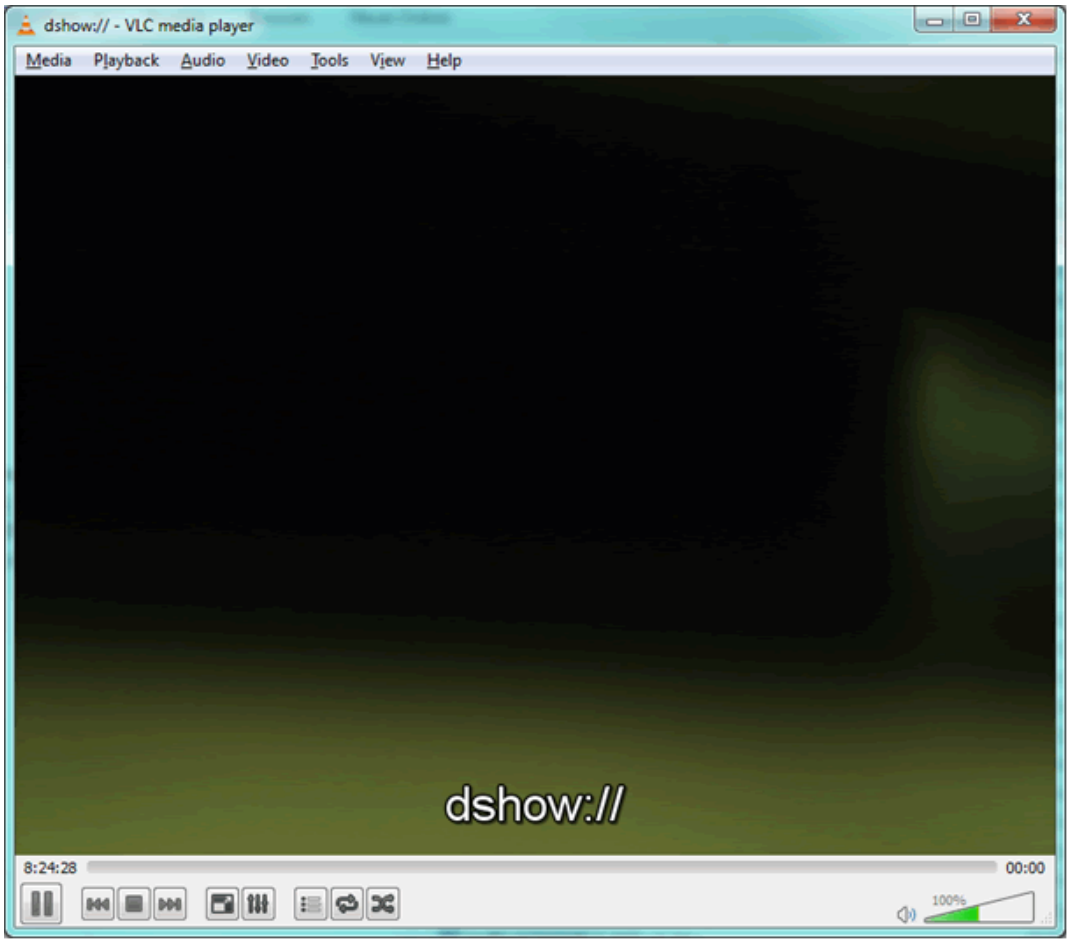


Figure 1: VLC Media Player with a connected device via DirectShow

19.9.1.1 System Requirements

It is necessary that the following drivers and programs are installed on the host device (laptop or PC):

- Windows 7 or higher, 32-bit or 64-bit
- up-do-date VLC Media Player, 32-bit or 64-bit (here: version 2.0.6)
- up-do-date Impact Acquire driver, 32-bit or 64-bit (here: version 2.5.6)

Note

Using **Windows 10** or **Windows 7**: VLC Media Player with versions 2.2.0 have been tested successfully with older versions of Impact Acquire. Since version 3.0.0 of VLC Impact Acquire will be needed to work with devices through the DirectShow interface!

19.9.1.2 Installing VLC Media Player

1. Download a suitable version of the VLC Media Player from the VLC Media Player website mentioned below.
2. Run the setup.
3. Follow the installation process and use the default settings.

A restart of the system is not required.

See also

http://www.videolan.org/

19.9.1.3 Setting Up A Device For DirectShow For using devices, DirectShow needs a public driver that must be registered with the operating system. As described in Registering Devices , this can be done with the help of DeviceConfigure .

> **Note**
>
> If running a 64-bit DirectShow application, make sure to use a 64-bit version of **DirectShow_Acquire**, which can only be installed with a 64-bit version of DeviceConfigure . (And respectively for a 32-bit version)

1. Connect the device to the host.
2. Power the camera using a power supply at the power connector.
3. Wait until the status LED turns blue.
4. Open the tool DeviceConfigure ,
5. set a friendly name ,
6. and register the device for DirectShow .

> **Note**
>
> In some cases it could be necessary to repeat step 5.

19.9.1.4 Working With VLC Media Player

1. Start VLC Media Player.
2. Click on **"Media ->** **Open Capture Device..."** .

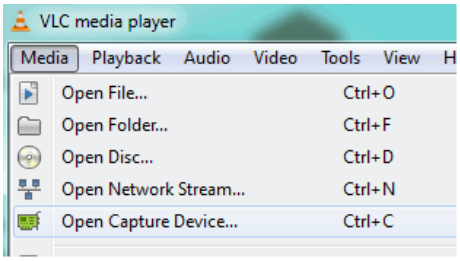


Figure 2: Open Capture Device...

3. Select the tab **"Device Selection"** .
4. In the section **"Video device name"** , select the friendly name of the device:

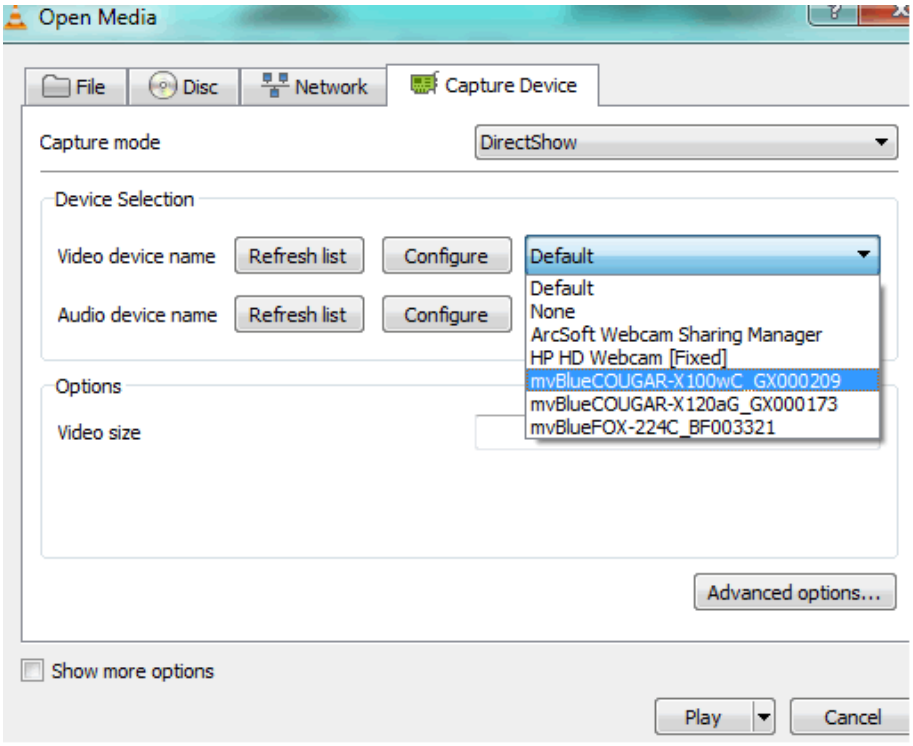


Figure 3: Video device name

5. Finally, click on **"Play"** .
   After a short delay you will see the live image of the camera.

19.9.1.5 Changing Camera Properties When selecting a **DirectShow_Acquire** camera as a DirectShow capture device (in the "Open Capture Device" dialog), there are two posibilities to display the camera property dialog:

A. Click the checkbox "Show more options" and append " :dshow-config" to the options in the bottom edit box:

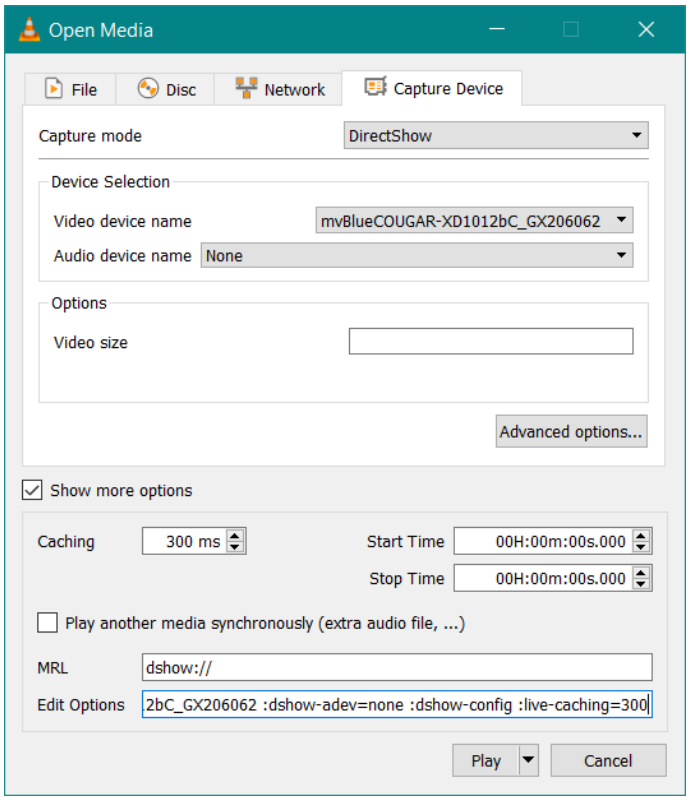


Figure 1: Entering additional options for displaying the DirectShow_Acquire property dialog

B. Alternatively, press the button "Advanced options" and, in the window that is going to be displayed, click the checkbox "Device properties" followed by pressing "OK". This fills in all the relevant settings of the dialog into the edit box in the "Open Media : Capture Device" dialog from solution A above.

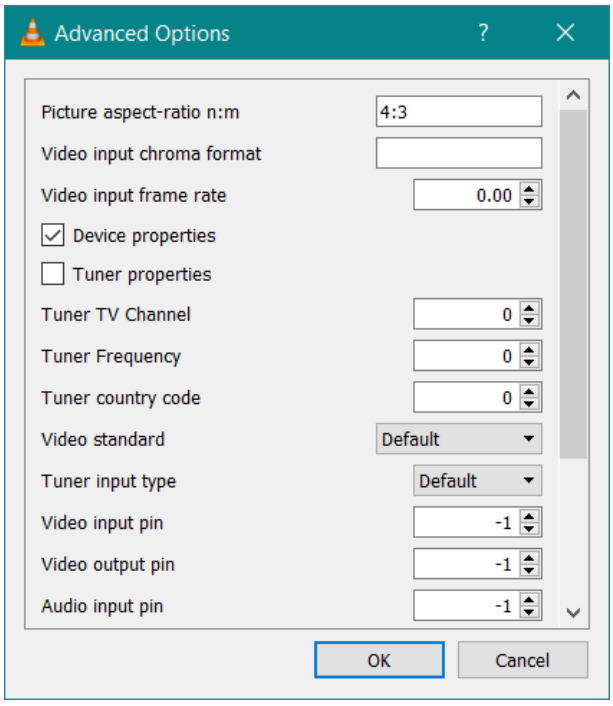


Figure 2: Advanced options dialog invokrd by the Media/Open Capture dialog

After selecting the activity to invoke (Play/Stream/Enqueue/Convert), the **DirectShow_Acquire** property dialog will open.

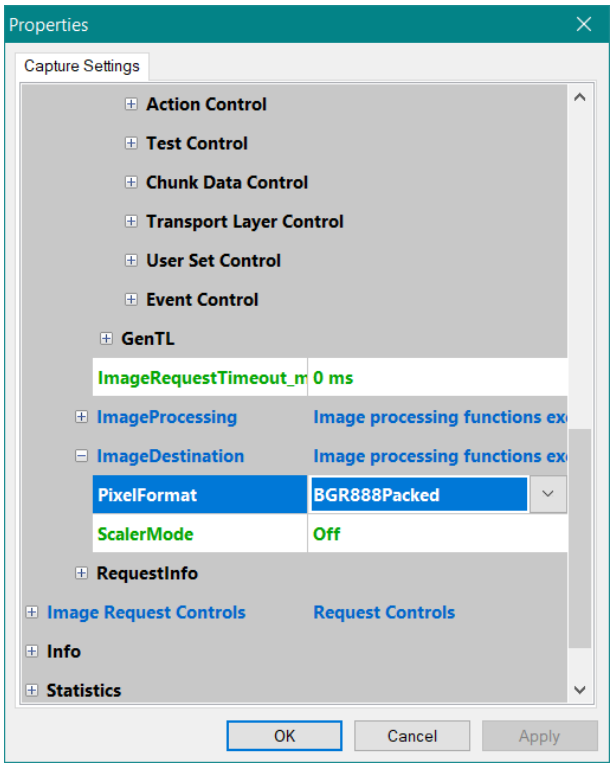


Figure 3: The property dialog for the DirectShow_Acquire capture device

Note

Most properties in this dialog control the camera directly, so they stay the same as long as the camera is not reset. Properties that control streaming (like the ImageDestination::PixelFormat) are set by the VLC Media Player each time the capture device is started.

After pressing "OK", the selected activity will start.

See also

If there are wrong colors in the resulting images, refer to Wrong Colors in the VLC Media Player .

Possibly there will be an error box like the one shown below, but you can safely ignore it by pressing "No" or "Cancel".

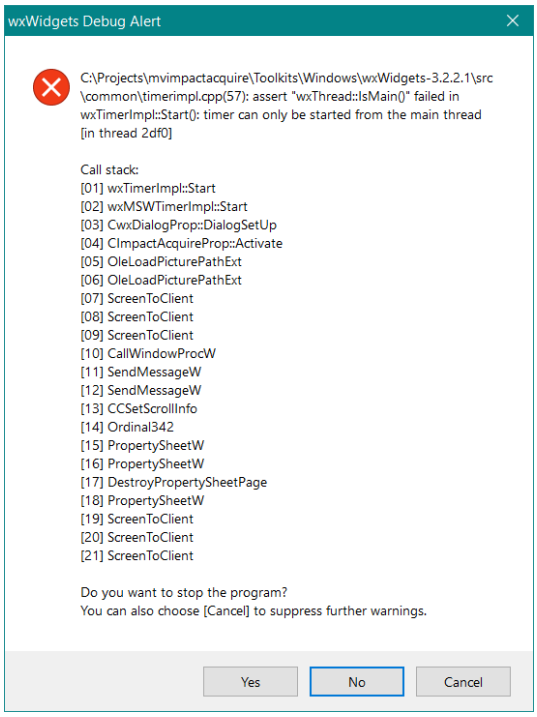


Figure 4: Error message during startup of capture property dialog

19.9.2 Using USB2 Cameras In A Docker Container

When developing machine vision applications using Docker containers, it might be required to access the cameras inside the container. With the Impact Acquire driver stack this can be achieved fairly easily and this chapter will demonstrate how to build a basic Docker container where the cameras can be used.

19.9.2.1 Host Preparation

19.9.2.1.1 Linux

19.9.2.1.2 Windows

19.9.2.1.3 Host system requirements

- Windows 11 64-bit: Home or Pro version 21H2 or higher, or Enterprise or Education version 21H2 or higher (Build 22000 or later)
- Windows 10 64-bit: Home or Pro 21H1 (build 19043) or higher, or Enterprise or Education 20H2 (build 19042) or higher
- WSL2 backend (For installation please follow: `Docker Window Install`)
- Impact Acquire driver package >= 2.48.0 recommended

19.9.2.1.4 Attach the camera to the WSL2 Linux distro via USB/IP USB devices physically connected to the host system are not automatically accessible in the WSL2 Linux distro. They need to be first attached from the Windows host to the default Linux distro via USB/IP. Please follow Connect USB devices WSL2 for implementation guidance.

19.9.2.1.5 Start udev manually udev is needed to identify attached USB devices and to access USB3 Vision™ devices as non-root users with the help of the udev-rules shipped by the Impact Acquire driver package. However, systemd, which starts udev automatically, is by default not supported in WSL2 distros. Besides, udev doesn't support containers. Since WSL2 distros themselves are technically containers, they are not supported by udev. In order for udev to work in WSL2 distros, the following lines need to be commented out in `/etc/init.d/udev` before manually starting udev, as shown below:

```
#if [ ! -w /sys ]; then
#    log_warning_msg "udev does not support containers, not started"
#    exit 0
#fi
```

Then start udev in the WSL2 default Linux distro:

```
$ sudo /etc/init.d/udev start
```

19.9.2.2 Building A Docker Image The following demo Dockerfile builds a basic Docker image based on a slim version of Debian, where the Impact Acquire driver package for the cameras and its sample programs are installed. This Dockerfile can be used in many ways:

- Use it directly to test your device in a Docker container.
- Use it as a base image for your device applications.
- Use it as an inspiration for building your own Dockerfile.

Before building the Dockerfile, please download the required Impact Acquire driver installation files from Balluff website (https://www.balluff.com/en-de/downloads/software) (user login is required):

- The installation script: *install_mvBlueFOX.sh*
- The installation package: *mvBlueFOX-x86_64_ABI2-*.tgz* (* should be replaced by the version number)

Create a directory called *Impact* Acquire (as used in this demo Dockerfile) and move both installation files into this directory. In this example, both files are downloaded into the *Downloads* directory and the *Impact* Acquire directory is created inside the *Downloads:*

- `$ cd ~/Downloads`
- `$ mkdir Impact_Acquire`
- `$ mv install_mvBlueFOX.sh mvBlueFOX-x86_64_ABI2-*.tgz Impact_Acquire/`

Make the installation script *install_mvBlueFOX.sh* executable:

- `$ cd Impact_Acquire`
- `$ chmod a+x install_mvBlueFOX.sh`

Navigate back into the directory where *product_name* resides (e.g. *Downloads*) and create your Dockerfile:

- `$ cd ~/Downloads`
- `$ touch Dockerfile`

Create the content of your Dockerfile. Our demo Dockerfile looks as follows:

```
# start with slim version of actual Debian
FROM debian:9-slim

ENV LC_ALL C
ENV DEBIAN_FRONTEND noninteractive

# entrypoint of Docker
CMD ["/bin/bash"]

# set environment variables
ENV TERM linux
ENV MVIMPACT_ACQUIRE_DIR /opt/Impact_Acquire
ENV MVIMPACT_ACQUIRE_DATA_DIR /opt/Impact_Acquire/data
ENV container docker

# update packets and install minimal requirements
# after installation it will clean apt packet cache
RUN apt-get update && apt-get -y install build-essential && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

# move the directory Impact_Acquire with *.tgz and *.sh files to the container
COPY Impact_Acquire /var/lib/Impact_Acquire

# execute the setup script in an unattended mode
RUN cd /var/lib/Impact_Acquire && \
    ./install_mvBlueFOX.sh -u && \
    rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

At last, build a Docker image using this Dockerfile:

```
$ sudo docker build -t [image_name] .
```

Note

Please make sure to call *docker build* from within the directory where the Dockerfile resides. An Internet access is required for the *docker build*.

If built successfully, the newly built [image_name] will be listed when calling:

```
$ sudo docker images
```

19.9.2.3 Starting The Docker Container Since the Docker container is isolated from the host system, it needs to be started with volume mount of */dev* and certain cgroup permissions for it to access the cameras. In order to avoid running the container in privileged mode, which is not secure, it can be started like this:

```
$ sudo docker run -ti -v /dev:/dev --device-cgroup-rule 'a 189:* rwm' [image_name] /bin/bash
```

Where:

- **-v /dev:/dev**: use volume mount to map the host */dev* directory to the container, so the container will be able to always detect devices also when they get unplugged and re-plugged at any time.
- **–device-cgroup-rule 'a 189:* rwm'**: with the *--device-cgroup-rule* flag, specific permission rules can be added to a device list that is allowed by the container's cgroup. Here in this example, *189* is the major number of the USB bus, * means all minor numbers, and *rwm* are respectively read, write, mknod accesses. By doing so, all USB devices will get read, write, mknod access. The camera can thus be enumerated successfully.

19.9.2.4 Validation After starting the container, the correct operation of cameras can be validated by running one of the sample programs provided by the Impact Acquire (e.g. *SingleCapture*):

- `$ cd /opt/Impact_Acquire/apps/SingleCapture/x86_64`
- `$ ./SingleCapture` If the attached camera appears in the device list of the program's output, access to it in the container by using the Impact Acquire has been established. Now the camera can be used inside the Docker container for your machine vision applications.

19.10 Working with the Hardware Real-Time Controller (HRTC)

> **Note**
>
> Please have a look at the Hardware Real-Time Controller (HRTC) chapter for basic information.

There are several use cases concerning the Hardware Real-Time Controller (HRTC):

- **"Using single camera"**:
  - Achieve a defined image frequency (HRTC)
  - Delay the external trigger signal (HRTC)
  - Creating double acquisitions (HRTC)
  - Take two images after one external trigger (HRTC)
  - Take two images with different expose times after an external trigger (HRTC)
  - Edge controlled triggering (HRTC)
- **"Using multiple cameras"**:
  - Delay the expose start of the following camera (HRTC)

19.10.1 Achieve a defined image frequency (HRTC)

> **Note**
>
> Please have a look at the Hardware Real-Time Controller (HRTC) chapter for basic information.

With the use of the HRTC, any feasible frequency with the accuracy of micro seconds(us) is possible. The program to achieve this roughly must look like this (with the trigger mode set to ctmOnRisingEdge):

```
0. WaitClocks( <frame time in us> - <trigger pulse width in us>) )
1. TriggerSet 1
2. WaitClocks( <trigger pulse width in us> )
3. TriggerReset
4. Jump 0
```

So to get e.g. exactly 10 images per second from the camera the program would somehow look like this(of course the expose time then must be smaller or equal then the frame time in normal shutter mode):

```
0. WaitClocks 99000
1. TriggerSet 1
2. WaitClocks 1000
3. TriggerReset
4. Jump 0
```

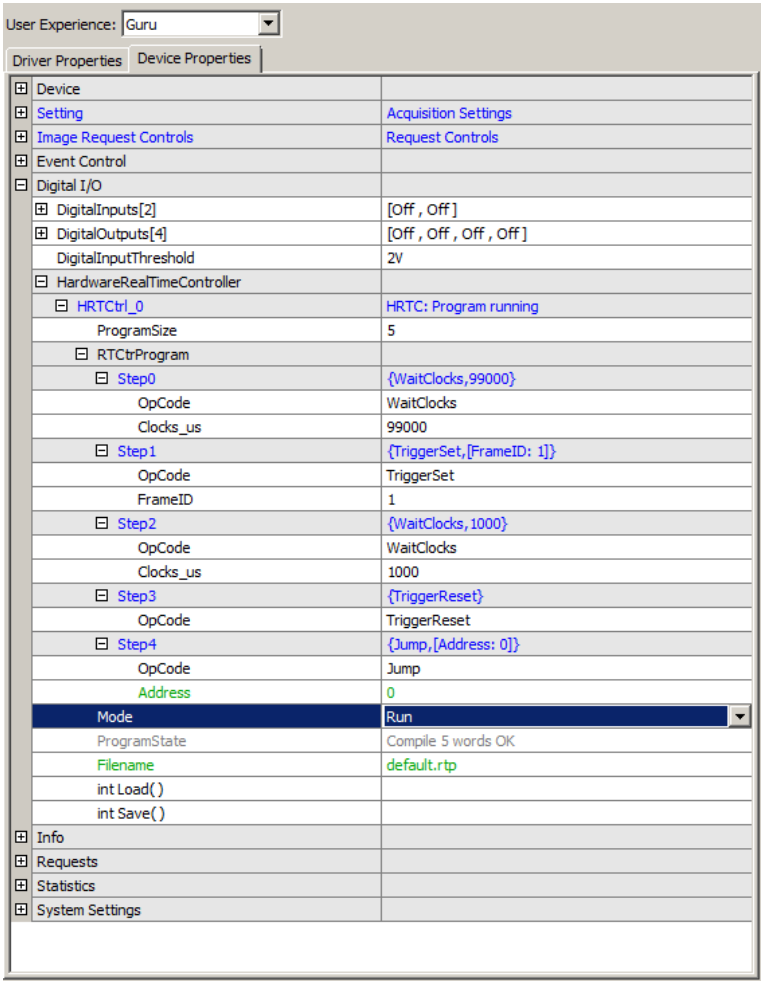


Figure 1: ImpactControlCenter - Entering the sample "Achieve a defined image frequency"

Note

Please note the max. frame rate of the corresponding sensor!

To see a code sample (in C++) how this can be implemented in an application see the description of the class `mvIMPACT::acquire::RTCtrProgram` (C++ developers)

19.10.2 Delay the external trigger signal (HRTC)

Note

Please have a look at the Hardware Real-Time Controller (HRTC) chapter for basic information.

```
0. WaitDigin DigIn0->On
1. WaitClocks <delay time>
2. TriggerSet 0
3. WaitClocks <trigger pulse width>
4. TriggerReset
5. Jump 0

<trigger pulse width> should not less than 100us.
```

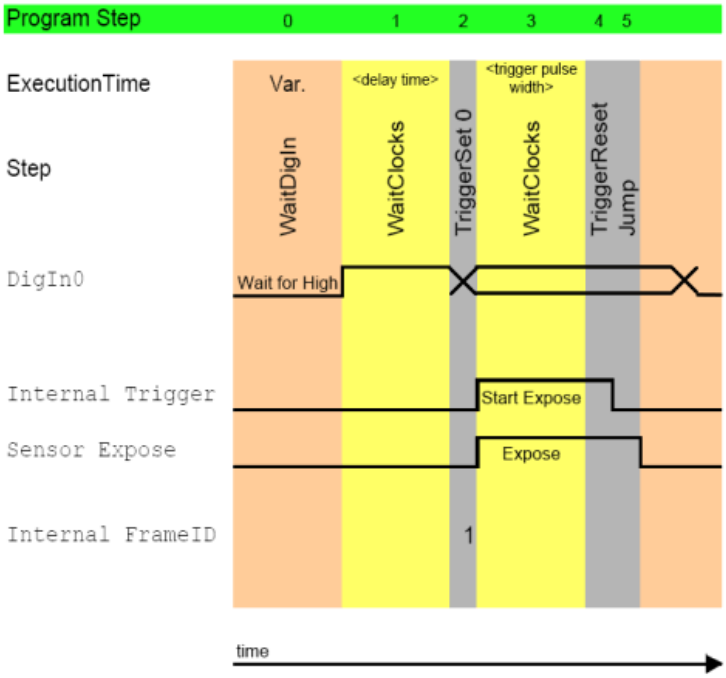


Figure 1: Delay the external trigger signal

As soon as digital input 0 changes from high to low (0), the HRTC waits the < delay time > (1) and starts the image expose. The expose time is used from the expose setting of the camera. Step (5) jumps back to the beginning to be able to wait for the next incoming signal.

Note

WaitDigIn waits for a state.

Between TriggerSet and TriggerReset has to be a waiting period.

If you are waiting for an external edge in a HRTC sequence like

```
WaitDigIn[On,Ignore]
WaitDigIn[Off,Ignore]
```

the minimum pulse width which can be detected by **HRTC** has to be at least 5 us.

19.10.3 Creating double acquisitions (HRTC)

Note

Please have a look at the Hardware Real-Time Controller (HRTC) chapter for basic information.

If you need a double acquisition, i.e. take two images in a very short time interval, you can achieve this by using the HRTC.

With the following HRTC code, you will

- take an image using **TriggerSet** and after **TriggerReset** you have to
- set the camera to **ExposeSet** immediately.
- Now, you have to wait until the first image was read-out and then
- set the second **TriggerSet**.

The **ExposureTime** was set to 200 us.

```
0    WaitDigin  DigitalInputs[0] - On
1    TriggerSet  1
2    WaitClocks 200
3    TriggerReset
4    WaitClocks 5
5    ExposeSet
6    WaitClocks 60000
7    TriggerSet 2
8    WaitClocks 100
9    TriggerReset
10   ExposeReset
11   WaitClocks 60000
12   Jump 0
```

19.10.4 Take two images after one external trigger (HRTC)

> **Note**
>
> Please have a look at the Hardware Real-Time Controller (HRTC) chapter for basic information.

```
0. WaitDigin DigIn0->Off
1. TriggerSet 1
2. WaitClocks <trigger pulse width>
3. TriggerReset
4. WaitClocks <time between 2 acquisitions - 10us> (= WC1)
5. TriggerSet 2
6. WaitClocks <trigger pulse width>
7. TriggerReset
8. Jump 0

<trigger pulse width> should not less than 100us.
```

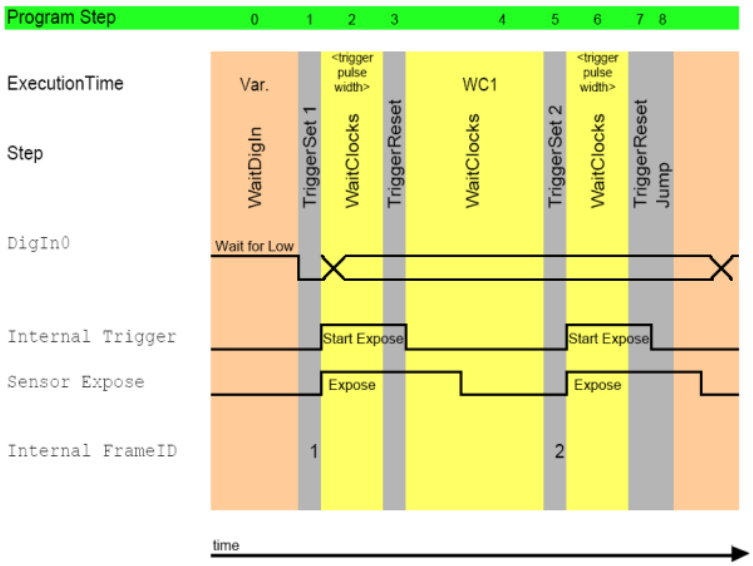


Figure 1: Take two images after one external trigger

This program generates two internal trigger signals after the digital input 0 is going to low. The time between those internal trigger signals is defined by step (4). Each image is getting a different frame ID. The first one has the number 1, defined in the command (1) and the second image will have the number 2. The application can ask for the frame ID of each image, so well known which image is the first and the second one.

19.10.5 Take two images with different expose times after an external trigger (HRTC)

Note

Please have a look at the Hardware Real-Time Controller (HRTC) chapter for basic information.

The following code shows the solution in combination with a **CCD** model of the camera. With CCD models you have to set the exposure time using the trigger width.

```
0.  WaitDigin DigIn0->Off
1.  ExposeSet
2.  WaitClocks <expose time image1 - 10us> (= WC1)
3.  TriggerSet 1
4.  WaitClocks <trigger pulse width>
5.  TriggerReset
6.  ExposeReset
7.  WaitClocks <time between 2 acquisitions - expose time image1 - 10us> (= WC2)
8.  ExposeSet
9.  WaitClocks <expose time image2 - 10us> (= WC3)
10. TriggerSet 2
11. WaitClocks <trigger pulse width>
12. TriggerReset
13. ExposeReset
14. Jump 0

<trigger pulse width> should not less than 100us.
```

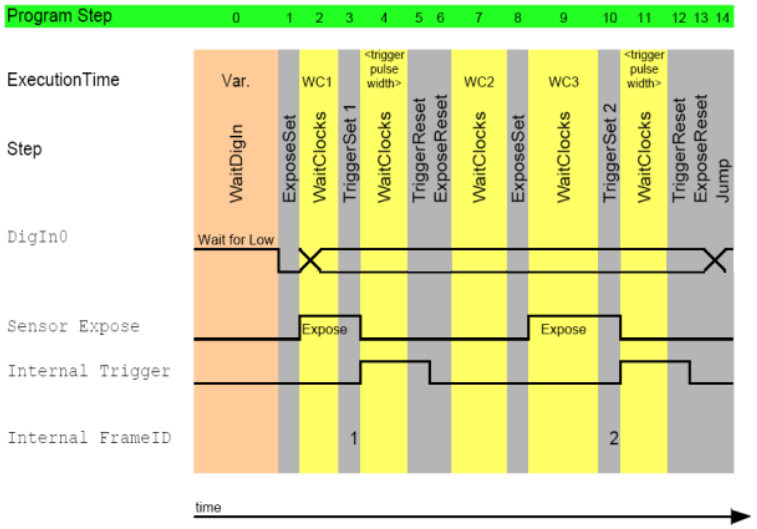


Figure 1: Take two images with different expose times after an external trigger

Note

Due to the internal loop to wait for a trigger signal, the *WaitClocks* call between *"TriggerSet 1"* and *"TriggerReset"* constitute *100* . For this reason, the trigger signal cannot be missed.

Before the *ExposeReset*, you have to call the *TriggerReset* otherwise the normal flow will continue and the image data will be lost!

The sensor expose time after the TriggerSet is *0* .

Using a **CMOS** model (e.g. the mvBlueFOX-MLC205), a sample with four consecutive exposure times (10ms / 20ms / 40ms / 80ms) triggered just by one hardware input signal would look like this:

```
0.  WaitDigin DigIn0->On
1.  TriggerSet
2.  WaitClocks 10000 (= 10 ms)
3.  TriggerReset
4.  WaitClocks 1000000 (= 1 s)
5.  TriggerSet
6.  WaitClocks 20000 (= 20 ms)
7.  TriggerReset
8.  WaitClocks 1000000 (= 1 s)
9.  TriggerSet
10. WaitClocks 40000 (= 40 ms)
11. TriggerReset
12. WaitClocks 1000000 (= 1 s)
13. TriggerSet
14. WaitClocks 80000 (= 40 ms)
15. TriggerReset
16. WaitClocks 1000000 (= 1 s)
17. Jump 0
```

See also

This second sample is also available as an `rtp` file: `MLC205_four_images_diff_exp.rtp`.

19.10.6 Edge controlled triggering (HRTC)

> **Note**
>
> Please have a look at the Hardware Real-Time Controller (HRTC) chapter for basic information.

To achieve an edged controlled triggering, you can use HRTC. Please follow these steps:

1. First of all, you have to set the **TriggerMode** to **OnHighLevel** .
2. Then, set the **TriggerSource** to **RTCtrl** .

OffsetAutoBlackLevel	2
OffsetAutoBlackSpeed	Medium
TriggerMode	OnHighLevel
TriggerSource	RTCtrl
PixelClock_KHz	27000
Expose_us	2000
BinningMode	Off
FrameDelay_us	0
LineDelay_clk	0
FlashMode	Off
FlashType	Standard
TestMode	Off
ShutterMode	FrameShutter
ImageRequestTimeout_ms	0 ms
FlashToExposeDelay_us	0
AdvancedOptions	Off

Figure 1: ImpactControlCenter - TriggerMode and TriggerSource

Afterwards you have to configure the HRTC program:

1. The HRTC program waits for a rising edge at the digital input 0 (step 1).
2. If there is a rising edge, the trigger will be set (step 2).
3. After a short wait time (step 3),
4. the trigger will be reset (step 4).
5. Now, the HRTC program waits for a falling edge at the digital input 0 (step 5).
6. If there is a falling edge, the trigger will jump to step 0 (step 6).

Note

The waiting time at step 0 is necessary to debounce the signal level at the input (the duration should be shorter than the frame time).

⊟ HardwareRealTimeController	
⊟ HRTCtrl_0	HRTC: Program running
ProgramSize	7
⊟ RTCtrProgram	
⊟ Step0	{WaitClocks,1000}
OpCode	WaitClocks
Clocks_us	1000
⊟ Step1	{WaitDigin,[On, Ignore]}
OpCode	WaitDigin
⊟ DigitalInputs[2]	[On, Ignore]
DigitalInputs[0]	On
DigitalInputs[1]	Ignore
⊟ Step2	{TriggerSet,[FrameID: 0]}
OpCode	TriggerSet
FrameID	0
⊟ Step3	{WaitClocks,100}
OpCode	WaitClocks
Clocks_us	100
⊟ Step4	{TriggerReset}
OpCode	TriggerReset
⊟ Step5	{WaitDigin,[Off, Ignore]}
OpCode	WaitDigin
⊟ DigitalInputs[2]	[Off, Ignore]
DigitalInputs[0]	Off
DigitalInputs[1]	Ignore
⊟ Step6	{Jump,[Address: 0]}
OpCode	Jump
Address	0
Mode	RunRestart
ProgramState	Compile 8 words OK

Figure 2: ImpactControlCenter - Edge controller triggering using HRTC

How you can work with capture settings in ImpactControlCenter is described in **"Setting Up Multiple Display Support, Working With Several Capture Settings In Parallel"** in the **"Impact Acquire SDK GUI Applications"** manual.

To see a code sample (in C++) how this can be implemented in an application see the description of the class `mvIMPACT::acquire::RTCtrProgram` (C++ developers)

19.10.7 Delay the expose start of the following camera (HRTC)

Note

Please have a look at the Hardware Real-Time Controller (HRTC) chapter for basic information.

The use case Synchronize the cameras to expose at the same time shows how you have to connect the cameras.

If a defined delay should be necessary between the cameras, the HRTC can do the synchronization work.

In this case, one camera must be the master. The external trigger signal that will start the acquisition must be connected to one of the cameras digital inputs. One of the digital outputs then will be connected to the digital input of the next camera. So camera one uses its digital output to trigger camera two. How to connect the cameras to one another can also be seen in the following image:

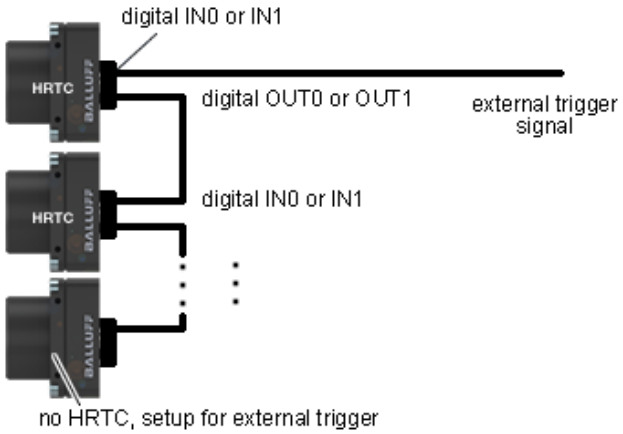


Figure 1: Connection diagram for a defined delay from the exposure start of one camera relative to another

Assuming that the external trigger is connected to digital input 0 of camera one and digital output 0 is connected to digital input 0 of camera two. Each additional camera will then be connected to it predecessor like camera 2 is connected to camera 1. The HRTC of camera one then has to be programmed somehow like this:

```
0. WaitDigin DigIn0->On
1. TriggerSet 0
2. WaitClocks <trigger pulse width>
3. TriggerReset
4. WaitClocks <delay time>
5. SetDigout DigOut0->On
6. WaitClocks 100us
7. SetDigout DigOut0->Off
8. Jump 0

<trigger pulse width> should not less than 100us.
```

When the cameras are set up to start the exposure on the rising edge of the signal <*delay* time> of course is the desired delay time minus <*trigger* pulse width>.

If more than two cameras shall be connected like this, every camera except the last one must run a program like the one discussed above. The delay times of course can vary.

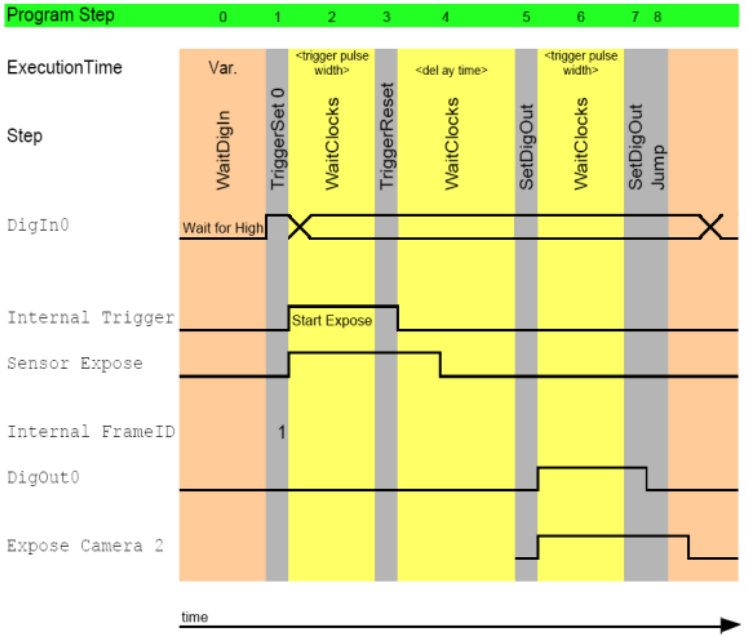


Figure 2: Delay the expose start of the following camera

20 Appendix A. Specific Camera / Sensor Data

20.1 A.1 CCD

20.1.1 mvBlueFOX-[Model]220 (0.3 Mpix [640 x 480])

20.1.1.1 Introduction

The CCD sensor is a highly programmable imaging module which will, for example, enable the following type of applications

Industrial applications:

- triggered image acquisition with precise control of image exposure start by hardware trigger input.
- image acquisition of fast moving objects due to:

  - frame exposure, integrating all pixels at a time in contrast to CMOS imager which typically integrate line-by-line.
  - short shutter time, to get sharp images.
  - flash control output to have enough light for short time.

Scientific applications:

- long time exposure for low light conditions.
- optimizing image quality using the variable shutter control.

20.1.1.2 Details of operation

The process of getting an image from the CCD sensor can be separated into three different phases.

20.1.1.2.1 Trigger

When coming out of reset or ready with the last readout the CCD controller is waiting for a Trigger signal.

The following trigger modes are available:

Mode	Description
Continuous	Free running, no external trigger signal needed.
OnDemand	Image acquisition triggered by command (software trigger).
OnLowLevel	Start an exposure of a frame as long as the trigger input is below the trigger threshold.
OnHighLevel	Start an exposure of a frame as long as the trigger input is above the trigger threshold.
OnFallingEdge	Each falling edge of trigger signal acquires one image.
OnRisingEdge	Each rising edge of trigger signal acquires one image.
OnHighExpose	Each rising edge of trigger signal acquires one image, exposure time corresponds to pulse width.

See also

For detailed description about the trigger modes (https://www.balluff.com/en-de/documentation-for-you [Impact Acquire API])

- **C: TCameraTriggerMode**
- **C++**: **mvIMPACT::acquire::TCameraTriggerMode**

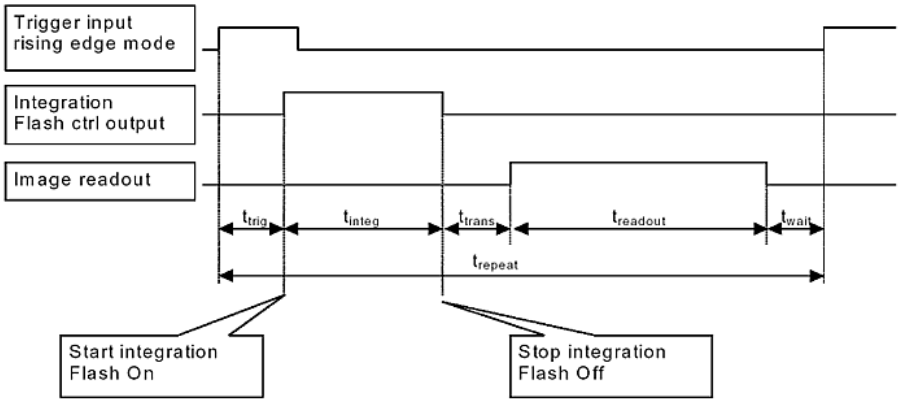
20.1.1.2.2 Exposure aka Integration

After an active trigger, the exposure phase starts with a maximum jitter of t_{trig}. If flash illumination is enabled in software the flash output will be activated exactly while the sensor chip is integrating light. Exposure time is adjustable by software in increments of $t_{readline}$.

20.1.1.2.3 Readout

When exposure is finished, the image is transferred to hidden storage cells on the CCD. Image data is then shifted out line-by-line and transferred to memory. Shifting out non active lines takes tvshift, while shifting out active lines will consume $t_{readline}$. The number of active pixels per line will not have any impact on readout speed.

20.1.1.3 CCD Timing



Name	Description	Pixel clock	
		12 MHz	24 MHz
t_{trig}	Time from trigger (internal or external) to exposure start	10us	
t_{trans}	Image transfer time (move image to readout cells in CCD)	64us	32us
$t_{readline}$	time needed to readout a line	64us	32us
t_{vshift}	time needed to shift unused lines away	3.15us	1.6us
t_{wait}	minimal time to next trigger	64us	32us
$t_{exposure}$	Exposure time	2uss - 128s	
$t_{readout}$	Image readout time (move image from readout cells to memory	$t_{readout}$ = (ActiveLines $*$ $t_{readline}$) + (510 - ActiveLines) $*$ t_{vshift}	

20.1.1.3.1 Timings

Note

In partial scan mode (readout window ysize $<$ 480 lines).

To calculate the maximum frames per second (FPS_{max}) you will need following formula (ExposeMode: Standard):

$$FPS\_max = \frac{1}{t\_trig + t\_readout + t\_exposure + t\_trans + t\_wait}$$

(ExposeMode: Overlapped):

$t\_trig + t\_readout + t\_trans + t\_wait < t\_exposure:$ $FPS\_max = \frac{1}{t\_integ}$

$t\_trig + t\_readout + t\_trans + t\_wait > t\_exposure:$ $FPS\_max = \frac{1}{t\_trig + t\_readout + t\_trans + t\_wait}$

Example: Frame rate as function of lines & exposure time

Now, when we insert the values using exposure time of, for example, 65 us, 100 lines and 12MHz pixel clock (ExposeMode: Standard):

```
FPS_max =                                                1
          ------------------------------------------------------------------------------
          10 us + ((100 * 64 us) + ((510 - 100) * 4.85 us) + 3.15 us) + 65 us + 64 us + 64 us
        = 0.000126670466780670086 8   1 / us
        = 126.7
```

Note

The calculator returns the max. frame rate supported by the sensor. Please keep in mind that it will depend on the interface and the used image format if this frame rate can be transferred.

See also

To find out how to achieve any defined freq. below or equal to the achievable max. freq., please have a look at Achieve a defined image frequency (HRTC).

20.1.1.4 Reprogramming CCD Timing

Reprogramming the CCD Controller will happen when the following changes occur

- Changing the exposure time
- Changing the capture window
- Changing Trigger Modes

Reprogram time consists of two phases

1. Time needed to send data to the CCD controller depending on what is changed
   exposure : abt 2..3ms
   window: abt 4..6ms
   trigger mode: from 5..90ms,
   varies with oldmode/newmode combination
2. Time to initialize (erase) the CCD chip after reprogramming this is fixed, abt 4.5 ms

So for example when reprogramming the capture window you will need (average values)

$t_{regprog}$ = change_window + init_ccd

$t_{regprog}$ = 5ms + 4.5ms

$t_{regprog}$ = 9.5ms

20.1.1.5 CCD Sensor Data

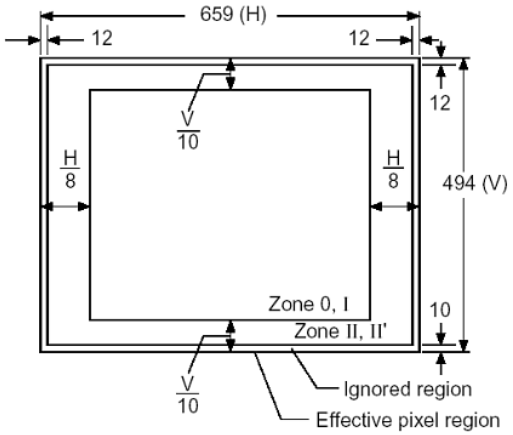
Device Structure

- Interline CCD image sensor
- Image size: Diagonal 4.5mm (Type 1/4)
- Number of effective pixels: 659 (H) x 494 (V) approx. 330K pixels
- Total number of pixels: 692 (H) x 504 (V) approx. 350K pixels
- Chip size: 4.60mm (H) x 3.97mm (V)
- Unit cell size: 5.6um (H) x 5.6um (V)
- Optical black:
  - Horizontal (H) direction: Front 2 pixels, rear 31 pixels
  - Vertical (V) direction: Front 8 pixels, rear 2 pixels
- Number of dummy bits: Horizontal 16 Vertical 5
- Substrate material: Silicon

20.1.1.5.1 Characteristics

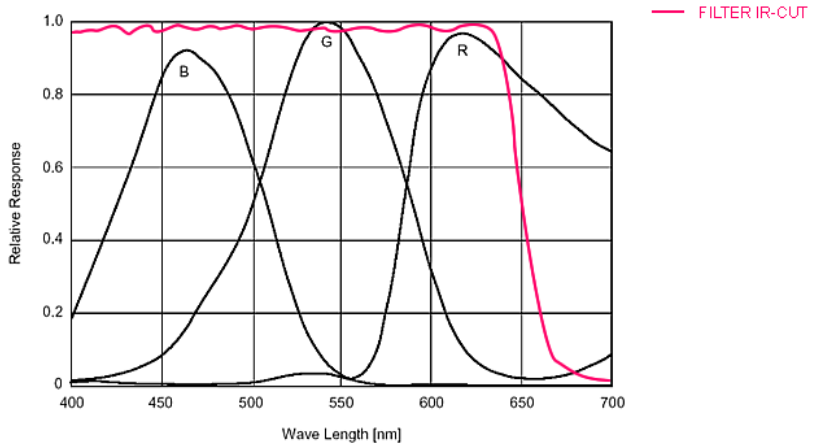
These zone definitions apply to both the color and gray scale version of the sensor.

Zone Definition of Video Signal Shading



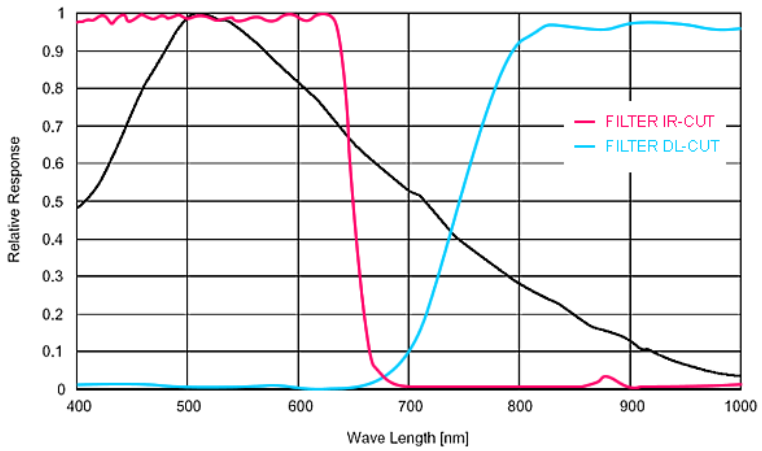
20.1.1.5.2 Color version

Spectral Sensitivity Characteristics (excludes lens characteristics and light source characteristics)



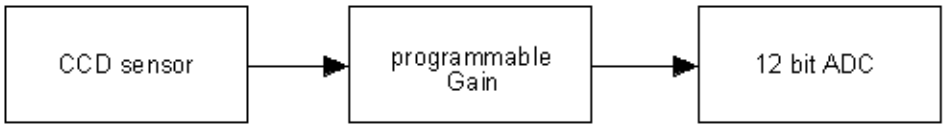
20.1.1.5.3 Gray scale version

Spectral Sensitivity Characteristics (excludes lens characteristics and light source characteristics)



20.1.1.6 CCD Signal Processing

The CCD signal is processed with an analog front-end and digitized by an 12 bit analog-to-digital converter (ADC). The analog front-end contains a programmable gain amplifier which is variable from 0db (gain=0) to 30dB (gain=255).



The 8 most significant bits of the ADC are captured to the frame buffer. This will give the following transfer function (based on the 8 bit digital code): Digital_code [lsb] = ccd_signal[V] ∗ 256[lsb/V] ∗ exp(gain[bB]/20) lsb : least significant bit (smallest digital code change)

Device Feature And Property List

20.1.1.7 Device Feature And Property List

- mvBlueFOX-220G Features
- mvBlueFOX-220C Features

20.1.1.7.1 mvBlueFOX-220G Features

20.1.1.7.2 mvBlueFOX-220C Features

20.1.2 mvBlueFOX-[Model]220a (0.3 Mpix [640 x 480])

20.1.2.1 Introduction

The CCD sensor is a highly programmable imaging module which will, for example, enable the following type of applications

Industrial applications:

- triggered image acquisition with precise control of image integration start by hardware trigger input.
- image acquisition of fast moving objects due to:
  - frame integration, integrating all pixels at a time in contrast to CMOS imager which typically integrate line-by-line.
  - short shutter time, to get sharp images.
  - flash control output to have enough light for short time.

Scientific applications:

- long time integration for low light conditions.
- optimizing image quality using the variable shutter control.

20.1.2.2 Details of operation

The process of getting an image from the CCD sensor can be separated into three different phases.

20.1.2.2.1 Trigger

When coming out of reset or ready with the last readout the CCD controller is waiting for a Trigger signal.

The following trigger modes are available:

Mode	Description
Continuous	Free running, no external trigger signal needed.
OnDemand	Image acquisition triggered by command (software trigger).
OnLowLevel	As long as trigger signal is *Low* camera acquires images with own timing.
OnHighLevel	As long as trigger signal is *High* camera acquires images with own timing.
OnFallingEdge	Each falling edge of trigger signal acquires one image.
OnRisingEdge	Each rising edge of trigger signal acquires one image.
OnHighExpose	Each rising edge of trigger signal acquires one image, exposure time corresponds to pulse width.

TriggerSource Impact Acquire	TriggerSource GenICam(BCX)
GP-IN0	Line4
GP-IN1	Line5

See also

For detailed description about the trigger modes (https://www.balluff.com/en-de/documentation-for-you [Impact Acquire API])

- **C: TCameraTriggerMode**
- **C++**: **mvIMPACT::acquire::TCameraTriggerMode**

Note

Trigger modes which use an external input (**ctmOnLowLevel**, **ctmOnHighLevel**, **ctmOnRisingEdge**, **ctmOnFallingEdge**) will use digital input 0 as input for the trigger signal. Input 0 is not restricted to the trigger function. It can always also be used as general purpose digital input. The input switching threshold of all inputs can be programmed with write_dac(level_in_mV). The best is to set this to the half of the input voltage. So for example if you apply a 24V switching signal to the digital inputs set the threshold to 12000 mV.

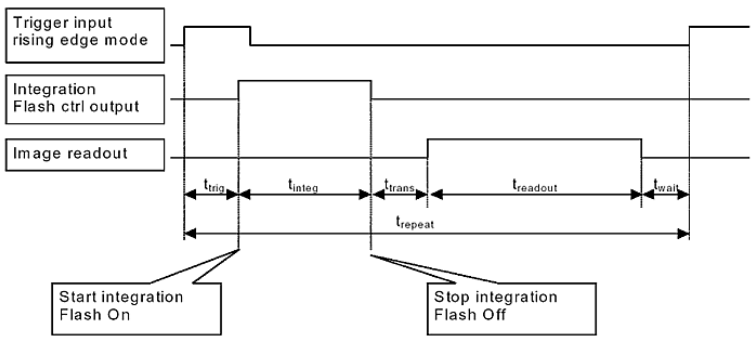
20.1.2.2.2 Exposure aka Integration

After an active trigger, the integration phase starts with a maximum jitter of t_{trig}. If flash illumination is enabled in software the flash output will be activated exactly while the sensor chip is integrating light. Exposure time is adjustable by software in increments of $t_{readline}$.

20.1.2.2.3 Readout

When integration is finished, the image is transferred to hidden storage cells on the CCD. Image data is then shifted out line-by-line and transferred to memory. Shifting out non active lines takes tvshift, while shifting out active lines will consume $t_{readline}$. The number of active pixels per line will not have any impact on readout speed.

20.1.2.3 CCD Timing



Name	Description	Pixel clock	
		20 MHz	40 MHz
t_{trig}	Time from trigger (internal or external) to exposure start	3.6us	1.8us
t_{trans}	Image transfer time (move image to readout cells in CCD)	42.6us	21.3us
$t_{readline}$	time needed to readout a line	39.05us	19.525us
t_{vshift}	time needed to shift unused lines away	3.6us	1.8us
t_{wait}	minimal time to next trigger	7.2us	3.6us
$t_{exposure}$	Exposure time	1us..10s	1us..10s
$t_{readout}$	Image readout time (move image from readout cells to memory	$t_{readout}$ = (ActiveLines $*$ $t_{readline}$) + (504 - ActiveLines) $*$ t_{vshift} + $t_{readline}$	

20.1.2.3.1 Timings

Note

In partial scan mode (readout window ysize $<$ 480 lines).

To calculate the maximum frames per second (FPS_{max}) you will need following formula (Expose mode: No overlap):

$$FPS\_max = \frac{1}{t\_trig + t\_readout + t\_exposure + t\_trans + t\_wait}$$

(Expose mode: Overlapped):

$t\_trig + t\_readout + t\_trans + t\_wait < t\_exposure$: $FPS\_max = \frac{1}{t\_exposure}$

$t\_trig + t\_readout + t\_trans + t\_wait > t\_exposure$: $FPS\_max = \frac{1}{t\_trig + t\_readout + t\_trans + t\_wait}$

20.1.2.3.2 Example: Frame rate as function of lines & exposure time

Now, when we insert the values using exposure time of, for example, 8000 us, 480 lines and 40MHz pixel clock (Expose mode: No overlap):

$$FPS\_max = \frac{1}{1.8\ us + ((480 * 19.525\ us) + ((504 - 480) * 1.80\ us) + 19.525\ us) + 8000\ us + 21.3\ us + 3.6\ us}$$
$$= 0.00005726909458993180 68\ \ 1\ /\ us$$
$$= 57.3$$

20.1.2.3.3 Frame rate calculator

Note

The calculator returns the max. frame rate supported by the sensor. Please keep in mind that it will depend on the interface and the used image format if this frame rate can be transferred.

See also

To find out how to achieve any defined freq. below or equal to the achievable max. freq., please have a look at Achieve a defined image frequency (HRTC).

20.1.2.4 Reprogramming CCD Timing

Reprogramming the CCD Controller will happen when the following changes occur

- Changing the exposure time
- Changing the capture window
- Changing Trigger Modes

Reprogram time consists of two phases

1. Time needed to send data to the CCD controller depending on what is changed
   exposure : abt 2..3ms
   window: abt 4..6ms
   trigger mode: from 5..90ms,
   varies with oldmode/newmode combination
2. Time to initialize (erase) the CCD chip after reprogramming this is fixed, abt 4.5 ms

So for example when reprogramming the capture window you will need (average values)

$t_{regprog}$ = change_window + init_ccd

$t_{regprog}$ = 5ms + 4.5ms

$t_{regprog}$ = 9.5ms

20.1.2.5 CCD Sensor Data

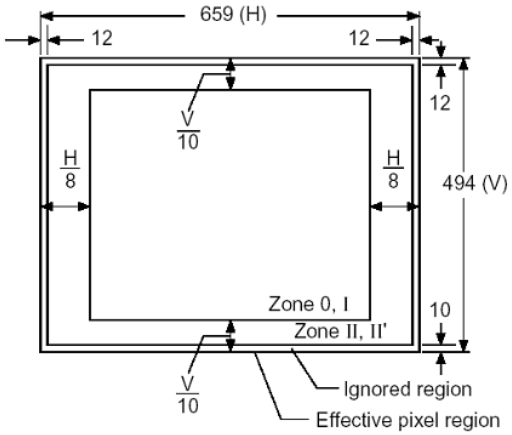
Device Structure

- Interline CCD image sensor
- Image size: Diagonal 6mm (Type 1/3)
- Number of effective pixels: 659 (H) x 494 (V) approx. 330K pixels
- Total number of pixels: 692 (H) x 504 (V) approx. 350K pixels
- Chip size: 5.79mm (H) x 4.89mm (V)
- Unit cell size: 7.4um (H) x 7.4um (V)
- Optical black:
  - Horizontal (H) direction: Front 2 pixels, rear 31 pixels
  - Vertical (V) direction: Front 8 pixels, rear 2 pixels
- Number of dummy bits: Horizontal 16 Vertical 5
- Substrate material: Silicon

20.1.2.5.1 Characteristics

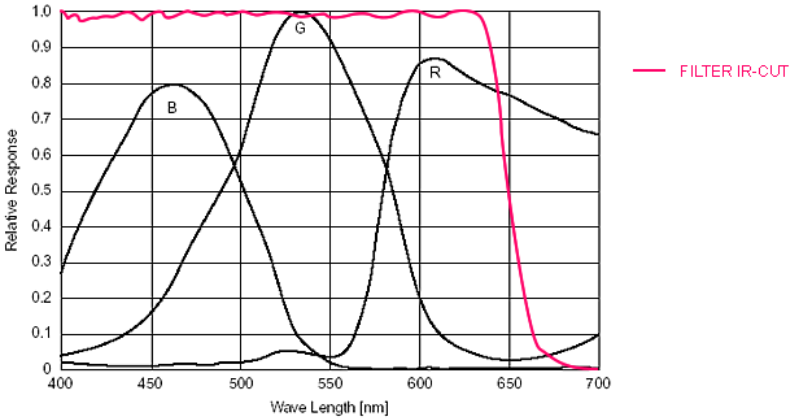
These zone definitions apply to both the color and gray scale version of the sensor.

Zone Definition of Video Signal Shading



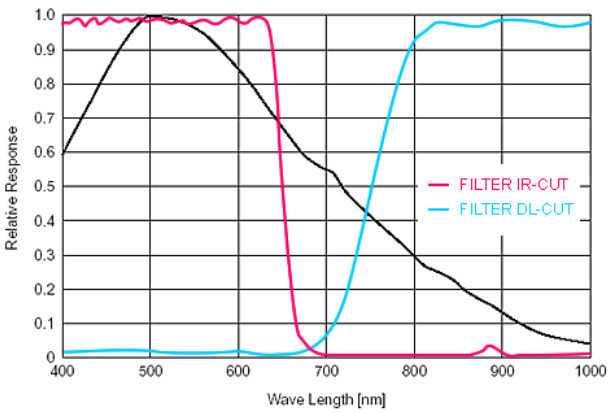
20.1.2.5.2 Color version

Spectral Sensitivity Characteristics (Includes lens characteristics, excludes light source characteristics)



20.1.2.5.3 Gray scale version

Spectral Sensitivity Characteristics (Excludes lens characteristics and light source characteristics)



Device Feature And Property List

20.1.2.6 Device Feature And Property List

- mvBlueFOX-220aG Features
- mvBlueFOX-220aC Features

20.1.2.6.1 mvBlueFOX-220aG Features

20.1.2.6.2 mvBlueFOX-220aC Features

20.1.3 mvBlueFOX-[Model]221 (0.8 Mpix [1024 x 768])

20.1.3.1 Introduction

The CCD sensor is a highly programmable imaging module which will, for example, enable the following type of applications

Industrial applications:

- triggered image acquisition with precise control of image exposure start by hardware trigger input.
- image acquisition of fast moving objects due to:
  - frame exposure, integrating all pixels at a time in contrast to CMOS imager which typically integrate line-by-line.
  - short shutter time, to get sharp images.
  - flash control output to have enough light for short time.

Scientific applications:

- long time exposure for low light conditions.
- optimizing image quality using the variable shutter control.

20.1.3.2 Details of operation

The process of getting an image from the CCD sensor can be separated into three different phases.

20.1.3.2.1 Trigger

When coming out of reset or ready with the last readout the CCD controller is waiting for a Trigger signal.

The following trigger modes are available:

Mode	Description
Continuous	Free running, no external trigger signal needed.
OnDemand	Image acquisition triggered by command (software trigger).
OnLowLevel	As long as trigger signal is *Low* camera acquires images with own timing.
OnHighLevel	As long as trigger signal is *High* camera acquires images with own timing.
OnFallingEdge	Each falling edge of trigger signal acquires one image.
OnRisingEdge	Each rising edge of trigger signal acquires one image.
OnHighExpose	Each rising edge of trigger signal acquires one image, exposure time corresponds to pulse width.
OnLowExpose	Each falling edge of trigger signal acquires one image, exposure time corresponds to pulse width.
OnAnyEdge	Start the exposure of a frame when the trigger input level changes from high to low or from

See also

For detailed description about the trigger modes (https://www.balluff.com/en-de/documentation-for-you [Impact Acquire API])

- **C: TCameraTriggerMode**
- **C++**: **mvIMPACT::acquire::TCameraTriggerMode**

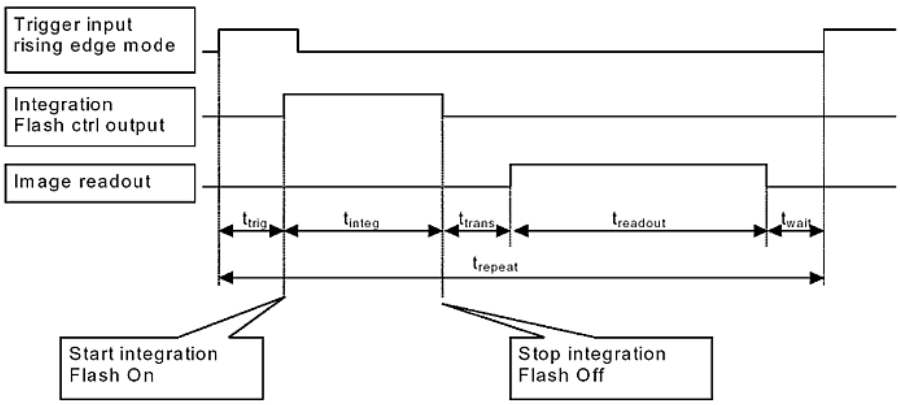
20.1.3.2.2 Exposure aka Integration

After an active trigger, the exposure phase starts with a maximum jitter of t_{trig}. If flash illumination is enabled in software the flash output will be activated exactly while the sensor chip is integrating light. Integration time is adjustable by software in increments of $t_{readline}$.

20.1.3.2.3 Readout

When exposure is finished, the image is transferred to hidden storage cells on the CCD. Image data is then shifted out line-by-line and transferred to memory. Shifting out non active lines takes tvshift, while shifting out active lines will consume $t_{readline}$. The number of active pixels per line will not have any impact on readout speed.

20.1.3.3 CCD Timing



Name	Description	Pixel clock	
		20 MHz	40 MHz
t_{trig}	Time from trigger (internal or external) to exposure start	9.7us	4.85us
t_{trans}	Image transfer time (move image to readout cells in CCD)	45us	22.5us
$t_{readline}$	time needed to readout a line	65.4us	32.7us
t_{vshift}	time needed to shift unused lines away	9.7us	4.85us
t_{wait}	minimal time to next trigger	116us	58us
$t_{exposure}$	Integration time	1us..10s	1us..10s
$t_{readout}$	Image readout time (move image from readout cells to memory	$t_{readout}$ = (ActiveLines $*$ $t_{readline}$) + (788 - ActiveLines) $*$ t_{vshift} + $t_{readline}$	

20.1.3.3.1 Timings

Note

In partial scan mode (readout window ysize $<$ 768 lines).

To calculate the maximum frames per second (FPS_{max}) you will need following formula (Expose mode: Sequential):

```
FPS_max =                        1
          ------------------------------------------------
          t_trig + t_readout + t_exposure + t_trans + t_wait
```

(Expose mode: Overlapped):

```
t_trig + t_readout + t_trans + t_wait < t_exposure:    FPS_max =        1
                                                                ---------------
                                                                    t_exposure

t_trig + t_readout + t_trans + t_wait  > t_exposure:    FPS_max =                    1
                                                                ------------------------------------
                                                                t_trig + t_readout + t_trans + t_wait
```

Example: Frame rate as function of lines & exposure time

Now, when we insert the values using exposure time of, for example, 8000 us, 768 lines and 40MHz pixel clock (Expose mode: Sequential):

```
FPS_max =                                                1
          ------------------------------------------------------------------------------------------
          4.85 us + ((768 * 32.7 us) + ((788 - 768) * 4.85 us) + 32.7 us) + 8000 us + 22.5 us + 58 us
        = 0.00003000042155922907 17   1 / us
        = 30
```

Note

The calculator returns the max. frame rate supported by the sensor. Please keep in mind that it will depend on the interface and the used image format if this frame rate can be transferred.

See also

To find out how to achieve any defined freq. below or equal to the achievable max. freq., please have a look at Achieve a defined image frequency (HRTC).

20.1.3.4 Reprogramming CCD Timing

Reprogramming the CCD Controller will happen when the following changes occur

- Changing the exposure time
- Changing the capture window
- Changing Trigger Modes

Reprogram time consists of two phases

1. Time needed to send data to the CCD controller depending on what is changed
   exposure : abt 2..3ms
   window: abt 4..6ms
   trigger mode: from 5..90ms,
   varies with oldmode/newmode combination
2. Time to initialize (erase) the CCD chip after reprogramming this is fixed, abt 4.5 ms

So for example when reprogramming the capture window you will need (average values)

$t_{regprog}$ = change_window + init_ccd

$t_{regprog}$ = 5ms + 4.5ms

$t_{regprog}$ = 9.5ms

20.1.3.5 CCD Sensor Data

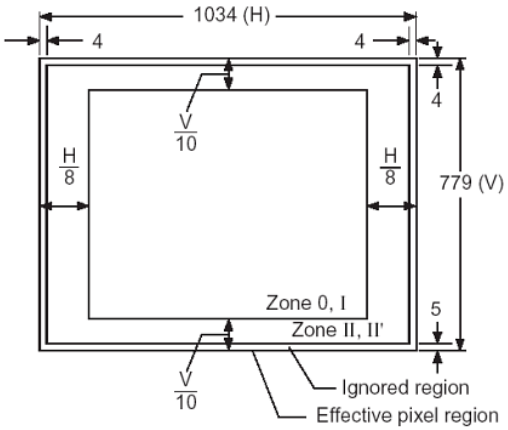
Device Structure

- Interline CCD image sensor
- Image size: Diagonal 6mm (Type 1/3)
- Number of effective pixels: 1025 (H) x 768 (V) approx. 790K pixels
- Total number of pixels: 1077 (H) x 788 (V) approx. 800K pixels
- Chip size: 5.80mm (H) x 4.92mm (V)
- Unit cell size: 4.65um (H) x 4.65um (V)
- Optical black:
  - Horizontal (H) direction: Front 3 pixels, rear 40 pixels
  - Vertical (V) direction: Front 7 pixels, rear 2 pixels
- Number of dummy bits: Horizontal 29 Vertical 1
- Substrate material: Silicon

20.1.3.5.1 Characteristics

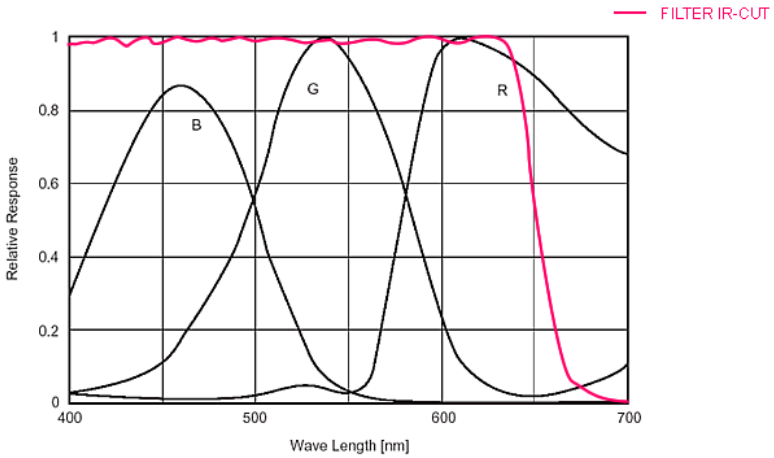
These zone definitions apply to both the color and gray scale version of the sensor.

Zone Definition of Video Signal Shading

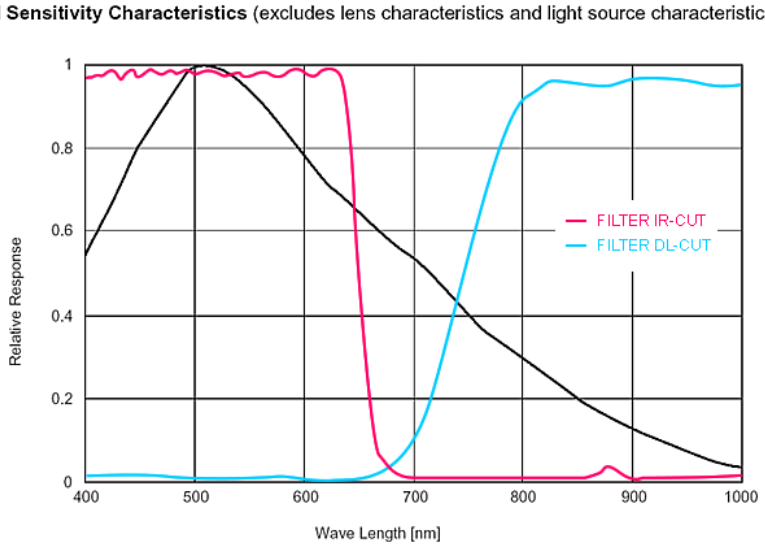


20.1.3.5.2 Color version

Spectral Sensitivity Characteristics (excludes lens characteristics and light source characteristics)



20.1.3.5.3 Gray scale version



20.1.3.6 CCD Signal Processing

The CCD signal is processed with an analog front-end and digitized by an 12 bit analog-to-digital converter (ADC). The analog front-end contains a programmable gain amplifier which is variable from 0db (gain=0) to 30dB (gain=255).

The 8 most significant bits of the ADC are captured to the frame buffer. This will give the following transfer function (based on the 8 bit digital code): Digital_code [lsb] = ccd_signal[V] ∗ 256[lsb/V] ∗ exp(gain[bB]/20) lsb : least significant bit (smallest digital code change)

Device Feature And Property List

20.1.3.7 Device Feature And Property List

- mvBlueFOX-221G Features
- mvBlueFOX-221C Features

20.1.3.7.1 mvBlueFOX-221G Features

20.1.3.7.2 mvBlueFOX-221C Features

20.1.4 mvBlueFOX-[Model]223 (1.4 Mpix [1360 x 1024])

20.1.4.1 Introduction

The CCD sensor is a highly programmable imaging module which will, for example, enable the following type of applications

Industrial applications:

- triggered image acquisition with precise control of image exposure start by hardware trigger input.

- image acquisition of fast moving objects due to:
  - frame exposure, integrating all pixels at a time in contrast to CMOS imager which typically integrate line-by-line.
  - short shutter time, to get sharp images.
  - flash control output to have enough light for short time.

Scientific applications:

- long time exposure for low light conditions.
- optimizing image quality using the variable shutter control.

20.1.4.2 Details of operation

The process of getting an image from the CCD sensor can be separated into three different phases.

20.1.4.2.1 Trigger

When coming out of reset or ready with the last readout the CCD controller is waiting for a Trigger signal.

The following trigger modes are available:

Mode	Description
Continuous	Free running, no external trigger signal needed.
OnDemand	Image acquisition triggered by command (software trigger).
OnLowLevel	As long as trigger signal is *Low* camera acquires images with own timing.
OnHighLevel	As long as trigger signal is *High* camera acquires images with own timing.
OnFallingEdge	Each falling edge of trigger signal acquires one image.
OnRisingEdge	Each rising edge of trigger signal acquires one image.
OnHighExpose	Each rising edge of trigger signal acquires one image, exposure time corresponds to pulse width.

See also

For detailed description about the trigger modes (https://www.balluff.com/en-de/documentation-for-you
[Impact Acquire API])

- **C: TCameraTriggerMode**
- **C++**: **mvIMPACT::acquire::TCameraTriggerMode**

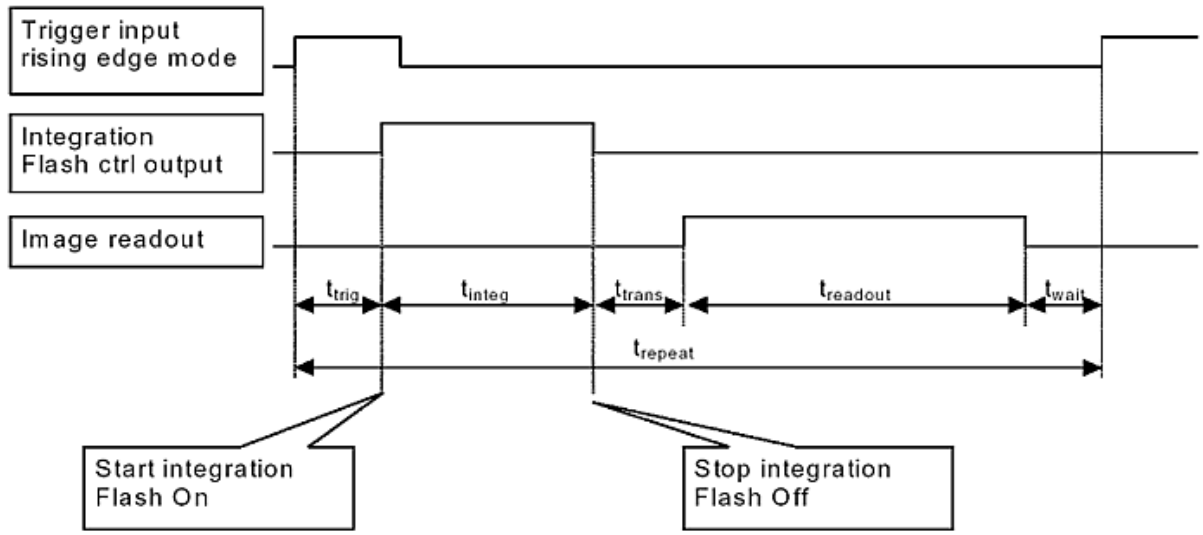
20.1.4.2.2 Exposure aka Integration

After an active trigger, the exposure phase starts with a maximum jitter of t_{trig}. If flash illumination is enabled in software the flash output will be activated exactly while the sensor chip is integrating light. Exposure time is adjustable by software in increments of $t_{readline}$.

20.1.4.2.3 Readout

When exposure is finished, the image is transferred to hidden storage cells on the CCD. Image data is then shifted out line-by-line and transferred to memory. Shifting out non active lines takes tvshift, while shifting out active lines will consume $t_{readline}$. The number of active pixels per line will not have any impact on readout speed.

20.1.4.3 CCD Timing



20.1.4.3.1 Timings

Note

In partial scan mode (readout window ysize $<$ 1024 lines).

To calculate the maximum frames per second (FPS_{max}) you will need following formula (Expose mode: No overlap):

20.1.4.3.2 Example: Frame rate as function of lines & exposure time Now, when we insert the values using exposure time of, for example, 8000 us, 1024 lines and 56MHz pixel clock (Expose mode: No overlap):

See also

To find out how to achieve any defined freq. below or equal to the achievable max. freq., please have a look at Achieve a defined image frequency (HRTC).

20.1.4.4 Reprogramming CCD Timing

Reprogramming the CCD Controller will happen when the following changes occur

- Changing the exposure time
- Changing the capture window
- Changing Trigger Modes

Reprogram time consists of two phases

1. Time needed to send data to the CCD controller depending on what is changed **exposure** : abt 2..3ms **window:** abt 4..6ms **trigger mode:** from 5..90ms, varies with oldmode/newmode combination
2. Time to initialize (erase) the CCD chip after reprogramming this is fixed, abt 4.5 ms

So for example when reprogramming the capture window you will need (average values)

$t_{regprog}$ = change_window + init_ccd

$t_{regprog}$ = 5ms + 4.5ms

$t_{regprog}$ = 9.5ms

20.1.4.5 CCD Sensor Data

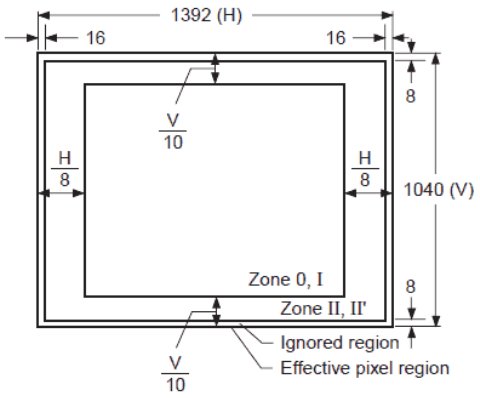
Device Structure

- Interline CCD image sensor
- Image size: Diagonal 8mm (Type 1/2)
- Number of effective pixels: 1392 (H) x 1040 (V) approx. 1.45M pixels
- Total number of pixels: 1434 (H) x 1050 (V) approx. 1.5M pixels
- Chip size: 7.60mm (H) x 6.2mm (V)
- Unit cell size: 4.65um (H) x 4.65um (V)
- Optical black:
  - Horizontal (H) direction: Front 2 pixels, rear 40 pixels
  - Vertical (V) direction: Front 8 pixels, rear 2 pixels
- Number of dummy bits: Horizontal 20 Vertical 3
- Substrate material: Silicon

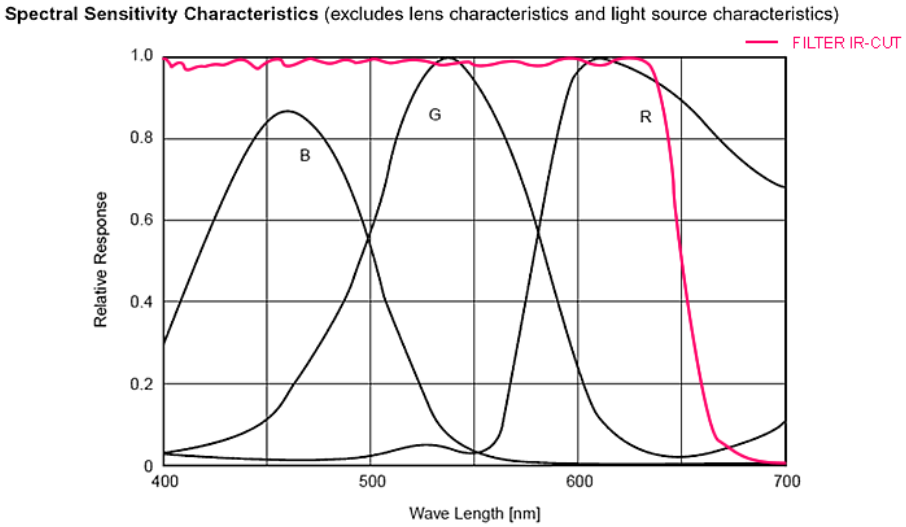
20.1.4.5.1 Characteristics

These zone definitions apply to both the color and gray scale version of the sensor.

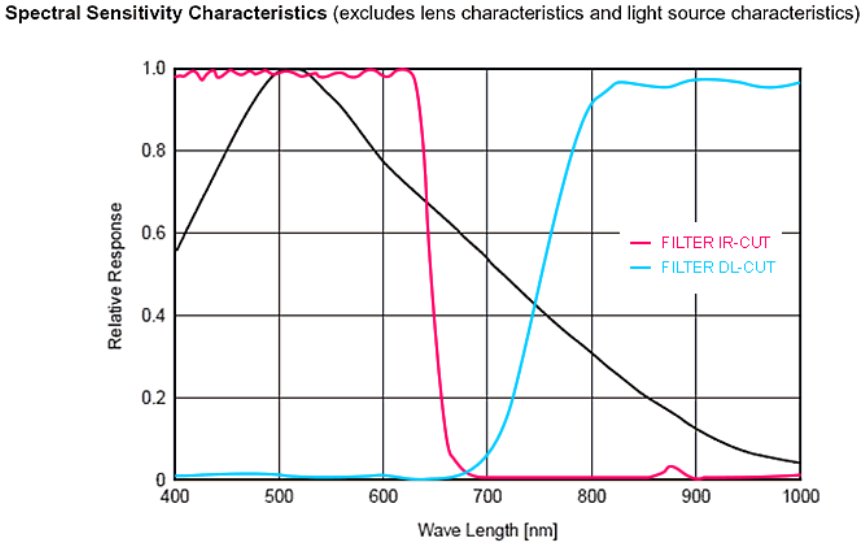
Zone Definition of Video Signal Shading



20.1.4.5.2 Color version

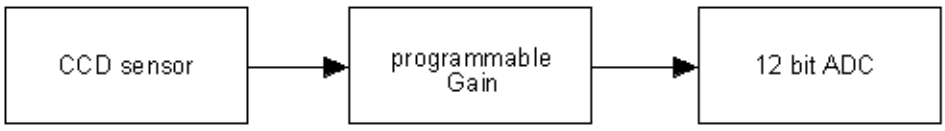


20.1.4.5.3 Gray scale version



20.1.4.6 CCD Signal Processing

The CCD signal is processed with an analog front-end and digitized by an 12 bit analog-to-digital converter (ADC). The analog front-end contains a programmable gain amplifier which is variable from 0db (gain=0) to 30dB (gain=255).



The 8 most significant bits of the ADC are captured to the frame buffer. This will give the following transfer function (based on the 8 bit digital code): Digital_code [lsb] = ccd_signal[V] ∗ 256[lsb/V] ∗ exp(gain[bB]/20) lsb : least significant bit (smallest digital code change)

Device Feature And Property List

20.1.4.7 Device Feature And Property List

- mvBlueFOX-223G Features
- mvBlueFOX-223C Features

20.1.4.7.1 mvBlueFOX-223G Features

20.1.4.7.2 mvBlueFOX-223C Features

20.1.5 mvBlueFOX-[Model]224 (1.9 Mpix [1600 x 1200])

20.1.5.1 Introduction

The CCD sensor is a highly programmable imaging module which will, for example, enable the following type of applications

Industrial applications:

- triggered image acquisition with precise control of image exposure start by hardware trigger input.
- image acquisition of fast moving objects due to:
  - frame exposure, integrating all pixels at a time in contrast to CMOS imager which typically integrate line-by-line.
  - short shutter time, to get sharp images.
  - flash control output to have enough light for short time.

Scientific applications:

- long time exposure for low light conditions.
- optimizing image quality using the variable shutter control.

20.1.5.2 Details of operation

The process of getting an image from the CCD sensor can be separated into three different phases.

20.1.5.2.1 Trigger

When coming out of reset or ready with the last readout the CCD controller is waiting for a Trigger signal.

The following trigger modes are available:

Mode	Description
Continuous	Free running, no external trigger signal needed.
OnDemand	Image acquisition triggered by command (software trigger).
OnLowLevel	As long as trigger signal is *Low* camera acquires images with own timing.
OnHighLevel	As long as trigger signal is *High* camera acquires images with own timing.
OnFallingEdge	Each falling edge of trigger signal acquires one image.
OnRisingEdge	Each rising edge of trigger signal acquires one image.
OnHighExpose	Each rising edge of trigger signal acquires one image, exposure time corresponds to pulse width.

TriggerSource Impact Acquire	TriggerSource GenICam(BCX)
GP-IN0	Line4
GP-IN1	Line5

See also

For detailed description about the trigger modes (https://www.balluff.com/en-de/documentation-for-you [Impact Acquire API])

- **C: TCameraTriggerMode**
- **C++**: **mvIMPACT::acquire::TCameraTriggerMode**

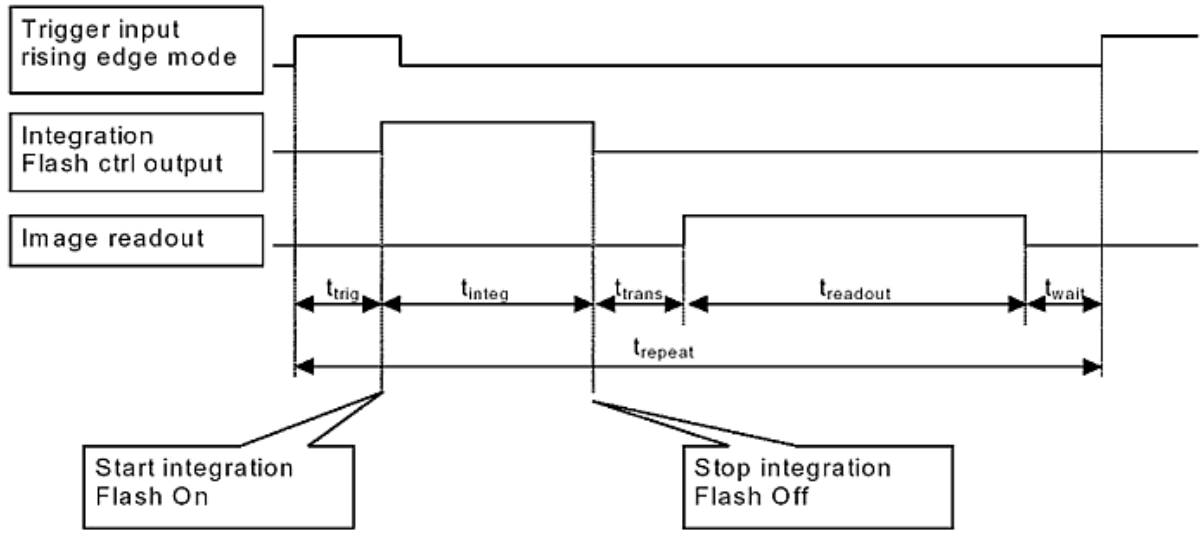
20.1.5.2.2 Exposure aka Integration

After an active trigger, the exposure phase starts with a maximum jitter of t_{trig}. If flash illumination is enabled in software the flash output will be activated exactly while the sensor chip is integrating light. Exposure time is adjustable by software in increments of $t_{readline}$.

20.1.5.2.3 Readout

When exposure is finished, the image is transferred to hidden storage cells on the CCD. Image data is then shifted out line-by-line and transferred to memory. Shifting out non active lines takes tvshift, while shifting out active lines will consume $t_{readline}$. The number of active pixels per line will not have any impact on readout speed.

20.1.5.3 CCD Timing



Name	Description	Pixel clock	
		20 MHz	40 MHz
t_{trig}	Time from trigger (internal or external) to exposure start	10.2us	5.1us
t_{trans}	Image transfer time (move image to readout cells in CCD)	96us	48us
$t_{readline}$	time needed to readout a line	96us	48us

t_{vshift}	time needed to shift unused lines away	10.2us	5.1us
t_{wait}	minimal time to next trigger	316us	158us
$t_{exposure}$	Exposure time	1us..10s	1us..10s
$t_{readout}$	Image readout time (move image from readout cells to memory	$t_{readout}$ = (ActiveLines $*$ $t_{readline}$) + (1248 - ActiveLines) $*$ t_{vshift} + $t_{readline}$	

20.1.5.3.1 Timings

Note

In partial scan mode (readout window ysize $<$ 1200 lines).

To calculate the maximum frames per second (FPS_{max}) you will need following formula (Expose mode: No overlap):

$$FPS\_max = \frac{1}{t\_trig + t\_readout + t\_exposure + t\_trans + t\_wait}$$

(Expose mode: Overlapped):

$t\_trig + t\_readout + t\_trans + t\_wait < t\_exposure$: $FPS\_max = \frac{1}{t\_exposure}$

$t\_trig + t\_readout + t\_trans + t\_wait > t\_exposure$: $FPS\_max = \frac{1}{t\_trig + t\_readout + t\_trans + t\_wait}$

20.1.5.3.2 Example: Frame rate as function of lines & exposure time

Now, when we insert the values using exposure time of, for example, 8000 us, 1200 lines and 40MHz pixel clock (Expose mode: No overlap):

$$\begin{aligned} FPS\_max &= \frac{1}{5.1\ us + ((1200 * 48\ us) + ((1248 - 1200) * 5.1\ us) + 48\ us) + 8000\ us + 48\ us + 158\ us} \\ &= 0.000015127700483632586\ \ 1\ /\ us \\ &= 15.1 \end{aligned}$$

20.1.5.3.3 Frame rate calculator

Note

The calculator returns the max. frame rate supported by the sensor. Please keep in mind that it will depend on the interface and the used image format if this frame rate can be transferred.

See also

To find out how to achieve any defined freq. below or equal to the achievable max. freq., please have a look at Achieve a defined image frequency (HRTC).

20.1.5.4 Reprogramming CCD Timing

Reprogramming the CCD Controller will happen when the following changes occur

- Changing the exposure time
- Changing the capture window
- Changing Trigger Modes

Reprogram time consists of two phases

1. Time needed to send data to the CCD controller depending on what is changed **exposure** : abt 2..3ms **window:** abt 4..6ms **trigger mode:** from 5..90ms, varies with oldmode/newmode combination
2. Time to initialize (erase) the CCD chip after reprogramming this is fixed, abt 4.5 ms

So for example when reprogramming the capture window you will need (average values)

$t_{regprog}$ = change_window + init_ccd

$t_{regprog}$ = 5ms + 4.5ms

$t_{regprog}$ = 9.5ms

20.1.5.5 CCD Sensor Data

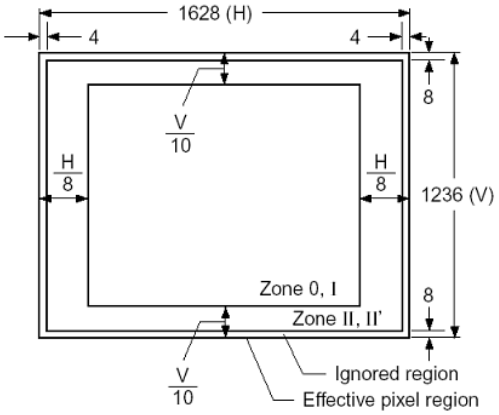
Device Structure

- Interline CCD image sensor
- Image size: Diagonal 8.923mm (Type 1/1.8)
- Number of effective pixels: 1600 (H) x 1200 (V) approx. 1.92M pixels
- Total number of pixels: 1688 (H) x 1248 (V) approx. 2.11M pixels
- Chip size: 8.50mm (H) x 6.8mm (V)
- Unit cell size: 4.4um (H) x 4.4um (V)
- Optical black:
  - Horizontal (H) direction: Front 12 pixels, rear 48 pixels
  - Vertical (V) direction: Front 10 pixels, rear 2 pixels
- Number of dummy bits: Horizontal 28 Vertical 1
- Substrate material: Silicon

20.1.5.5.1 Characteristics

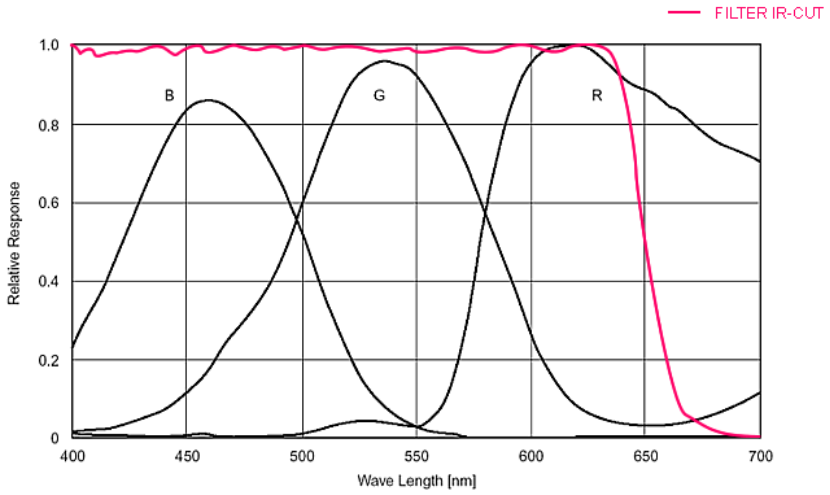
These zone definitions apply to both the color and gray scale version of the sensor.

Zone Definition of Video Signal Shading



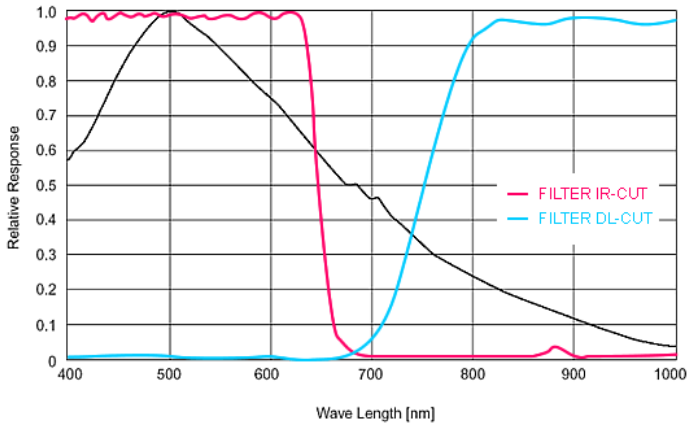
20.1.5.5.2 Color version

Spectral Sensitivity Characteristics (excludes lens characteristics and light source characteristics)



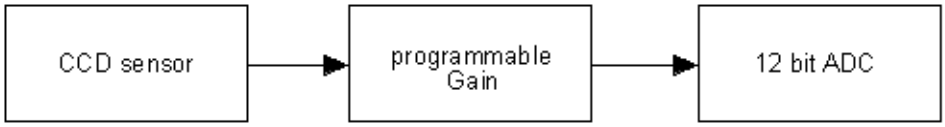
20.1.5.5.3 Gray scale version

Spectral Sensitivity Characteristics (excludes lens characteristics and light source characteristics)



20.1.5.6 CCD Signal Processing

The CCD signal is processed with an analog front-end and digitized by an 12 bit analog-to-digital converter (ADC). The analog front-end contains a programmable gain amplifier which is variable from 0db (gain=0) to 30dB (gain=255).



The 8 most significant bits of the ADC are captured to the frame buffer. This will give the following transfer function (based on the 8 bit digital code): Digital_code [lsb] = ccd_signal[V] ∗ 256[lsb/V] ∗ exp(gain[bB]/20) lsb : least significant bit (smallest digital code change)

Device Feature And Property List

20.1.5.7 Device Feature And Property List

- mvBlueFOX-224G Features
- mvBlueFOX-224C Features

20.1.5.7.1 mvBlueFOX-224G Features

20.1.5.7.2 mvBlueFOX-224C Features

20.2 A.2 CMOS

- mvBlueFOX-[Model]200w (0.4 Mpix [752 x 480])
- mvBlueFOX-[Model]202a (1.3 Mpix [1280 x 1024])
- BVS CA-[MLC|IGC]-0012V / mvBlueFOX-[MLC|IGC]202v (1.2 Mpix [1280 x 960])
- mvBlueFOX-[Model]202b (1.2 Mpix [1280 x 960])
- mvBlueFOX-[Model]202d (1.2 Mpix [1280 x 960])
- mvBlueFOX-[Model]205 (5.0 Mpix [2592 x 1944])

20.2.1 mvBlueFOX-[Model]200w (0.4 Mpix [752 x 480])

20.2.1.1 Introduction

The CMOS sensor module (MT9V034) incorporates the following features:

- resolution to 752 x 480 gray scale or RGB Bayer mosaic
- supports window AOI mode with faster readout
- high dynamic range 110 dB
- programmable analog gain (0..12 dB)
- progressive scan sensor (no interlaced problems!)
- full frame shutter
- programmable readout timing with free capture windows and partial scan
- many trigger modes (free-running, hardware-triggered)

20.2.1.2 Details of operation

The sensor uses a full **frame shutter** (ShutterMode = "`FrameShutter`"), i.e. all pixels are reset at the same time and the exposure commences. It ends with the charge transfer of the voltage sampling.
Furthermore, the sensor offers two different modes of operation:

- free running mode (Overlapping exposure and readout)
- snapshot mode (Sequential exposure and readout)

20.2.1.2.1 Free running mode

In free running mode, the sensor reaches its maximum frame rate. This is done by overlapping erase, exposure and readout phase. The sensor timing in free running mode is fixed, so there is no control when to start an acquisition. This mode is used with trigger mode **Continuous**.

To calculate the maximum frames per second (FPS_{max}) in free running mode you will need following formula:

```
FrameTime = (ImageWidth + 61) * ((ImageHeight + 45) / PixelClock)
```

If exposure time is lower than frame time:

$$FPS\_max = \frac{1}{FrameTime}$$

If exposure time is greater than frame time:

$$FPS\_max = \frac{1}{ExposureTime}$$

20.2.1.2.2 Snapshot mode

In snapshot mode, the image acquisition process consists off several sequential phases:

20.2.1.2.3 Trigger

Snapshot mode starts with a trigger. This can be either a hardware or a software signal.

The following trigger modes are available:

Mode	Description
Continuous	Free running, no external trigger signal needed.
OnDemand	Image acquisition triggered by command (software trigger).
OnLowLevel	As long as trigger signal is *Low* camera acquires images with own timing.
OnHighLevel	As long as trigger signal is *High* camera acquires images with own timing.

See also

Using external trigger with CMOS sensors

20.2.1.2.4 Erase, exposure and readout

All pixels are light sensitive at the same period of time. The whole pixel core is reset simultaneously and after the exposure time all pixel values are sampled together on the storage node inside each pixel. The pixel core is read out line-by-line after exposure.

> **Note**
>
> Exposure and read out cycle is carry-out in serial; that causes that no exposure is possible during read out.

The step width for the exposure time is 1 us.

Image data is then shifted out line-by-line and transferred to memory.

To calculate the maximum frames per second (FPS_{max}) in snapshot mode you will need following formula:

```
FrameTime = (ImageWidth + 61) * ((ImageHeight + 45) / PixelClock)
```

$$FPS\_max = \frac{1}{FrameTime + ExposureTime}$$

AOI	PixelClock (MHz)	Exposure Time (us)	Maximal Frame Rate (fps)	PixelFormat
Maximum	40	100	93.7	Mono8
W:608 x H:388	40	100	131.4	Mono8
W:492 x H:314	40	100	158.5	Mono8
W:398 x H:206	40	100	226.7	Mono8

20.2.1.3 Measured frame rates

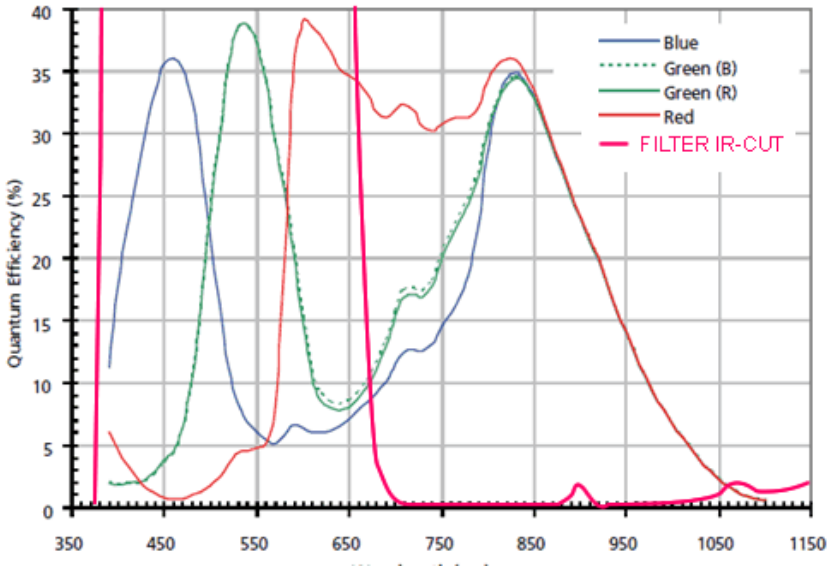
20.2.1.4 Sensor Data

Device Structure

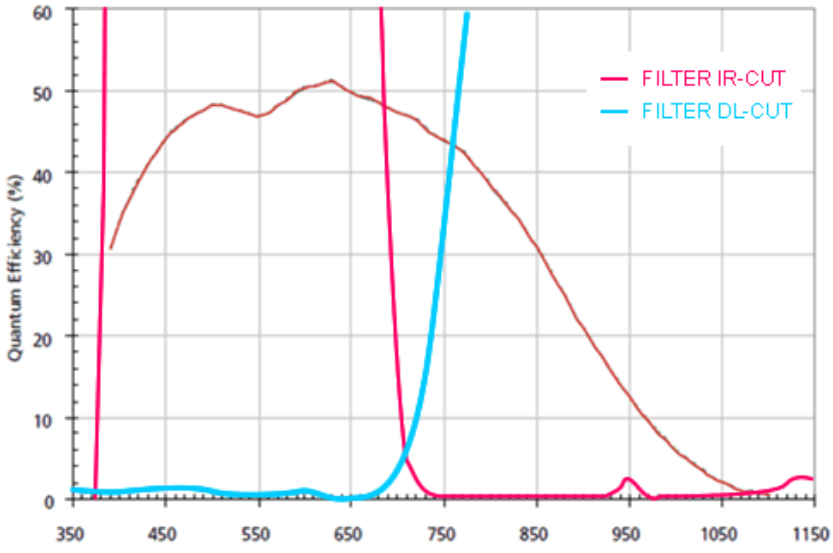
- Progressive scan CMOS image sensor
- Image size: 4.51(H)x2.88(V)mm (Type 1/3")
- Number of effective pixels: 752 (H) x 480 (V)
- Unit cell size: 6um (H) x 6um (V)

20.2.1.4.1 Characteristics

20.2.1.4.2 Color version



20.2.1.4.3 Gray scale version



Device Feature And Property List

20.2.1.5 Device Feature And Property List

- mvBlueFOX-200wG Features
- mvBlueFOX-200wC Features

20.2.1.5.1 mvBlueFOX-200wG Features

20.2.1.5.2 mvBlueFOX-200wC Features

20.2.2 mvBlueFOX-[Model]202a (1.3 Mpix [1280 x 1024])

20.2.2.1 Introduction

The CMOS sensor module (MT9M001) incorporates the following features:

- resolution to 1280 x 1024 gray scale
- supports window AOI mode with faster readout
- dynamic range 61dB
- programmable analog gain (0..12dB)
- progressive scan sensor (no interlaced problems!)
- rolling shutter
- programmable readout timing with free capture windows and partial scan
- many trigger modes (free-running, hardware-triggered)

20.2.2.2 Details of operation

The sensor uses following acquisition mode:

- **rolling shutter** (ShutterMode = "`ElectronicRollingShutter`").

With the **rolling shutter** the lines are exposed for the same duration, but at a slightly different point in time.

> **Note**
>
> Moving objects together with a rolling shutter can cause a shear in moving objects.

Furthermore, the sensor offers one operating mode:

- snapshot mode (which means sequential exposure and readout)

20.2.2.2.1 Snapshot mode

In snapshot mode, the image acquisition process consists off several sequential phases:

20.2.2.2.2 Trigger

Snapshot mode starts with a trigger. This can be either a hardware or a software signal.

The following trigger modes are available:

Mode	Description
Continuous	Free running, no external trigger signal needed.
OnDemand	Image acquisition triggered by command (software trigger).
OnLowLevel	As long as trigger signal is *Low* camera acquires images with own timing.
OnHighLevel	As long as trigger signal is *High* camera acquires images with own timing.
OnFallingEdge	Each falling edge of trigger signal acquires one image.
OnRisingEdge	Each rising edge of trigger signal acquires one image.
OnHighExpose	Each rising edge of trigger signal acquires one image, exposure time corresponds to pulse width.
OnLowExpose	Each falling edge of trigger signal acquires one image, exposure time corresponds to pulse width.
OnAnyEdge	Start the exposure of a frame when the trigger input level changes from high to low or from low to high.

See also

For detailed description about the trigger modes (https://www.balluff.com/en-de/documentation-for-you [Impact Acquire API])

- **C: TCameraTriggerMode**
- **C++**: **mvIMPACT::acquire::TCameraTriggerMode**

20.2.2.2.3 Erase, exposure and readout

After the trigger pulse, the complete sensor array is erased. This takes some time, so there is a fix delay from about 285 us between the trigger pulse on digital input 0 and the start of exposure of the first line.
The exact time of exposure start of each line (except the first line) depends on the exposure time and the position of the line. The exposure of a particular line N is finished when line N is ready for read-out. Image data is read out line-by-line and transferred to memory (see: https://www.balluff.↩com/de-en/whitepapers/cmos-sensors-with-rolling-shutter).
Exposure time is adjustable by software and depends on the image width. To calculate the exposure step size you will need following formula:

LineDelay = 0

$$PixelClkPeriod = \frac{1}{PixelClk}$$

$$RowTime = (ImageWidth + 244 + LineDelay) * PixelClkPeriod$$

$$RowTime = MinExposurTime = ExposureStepSize$$

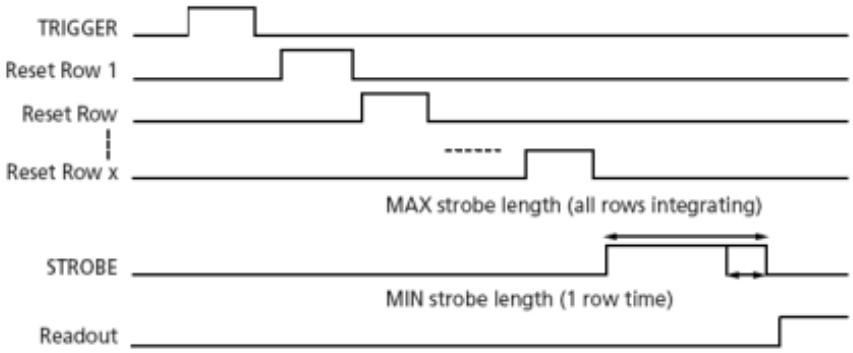
Image data is then shifted out line-by-line and transferred to memory.

To calculate the maximum frames per second (FPS_{max}) in snapshot mode you will need following formula:

$$FrameTime = (ImageWidth + 244) * ((ImageHeight + 16) / PixelClock)$$

$$FPS\_max = \frac{1}{FrameTime + ExposureTime}$$

20.2.2.2.4 CMOS Timing in Snapshot mode



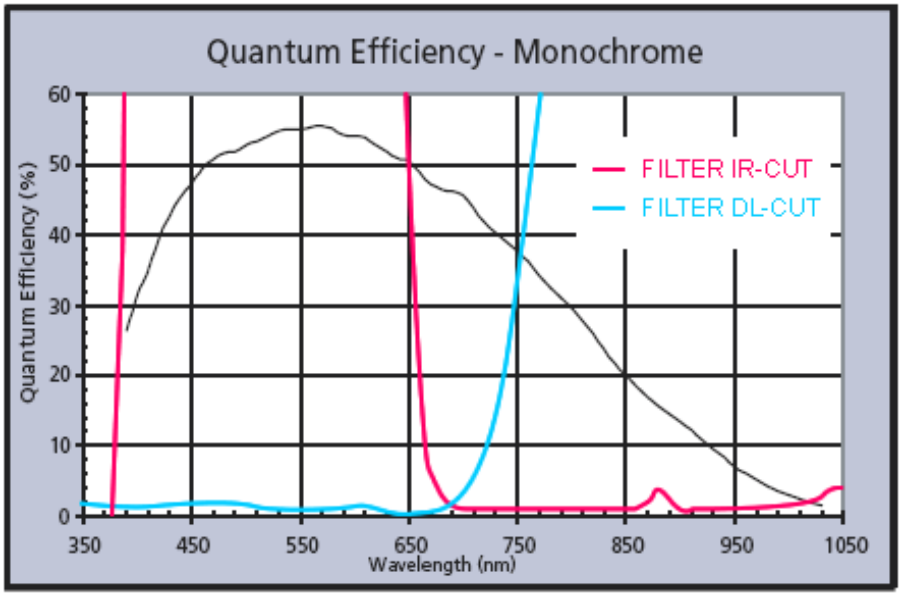
20.2.2.3 Sensor Data

Device Structure

- Progressive scan CMOS image sensor
- Image size: 6.66(H)x5.32(V)mm (Type 1/2")
- Number of effective pixels: 1280 (H) x 1024 (V)
- Unit cell size: 5.2um (H) x 5.2um (V)

20.2.2.4 Characteristics

20.2.2.4.1 Gray scale version



Device Feature And Property List

20.2.2.5 Device Feature And Property List

- mvBlueFOX-202aG Features

20.2.2.5.1 mvBlueFOX-202aG Features

20.2.3 BVS CA-[MLC|IGC]-0012V / mvBlueFOX-[MLC|IGC]202v (1.2 Mpix [1280 x 960])

20.2.3.1 Introduction

The CMOS sensor module (AR0135) incorporates the following features:

- resolution to 1280 x 960 gray scale or RGB Bayer mosaic
- supports window AOI mode with faster readout
- programmable analog gain (0..12 dB)
- progressive scan sensor (no interlaced problems!)
- pipelined global shutter
- programmable readout timing with free capture windows and partial scan
- many trigger modes (free-running, hardware-triggered)

20.2.3.2 Details of operation

The sensor uses a pipelined global snapshot shutter (`ShutterMode` = `"FrameShutter"`) , i.e. light exposure takes place on all pixels in parallel, although subsequent readout is sequential.
Therefore the sensor offers two different modes of operation:

- free running mode (Overlapping exposure and readout)
- snapshot mode (Sequential exposure and readout)

20.2.3.2.1 Free running mode

In free running mode, the sensor reaches its maximum frame rate. This is done by overlapping erase, exposure and readout phase. The sensor timing in free running mode is fixed, so there is no control when to start an acquisition. This mode is used with trigger mode **Continuous**.

To calculate the maximum frames per second (FPS_{max}) in free running mode you will need following formula:

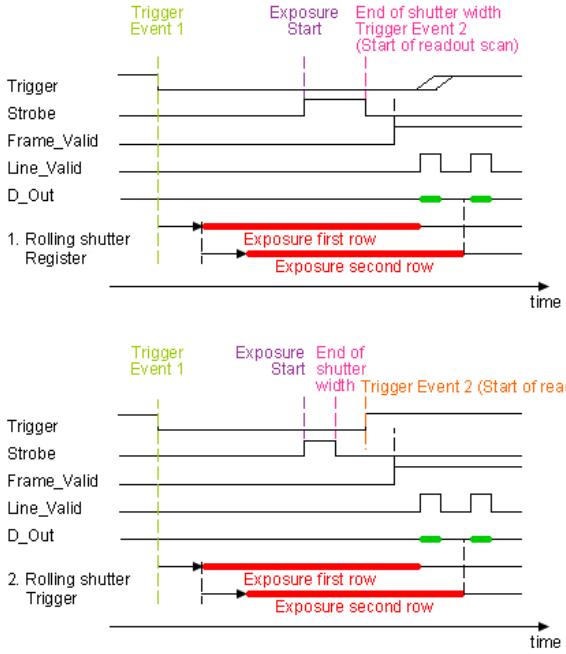
```
FrameTime = (ImageHeight * (1650 / PixelClock)) + (25 * (1650 / PixelClock))
```

If exposure time is lower than frame time:

$$FPS\_max = \frac{1}{FrameTime}$$

If exposure time is greater than frame time:

```
FPS_max =            1
          ------------------------
               ExposureTime
```



20.2.3.2.2 Snapshot mode

In snapshot mode, the image acquisition process consists off several sequential phases:

20.2.3.2.3 Trigger

Snapshot mode starts with a trigger. This can be either a hardware or a software signal.

The following trigger modes are available:

Mode	Description
Continuous	Free running, no external trigger signal needed.
OnLowLevel	As long as trigger signal is *Low* camera acquires images with own timing.
OnHighLevel	As long as trigger signal is *High* camera acquires images with own timing.

See also

Using external trigger with CMOS sensors

20.2.3.2.4 Erase, exposure and readout

All pixels are light sensitive at the same period of time. The whole pixel core is reset simultaneously and after the exposure time all pixel values are sampled together on the storage node inside each pixel. The pixel core is read out line-by-line after exposure.

Note

Exposure and read out cycle is carry-out in serial; that causes that no exposure is possible during read out.

The step width for the exposure time is 1 us.

Image data is then shifted out line-by-line and transferred to memory.

To calculate the maximum frames per second (FPS_{max}) in snapshot mode you will need following formula:

```
FrameTime = (ImageHeight * (1650 / PixelClock)) + (25 * (1650 / PixelClock))
```

$$FPS_{max} = \frac{1}{FrameTime + ExposureTime}$$

AOI	PixelClock (MHz)	Exposure Time (us)	Maximal Frame Rate (fps)	PixelFormat
Maximum	40	100	24.6	Mono8
W:1036 x H:776	40	100	30.3	Mono8
W:838 x H:627	40	100	37.1	Mono8
W:678 x H:598	40	100	38.9	Mono8
W:550 x H:484	40	100	47.6	Mono8

20.2.3.3 Measured frame rates

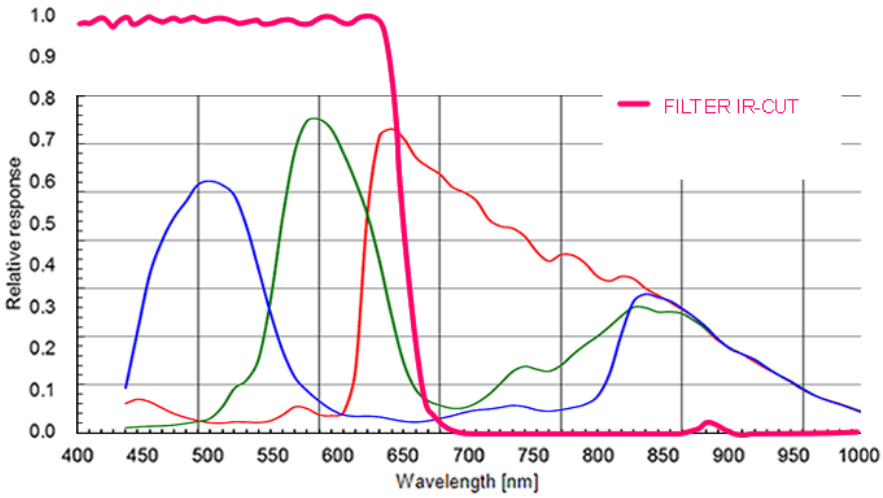
20.2.3.4 Sensor Data

Device Structure

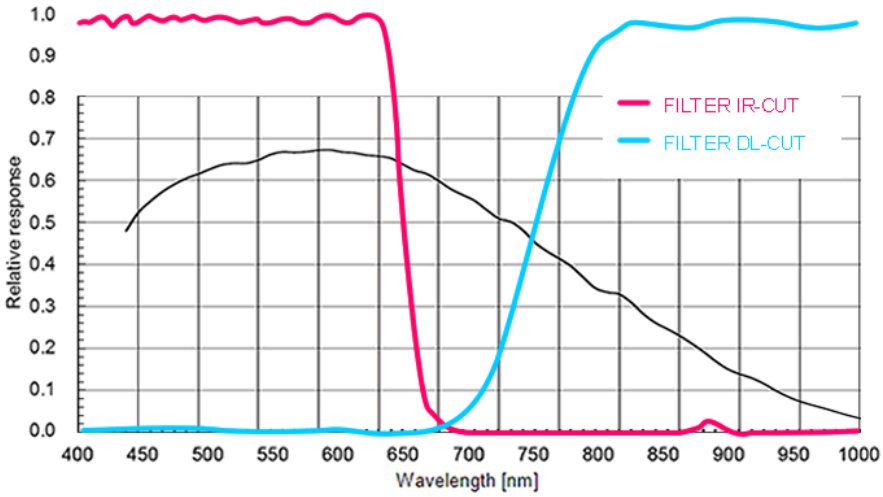
- CMOS image sensor (Type 1/3")
- Number of effective pixels: 1280 (H) x 960 (V)
- Unit cell size: 3.75um (H) x 3.75um (V)

20.2.3.4.1 Characteristics

20.2.3.4.2 Color version



20.2.3.4.3 Gray scale version



Device Feature And Property List

20.2.3.5 Device Feature And Property List

- BVS CA-[MLC|IGC]-0012VG / mvBlueFOX-[MLC|IGC]202vG Features
- BVS CA-[MLC|IGC]-0012VC / mvBlueFOX-[MLC|IGC]202vC Features

20.2.3.5.1 BVS CA-[MLC|IGC]-0012VG / mvBlueFOX-[MLC|IGC]202vG Features

20.2.3.5.2 BVS CA-[MLC|IGC]-0012VC / mvBlueFOX-[MLC|IGC]202vC Features

20.2.4 mvBlueFOX-[Model]202b (1.2 Mpix [1280 x 960])

20.2.4.1 Introduction

The CMOS sensor module (MT9M021) incorporates the following features:

- resolution to 1280 x 960 gray scale or RGB Bayer mosaic
- supports window AOI mode with faster readout
- programmable analog gain (0..12 dB)
- progressive scan sensor (no interlaced problems!)
- pipelined global shutter
- programmable readout timing with free capture windows and partial scan
- many trigger modes (free-running, hardware-triggered)

20.2.4.2 Details of operation

The sensor uses a pipelined global snapshot shutter (`ShutterMode` = "`FrameShutter`") , i.e. light exposure takes place on all pixels in parallel, although subsequent readout is sequential.
Therefore the sensor offers two different modes of operation:

- free running mode (Overlapping exposure and readout)
- snapshot mode (Sequential exposure and readout)

20.2.4.2.1 Free running mode

In free running mode, the sensor reaches its maximum frame rate. This is done by overlapping erase, exposure and readout phase. The sensor timing in free running mode is fixed, so there is no control when to start an acquisition. This mode is used with trigger mode **Continuous**.

To calculate the maximum frames per second (FPS_{max}) in free running mode you will need following formula:

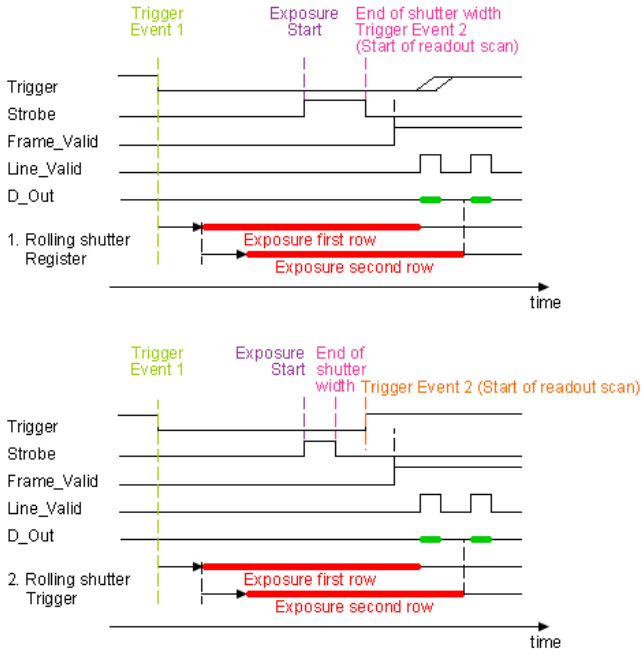
```
FrameTime = (ImageHeight * (1650 / PixelClock)) + (25 * (1650 / PixelClock))
```

If exposure time is lower than frame time:

$$FPS\_max = \frac{1}{FrameTime}$$

If exposure time is greater than frame time:

$$FPS\_max = \frac{1}{ExposureTime}$$



20.2.4.2.2 Snapshot mode

In snapshot mode, the image acquisition process consists off several sequential phases:

20.2.4.2.3 Trigger

Snapshot mode starts with a trigger. This can be either a hardware or a software signal.

The following trigger modes are available:

Mode	Description
Continuous	Free running, no external trigger signal needed.
OnLowLevel	As long as trigger signal is *Low* camera acquires images with own timing.
OnHighLevel	As long as trigger signal is *High* camera acquires images with own timing.

See also

Using external trigger with CMOS sensors

20.2.4.2.4 Erase, exposure and readout

All pixels are light sensitive at the same period of time. The whole pixel core is reset simultaneously and after the exposure time all pixel values are sampled together on the storage node inside each pixel. The pixel core is read out line-by-line after exposure.

Note

Exposure and read out cycle is carry-out in serial; that causes that no exposure is possible during read out.

The step width for the exposure time is 1 us.

Image data is then shifted out line-by-line and transferred to memory.

To calculate the maximum frames per second (FPS_{max}) in snapshot mode you will need following formula:

```
FrameTime = (ImageHeight * (1650 / PixelClock)) + (25 * (1650 / PixelClock))
```

$$FPS\_max = \frac{1}{FrameTime + ExposureTime}$$

AOI	PixelClock (MHz)	Exposure Time (us)	Maximal Frame Rate (fps)	PixelFormat
Maximum	40	100	24.6	Mono8
W:1036 x H:776	40	100	30.3	Mono8
W:838 x H:627	40	100	37.1	Mono8
W:678 x H:598	40	100	38.9	Mono8
W:550 x H:484	40	100	47.6	Mono8

20.2.4.3 Measured frame rates

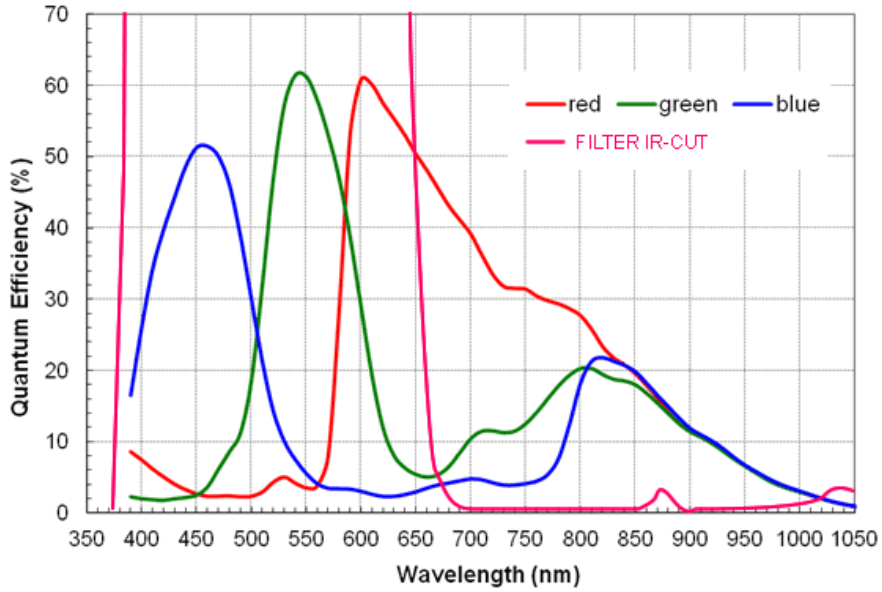
20.2.4.4 Sensor Data

Device Structure

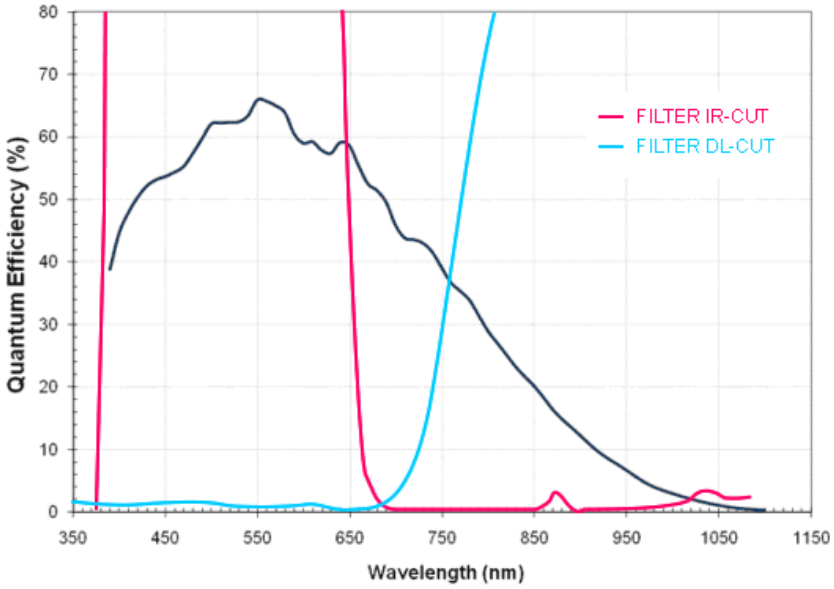
- CMOS image sensor (Type 1/3")
- Number of effective pixels: 1280 (H) x 960 (V)
- Unit cell size: 3.75um (H) x 3.75um (V)

20.2.4.4.1 Characteristics

20.2.4.4.2 Color version



20.2.4.4.3 Gray scale version



Device Feature And Property List

20.2.4.5 Device Feature And Property List

- mvBlueFOX-202bG Features
- mvBlueFOX-202bC Features

20.2.4.5.1 mvBlueFOX-202bG Features

20.2.4.5.2 mvBlueFOX-202bC Features

20.2.5 mvBlueFOX-[Model]202d (1.2 Mpix [1280 x 960])

20.2.5.1 Introduction

The CMOS sensor module (MT9M034) incorporates the following features:

- resolution to 1280 x 960 gray scale or RGB Bayer mosaic
- supports window AOI mode with faster readout
- programmable analog gain (0..12 dB)
- progressive scan sensor (no interlaced problems!)
- high dynamic range 115 dB (with gray scale version)
- rolling shutter
- programmable readout timing with free capture windows and partial scan
- many trigger modes (free-running, hardware-triggered)

20.2.5.2 Details of operation

The sensor uses following acquisition mode:

- **rolling shutter** (`ShutterMode` = `"ElectronicRollingShutter"`)

With the **rolling shutter** the lines are exposed for the same duration, but at a slightly different point in time.

> **Note**
>
> Moving objects together with a rolling shutter can cause a shear in moving objects.

Furthermore, the sensor offers following operating modes:

- free running mode (Overlapping exposure and readout)
- snapshot mode (Sequential exposure and readout)

20.2.5.2.1 Free running mode

In free running mode, the sensor reaches its maximum frame rate. This is done by overlapping erase, exposure and readout phase. The sensor timing in free running mode is fixed, so there is no control when to start an acquisition. This mode is used with trigger mode **Continuous**.

To calculate the maximum frames per second (FPS_{max}) in free running mode you will need following formula:

```
FrameTime = (ImageHeight * (1650 / PixelClock)) + (25 * (1650 / PixelClock))
```

If exposure time is lower than frame time:

$$FPS\_max = \frac{1}{FrameTime}$$

If exposure time is greater than frame time:

$$FPS\_max = \frac{1}{ExposureTime}$$

20.2.5.2.2 Snapshot mode

In snapshot mode, the image acquisition process consists off several sequential phases:

20.2.5.2.3 Trigger

Snapshot mode starts with a trigger. This can be either a hardware or a software signal.

The following trigger modes are available:

Mode	Description
Continuous	Free running, no external trigger signal needed.
OnLowLevel	As long as trigger signal is *Low* camera acquires images with own timing.
OnHighLevel	As long as trigger signal is *High* camera acquires images with own timing.

See also

Using external trigger with CMOS sensors

20.2.5.2.4 Erase, exposure and readout

All pixels are light sensitive at the same period of time. The whole pixel core is reset simultaneously and after the exposure time all pixel values are sampled together on the storage node inside each pixel. The pixel core is read out line-by-line after exposure.

Note

Exposure and read out cycle is carry-out in serial; that causes that no exposure is possible during read out.

The step width for the exposure time is 1 us.

Image data is then shifted out line-by-line and transferred to memory.

To calculate the maximum frames per second (FPS_{max}) in snapshot mode you will need following formula:

```
FrameTime = (ImageHeight * (1650 / PixelClock)) + (25 * (1650 / PixelClock))
```

$$FPS\_max = \frac{1}{FrameTime + ExposureTime}$$

AOI	PixelClock (MHz)	Exposure Time (us)	Maximal Frame Rate (fps)	PixelFormat
Maximum	40	100	24.6	Mono8
W:1036 x H:776	40	100	30.3	Mono8
W:838 x H:627	40	100	37.1	Mono8
W:678 x H:598	40	100	38.9	Mono8
W:550 x H:484	40	100	47.6	Mono8

20.2.5.3 Measured frame rates

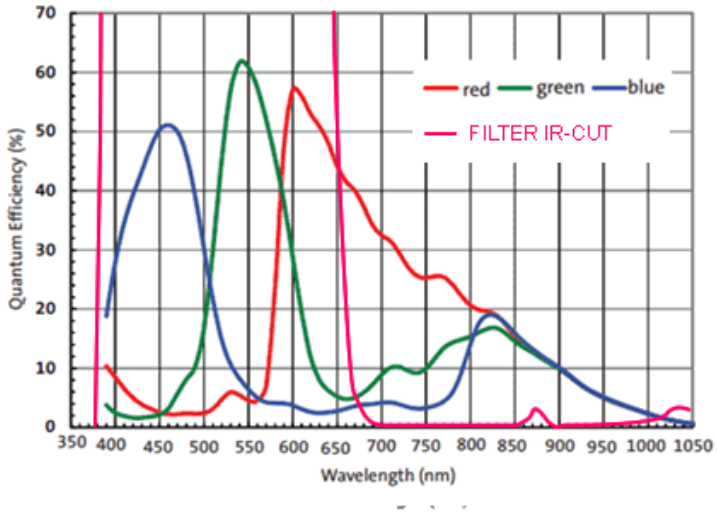
20.2.5.4 Sensor Data

Device Structure

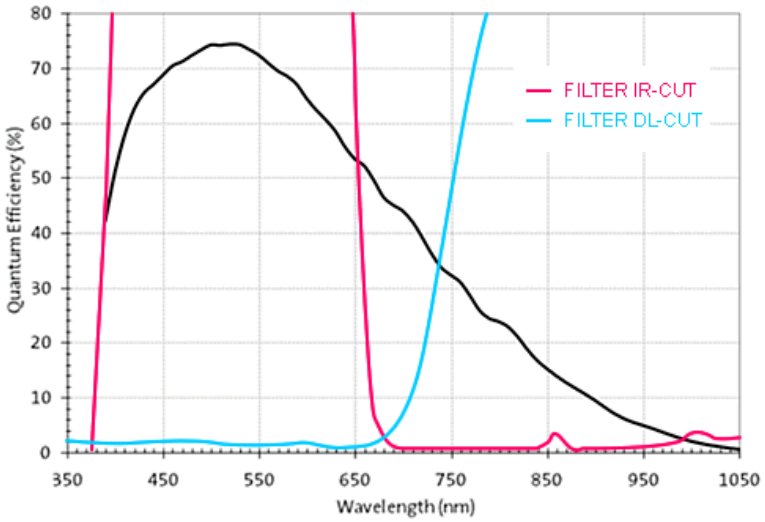
- CMOS image sensor (Type 1/3")
- Number of effective pixels: 1280 (H) x 960 (V)
- Unit cell size: 3.75um (H) x 3.75um (V)

20.2.5.4.1 Characteristics

20.2.5.4.2 Color version



20.2.5.4.3 Gray scale version



Device Feature And Property List

20.2.5.5 Device Feature And Property List

- mvBlueFOX-ML/IGC202dG Features
- mvBlueFOX-ML/IGC202dC Features

20.2.5.5.1 mvBlueFOX-ML/IGC202dG Features

20.2.5.5.2 mvBlueFOX-ML/IGC202dC Features

20.2.6 mvBlueFOX-[Model]205 (5.0 Mpix [2592 x 1944])

20.2.6.1 Introduction

The CMOS sensor module (MT9P031) incorporates the following features:

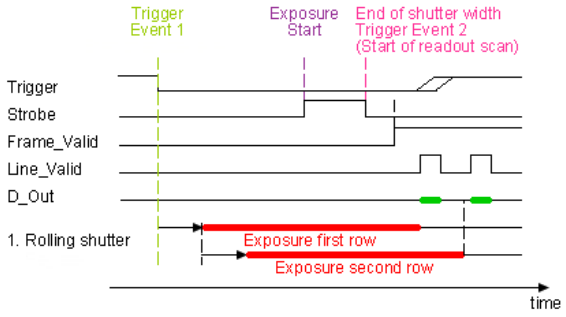
- resolution to 2592 x 1944 gray scale or RGB Bayer mosaic
- supports window AOI mode with faster readout
- programmable analog gain (0..32dB)
- progressive scan sensor (no interlaced problems!)
- rolling shutter / global reset release
- programmable readout timing with free capture windows and partial scan
- many trigger modes (free-running, hardware-triggered)

20.2.6.2 Details of operation

The sensor uses two acquisition modes:

- **rolling shutter** (ShutterMode = "`ElectronicRollingShutter`") and
- **global reset release shutter** (ShutterMode = "`GlobalResetRelease`").

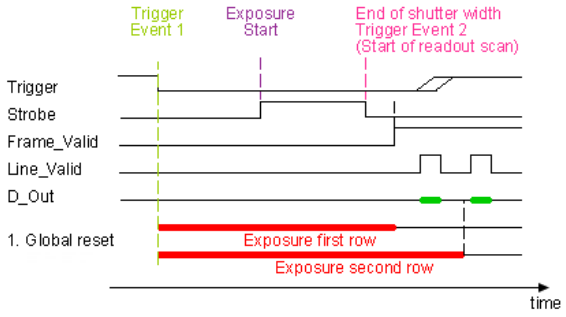
With the **rolling shutter** the lines are exposed for the same duration, but at a slightly different point in time:



Note

Moving objects together with a rolling shutter can cause a shear in moving objects.

The **global reset release shutter**, which is only available in triggered operation, starts the exposure of all rows simultaneously and the reset to each row is released simultaneously, too. However, the readout of the lines is equal to the readout of the rolling shutter: line by line:



Note

This means, the bottom lines of the sensor will be exposed to light longer! For this reason, this mode will only make sense, if there is no extraneous light and the flash duration is shorter or equal to the exposure time.

Furthermore, the sensor offers two operating modes:

- free running mode (Overlapping exposure and readout)
- snapshot mode (Sequential exposure and readout) in triggered operation

20.2.6.2.1 Free running mode

In free running mode, the sensor reaches its maximum frame rate. This is done by overlapping erase, exposure and readout phase. The sensor timing in free running mode is fixed, so there is no control when to start an acquisition. This mode is used with trigger mode **Continuous**.

To calculate the maximum frames per second (FPS_{max}) in free running mode you will need following formula:

```
FrameTime = (ImageWidth + 900) * ((ImageHeight + 9) / PixelClock)
```

If exposure time is lower than frame time:

$$FPS\_max = \frac{1}{FrameTime}$$

If exposure time is greater than frame time:

$$FPS\_max = \frac{1}{ExposureTime}$$

20.2.6.2.2 Snapshot mode

In snapshot mode, the image acquisition process consists off several sequential phases:

20.2.6.2.3 Trigger

Snapshot mode starts with a trigger. This can be either a hardware or a software signal.

The following trigger modes are available:

Mode	Description
Continuous	Free running, no external trigger signal needed.
OnDemand	Image acquisition triggered by command (software trigger).
OnLowLevel	Start an exposure of a frame as long as the trigger input is below the trigger threshold .
OnHighLevel	Start an exposure of a frame as long as the trigger input is above the trigger threshold.
OnHighExpose	Each rising edge of trigger signal acquires one image, exposure time corresponds to pulse width.

See also

Using external trigger with CMOS sensors

20.2.6.2.4 Erase, exposure and readout

All pixels are light sensitive at the same period of time. The whole pixel core is reset simultaneously and after the exposure time all pixel values are sampled together on the storage node inside each pixel. The pixel core is read out line-by-line after exposure.

Note

Exposure and read out cycle is carry-out in serial; that causes that no exposure is possible during read out.

The step width for the exposure time is 1 us.

Image data is then shifted out line-by-line and transferred to memory.

To calculate the maximum frames per second (FPS_{max}) in snapshot mode you will need following formula:

```
FrameTime = (ImageWidth + 900) * ((ImageHeight + 9) / PixelClock)


FPS_max =                  1
          ------------------------------------
               (FrameTime + ExposureTime)
```

20.2.6.2.5 Use Cases

As mentioned before, **"Global reset release"** will only make sense, if a flash is used which is brighter than the ambient light. The settings in ImpactControlCenter will look like this:

In this case, *DigOut0* gets a high signal as long as the exposure time (which is synchronized with the GlobalReset↩ Release). This signal can start a flash light.

AOI	PixelClock (MHz)	Exposure Time (us)	Maximal Frame Rate (fps)	PixelFormat
Maximum	40	100	5.9	Mono8
W:2098 x H:1574	40	100	8.4	Mono8
W:1696 x H:1272	40	100	12.0	Mono8
W:1376 x H:1032	40	100	16.9	Mono8
W:1104 x H:832	40	100	23.7	Mono8
W:800 x H:616	40	100	32	Mono8

20.2.6.3 Measured frame rates

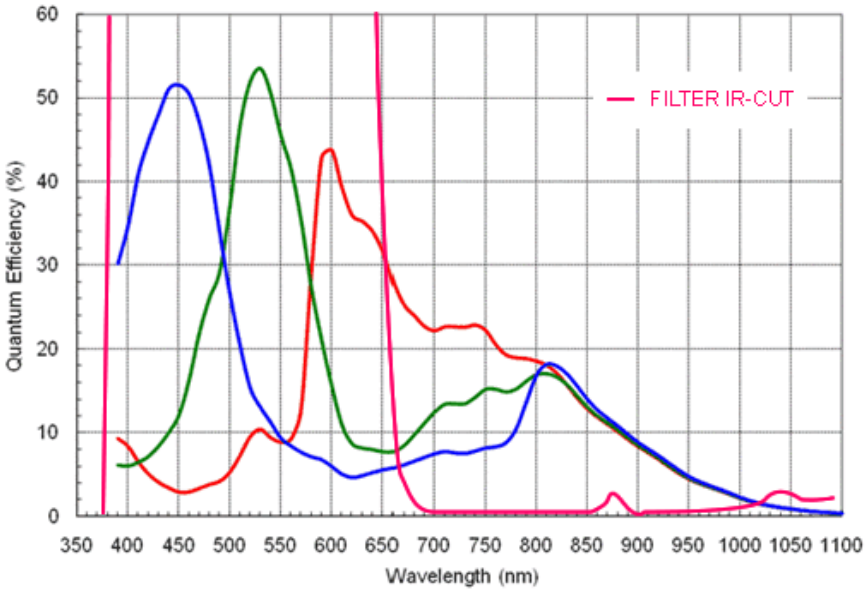
20.2.6.4 Sensor Data

Device Structure

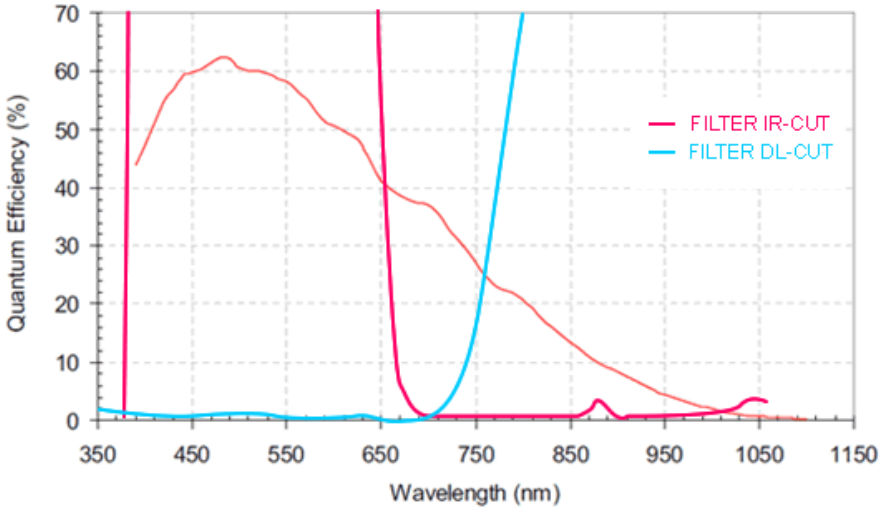
- Progressive scan CMOS image sensor
- Image size: 5.70(H)x4.28(V)mm (Type 1/2.5")
- Number of effective pixels: 2592 (H) x 1944 (V)
- Unit cell size: 2.2um (H) x 2.2um (V)

20.2.6.4.1 Characteristics

20.2.6.4.2 Color version



20.2.6.4.3 Gray scale version



Device Feature And Property List

20.2.6.5 Device Feature And Property List

- mvBlueFOX-205G Features
- mvBlueFOX-205C Features

20.2.6.5.1 mvBlueFOX-205G Features

20.2.6.5.2 mvBlueFOX-205C Features

21 Appendix B. Product Comparison

22 Appendix C. Tested ARM platforms

Balluff/MATRIX VISION devices can run on ARM-based Linux platforms without limitations regarding available feature sets or API functions. However, each platform may have its own limits in terms of achievable data throughput, RAM or bus speeds. Apart from that, each platform may also come with its own specific set of challenges. Therefore, certain modifications may need to be adapted in order to get your devices run at maximum performance.

This chapter contains test results from different ARM platforms, as well as the specific information on each platform, especially changes that need to be applied to achieve better performance.

The following platforms have been tested by Balluff:

System	ARM-Architecture	Technology Test Results					Performance	Suitable for	More information
		USB2.0	USB3.0	GigE	10GigE	PCIe			
NVIDIA Jetson AGX Xavier	NVIDIA Carmel ARMv8.2							Demanding Applications	nvidia.com
NVIDIA Jetson Xavier NX	NVIDIA Carmel ARMv8.2							Demanding Applications	nvidia.com
NVIDIA Jetson Nano	ARM Cortex-A57							Mid-Range Applications	nvidia.com
NVIDIA Jetson TX2	NVIDIA Denver 2 and ARM Cortex-A57							Demanding Applications	nvidia.com
Raspberry Pi 4	ARM Cortex-A72							Price Sensitive Projects	raspberrypi.org
Raspberry Pi Compute Module 4	ARM Cortex-A72							Price Sensitive Projects	raspberrypi.org
i.MX8M Mini	ARM Cortex-A53							Mid-Range Applications	nxp.com

 The system delivers good performance with this device.

 The system doesn't work with this device.

 The developer kit doesn't work with this device because it provides no PCI Express interface.

 The system delivers limited performance with this device.

 The system hasn't been tested yet with this device.

Appendices:

- C.1 ARM64 based devices
- C.2 ARMhf based devices

22.1 C.1 ARM64 based devices

- NVIDIA Jetson AGX Xavier
- NVIDIA Jetson Xavier NX
- NVIDIA Jetson Nano
- NVIDIA Jetson TX2
- Raspberry Pi Compute Module 4
- i.MX8M Mini

22.1.1 NVIDIA Jetson AGX Xavier

CPU	NVIDIA Carmel ARMv8.2 @ 2.26GHz
Cores	8
RAM	32GB
USB2.0 Interfaces	4
USB3.1 Interfaces	3
Ethernet	10/100/1000 MBit
PCIe	1x8 + 1x4 + 1x2 + 2x1 Gen 4.0

22.1.1.1 General

Note

The above table describes the specification of the NVIDIA Jetson AGX Xavier Developer Kit.

The following tests were conducted on JetPack 4.6.0.

22.1.1.2 Test Setup

Test setup

22.1.1.3 Additional Settings Impact Acquire - System Settings

Setting	Value
Request Count	20

Note

A `Request` in the Impact Acquire API represents a buffer where an image with the current device configuration is captured into. In order to avoid losing images at a high FPS, it's recommended to increase the number of these request buffers (i.e. '`RequestCount`' in '`SystemSettings`', by default the value is set to 4 for Balluff USB2.0 cameras), so that the driver can continue capturing image data even if the host application is sometimes slower at processing an image than the camera at transferring one. As a rule of thumb, the number of capture buffers should be configured roughly within the range of FPS/2 and FPS/5. In the following test, the `RequestCount` is set to 20 which is roughly FPS/5.

Camera	Resolution	Pixel Format	Frame Rate [Frames/s]	Bandwidth [MB/s]	CPU Load
mvBlueFOX-IGC200wC	752 x 480	Mono8	93.71	33.88	∼16%

22.1.1.4 Benchmarks

22.1.2 NVIDIA Jetson Xavier NX

CPU	NVIDIA Carmel ARMv8.2 @ 1.9GHz
Cores	6
RAM	8GB
USB3.1 Interfaces	4
Ethernet	10/100/1000 MBit
PCIe	1x1 + 1x4 Gen 3.0

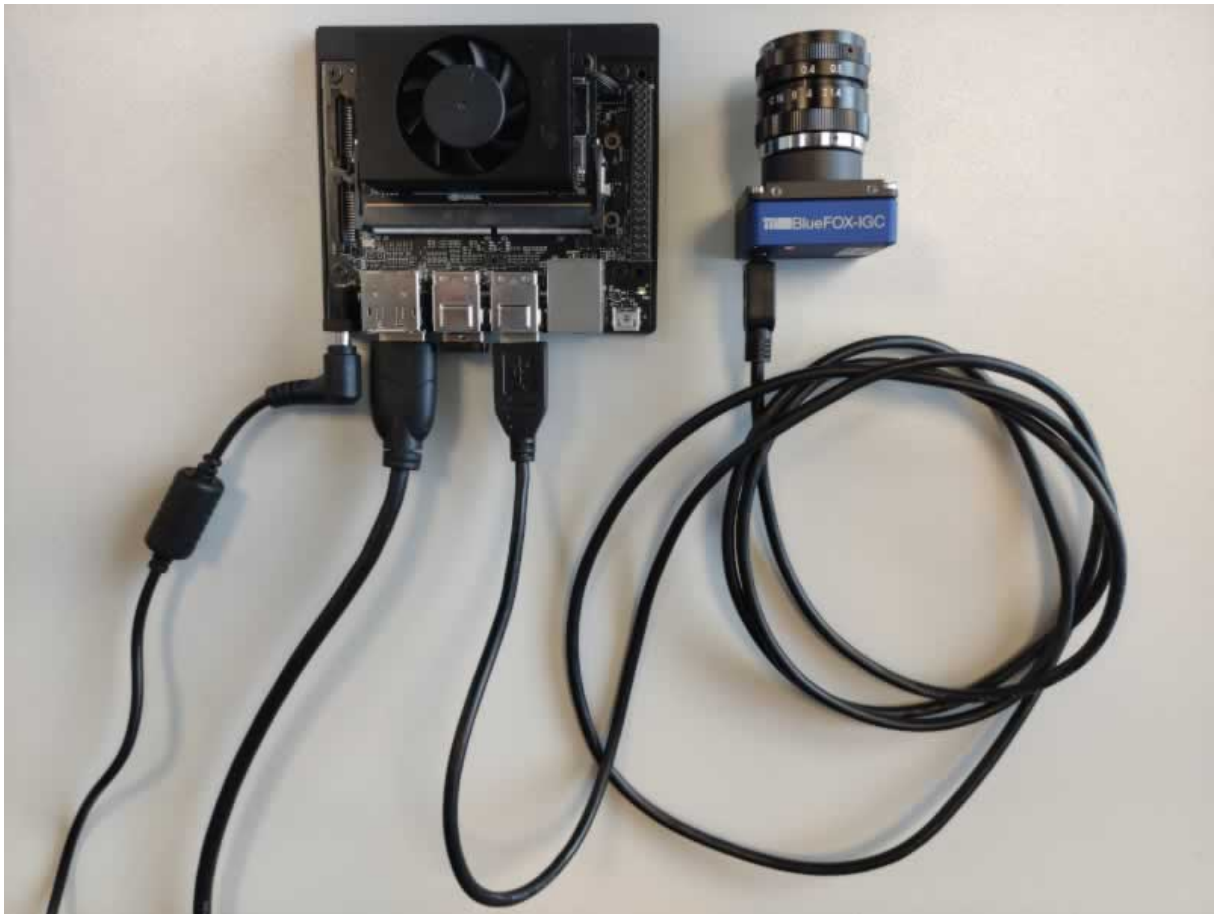
22.1.2.1 General

Note

The above table describes the specification of the NVIDIA Jetson Xavier NX Developer Kit.

The following tests were conducted on JetPack 4.6.0.

22.1.2.2 Test Setup



Test setup

22.1.2.3 Additional Settings Impact Acquire - System Settings

Setting	Value
Request Count	20

Note

A `Request` in the Impact Acquire API represents a buffer where an image with the current device configuration is captured into. In order to avoid losing images at a high FPS, it's recommended to increase the number of these request buffers (i.e. '`RequestCount`' in '`SystemSettings`', by default the value is set to 4 for Balluff USB2.0 cameras), so that the driver can continue capturing image data even if the host application is sometimes slower at processing an image than the camera at transferring one. As a rule of thumb, the number of capture buffers should be configured roughly within the range of FPS/2 and FPS/5. For example, if the capturing FPS is around 100Hz, it is recommended to set the `RequestCount` to 20 which is roughly FPS/5.

Camera	Resolution	Pixel Format	Frame Rate [Frames/s]	Bandwidth [MB/s]	CPU Load
mvBlueFOX-↩ IGC202dC	1280 x 960	BayerRG8 (on camera) -> RGB8 (on host)	24.61	30.26	∼25%

22.1.2.4 Benchmarks

22.1.3 NVIDIA Jetson Nano

CPU	Cortex-A57 @ 1.43 GHz
Cores	4
RAM	4GB
USB2.0 Interfaces	1
USB3.0 Interfaces	4
Ethernet	10/100/1000 MBit
PCIe	x1/x2/x4 Gen 2.0

22.1.3.1 General

Note

The above table describes the specification of the NVIDIA Jetson Nano Developer Kit.

22.1.3.2 Additional Settings Impact Acquire - System Settings

Setting	Value
Request Count	20

Note

A `Request` in the Impact Acquire API represents a buffer where an image with the current device configuration is captured into. In order to avoid losing images at a high FPS, it's recommended to increase the number of these request buffers (i.e. '`RequestCount`' in '`SystemSettings`', by default the value is set to 4 for Balluff USB2.0 cameras), so that the driver can continue capturing image data even if the host application is sometimes slower at processing an image than the camera at transferring one. As a rule of thumb, the number of capture buffers should be configured roughly within the range of FPS/2 and FPS/5. In the following test, the `RequestCount` is set to 20 which is roughly FPS/5.

Camera	Resolution	Pixel Format	Frame Rate [Frames/s]	Bandwidth [MB/s]	CPU Load
mvBlueFOX-IGC200wG	752 x 480	Mono8	93.72	33.88	∼31%

22.1.3.3 Benchmarks

22.1.3.4 Remarks

22.1.3.4.1 Choose the right power supply The Jetson Nano has 2 power supply possibilities: via the micro-USB connection or via the Barrel Jack connection.

The power (by default 10W) via the micro-USB connector is not sufficient if peripherals (e.g. keyboard, mouse, cameras, etc...) are connected. To avoid the system from throttling due to over-current, please supply the board with power through the Barrel Jack connector (4A@5V), when powering the USB/USB3 camera through the USB bus.

22.1.4 NVIDIA Jetson TX2

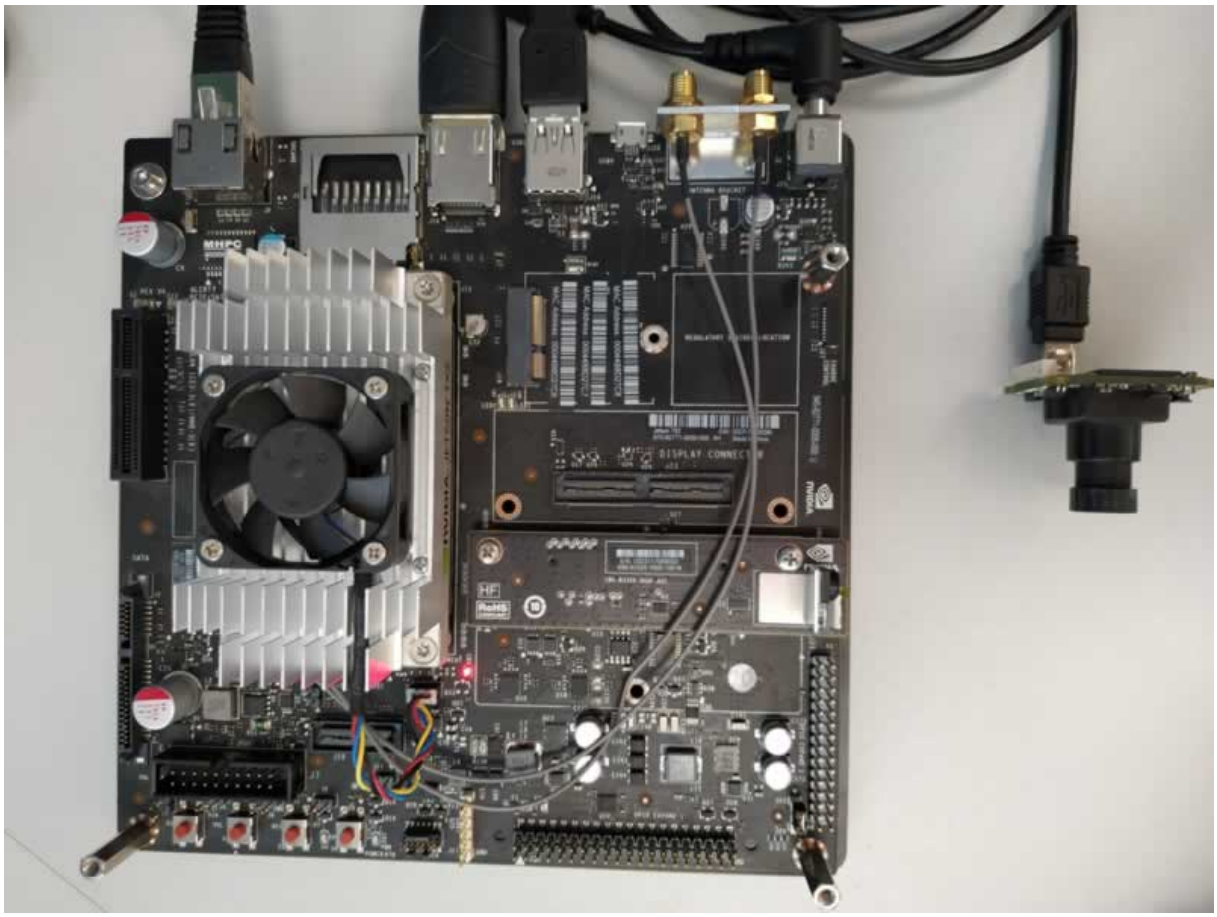
CPU	ARM Cortex-A57 @ 2GHz NVIDIA Denver2 @ 2GHz
Cores	4 2
RAM	8GB
USB2.0 Interfaces	1
USB3.0 Interfaces	1
Ethernet	10/100/1000 MBit
PCIe	1x4 + 1x1 | 2x1 + 1x2 Gen 2.0

22.1.4.1 General

Note

The above table describes the specification of the NVIDIA Jetson TX2 Developer Kit.

22.1.4.2 Test Setup



Test setup

22.1.4.3 Additional Settings Impact Acquire - System Settings

Setting	Value
Request Count	20

Note

A `Request` in the Impact Acquire API represents a buffer where an image with the current device configuration is captured into. In order to avoid losing images at a high FPS, it's recommended to increase the number of these request buffers (i.e. '`RequestCount`' in '`SystemSettings`', by default the value is set to 4 for Balluff USB2.0 cameras), so that the driver can continue capturing image data even if the host application is sometimes slower at processing an image than the camera at transferring one. As a rule of thumb, the number of capture buffers should be configured roughly within the range of FPS/2 and FPS/5. In the following test, the `RequestCount` is set to 20 which is roughly FPS/5.

Camera	Resolution	Pixel Format	Frame Rate [Frames/s]	Bandwidth [MB/s]	CPU Load
mvBlueFOX-IGC200wG	752 x 480	Mono8	94.72	33.88	∼32%

22.1.4.4 Benchmarks

22.1.5 Raspberry Pi Compute Module 4

22.1.5.1 General The Raspberry Pi Compute Module 4 is a well priced platform regarding its performance.

CPU	Cortex-A72 @ 1500MHz
Cores	4
RAM	4 GB
USB2.0 Interfaces	1
Ethernet	10/100/1000 MBit
PCIe	1 x 1 Lane Gen 2.0

Note

The above table describes the specification of the 4GB version Raspberry Pi Compute Module 4.

The Raspberry Pi Compute Module 4 IO Board is used as carrier board for the following test.

Since the Raspberry Pi Compute Module uses the same processor as the Raspberry Pi 4, it is a stripped-down module version of the Raspberry Pi 4. Please refer to Raspberry Pi 4 for benchmark results.

22.1.6 i.MX8M Mini

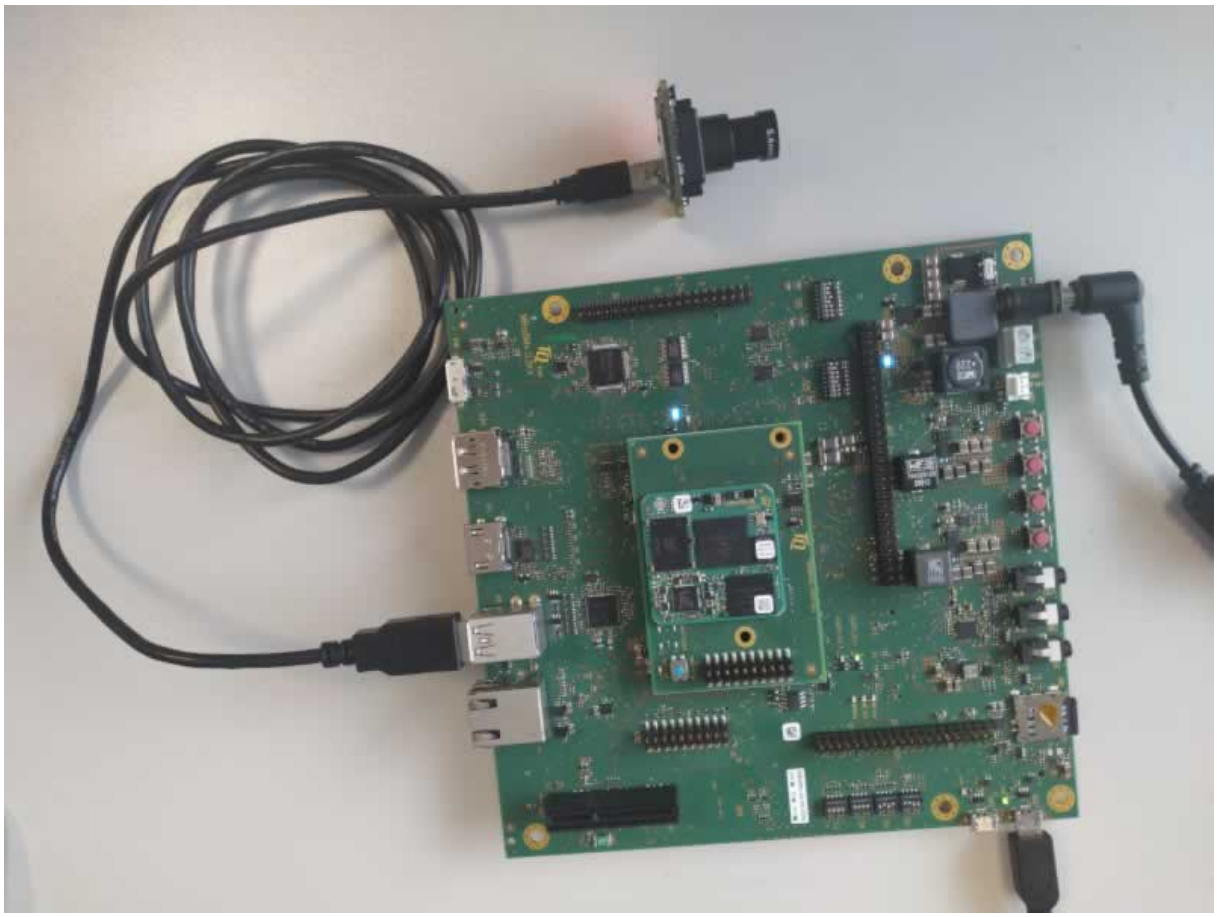
CPU	ARM Cortex®-A53 @ 1.6GHz
Cores	4
RAM	1 GB
USB2.0 Interfaces	2
USB3.0 Interfaces	None
Ethernet	1000 MBit
PCIe	1 x 1 Lane Gen 2.0

22.1.6.1 General The carrier-board used in this test: MBa8Mx from TQ-Systems GmbH

Note

If you are looking for more information and guidance about installing Impact Acquire driver packages via the Yocto Project, please choose an API-manual suited for your programming language and then go to chapter "Installation From Private Setup Routines -> Embedded Linux -> Yocto Project". All API-manuals can be found under https://www.balluff.↩com/en-de/documentation-for-your-balluff-product.

22.1.6.2 Test Setup



Test setup

22.1.6.3 Additional Settings Impact Acquire - System Settings

Setting	Value
Request Count	20

Note

A `Request` in the Impact Acquire API represents a buffer where an image with the current device configuration is captured into. In order to avoid losing images at a high FPS, it's recommended to increase the number of these request buffers (i.e. '`RequestCount`' in '`SystemSettings`', by default the value is set to 4 for Balluff USB2.0 cameras), so that the driver can continue capturing image data even if the host application is sometimes slower at processing an image than the camera at transferring one. As a rule of thumb, the number of capture buffers should be configured roughly within the range of FPS/2 and FPS/5. In the following test, the `RequestCount` is set to 20 which is roughly FPS/5.

Camera	Resolution	Pixel Format	Frame Rate [Frames/s]	Bandwidth [MB/s]	CPU Load (averaged over 4 cores)
mvBlueFOX-↩ MLC200WG	752 x 480	Mono8	93.7	33.8	∼6%

22.1.6.4 Benchmarks

22.2 C.2 ARMhf based devices

- Raspberry Pi 4

22.2.1 Raspberry Pi 4

22.2.1.1 General

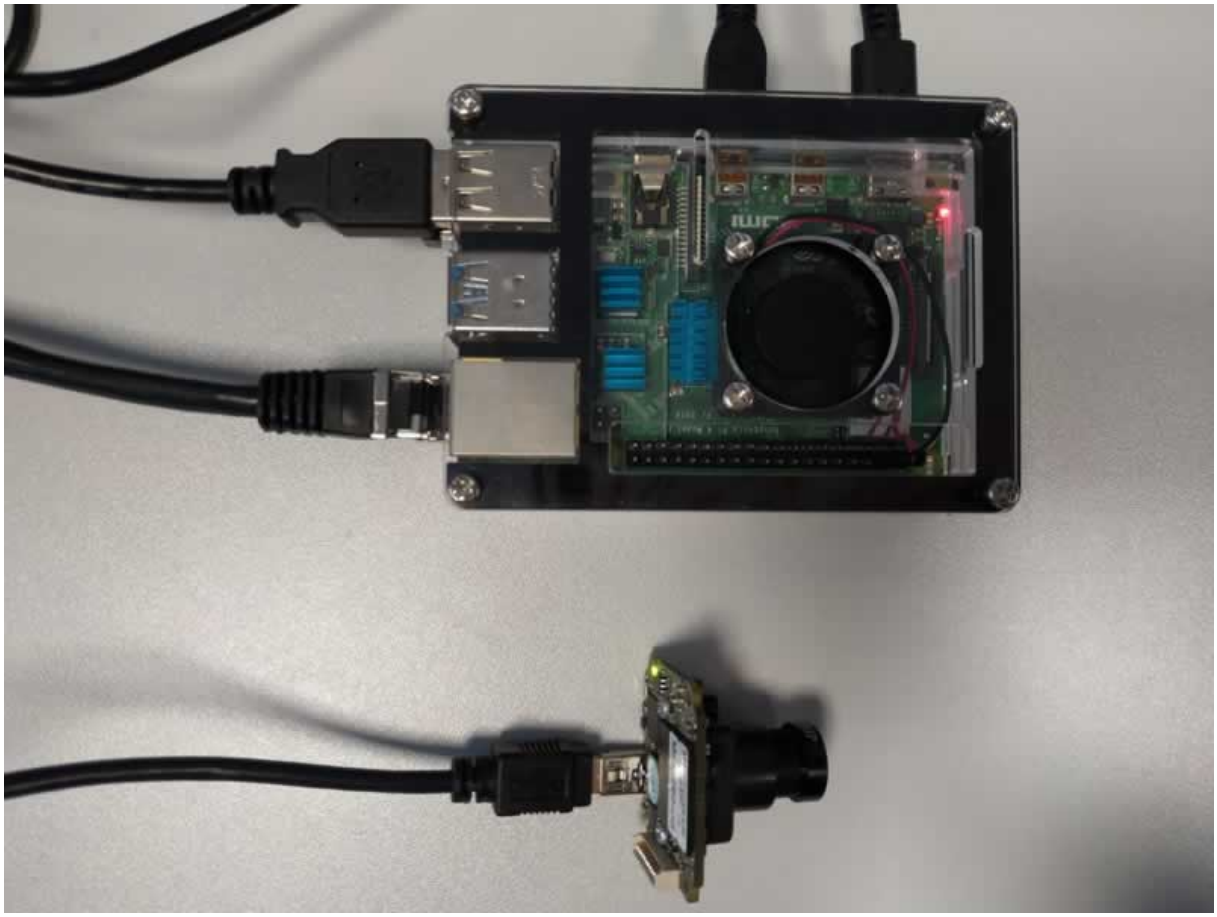
The Raspberry Pi 4 is a well priced platform regarding its performance.

CPU	Cortex-A72 @ 1500MHz
Cores	4
RAM	1/2/4/8 GB
USB2.0 Interfaces	2
USB3.0 Interfaces	2
Ethernet	10/100/1000 MBit

Note

For the following benchmark the 4GB version of the Raspberry Pi 4 with Raspbian OS has been used.

22.2.1.2 Test Setup



Test setup

22.2.1.3 Additional Settings Impact Acquire - System Settings

Setting	Value
Request Count	20

Note

A `Request` in the Impact Acquire API represents a buffer where an image with the current device configuration is captured into. In order to avoid losing images at a high FPS, it's recommended to increase the number of these request buffers (i.e. '`RequestCount`' in '`SystemSettings`', by default the value is set to 4 for Balluff USB2.0 cameras), so that the driver can continue capturing image data even if the host application is sometimes slower at processing an image than the camera at transferring one. As a rule of thumb, the number of capture buffers should be configured roughly within the range of FPS/2 and FPS/5. In the following test, the `RequestCount` is set to 20 which is roughly FPS/5.

Camera	Resolution	Pixel Format	Frame Rate [Frames/s]	Bandwidth [MB/s]	CPU Load
mvBlueFOX-IGC200wG	752 x 480	Mono8	93.72	33.88	∼29%

22.2.1.4 Benchmarks