Basler eXcite

OPERATING MANUAL

Document Number: DA000745
Version: 07   Language: 000 (English)
Release Date: 15 May 2007

Product Discontinued

This Product Has Reached Its End of Life and Will Only Be Available Until the End of 2009

For customers in the U.S.A.

This equipment has been tested and found to comply with the limits for a Class A 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 commercial 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. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

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 J of Part 15 of FCC Rules.

For customers in Canada

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

Pour utilisateurs au Canada

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

Life Support Applications

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Basler customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Basler for any damages resulting from such improper use or sale.

Warranty Note

Do not open the housing of the camera. The warranty becomes void if the housing is opened.

Contacting Basler Support Worldwide

Europe:

Basler AG
Ander Strusbek 60 - 62
22926 Ahrensburg
Germany

Tel.: +49-4102-463-500
Fax.: +49-4102-463-599

vc.support.europe@baslerweb.com

Americas:

Basler, Inc.
855 Springdale Drive, Suite 160
Exton, PA 19341
U.S.A.

Tel.: +1-877-934-8472
Fax.: +1-877-934-7608

vc.support.usa@baslerweb.com

Asia:

Basler Asia Pte Ltd
8 Boon Lay Way,
# 03 - 03 Tradehub 21
Singapore 609964

Tel.: +65-6425-0472
Fax.: +65-6425-0473

vc.support.asia@baslerweb.com

www.basler-vc.com

Table of Contents

1 Specifications, Requirements, & Precautions

1.1 eXcite Models 1-1
1.2 General Specifications 1-2
1.2.1 Camera Section 1-2
1.2.2 Processor Section 1-4
1.2.3 Interface 1-4
1.3 Spectral Response 1-5
1.3.1 CMOS Camera Models 1-5
1.3.2 CCD Camera Models 1-7
1.4 Mechanical Specifications 1-11
1.4.1 Sensor Positioning Accuracy. 1-11
1.4.1.1 Housing with Cooling Fins 1-11
1.4.1.2 Housing with Contact Plate 1-12
1.4.2 Camera Dimensions and Mounting Points 1-13
1.4.2.1 Housing with Cooling Fins 1-13
1.4.2.2 Housing with Contact Plate 1-14
1.4.3 Maximum Lens Thread Length on Color Cameras 1-15
1.4.4 Mechanical Stress Test Results 1-16
1.5 User Requirements 1-17
1.6 Environmental Requirements 1-18
1.6.1 Temperature and Humidity 1-18
1.6.2 Heat Dissipation 1-18
1.7 Precautions 1-19

2 eXcite Hardware and Software Installation

2.1 eXcite Hardware Installation 2-1
2.1.1 Connecting the eXcite 2-1
2.1.1.1 Connecting to a LAN with Static IP Addressing 2-2
2.1.1.2 Connecting to a LAN with Dynamic IP Addressing 2-3
2.1.1.3 Connecting Peer-to-peer 2-4
2.1.1.4 Changing the eXcite’s Default Network Connection Behavior 2-5
2.1.1.5 Finding an eXcite’s Network IP Address 2-10
2.1.1.6 Configuring Network Speed and Duplex Mode 2-12
2.1.1.7 Shutting the Camera Down 2-14
2.2 Running a Simple Functionality Test 2-15
2.3 Installing the IDE, Tool Chain, and Sample Code on a PC with a Linux OS 2-16
2.3.1 Assumptions and Basics 2-16
2.3.2 Installing Eclipse 2-16
2.3.3 Installing the Tool Chain 2-18
2.3.4 Installing the Code Samples 2-19

2.4 Installing the IDE, Tool Chain, and Sample Code on a Windows PC 2-20
2.4.1 Assumptions and Basics 2-20
2.4.2 Installing coLinux, Eclipse, the Tool Chain, and the Code Samples 2-21
2.4.2.1 Disabling Windows Data Execution Prevention 2-29
2.4.3 Installing the VNC Viewer 2-30
2.4.4 Configuring for Dynamic IP Addressing. 2-36
2.4.4.1 Setting the eXcite to Dynamic IP Addressing 2-36
2.4.4.2 Setting coLinux to Dynamic IP Addressing 2-36
2.4.4.3 Setting the Network Bridge to Dynamic IP Addressing 2-38
2.4.4.4 Adjusting the Display ID of the VNC Server 2-39

3 Learning to Use the eXcite

3.1 The Basler eXcite library provides the APIs used to: 3-2
3.2 Building, Loading, & Running a Program 3-3
3.2.1 Building the SimpleGrab Sample Program 3-3
3.2.2 Loading the SimpleGrab Executable onto Your eXcite 3-8
3.2.3 Running the SimpleGrab Executable on Your eXcite 3-9
3.2.4 Building Debug or Release Versions & Building for Different Targets. 3-10
3.3 Debugging the SimpleGrab Application Running on Your eXcite 3-14
3.3.1 Initial Steps 3-14
3.3.2 Using the Gdb Debugger from the Command Line 3-19
3.3.3 Using the Gdb Debugger with Eclipse. 3-20
3.4 Introduction to the Sample Programs 3-23
3.4.1 Assumptions and Basics 3-23
3.4.2 Overview 3-23
3.5 The eXcite Sample Programs 3-25
3.5.1 The “First” Sample Program 3-25
3.5.2 SimpleGrab 3-26
3.5.3 MultiGrab. 3-32
3.5.4 SimpleScalar 3-38
3.5.5 SimpleTrigger 3-40
3.5.6 SimpleDio 3-44
3.5.7 SimpleWatchDog. 3-47
3.5.8 RsReceive and RsSend 3-51
3.5.9 BsReceive and BsSend. 3-58
3.5.10 StreamingClient and StreamingServer 3-64

4 eXcite Interface

4.1 Connections 4-1
4.1.1 General Description. 4-1
4.1.2 Pin Assignments and Numbering 4-2
4.1.3 Connector Types 4-4
4.2 LED Indicators 4-5
4.3 Input Power 4-6

4.4 Ports 4-10
4.4.1 RS-232 Serial Port. 4-10
4.4.2 USB Ports 4-10
4.4.3 Ethernet Port 4-11
4.4.4 Input and Output Ports 4-12
4.4.4.1 Input Ports 4-13
4.4.4.2 Output Ports 4-14
4.4.4.3 I/O Port Cabling 4-16

5 Operation and Features

5.1 Terminology 5-1
5.2 Functional Description 5-1
5.2.1 CMOS Camera Models 5-1
5.2.2 CCD Camera Models 5-4
5.3 Exposure Control 5-6
5.3.1 Controlling Exposure with Shot Parameters Only (No Triggering) 5-6
5.3.2 Controlling Exposure with a Software Trigger 5-7
5.3.3 Controlling Exposure with an ExTrig Signal 5-9
5.3.4 Setting the Exposure Time 5-14
5.3.5 Overlapped and Non-overlapped Exposure 5-15
5.3.6 Trigger Ready Signal. 5-16
5.3.7 Integrate Enabled Signal 5-18
5.3.8 A Recommended Method for Controlling Exposure 5-19
5.3.9 Exposure Timing Charts 5-20
5.3.9.1 CMOS Camera Models 5-20
5.3.9.2 CCD Camera Models 5-21
5.4 Maximum Allowed Frame Capture Rate 5-23
5.4.1 On exA640-180m/c and exA640-120m/c Models 5-23
5.4.2 On exA640-60m/c Models 5-25
5.4.3 On exA1390-19m/c and exA1600-14m/c Models 5-27
5.4.4 What Does the Max Allowed Frame Capture Rate Mean to Me? 5-29
5.4.5 Using the Bandwidth Parameter 5-30
5.5 Image Data Output Formats (on Monochrome Models) 5-31
5.6 Image Data Output Formats (on Color Models) 5-32
5.6.1 The “Bayer” Color Filter 5-32
5.6.2 Output Formats 5-33
5.7 Video Modes 5-36
5.8 Image Information Parameters 5-37
5.9 Standard Features 5-38
5.9.1 Gain and Brightness 5-38
5.9.1.1 CMOS Camera Models 5-38
5.9.1.2 CCD Camera Models 5-41
5.9.2 White Balance (on Color Models) 5-44
5.9.3 Integrated IR Cut Filter (on Color Models). 5-45

5.9.4 Area of Interest (AOI) 5-46
5.9.4.1 Changing AOI Parameters “On-the-Fly” 5-48
5.9.5 RS-232 Serial Port. 5-49
5.9.6 USB Ports 5-54
5.9.7 Ethernet Port 5-54
5.9.8 Input Ports 5-55
5.9.9 Output Ports 5-57
5.9.9.1 Individual Output Port Control 5-57
5.9.9.2 Parallel Output Port Control 5-59
5.9.10 I/O Port Response Times 5-60
5.9.11 The Watchdog Timer. 5-60
5.9.12 Configuration Sets and Memory Channels 5-61
5.9.13 Error Monitoring and Error Indicators 5-63
5.9.13.1 Temperature Sensors 5-63
5.9.13.2 Overtemperature Condition 5-63
5.9.13.3 LEDs 5-63
5.9.13.4 Undervoltage Lockout Condition 5-63
5.9.13.5 Error Flags 5-64
5.10 Smart Features 5-66
5.10.1 What are Smart Features 5-66
5.10.2 Non-reporting Smart Features on the eXcite. 5-66
5.10.2.1 Test Images 5-66
5.10.2.2 Shutter Time Base 5-69
5.10.2.3 Trigger Counter and Trigger Flag 5-70
5.10.2.4 Extended Version Information 5-70

6 Image Data Formats and Ranges

6.1 Image Data Basics 6-1
6.1.1 Pixel Transfer Sequence 6-1
6.2 Image Data Formats 6-2
6.2.1 Data Format with the Camera Set for Mono 8 Output 6-2
6.2.2 Data Format with the Camera Set for Mono 16 Output 6-3
6.2.3 Data Format with the Camera Set for Raw 8 Output. 6-4
6.2.4 Data Format with the Camera Set for Raw 16 Output. 6-5
6.2.5 Data Format with the Camera Set for YUV 4:2:2 Output. 6-6
6.3 Image Data Ranges 6-7
6.3.1 Data Range for a Mono 8 or a Raw 8 Component 6-7
6.3.2 Data Range for a Mono 16 or a Raw 16 Component 6-7
6.3.3 Data Range for a U or a V Component 6-9

7 Troubleshooting and Support

7.1 Technical Support Resources 7-1
7.2 Before Contacting Basler Technical Support 7-2

Appendix A
License Information

A.1 GPL and GenICam Licensing A-1

Revision History i

Feedback iii

Index v

1 Specifications, Requirements, & Precautions

1.1 eXcite Models

The Basler eXcite is available in ten different models. The model depends on the camera’s sensor type, the camera’s maximum frame rate at full resolution and whether the sensor is monochrome or color. Table 1-1 shows the available eXcite models.

The standard eXcite models are equipped with cooling fins for heat dissipation. However, all eXcite models mentioned above are also available with an alternate housing variant (see Section 1.4) where the cooling fins are replaced by a contact plate for optimum contact with a heat sink. The camera models featuring the contact plate are indicated by ”CP“ appended to the camera’s name, e.g. exA640-60m CP.

Throughout this manual, the product will be referred to as the eXcite. Passages that are only valid for a specific model or specific housing variant will be so indicated.

Model	Sensor Type	Mono / Color	Max. Frame Rate (at full resolution with 8 bit output)
exA640-60m	CMOS	Mono	60 frames per second
exA640-60c	CMOS	Color	60 frames per second
exA640-120m	CMOS	Mono	132 frames per second
exA640-120c	CMOS	Color	132 frames per second
exA640-180m	CMOS	Mono	176 frames per second
exA640-180c	CMOS	Color	176 frames per second
exA1390-19m	CCD	Mono	18 frames per second
exA1390-19c	CCD	Color	18 frames per second
exA1600-14m	CCD	Mono	14 frames per second
exA1600-14c	CCD	Color	14 frames per second

Table 1-1: eXcite Models

1.2 General Specifications

1.2.1 Camera Section

Specification	exA640-60m/c	exA640-120m/c	exA640-180m/c
Sensor Type	Micron MT9V403 - 1/2 inch, progressive scan CMOS		
Pixels	Mono models: 656 (H) x 491 (V)		
	Color models: 656 (H) x 490 (V)		
Pixel Size	9.9 µm (H) x 9.9 µm (V)		
Max. Frame Capture Rate			
at 656 x 491 - 8 bits	60 fps	132fps	176 fps
at 640 x 480 - 8 bits	60 fps	135 fps	180 fps
at 320 x 240 - 8 bits	60 fps	269 fps	359 fps
Image Data Output Formats			
Mono Models:	Mono 8 Mono 16 *	Mono 8 Mono 16 *	Mono 8
Color Models:	Raw 8 Raw 16 * YUV 4:2:2 Mono 8 Mono 16 *	Raw 8 Raw 16 * YUV 4:2:2 Mono 8 Mono 16 *	Raw 8 Mono 8
Gain and Brightness	Programmable via the eXcite API		
Exposure Time Control	Programmable via the eXcite API		
Synchronization	External via external trigger signal or via software		
Power Requirements	+12.0 (+/- 0.5) VDC, 14 W ** (typical) @ 12 VDC Input current must not exceed 1.8 A (See Section 4.3 for more information.)		
Lens Adapter	C-mount		
Housing Size (L x W x H)			
Housing with Cooling Fins:	Without lens adapter: 136.5 mm x 55.0 mm x 59.5 mm		
	With C-mount adapter: 144.3 mm x 55.0 mm x 59.5 mm		
Housing with Contact Plate:	Without lens adapter: 136.5 mm x 71.6 mm x 44.0 mm		
	With C-mount adapter: 144.3 mm x 71.6 mm x 44.0 mm		
Weight	Max. 600 g (typical)		
Conformity ***	CE, FCC		

Table 1-2: Camera Section Specifications: CMOS sensors

* 16 bits per pixel are output from the camera with 10 bits effective.

** For operation at max. frame capture rate at full resolution combined with high processor load.

*** For full conformity use a noise suppression choke between the eXcite and the power supply, e.g., model R1405XKS from NKL GmbH.

Specification	exA1390-19m/c	exA1600-14m/c
Sensor Type	Sony ICX267 - 1/2 inch, progressive scan CCD	Sony ICX274 - 1/1.8 inch, progressive scan CCD
Pixels		
Mono models:	1392 (H) x 1040 (V)	1624 (H) x 1236 (V)
Color models:	1388 (H) x 1038 (V)	1624 (H) x 1234 (V)
Pixel Size	4.65 µm (H) x 4.65 µm (V)	4.4 µm (H) x 4.4 µm (V)
Max. Frame Capture Rate	at full resolution - 8 bits: 18.7 fps at full resolution - 16 bits: 16.1 fps at 1280 x 1024 - 8 bits: 18.9 at 1280 x 1024 - 16 bits: 17.7 at 640 x 480 - 8 bits: 35.2 at 640 x 480 - 16 bits: 35.2	at full resolution - 8 bits: 14 fps at full resolution - 16 bits: 11.6 fps at 1280 x 1024 - 8 bits: 16.5 at 1280 x 1024 - 16 bits: 16.5 at 640 x 480 - 8 bits: 31.5 at 640 x 480 - 16 bits: 31.5
Image Data Output Formats		
Mono models:	Mono 8 Mono 16 *	
Color models:	Raw 8 Raw 16 * YUV 4:2:2 Mono 8 Mono 16 *	
Gain and Brightness	Programmable via the eXcite API	
Exposure Time Control	Programmable via the eXcite API	
Synchronization	External via external trigger signal or via software	
Power Requirements	+12.0 (+/- 0.5) VDC, 16 W ** (typical) @ 12 VDC Input current must not exceed 1.8 A (See Section 4.3 for more information.)	
Lens Adapter	C-mount	
Housing Size (L x W x H)		
Housing with Cooling Fins:	Without lens adapter: 136.5 mm x 55.0 mm x 59.5 mm With C-mount adapter: 144.3 mm x 55.0 mm x 59.5 mm	
Housing with Contact Plate:	Without lens adapter: 136.5 mm x 71.6 mm x 44.0 mm With C-mount adapter: 144.3 mm x 71.6 mm x 44.0 mm	
Weight	Max. 600 g (typical)	
Conformity ***	CE, FCC	

Table 1-3: Camera Section Specifications: CCD sensors

* 16 bits per pixel are output from the camera with 12 bits effective.

** For operation at max. frame capture rate at full resolution combined with high processor load.

*** For full conformity use a noise suppression choke between the eXcite and the power supply, e.g., model R1405XKS from NKL GmbH.

1.2.2 Processor Section

Specification	All Cameras
CPU Type	64 Bit - MIPS Processor (PMC Sierra - RM 9000)
CPU Speed	1.0 GHz
RAM	128 MB
Flash Memory	128 MB
Operating System	Linux (Kernel 2.6)

Table 1-4: Processor Section Specifications

1.2.3 Interface

Type	All Cameras
USB	2 x Version 2.0, Type A (USB devices that draw more than 100 mA must not be used with the eXcite.)
LAN	1 x Ethernet 10/100/1000 Mbit
Serial	1 x RS 232, 115 kBaud max.
I/O	4 x Input: opto-isolated, +5.2 to +30 VDC, 1.8 mA @ +30 VDC 4 x Output: opto-isolated, +10 to +30 VDC max. forward voltage, 500 mA max. output current See Section 4.4 for more details.

Table 1-5: Interface Specifications

1.3 Spectral Response

1.3.1 CMOS Camera Models

The spectral response for monochrome CMOS eXcite models is shown in Figure 1-1.

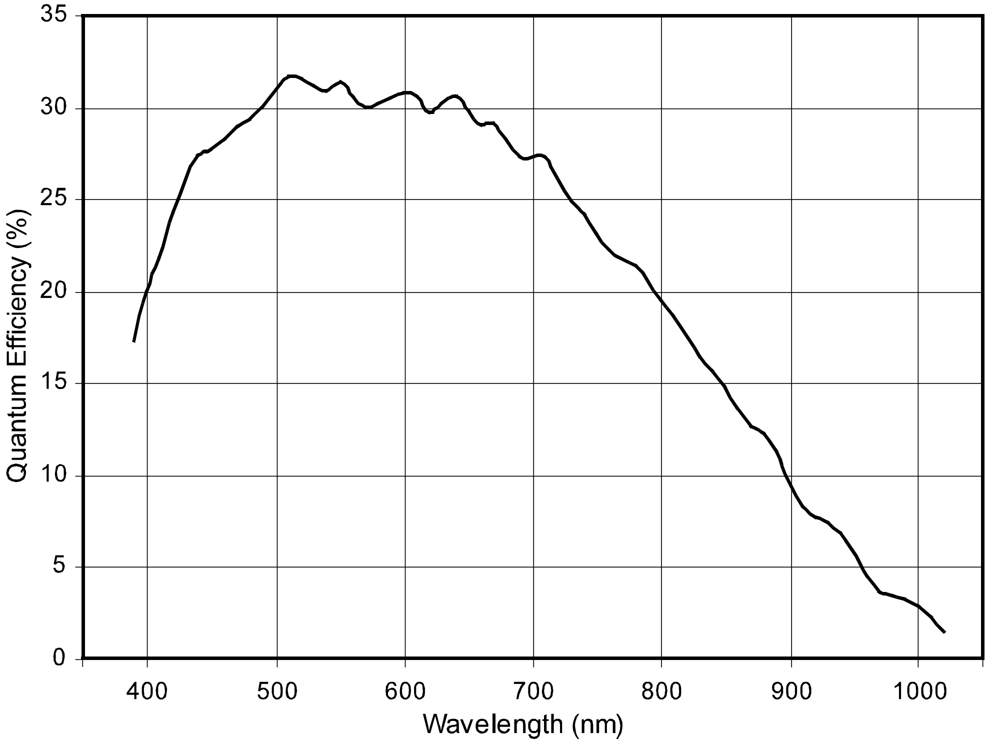


Figure 1-1: Monochrome Spectral Response

ⓘ	The spectral response curve excludes lens characteristics and light source characteristics.

The spectral response for color CMOS eXcite models is shown in Figure 1-2.

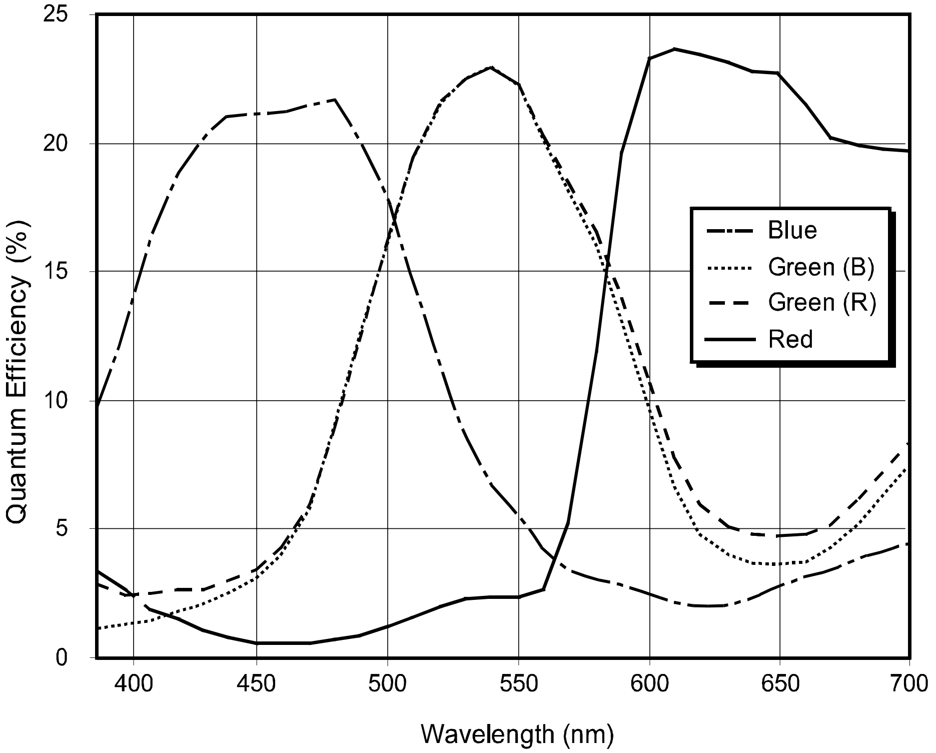


Figure 1-2: Color Spectral Response

ⓘ	The spectral response curves exclude lens characteristics, light source characteristics, and IR cut-off filter characteristics. To obtain the best performance from color models of the eXcite, use of a dielectric IR cut-off filter is recommended. The filter should transmit in a range of 400 nm to 700...720 nm, and it should cut off from 700...720 nm to 1100 nm. A suitable IR cut filter is included in the standard C-mount adapter on color models of the eXcite.

1.3.2 CCD Camera Models

The spectral response for monochrome eXcite model exA1390-19m is shown in Figure 1-3.

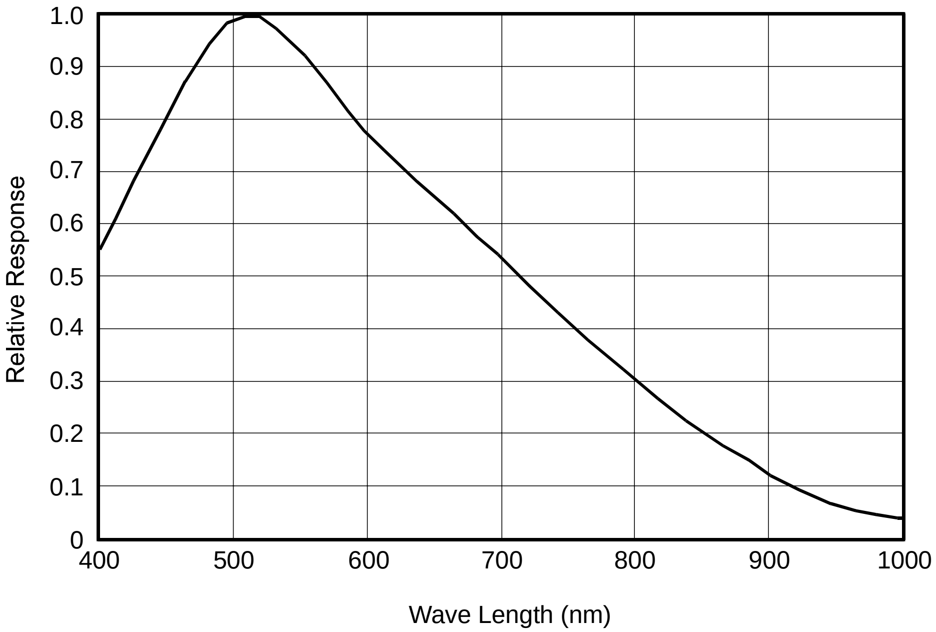


Figure 1-3: Monochrome Spectral Response for exA1390-19m

The spectral response for monochrome eXcite model exA1600-14m is shown in Figure 1-4.

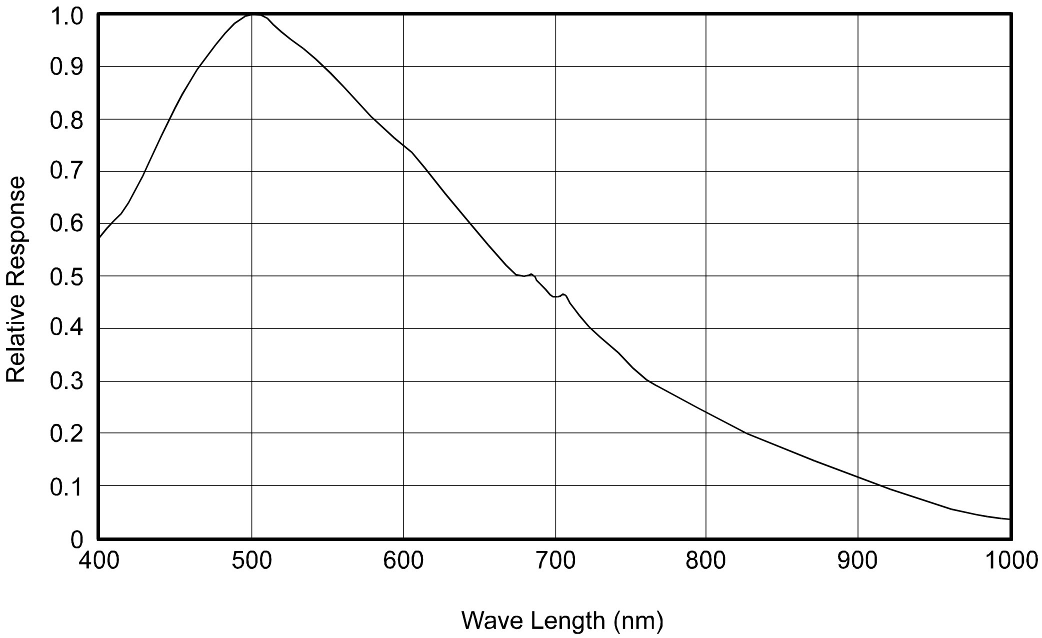


Figure 1-4: Monochrome Spectral Response for exA1600-14m

ⓘ	The spectral response curve excludes lens characteristics and light source characteristics.

The spectral response for color eXcite model exA1390-19c is shown in Figure 1-5.

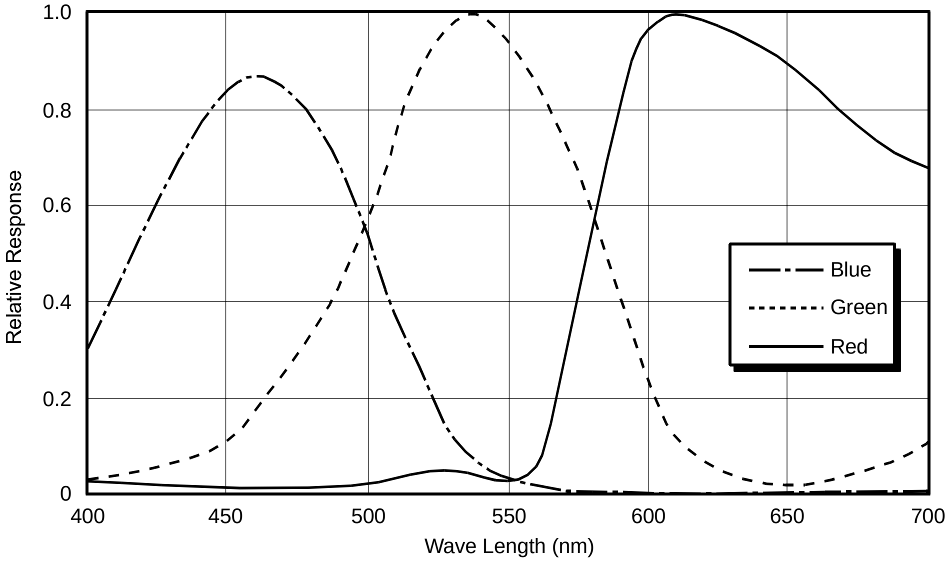


Figure 1-5: Color Spectral Response for exA1390-19c

The spectral response for color eXcite model exA1600-14c is shown in Figure 1-6.

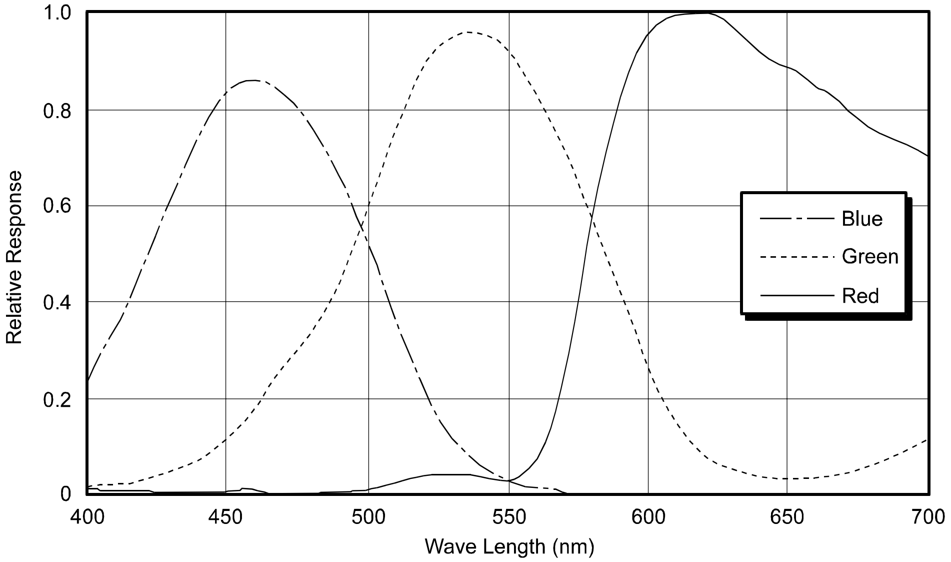


Figure 1-6: Color Spectral Response for exA1600-14c

The spectral response curves exclude lens characteristics, light source characteristics, and IR cut-off filter characteristics.

To obtain the best performance from color models of the eXcite, use of a dielectric IR cut-off filter is recommended. The filter should transmit in a range of 400 nm to 700...720 nm, and it should cut off from 700...720 nm to 1100 nm.

A suitable IR cut filter is included in the standard C-mount adapter on color models of the eXcite.

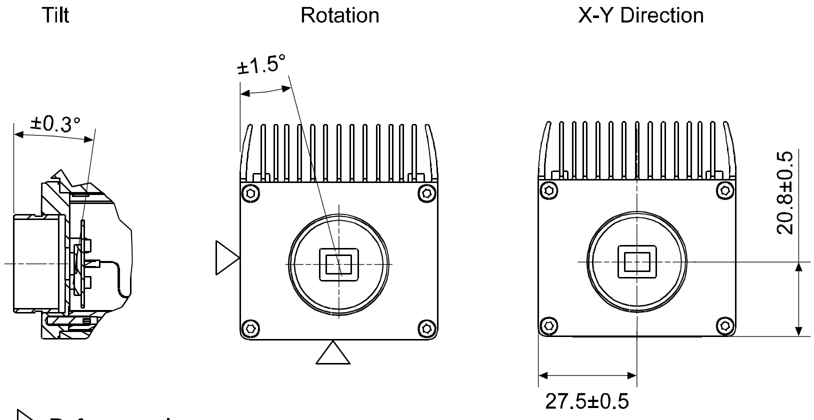
1.4 Mechanical Specifications

The eXcite housing is manufactured with high precision. Planar, parallel, and angular sides guarantee precise mounting with high repeatability.

1.4.1 Sensor Positioning Accuracy

1.4.1.1 Housing with Cooling Fins

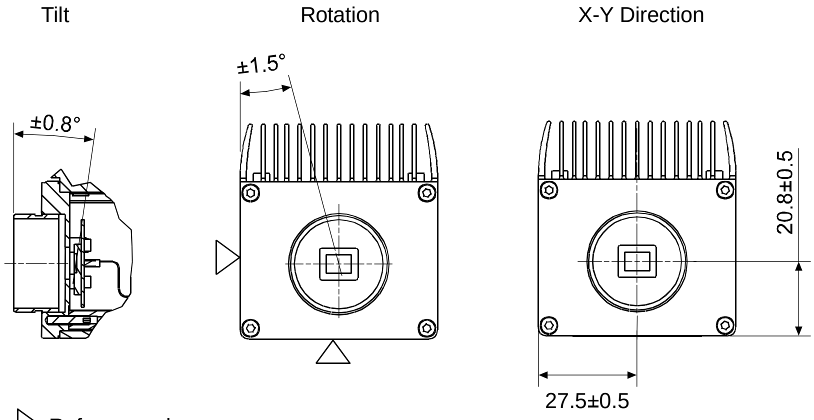
The sensor positioning accuracy in the horizontal and vertical directions is as shown in Figure 1-7 (CMOS sensor) and Figure 1-8 (CCD sensor). Rotational accuracy is also shown in each figure.



▷ Reference planes

Sensor tolerances are typical
Drawings are not to scale

Figure 1-7: CMOS Sensor Positioning Accuracy



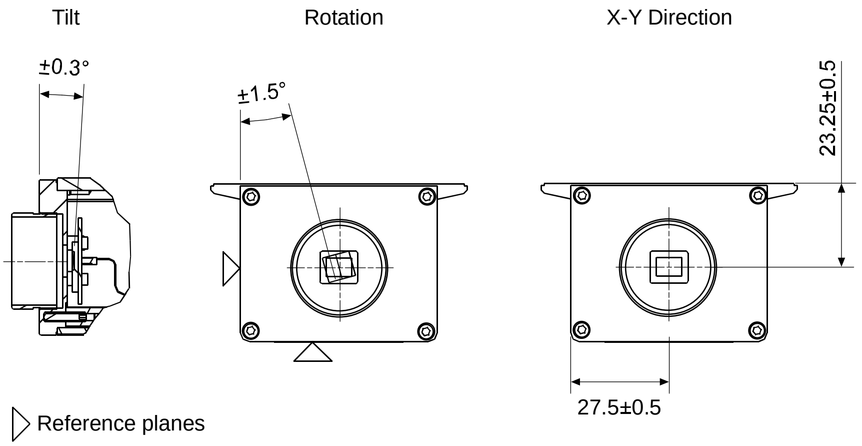
▷ Reference planes

Sensor tolerances are typical
Drawings are not to scale

Figure 1-8:CCD Sensor Positioning Accuracy

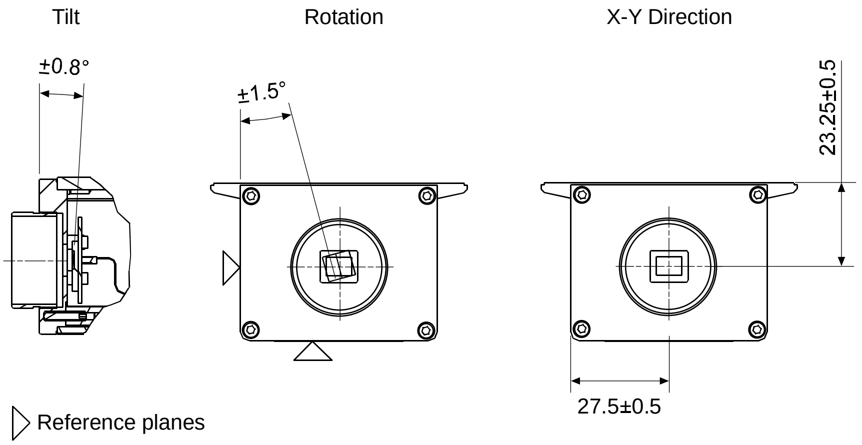
1.4.1.2 Housing with Contact Plate

he sensor positioning accuracy in the horizontal and vertical directions is as shown in Figure 1-9 (CMOS sensor) and Figure 1-10 (CCD sensor). Rotational accuracy is also shown in each figure.



Sensor tolerances are typical
Drawings are not to scale

Figure 1-9: CMOS Sensor Positioning Accuracy



Sensor tolerances are typical
Drawings are not to scale

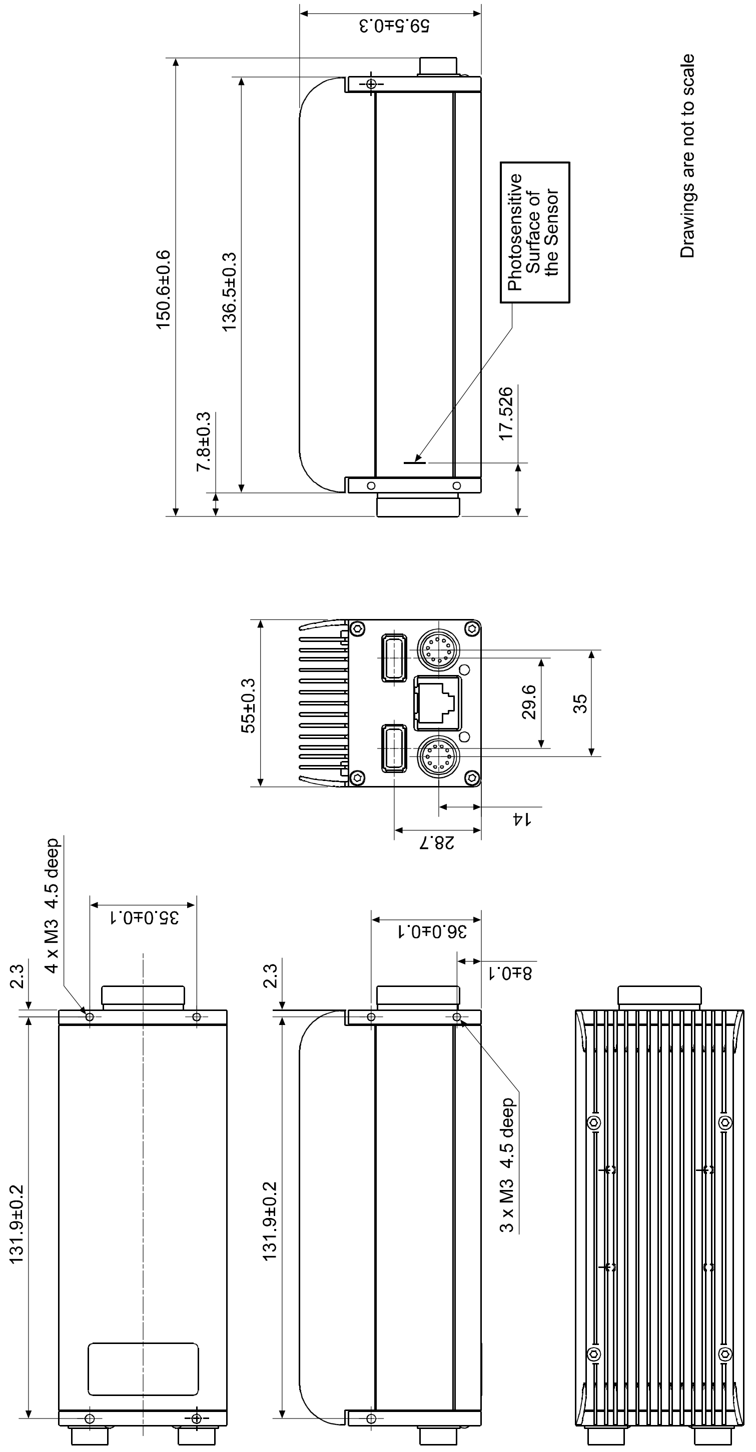
Figure 1-10:CCD Sensor Positioning Accuracy

1.4.2 Camera Dimensions and Mounting Points

The eXcite’s dimensions in millimeters are as shown in Figure 1-11.

1.4.2.1 Housing with Cooling Fins

The housing is equipped with three M3 mounting holes in each side and four M3 mounting holes on the bottom as shown in the drawings.



Drawings are not to scale

Figure 1-11: Mechanical Dimensions (in mm)

1.4.2.2 Housing with Contact Plate

The housing is equipped with three M3 mounting holes in each side, four M3 mounting holes on the bottom, and four mounting holes in the contact plate as shown in the drawings.

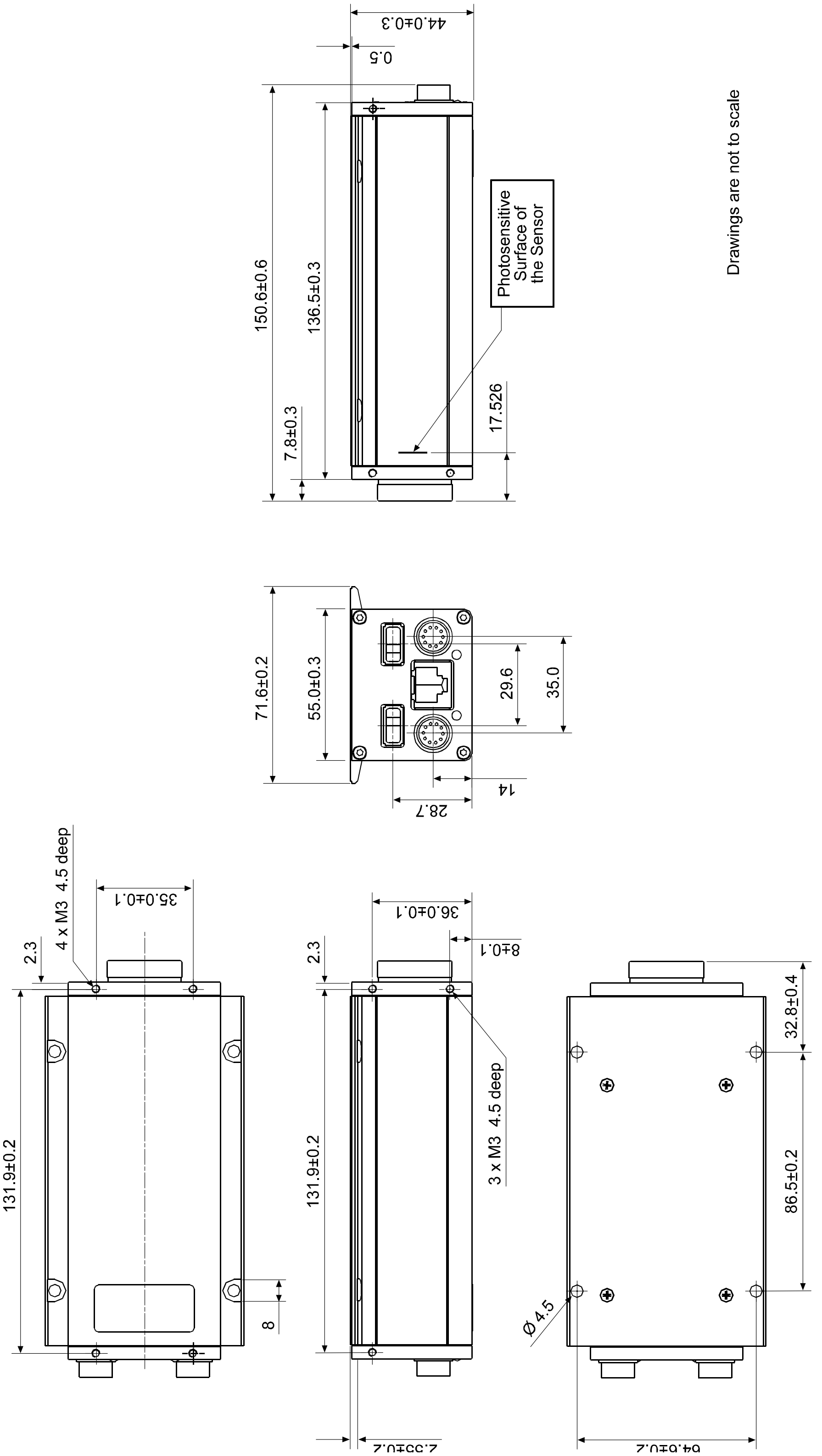


Figure 1-12: Mechanical Dimensions (in mm)

1.4.3 Maximum Lens Thread Length on Color Cameras

The C-mount lens adapter on color models of the eXcite is normally equipped with an internal IR cut filter. As shown in Figure 1-13, the length of the threads on any lens you use with a color eXcite must be less than 7.5 mm. If a lens with a longer thread length is used, the IR cut filter will be damaged or destroyed and the eXcite will no longer operate.

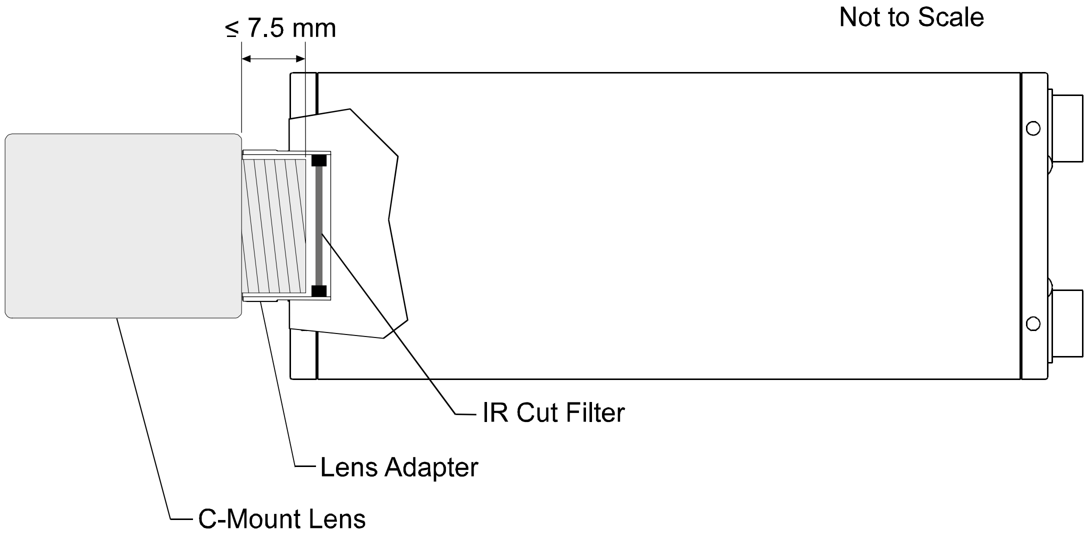


Figure 1-13: Maximum Lens Thread Length on Color Models

1.4.4 Mechanical Stress Test Results

The eXcite was submitted to an independent mechanical testing laboratory and subjected to the stress tests listed below. After mechanical testing, the camera exhibited no detectable physical damage and produced normal images during standard operational testing.

Test	**Standard**	**Conditions**
Vibration (each axis)	IEC 60068-2-6	10-58 Hz / 1.5 mm_58-500 Hz / 20 g_1 Octave/Minute 10 repetitions
Shock (each axis)	IEC 60068-2-27	20 g / 11 ms / 10 shocks positive 20 g / 11 ms / 10 shocks negative
Bump (each axis)	IEC 60068-2-29	20 g / 11 ms / 100 shocks positive 20 g / 11 ms / 100 shocks negative

Table 1-6: Mechanical Tests

1.5 User Requirements

Three user levels – expert, normal, and casual – have been defined for the eXcite. The requirements and the expected performance for each type of user are shown in Table 1-7.

Level	Requirements	Expected Performance
Expert	Extensive C++ programming experience. Highly skilled with the Linux operating system. Extensive experience with industrial digital imaging devices. Extensive experience writing image processing programs for use in a Linux environment.	Will be using the full capabilities of the eXcite and will be writing complex image processing programs after a short learning curve.
Normal	Experience with programming in the C++ language. Experience with the Linux operating system. Experience with industrial digital imaging devices. Some experience with writing image processing programs.	Will require a longer learning curve to become familiar with the full capabilities of the eXcite. Will require more hands-on experience with the eXcite before writing complex image processing programs.
Casual	Experience with programming in the C++ language. Familiar with the Linux operating system or familiar with other operating systems and willing to learn Linux. Familiar with industrial digital imaging devices. Minimal experience with writing image processing programs.	Steep learning curve to become familiar with the basic capabilities of the eXcite. Will require extensive hands-on experience before writing useful image processing programs.

Table 1-7: User Requirements and Performance

1.6 Environmental Requirements

1.6.1 Temperature and Humidity

Housing temperature during operation:	0° C … + 45° C (+ 32° F … + 113° F)
Humidity during operation:	20% … 80%, relative, non-condensing
Storage temperature:	- 20° C ... + 80° C (- 4° F ... + 176° F)
Storage humidity:	20% ... 80%, relative, non-condensing

1.6.2 Heat Dissipation

You must provide sufficient heat dissipation to maintain the temperature of the eXcite housing at 45° C or less. Since each installation is unique, Basler does not specify a strictly required technique for proper heat dissipation. Instead, we provide the following general guidelines:

- In all cases, you should monitor the temperature of the eXcite housing and make sure that the temperature does not exceed 45° C. Keep in mind that the camera will gradually become warmer during its first 1.5 hours of operation. After 1.5 hours, the housing temperature should stabilize and no longer increase.
- If you use the housing variant with cooling fins use of a fan to provide air flow over the cooling fins provides the best heat dissipation. We recommend that you use a fan. (Using a fan for the CP housing variant will provide some but insufficient heat dissipation.)
- If your eXcite is mounted on a large metal component in your system, this may provide sufficient heat dissipation. In this case, we recommend choosing the CP housing variant for optimum contact between the eXcite and the metal component.

ⓘ	The contact plate of the CP housing variant can not act as a heat sink. If the CP housing variant is used, the component the eXcite is mounted on must be able to provide sufficient heat dissipation.

A fan kit is available from Basler. Contact your Basler sales representative for more information.

The eXcite includes a feature that lets you monitor the temperature of the eXcite's processor and core boards. See Section 5.9.13.1 for more information.

The eXcite also includes an overtemperature protection function that will switch off internal power if the temperature of the processor or the core boards is too high. See Section 5.9.13.2 for more information.

1.7 Precautions

Input Voltageinput voltage

⚠

Warning!

For input power, the camera has overvoltage protection up to 30 VDC. An input voltage higher than 30 VDC will cause damage leaving the camera nonoperational.

Heat

⚠

Warning!

If heat is not properly dissipated, the eXcite can get hot enough during operation to cause burning when touched. See Section 1.6.2 for more information about heat dissipation.

Dust

⚠

Caution!

The eXcite is shipped with a cap on the lens mount. To avoid collecting dust on the sensor, make sure that you always put the cap in place when there is no lens mounted on the camera.

Lens Thread Length

⚠

Caution!

Color models of the eXcite are equipped with an IR cut filter mounted in the lens adapter. The location of this filter limits the length of the threads on any lens you use with the eXcite. The thread length on your lens must be less than 7.5 mm. If a lens with a longer thread length is used, the IR cut filter will be damaged or destroyed and the eXcite will no longer operate. See Section 1.4.3 for more information.

Security

⚠

Caution!

The operating system on the eXcite has limited security against malicious attack. If your eXcite is part of a network, you should take appropriate security measures.

Electromagnetic Compatibility

ⓘ	Generally, the eXcite is electromagnetically highly compatible with its electromagnetic environment. Please contact Basler technical support if you observe any EMI effects in an environment particularly sensitive to EMI.

To ensure that your warranty remains in force:

Do not open the eXcite housing

Do not open the housing. Touching internal components may damage them.

Keep foreign matter outside of the camera

Be careful not to allow liquid, flammable, or metallic material inside of the camera housing. If operated with any foreign matter inside, the camera may fail or cause a fire.

Electromagnetic fields

Do not operate the camera in the vicinity of strong electromagnetic fields. Avoid electrostatic charging.

Transporting

Transport the camera in its original packaging only. Do not discard the packaging.

Cleaning

Avoid cleaning the surface of the sensor if possible. If you must clean it, use a soft, lint free cloth dampened with a small quantity of high quality window cleaner. Because electrostatic discharge can damage the sensor, you must use a cloth that will not generate static during cleaning (cotton is a good choice).

To clean the surface of the camera housing, use a soft, dry cloth. To remove severe stains, use a soft cloth dampened with a small quantity of neutral detergent, then wipe dry.

Do not use volatile solvents such as benzine and thinners; they can damage the surface finish.

Read the manual

Read the manual carefully before using the camera!

2 eXcite Hardware and Software Installation

2.1 eXcite Hardware Installation

To complete the eXcite hardware installation you will need:

- an eXcite camera.
- a power supply to supply 12 VDC input power to the eXcite as described in Section 4.3.
- an appropriate network cable as described in Section 4.4.3.

You will also usually need:

- a Y cable for the eXcite as described in Section 4.4.1.
- a null modem cable.
- a development PC with an available serial port connection.

2.1.1 Connecting the eXcite

This procedure assumes that you will be making a network connection between your eXcite and your development PC. Before you begin, you must decide how you will make the network connection between the eXcite and your PC:

- If you want to connect the eXcite to the PC via a LAN that uses static IP addressing, go to Section 2.1.1.1.
- If you want to connect the eXcite to the PC via a LAN that uses dynamic IP addressing (i.e., has a DHCP server), go to Section 2.1.1.2.
- If you want to make a direct (peer-to-peer) connection between the eXcite and your development PC, go to Section 2.1.1.3.

2.1.1.1 Connecting to a LAN with Static IP Addressing

As a default, each eXcite is configured to first make several attempts to connect to a network with dynamic IP addressing (i.e., a network with a DHCP server). If these connection attempts fail, the eXcite then sets itself to a static IP address of 192.168.0.173 and a subnet mask of 255.255.255.0.

Check with your network administrator and make sure that this behavior is acceptable on your LAN:

- Make sure that the attempts to connect to a DHCP server will not cause any problems on your LAN.
- Make sure that the subnet portion of the eXcite's default IP address (192.168.0) is compatible with your network.
- Make sure that the device number part of the eXcite's default IP address (the 173 at the end) is not being used by any other devices.
- Make sure that the subnet mask (255.255.255.0) is OK for use on your network.

If there is a conflict, go to Section 2.1.1.4 and use the procedure described there to make the necessary changes to the default behavior and/or the static IP address. (If you are using multiple eXcites in a static IP network, you will absolutely need to change each eXcite's static IP address.) Once you have made any necessary changes to the default behavior, return here and complete the procedure below.

To start the eXcite and connect to your LAN:

1. If you will be working with the eXcite on a workbench, obtain a small fan that you can use on your bench top. Start the fan and aim it so that it will blow across the top of the eXcite. This will ensure proper heat dissipation.

   If you will be placing the eXcite in a system, be sure to follow the heat dissipation requirements in Section 1.6.2.
2. Connect an Ethernet cable from the RJ-45 (Figure 4-1) jack on the eXcite to your network.
3. Connect the 10-pin plug on the power supply cable to the 10-pin receptacle (Figure 4-1) on the eXcite.
4. Connect the power supply to an AC outlet.
5. Wait about 15 seconds for the eXcite to boot up.
6. From the command line, ping the eXcite's IP address.

   If the ping was successful, go on to Section 2.2.

   If the ping was not successful, double-check your connections and make sure that there are no device conflicts on your network.

2.1.1.2 Connecting to a LAN with Dynamic IP Addressing

As a default, each eXcite is configured to first make several attempts to connect to a network with dynamic IP addressing (i.e., a network with a DHCP server). If these attempts fail, the eXcite then sets itself to a static IP address of 192.168.0.173 and a subnet mask of 255.255.255.0.

Check with your network administrator and make sure that this behavior is acceptable on your LAN:

- Make sure that the eXcite's behavior of setting itself to a fixed address if the DHCP connection attempts fail will not cause any problems on your LAN.

If there is a conflict, go to Section 2.1.1.4 and use the procedure described there to make the necessary changes to the default behavior. Once you have made any necessary changes to the default behavior, return here and complete the procedure below.

To start the eXcite and connect to your LAN:

1. If you will be working with the eXcite on a workbench, obtain a small fan that you can use on your bench top. Start the fan and aim it so that it will blow across the top of the eXcite. This will ensure proper heat dissipation.

   If you will be placing the eXcite in a system, be sure to follow the heat dissipation requirements in Section 1.6.2.
2. Connect an Ethernet cable from the RJ-45 jack (Figure 4-1) on the eXcite to your network.
3. Connect the 10-pin plug on the power supply cable to the 10-pin receptacle (Figure 4-1) on the eXcite.
4. Connect the power supply to an AC outlet.
5. Wait about 15 seconds for the eXcite to boot up.
6. Use the "find_excite" utility described in Section 2.1.1.5 to determine the IP address that was assigned to the eXcite by the DHCP server.
7. From the command line, ping the eXcite's IP address.

   If the ping was successful, go on to Section 2.2.

   If the ping was not successful, double-check that there are no device conflicts on your network.

2.1.1.3 Connecting Peer-to-peer

As a default, each eXcite is configured to first make several attempts to connect to a network with dynamic IP addressing (i.e., a network with a DHCP server). If these attempts fail, the eXcite then sets itself to a static IP address of 192.168.0.173 and a subnet mask of 255.255.255.0. We assume that this behavior is acceptable to you when you are using an eXcite in a peer-to-peer network with a PC.

If you want to change the behavior of the eXcite so that it does not make attempts to connect to a DHCP network or so that it uses a different static IP address, follow the procedure in Section 2.1.1.4. Once you have made any necessary changes to the default behavior, return here and complete the procedure below.

To connect an eXcite directly to a PC (a peer-to-peer connection), the PC must be equipped with a functioning 10/100 MBit Ethernet network adapter or a 1000 MBit Ethernet adapter.

If the PC is equipped with a 10/100 adapter, you **must use an Ethernet cross-over cable** between the eXcite and your PC.

If the PC is equipped with a 1000 MBit adapter, you can use either a straight through Ethernet cable or a cross-over cable.

To make a peer-to-peer connection:

1. Configure the network adapter in your PC so that it has a static IP address of 192.168.0.X (where X is any number between 0 and 255 other than 173). Also configure the adapter with a subnet mask of 255.255.255.0.
2. If you will be working with the eXcite on a workbench, obtain a small fan that you can use on your bench top. Start the fan and aim it so that it will blow across the top of the eXcite. This will ensure proper heat dissipation.

   If you will be placing the eXcite in a system, be sure to follow the heat dissipation requirements in Section 1.6.2.
3. Connect the appropriate Ethernet cable from the RJ-45 jack (Figure 4-1) on the eXcite to the network adapter in the PC.
4. Connect the 10-pin plug on the power supply cable to the 10-pin receptacle (Figure 4-1) on the eXcite.
5. Connect the power supply to an AC outlet.
6. Wait about 15 seconds for the eXcite to boot up.
7. From the command line, ping the eXcite’s IP address.

   If the ping was successful, go on to Section 2.2.

   If the ping was not successful, double-check that the configuration of your network adapter and make sure that you are using the correct type of Ethernet cable.

2.1.1.4 Changing the eXcite’s Default Network Connection Behavior

As a default, each eXcite is configured to first make several attempts to connect to a network with dynamic IP addressing (i.e., a network with a DHCP server). If these attempts fail, the eXcite then sets itself to a static IP address of 192.168.0.173 and a subnet mask of 255.255.255.0.

This section describes how to change this behavior. You can change the behavior in these ways:

- You can change the static IP address and/or the subnet mask that the eXcite will use if its attempts to make a DHCP connection fail.
- You can change the behavior so that the eXcite does not attempt to make a DHCP connection, but simply sets itself for a static IP address and subnet mask of your choice.
- You can change the behavior so that the eXcite attempts to make a DHCP connection, but does not set itself to a static IP address if the DHCP connection attempts fail.

Before you begin the procedure, you should decide how you would like the eXcite to behave.

Changing the network connection behavior is a two part process. The first part is to make an RS-232 serial connection between the eXcite and your development PC. The second part is to edit the startup configuration file on the eXcite.

Establishing an RS-232 Serial Connection between the eXcite and the PC

1. Start a terminal emulation program such as Minicom (Linux) or Hyperterminal (Windows® OS) on your development PC. The emulator should have the following settings:

Bps = 57600	Data bits = 8
Parity = none	Stop bits = 1
Flow control = none	Emulation = VT100

   Com Port = the serial port on your PC that you will use to connect to the eXcite

2. Get your power supply, Y cable, and null modem cable.

   a) If you will be working with the eXcite on a workbench, obtain a small fan that you can use on your bench top. Start the fan and aim it so that it will blow across the top of the eXcite. This will ensure proper heat dissipation.

      If you will be placing the eXcite in a system, be sure to follow the heat dissipation requirements in Section 1.6.2.

   b) Connect the 9-pin D connector on the Y cable to one end of a null modem cable as shown in Figure 2-1.

   c) Connect the other end of the null modem cable to the serial port on your PC.

   d) Connect the 10-pin plug on the Y cable to the 10-pin receptacle on the eXcite.

   e) Connect the 10-pin jack on the Y cable to the 10-pin plug on the power supply.

   f) Connect the power supply to an AC outlet.

3. The eXcite will begin to boot up and you can observe the eXcite processor’s bootup process on the emulator screen.

   When the login prompt appears, the bootup process is complete.

   (If you don’t see the bootup, double check the settings on your terminal emulator and make sure that the emulator is “connected” to the serial port.)

4. Log into the Linux operating system on the eXcite processor (login = root, pwd = root).

   The serial connection is complete. You now have access to the Linux OS and to the file system on the eXcite’s processor.

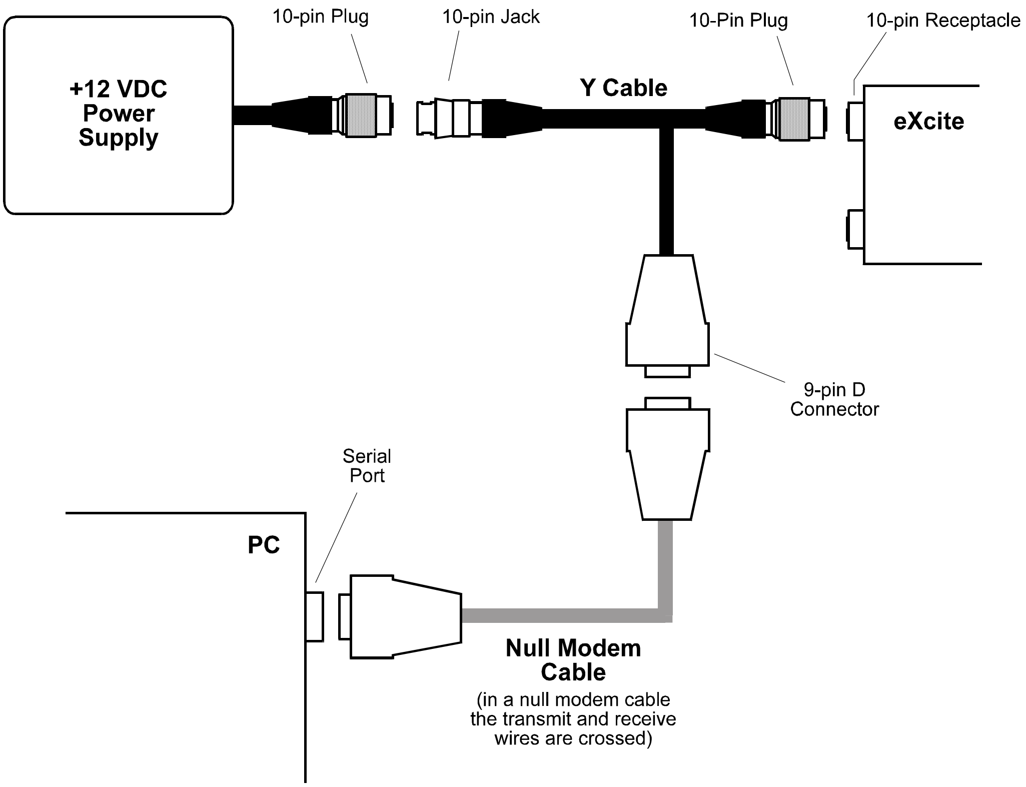


Figure 2-1: Using a Y Cable to Make a Serial Connection to the eXcite

Editing the Startup Configuration File

Note: In this procedure, you will be using the nano editor to edit the eXcite's startup configuration file. A brief introduction to the editor appears on page 2-9 and you can find many introductions to the nano editor on the web.

1. The startup configuration file for the eXcite is called `rcS` and the file is located in the `/etc/init.d` directory on the eXcite.

   Execute the following commands to navigate to this directory and to make a backup copy of the unmodified `rcS` file:

   a) `cd /etc/init.d ↵`

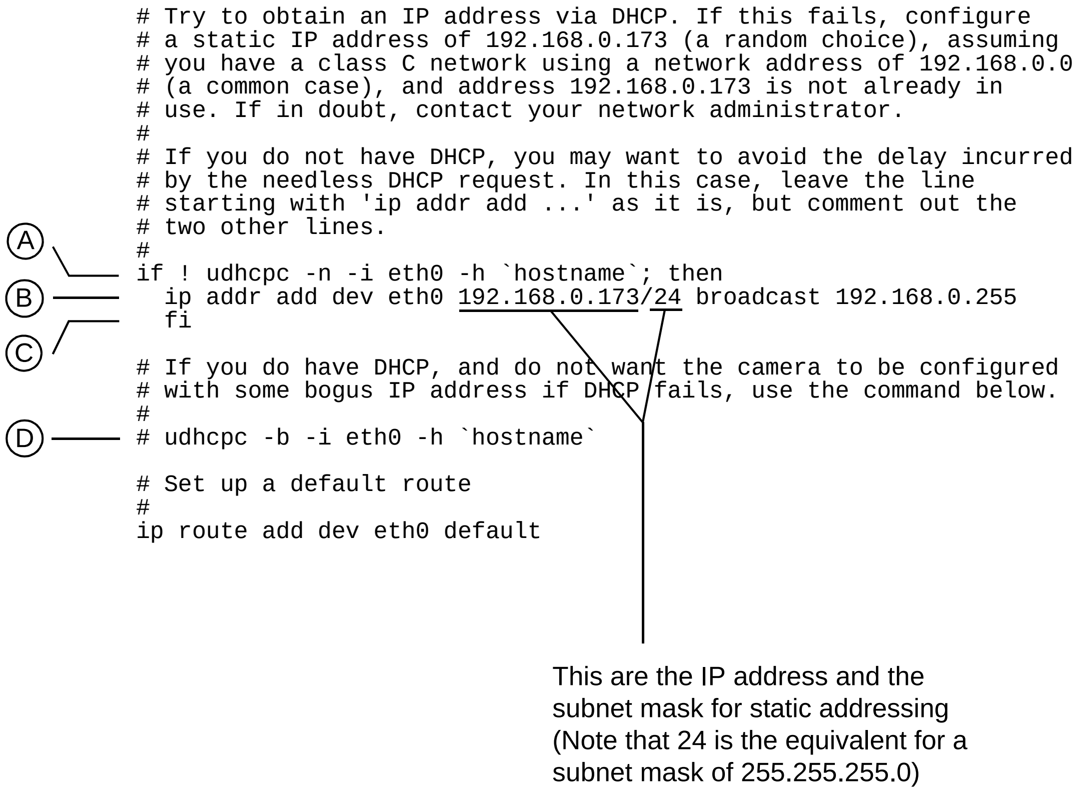
   b) `cp rcS rcS_bu ↵`

2. Execute the following command to open the `rcS` file with the nano editor:

   a) `nano -w rcS ↵`

3. Take a look at Figure 2-2 showing part of the eXcite's startup configuration file rcS. The four lines in the figure marked A, B, C, and D are the lines in the rcS file that must be modified to change network connection behavior. Scroll down until you find those lines in the file. To change the startup behavior, do one of the following:

   - If you want to change the static IP address and the subnet mask that the eXcite will use if its attempts to make a DHCP connection fail, use the editor to modify the IP address and subnet mask entries in line B.
   - If you want to change the startup behavior so that the eXcite does not attempt to make a DHCP connection, but simply sets itself for a static IP address and subnet mask, use the editor to put hash (#) marks in front of lines A and C. You can also change the IP address and subnet mask entries if you desire.
   - If you want to change the behavior so that the eXcite attempts to make a DHCP connection, but does not set itself to a static IP address if the DHCP connection attempts fail, use the editor to put hash marks in front of lines A, B, and C. Also remove the hash mark in front of line D.



```
# Try to obtain an IP address via DHCP. If this fails, configure
# a static IP address of 192.168.0.173 (a random choice), assuming
# you have a class C network using a network address of 192.168.0.0
# (a common case), and address 192.168.0.173 is not already in
# use. If in doubt, contact your network administrator.
#
# If you do not have DHCP, you may want to avoid the delay incurred
# by the needless DHCP request. In this case, leave the line
# starting with 'ip addr add ...' as it is, but comment out the
# two other lines.
#
if ! udhcpc -n -i eth0 -h `hostname`; then
  ip addr add dev eth0 192.168.0.173/24 broadcast 192.168.0.255
  fi

# If you do have DHCP, and do not want the camera to be configured
# with some bogus IP address if DHCP fails, use the command below.
#
# udhcpc -b -i eth0 -h `hostname`

# Set up a default route
#
ip route add dev eth0 default
```

This are the IP address and the subnet mask for static addressing (Note that 24 is the equivalent for a subnet mask of 255.255.255.0)

Figure 2-2: Extract of the Configuration File

When you are finished making your desired changes, press the Control+X keys and when you are asked to save the modified buffer, execute y ↵.

4. You can leave the camera running or you can shut the camera down.

   If you want to leave the camera running excute the following command to boot the eXcite again and to make the changes become effective:

   a) `reboot` ↵

If you want to shut the camera down:

a) Follow the shutdown procedure (Section 2.1.1.7).

b) Disconnect and close the terminal emulation program.

ⓘ	Each hardware device in an Ethernet network – such as an eXcite camera – must have a unique network host name and a unique MAC address. Each eXcite has a unique six digit hexadecimal string stored in the `/proc/excite/unit_id` file. This string is used to form a unique host name and MAC address for the eXcite. For example, assume the contents of the `/proc/excite/unit_id` file is “00001e.” In this case, the MAC address for the eXcite would be “00:30:53:00:00:1e” and the host name would be “eXcite-00001e.”

Some Basics of the nano Editor

The nano editor is a simple-to-use text editor. A copy of the nano editor is included on the eXcite and in coLinux. You can obtain more information about the nano editor e.g. from http://www.nano-editor.org.

Note: The VI editor is supplied as well. However, we recommend to use the nano editor due to its ease uf use.

Here are some useful commands (options) when using the nano editor:

Options

-c constantly show the cursor position

-w disable wrapping of long lines

> ⓘ We strongly recommend to use the -w option to prevent unintentional wrapping of long lines. A file with wrapped lines may not operate properly. In this document, use of the -w option is illustrated in all code samples referring to the nano editor.

Exit and Save Commands

Control+O keys save the modified file

Control+X keys quit nano; you will be asked wheather you want to save the modified file (Y) or not (N). Enter Y and execute ↵ if you want to save the modified file, enter N and execute ↵ if you do not want to save the modified file.

2.1.1.5 Finding an eXcite’s Network IP Address

To follow many of the procedures in this manual, you will need to know the IP address that is currently assigned to your eXcite. If your eXcite is set for dynamic (DHCP) addressing, you usually won’t know exactly what IP address the DHCP server has assigned to the eXcite. If you are working with an eXcite that has a fixed IP address, you may simply not know the address.

There are two ways to find the IP address of an eXcite connected to your PC via an Ethernet network. The first way is to use Basler’s “find_excite” utility and the second way is to use an RS-232 serial connection to the eXcite.

(To get the “find_excite” utility, you will need a copy of the eXcite CD. If you don’t have the CD, you can order one from your sales representative.)

Using the “find_excite” Utility

Using the Utility on a PC with a Linux OS

1. You will find a copy of the “find_excite” utility for on the CD. The utility should be in the `software/linux/utilities` folder. If you have not already done so, copy this utility from the CD to a convenient location on your PC.
2. Before you use the utility, you must know the broadcast address for your network subnet. If you don’t know the broadcast address, execute the following from the command line:

   ```
   ifconfig ↵
   ```

   The system will return information for the Ethernet adapter in your PC and for your loopback connection. If you look at the information for the Ethernet adapter, you will see an entry similar to this:

   ```
   Bcast:172.17.255.255
   ```

   The numbers that appear after “Bcast:” are the broadcast address for you subnet. (If you have more than on Ethernet adapter in your PC, make sure that you get the information for the adapter to which the eXcite is connected.)

   Once you know the broadcast address for your subnet, you can use it whenever you run the “find_excite” utility. (The broadcast address will only change if changes are made to your basic network setup.)
3. To run the “find_excite” utility:
   a) From the command line, navigate to the directory where you copied the utility.
   b) Execute the following command:

      ```
      find_excite [broadcast address] ↵
      ```

      The utility will poll the network and will list the network name and IP address of each eXcite connected to the network.

Using the Utility on a PC with a Windows OS

1. You will find a copy of the “find_excite.exe” utility for on the CD. The utility should be in the `software\windows\utilities` folder. If you have not already done so, copy this utility from the CD to a convenient location on your PC.
2. Before you use the utility, you must know the broadcast address for your network subnet. If you don’t know the broadcast address, open a command prompt window and execute the following command:

   ```
   ipconfig ↵
   ```

   The system will return the IP address and subnet mask for the Ethernet adapter in your PC. You can use this information to determine the broadcast address for your subnet. (If you have more than on Ethernet adapter in your PC, make sure that you get the information for the adapter to which the eXcite is connected.)

   You determine the broadcast address in the following way: for any group of numbers in the subnet mask that shows 255, you use the corresponding group of numbers in the IP address as part of the broadcast address. And for any group of numbers in the subnet mask that shows 0, you use 255 as part of the broadcast address. Take a look at an example below:

IP address returned by the ipconfig command:	172.17.100.6
Subnet mask returned by the ipconfig command:	255.255.0.0
Broadcast address:	172.17.255.255

   Once you know the broadcast address for your subnet, you can use it whenever you run the “find_excite” utility. (The broadcast address will only change if changes are made to your basic network setup.)
3. To run the “find_excite” utility:
   a) From the command line, navigate to the directory where you copied the utility.
   b) Execute the following command:

   ```
   find_excite [broadcast address] ↵
   ```

   The utility will poll the network and will list the network name and IP address of each eXcite connected to the network.

Using a Serial Connection

You can use a serial connection between your eXcite and your PC to find the eXcite’s IP address. To find the IP address with a serial connection:

1. If you have not already done so, follow the “Establishing an RS-232 Serial Connection between the eXcite and the PC” procedure described on page 2-5 to create an RS-232 connection to the eXcite.
2. Use the terminal emulator to log into the eXcite’s OS (login = root, pwd = root).
3. Execute the following command:
   a)
   ```
   ip addr show ↵
   ```

   The IP address will be returned in the emulator window. (The Ethernet IP address is the address shown in the last line returned.)

2.1.1.6 Configuring Network Speed and Duplex Mode

This section describes how to configure the Ethernet adapter of the eXcite to match the Ethernet adapter of the PC (peer-to-peer) or of the network it is connected to. The network speed and duplex mode of the Ethernet adapter of the eXcite must be set identical to the setting for the connected Ethernet adapter.

The settings for the Ethernet adapter of the eXcite are included in the eXcite's startup configuration file `/etc/init.d/rcS` involving the parameters `autoneg`, `speed`, and `duplex_mode`.

Autonegotiation

The `autoneg` parameter enables autonegotiation (`autoneg`=1) and disables autonegotiation (`autoneg`=0).

By default, the Ethernet adapter of the eXcite is set to autonegatiation. If the Ethernet adapter of the connected PC is also set to autonegotiation, the identical network speed and duplex mode are set automatically for the eXcite and the connected PC. In addition, compatibility of the settings are automatically ensured. For example, if the PC is equipped with a 10 MBit Ethernet network adapter, the network speed of the eXcite's Ethernet network adapter will be set to10 MBit/s despite its capability to work at 1000 MBit/s.

If the Ethernet adapter of the connected PC is not set to autonegotiation, autonegotiation must be disabled in the eXcite's startup configuration file `rcS` and network speed and duplex mode must be set for the eXcite's Ethernet adapter exactly as in the PC.

Network Speed

The `speed` parameter sets the network speed to 10 MBit/s (`speed`=0), 100 MBit/s (`speed`=1), or 1000 MBit/s (`speed`=2).

Duplex Mode

The `duplex_mode` parameter sets half duplex mode (`duplex_mode`=0) or full duplex mode (`duplex_mode`=1).

> The eXcite can be operated at half duplex mode. However, for reasons of performance, we strongly recommend using the eXcite in a peer-to-peer connection or in a network that supports full duplex mode.

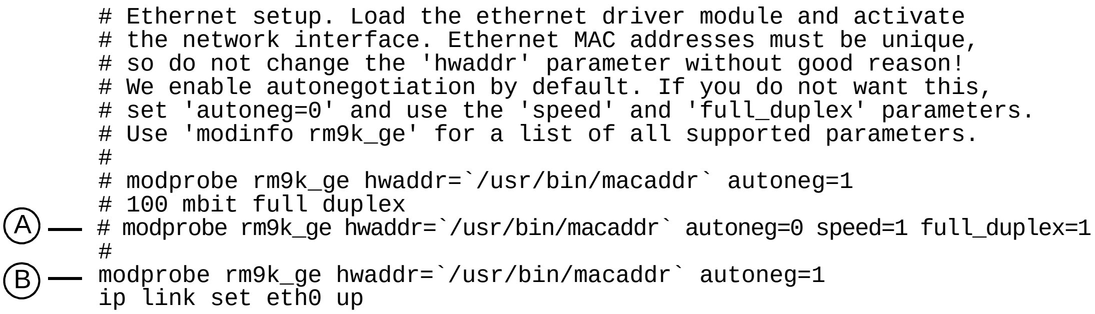
This following procedure assumes that you have determined the settings of the Ethernet adapter of your PC that connects to the eXcite, that the PC and the eXcite are connected, and that you know what settings to make.

Checking settings and configuring the eXcite's startup configuration file `rcS`:

Note: In this procedure, you will be using the nano editor to edit the eXcite's startup configuration file. A brief introduction to the editor appears on page 2-9 and you can find many introductions to the nano editor on the web.

1. Determine the settings of the Ethernet adapter of your PC that connects to the eXcite via peer-to-peer connection or Ethernet network.
2. If you have not already done so, connect the eXcite and the PC as described earlier in Section 2.1.1 and make sure to know the eXcite's IP address (see Section 2.1.1.5).

3. Log on to the eXcite's OS.
4. Execute the following commands to navigate to the `/etc/init.d` directory and to make a backup copy of the unmodified `rcS` file:
   a) `cd /etc/init.d` ↵
   b) `cp rcS rcS_bu` ↵
5. Execute the following command to open the `rcS` file with the nano editor:
   a) `nano -w rcS` ↵
6. Move the cursor to the portion of the code shown below to find the current settings:



```
# Ethernet setup. Load the ethernet driver module and activate
# the network interface. Ethernet MAC addresses must be unique,
# so do not change the 'hwaddr' parameter without good reason!
# We enable autonegotiation by default. If you do not want this,
# set 'autoneg=0' and use the 'speed' and 'full_duplex' parameters.
# Use 'modinfo rm9k_ge' for a list of all supported parameters.
#
# modprobe rm9k_ge hwaddr=`/usr/bin/macaddr` autoneg=1
# 100 mbit full duplex
# modprobe rm9k_ge hwaddr=`/usr/bin/macaddr` autoneg=0 speed=1 full_duplex=1
#
modprobe rm9k_ge hwaddr=`/usr/bin/macaddr` autoneg=1
ip link set eth0 up
```

Figure 2-3: Extract of the eXcite's Configuration File

As is apparent from line B of Figure 2-3, the eXcite's Ethernet adapter is currently set to autonegotiation enabled (`autoneg=1`).

7. If you want autonegotiation on the eXcite:
   a) Make sure line B is uncommented (as shown in Figure 2-3).
   b) Press the Control+X keys to leave the nano editor.
   c) When you are asked to save the modified file, execute:

      `y` ↵

   d) `logout` ↵ (This will log you out of the eXcite's OS and quit telnet.)
8. If you do not want autonegotiation on the eXcite:
   a) Put a hash mark in front of line B (Figure 2-3) to comment it out.
   b) Remove the hash mark in front of line A.
   c) Set the parameters to match the settings of the Ethernet adapter of the PC.
   d) Press the Control+X keys to leave the nano editor.
   e) When you are asked to save the modified file, execute:

      `y` ↵

   f) `logout` ↵ (This will log you out of the eXcite's OS and quit telnet.)

2.1.1.7 Shutting the Camera Down

This procedure assumes that the eXcite is running and that the eXcite’s OS can be accessed from a command prompt window.

Shutting the camera down:

1. From the command line, execute the following command:
   a) `halt` ↵
2. Unplug the eXcite’s power supply.
3. If you no longer need to use the Y cable remove the Y cable connecting the eXcite and the power supply.

2.2 Running a Simple Functionality Test

You can use a network connection to the eXcite to run a simple program that has been pre-installed in the file system on the eXcite's processor. Running the program will test the eXcite's basic functionality, i.e., its ability to grab an image and transmit it to the processor. To run the test, you need to know the IP address of the eXcite you want to connect to. If you do not know the IP address of the eXcite, use one of the methods described in Section 2.1.1.5 to find it.

Once you have the IP address in hand:

1. Put a lens on the eXcite.
2. Set up your lighting as you normally would.
3. Put a target object in the eXcite's field of view.
4. Make a "best guess" adjustment to the lens aperture and focus.
5. On any PC in the same network subnet as the eXcite, execute the following from the command line:

   a) `telnet [eXcite IP address]` ↵

   b) `root` ↵ (This the login for the eXcite's OS.)

   c) `root` ↵ (This is the password for the eXcite's OS.)

   d) `cd /opt/excite/bin` ↵

   e) `./SimpleGrab` ↵

   This will start the "SimpleGrab" program. The program will capture one image and will save it to a file named "image.pgm" in the `/opt/excite/bin` directory.

   f) `logout` ↵ (This will log you out of the eXcite's OS and quit telnet.)

6. Execute the following from the command line:

   a) `ftp [eXcite IP address]` ↵

   b) `root` ↵ (This the login for the eXcite's OS.)

   c) `root` ↵ (This is the password for the eXcite's OS.)

   d) `cd /opt/excite/bin` ↵

   e) `binary` ↵

   f) `get image.pgm` ↵

   g) `bye` ↵ (This will log you out of the eXcite's OS and quit ftp.)

   These actions will copy the file from the eXcite to the current working directory on your PC.

7. You can now use a web browser or an image viewer program to view the image.pgm file that you transferred to the PC.

   You will probably find that the image is out of focus or is too light or too dark. This happens simply because you "best guessed" the aperture and focus settings. If you care too, you can adjust the aperture and focus, repeat steps 5 and 6 to capture another image, and transfer the image to your PC. (If you capture another image, the existing image.pgm file in the eXcite will simply be overwritten.)

Installation and basic testing of the eXcite is now complete. Go on to Section 2.3.

2.3 Installing the IDE, Tool Chain, and Sample Code on a PC with a Linux OS

2.3.1 Assumptions and Basics

We assume that you will be using a system configuration where your programs are written and compiled on a development PC and the compiled executable runs directly on the MIPS processor in the eXcite. We also assume that the development PC has a Linux operating system.

Our recommend Integrated Development Environment (IDE) is “Eclipse.” We assume that you will install Eclipse and the CDT plugin as described below or that Eclipse and the plugin are already installed on your PC.

> ⓘ The procedures in this section were developed on a system with a Red Hat Linux distribution. The procedures can be used with other Linux distributions, but be aware that there may be some differences in file locations.

2.3.2 Installing Eclipse

This section describes installing the Eclipse IDE and the required supporting software on your development PC. The Eclipse software package includes:

- The Eclipse IDE (version 3.0.x for Linux)
- A CDT plug-in (version 2.x, compatible with Eclipse 3.0)
- A script to start Eclipse
- The Java Runtime Environment (version 1.5)

The installation files for this software are located a the `software/linux/` directory on the eXcite CD. (If you don’t have the CD, you can order one from your sales representative.)

To install Eclipse and Its Supporting Software:

1. Locate the “eclipse-platformXXX-gtk.zip” file in the `software/linux/` directory on the CD. (XXX = the version designation)

   This zip file is the installation package for the Eclipse software.

2. Execute the following commands to unpack the Eclipse zip from the CD to the `/opt` directory on your PC:

   a) `cd /opt ↵`

   b) `unzip [pathname]/eclipse-platformXXX-gtk.zip ↵`

      (XXX is version number of the actual file on the CD and [pathname] is the absolute path name to the file on the CD.)

   These actions will create an `/opt/eclipse` subdirectory and will install the Eclipse IDE there.

3. Locate the “org.eclipse.cdt-2.XXX-linux.x86.zip” file in the `software/linux/` directory on the CD.

   This zip is the installer for the Eclipse CDT plug-in.

4. Execute the following commands to unpack the plug-in from the CD to the `/opt` directory on your PC:

   a) `cd /opt ↵`

   b) `unzip [pathname]/org.eclipse-CDT-2.XXX-linux.x86.zip ↵`

      (XXX is version number of the actual file on the CD and [pathname] is the absolute path name to the file on the CD.)

   These actions will unpack the CDT plug-in and make the plug-in active.

5. Locate the “eclipse” file in the `software/linux/` directory on the CD.

   This is a UNIX shell script that should be used to start the Eclipse IDE. This script sets some parameters and then starts Eclipse.

6. Execute the following commands to copy the script from the CD to the `/usr/bin` directory on your PC:

   a) `cp [pathname]/eclipse  /usr/bin ↵`

      (Where [pathname] is the absolute path name to the file on the CD.)

   b) `chmod a+x /usr/bin/eclipse ↵`

   These actions will copy the script and set the “executable” flag for the script file.

7. Execute the following command to check if the correct version of the Java Runtime Engine is installed on your PC:

   a) `ls -l /usr/bin/java ↵`

   If this command returns the information that “/usr/bin/java” is a link to “jre-1.5.XXX”, the correct version of the runtime engine is installed. Go on to step 10.

   If the command returns an indication that “usr/bin/java” is a link to “gij-XXX”, you must install a newer version of the runtime engine. Go to step 8.

8. Locate the “jre-XXX.rpm” file in the `software/linux/` directory on the CD.

   This is an installation package for the Java Runtime Environment V1.5.

9. Execute the following commands to install the JRE on your PC:

   a) `rpm -i [pathname]/jre-XXX.rpm ↵`

      (XXX is version number of the actual file on the CD and [pathname] is the absolute path name to the file on the CD.)

   b) `rm /usr/bin/java ↵`

      (This removes the existing symbolic link to the “gij” file.)

   c) `ln -s /usr/java/jre1.5XXX/bin/java /usr/bin/java ↵`

      (XXX is version number of the actual file on the CD. This creates a symbolic link to the new “jre” file.)

10. (Optional) Create a desk top shortcut to start Eclipse:

    a) Open your file browser and navigate to the `usr/bin` directory.

    b) Drag and drop the Eclipse file onto the desktop to create the shortcut, for example, with the Gnome windowing system in Red Hat Linux, you press the Control + Shift keys and then you click on the Eclipse file and drag it to the desk top.

To start Eclipse, simply double-click the Eclipse desktop shortcut and select “Run in terminal” from the message box that appears.

If you did not create a shortcut, you can always start Eclipse by and typing in: `eclipse ↵` from the command line. This will start the eclipse script installed in steps 5 and 6.)

2.3.3 Installing the Tool Chain

This section describes installing the Basler tool chain on your development PC. The tool chain package includes:

- the compiler and linker programs necessary for Eclipse to “cross-compile” your programs.
- the Basler library files.

The installation files for this software are located in the `software/linux/` directory on the eXcite CD. (If you don’t have the CD, you can order one from your sales representative.)

To Install the Tool Chain:

1. Locate the “excite-toolchain-XXX.tar.bz2” file in the `software/linux/` directory on the CD. (XXX = the version designation.)

   This bz2 file contains the tool chain.

2. Execute the following commands to unpack the tool chain file from the CD to the `/opt` directory on your PC:

   a) `cd /opt ↵`

   b) `tar -xjvf [pathname]/excite-toolchain-XXX.tar.bz2 ↵`

      (Where XXX is version number of the actual file on the CD and [pathname] is the absolute path name to the file on the CD.)

   These actions will create a subdirectory structure in the `/opt` directory and will install the tool chain there.

3. Create an external executable search path by adding the following lines to the end of the “profile” file in the `/etc` directory on your PC:

   ```
   PATH=$PATH:/opt/excite-tools/bin
   export PATH
   ```

4. To make sure that the path modification became effective, restart your computer, and execute the following command:

   ```
   echo $PATH ↵
   ```

   When you press the Enter key, the search paths will be displayed.
   You should now see `/opt/excite-tools/bin` included in the paths.

2.3.4 Installing the Code Samples

This section describes installing the Basler code samples on your development PC. The files for the code samples are located in the `software/samples/` directory on the eXcite CD. (If you don’t have the CD, you can order one from your sales representative.)

To install the sample code:

1. Locate the “samples.tar.bz2” file in the `software/linux/` directory on the CD.

   This bz2 file contains the code samples.

2. Execute the following commands to unpack the samples to your PC:

   a) `cd /[pathname]` ↵

      (Where [pathname] is the path to the directory on your PC where you want to install the code samples.

   b) `tar -xjvf [pathname]/samples.tar.bz2` ↵

      (Where [pathname] is the absolute path name to the file on the CD.)

   These actions will create a subdirectory structure on your PC and will install the code samples there.

2.4 Installing the IDE, Tool Chain, and Sample Code on a Windows PC

2.4.1 Assumptions and Basics

We assume that you will be using a system configuration where your programs are written and compiled on a development PC and the compiled executable runs directly on the MIPS processor in the eXcite. We also assume that the development PC has a Windows XP operating system and at least 512 MB of RAM.

The Integrated Development Environment (IDE) and the tool chain that we have selected for writing and cross-compiling programs for the eXcite is designed to run in a Linux environment. So the first part of installation procedure involves installing “coLinux” on your Windows PC. When installing coLinux, the IDE and the tool chain will be installed automatically along with coLinux.

coLinux is a port of the Linux operating system that runs cooperatively alongside the Windows OS on your PC. With coLinux installed, you will have a complete Linux OS environment available on your Windows PC. You can then run the IDE and tool chain in this environment to develop and compile programs for the eXcite.

The next part of the installation procedure involves extracting two Zip files to a location on your hard drive. One of the extracted files serves as a root partition for the coLinux OS. The other extracted file contains configuration information.

The final part of the procedure installs a graphical front end for use with coLinux.

Our recommend Integrated Development Environment (IDE) is “Eclipse”. A working version of Eclipse, along with all necessary compilers and tools, is part of the file system extracted during the coLinux installation procedure. So there is no need to install Eclipse separately. A copy of the eXcite sample programs is also included in the extracted files.

> Your development PC must have a Windows XP operating system and at least 512 MB of RAM.
>
> You will need approximately 2.5 GigaBytes of free space on your hard drive to install coLinux.

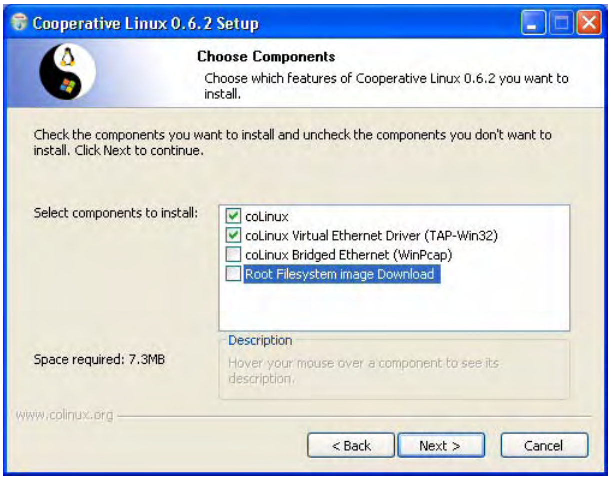
2.4.2 Installing coLinux, Eclipse, the Tool Chain, and the Code Samples

This part of the installation procedure assumes that your development PC has a network adapter card, that your eXcite is installed and has power, and that you are able to communicate with the eXcite either via a LAN or a peer-to-peer Ethernet connection (see Section 2.1). It also assumes that you have WinZip or an equivalent utility installed on your PC.

The installation files for the coLinux software are located in the `software\windows\` directory on the eXcite CD. (If you don't have the CD, you can order one from your sales representative.)

To install coLinux, Eclipse, the tool chain, and the sample code:

1. Make sure that all of the programs on your PC are closed.
2. Click Start and click Run.
   a) Click the Browse button, navigate to the `software\windows\` directory on the CD, and find the "coLinux-XXX.exe" file (XXX = the version designation).
   b) Click on the file, click Open, and click OK.
3. An install wizard will open and a Welcome window will appear.
   a) Click Next.
4. A License Agreement window will appear.
   a) Click I Agree to accept the license.
5. A Choose Components window will appear.
   a) Make sure that "coLinux" and "colinux Virtual Ethernet Driver" are checked as shown below.
   b) Make sure that "coLinux Bridged Ethernet" and "Root Filesystem image download" **are not** checked as shown below.
   c) Click Next.



Cooperative Linux 0.6.2 Setup

Choose Components

Choose which features of Cooperative Linux 0.6.2 you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select components to install:

☑ coLinux
☑ coLinux Virtual Ethernet Driver (TAP-Win32)
□ coLinux Bridged Ethernet (WinPcap)
□ Root Filesystem image Download

Description

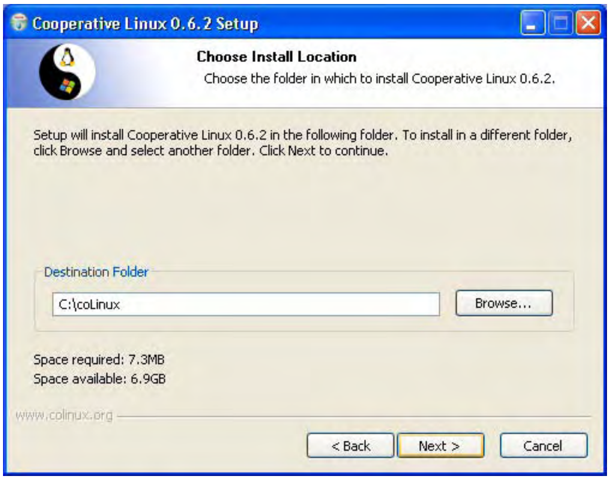
Hover your mouse over a component to see its description.

Space required: 7.3MB

6. A Choose Install Location window will appear.
   a) We strongly suggest that you change the destination folder to: C:\coLinux
   as shown below.

   (You are free to choose any location you want. But if you choose a different location, you must remember to change the coLinux configuration file. This will be covered in a later step.)

   b) When the location is correct, click Next.



Cooperative Linux 0.6.2 Setup

Choose Install Location

Choose the folder in which to install Cooperative Linux 0.6.2.

Setup will install Cooperative Linux 0.6.2 in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

C:\coLinux

Space required: 7.3MB

Space available: 6.9GB

7. A Get WinPCAP window will appear.
   a) WinPCAP is not needed for our installation. Ignore the message in this window and click Next.
8. An Installing window will appear showing the progress of the installation and then a Message window will appear stating that “The software you are installing has not passed Windows Logo testing”.
   a) Click Continue Anyway.
9. A Completing window will appear.
   a) Click finish.
10. Open Windows Explorer.
   a) Navigate to the `software\windows\` directory on the CD.
   b) Find the file called “Debian-XXX.zip” (where XXX is version number).
   c) Use WinZip to extract the contents of this file into the directory where you installed coLinux. (If you followed our suggestion in step 6, it would be the C:\coLinux directory.)

   The files in this zip are large and will take several minutes to extract.

11. In Windows Explorer, navigate to the folder where you installed coLinux. Find the file called “debian.xml.” Open this file with a plain text editor such as Notepad and do the following:

    a) Find this line in the file:

    ```
    <block_device index="0" path="\DosDevices\c:\coLinux\Debian_fs" enabled="true" />
    ```

    If you installed coLinux in a directory other than c:\coLinux, modify the path statement in this line to reflect the actual installation directory path.

    b) Find this line in the file:

    ```
    <memory size="256" />
    ```

    This line sets the amount of memory that will be available to coLinux. The memory size must be set to at least 256. If you have a large amount of RAM on your PC, you should consider increasing the memory size. Increasing the memory size can improve the performance of the Eclipse IDE. (As a general rule, the memory size should be set to use half of the available RAM.)

    c) Save the modified file.

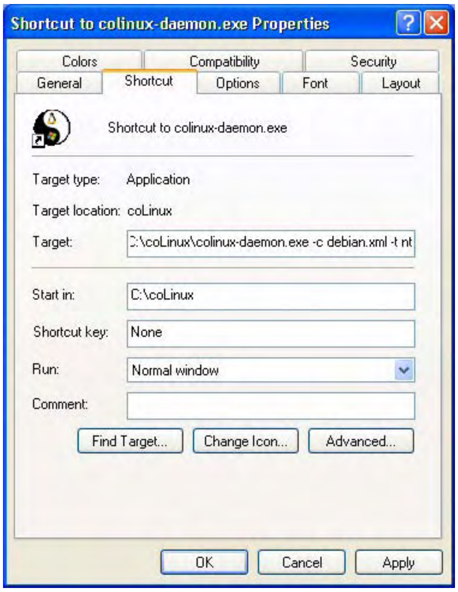
12. In Windows Explorer, navigate to the folder where you installed coLinux and find the file called colinux-daemon.exe. Create a shortcut to this file on your desktop.

13. Right click on the desktop shortcut and select Properties from the drop down menu.

    a) Add the following to the Target line: -c debian.xml -t nt

    When you finish modifying the Target line, it should look similar to the target line shown below.

    b) Click Apply and click OK.



Shortcut to colinux-daemon.exe Properties

Shortcut to colinux-daemon.exe

Target type: Application

Target location: coLinux

Target: :\coLinux\colinux-daemon.exe -c debian.xml -t nt

Start in: C:\coLinux

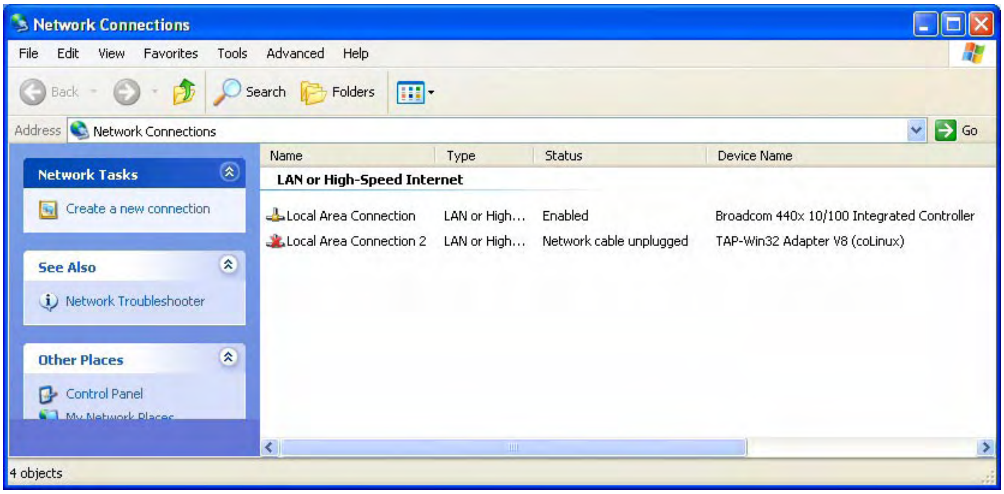
Shortcut key: None

Run: Normal window

Comment:

14. Open the Network Connections window. In the Network Connections window, find the LAN connection that uses the physical network adapter in your PC. And find the LAN connection that uses the virtual network adapter that was added during the coLinux installation.

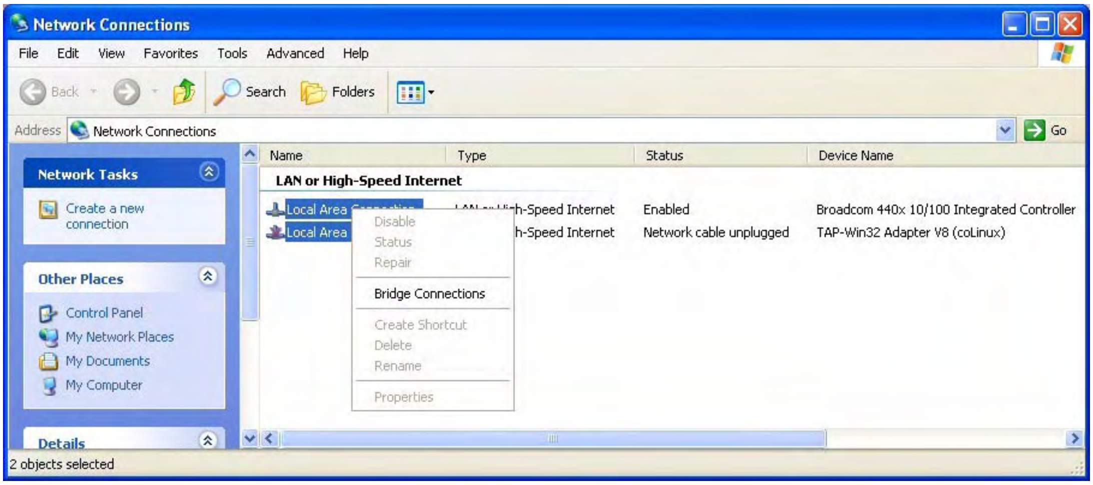
    (In the illustration below, “Local Area Connection” is the name of a connection that uses a physical Broadcom network adapter board. “Local Area Connection 2” is the name of a connection that uses a “TAP-Win32” virtual network adapter that was added during the coLinux install).



Name	Type	Status	Device Name
LAN or High-Speed Internet			
Local Area Connection	LAN or High...	Enabled	Broadcom 440x 10/100 Integrated Controller
Local Area Connection 2	LAN or High...	Network cable unplugged	TAP-Win32 Adapter V8 (coLinux)

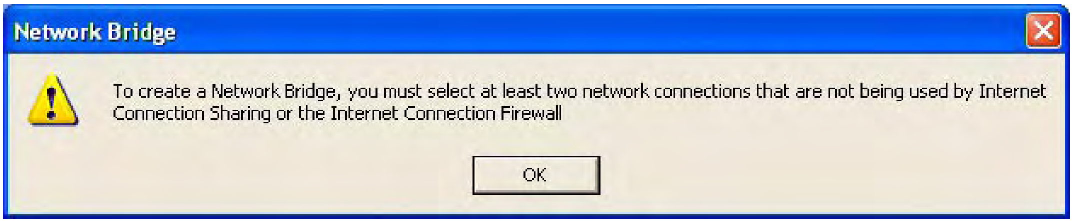
15. Do the following to create a network bridge, bridging the physical and the virtual adapter:

    a) Select the physical and the virtual adapter and right click. A drop down menu will open.



Name	Type	Status	Device Name
LAN or High-Speed Internet			
Local Area	h-Speed Internet	Enabled	Broadcom 440x 10/100 Integrated Controller
Local Area	h-Speed Internet	Network cable unplugged	TAP-Win32 Adapter V8 (coLinux)

    b) Select Bridge Connections in the drop down menu. The following error message may appear.



Network Bridge

To create a Network Bridge, you must select at least two network connections that are not being used by Internet Connection Sharing or the Internet Connection Firewall

This error message appears if you have changed the Windows default setting of the local area connection properties.

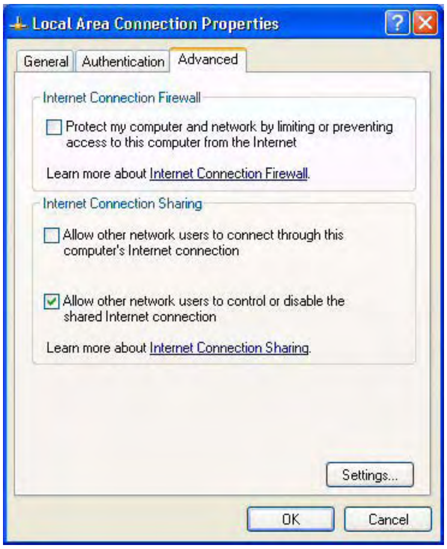
If the error message appears you must set the local area connection properties appropriate for a network bridge:

- Close the error message
- In the Network Connections window:

  Right click on the name of the physical network adapter.

  Select Properties from the drop down window. A Properties window will open.
- Click on the Advanced tab.
- In the “Internet Connection Sharing“ section of the tab:

  Make sure the box labeled “Allow other network users to connect...“ is not checked as shown below.



Local Area Connection Properties

Internet Connection Firewall

□ Protect my computer and network by limiting or preventing access to this computer from the Internet

Learn more about Internet Connection Firewall.

Internet Connection Sharing

□ Allow other network users to connect through this computer's Internet connection

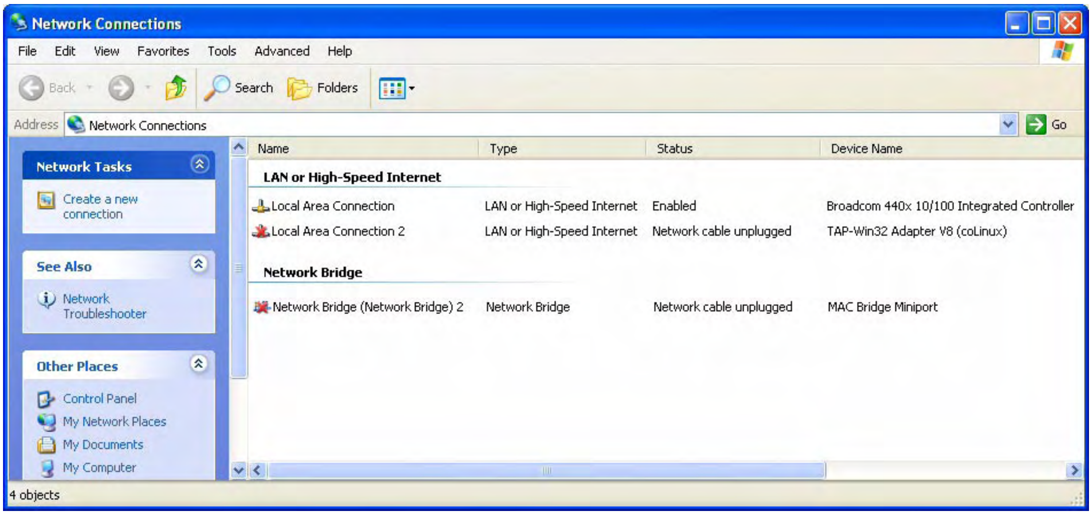
☑ Allow other network users to control or disable the shared Internet connection

Learn more about Internet Connection Sharing.

- Click the OK button.
- Select the physical and the virtual adapter in the Network Connections window (see figure after step 15a) and right click. A drop down menu will open.
- Select Bridge Connections in the drop down menu. An entry for the network bridge will appear in the Network Connections window.

c) Restart your computer. This will ensure that all of the network connections are properly reset.

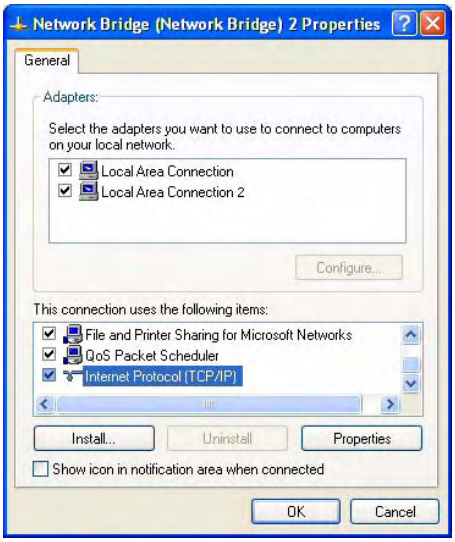
d) Open the Network Connections window. In the Network Connections window, find the name of the network bridge as shown below:



Name	Type	Status	Device Name
LAN or High-Speed Internet			
Local Area Connection	LAN or High-Speed Internet	Enabled	Broadcom 440x 10/100 Integrated Controller
Local Area Connection 2	LAN or High-Speed Internet	Network cable unplugged	TAP-Win32 Adapter V8 (coLinux)
Network Bridge			
Network Bridge (Network Bridge) 2	Network Bridge	Network cable unplugged	MAC Bridge Miniport

e) Right click on the name of the network bridge in the Network Connections window. A drop down menu will open.

f) Select Properties. A Network Bridge (Network Bridge) 2 Properties window will open.



Select the adapters you want to use to connect to computers on your local network.

☑ Local Area Connection
☑ Local Area Connection 2

This connection uses the following items:

☑ File and Printer Sharing for Microsoft Networks
☑ QoS Packet Scheduler
☑ Internet Protocol (TCP/IP)

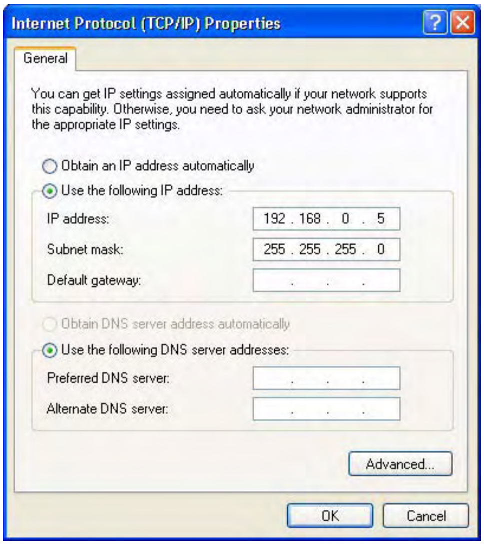
☐ Show icon in notification area when connected

g) Make sure the adapters of the bridge are checked in the edit field of the Adapter section as shown above.

h) Select Internet protocol (TCP/IP) in the lower edit field as shown above.

i) Click the Properties button. An Internet Protocol (TCP/IP) Properties window will open.

Note: The following steps j through o will set the network bridge to static IP addressing. If the physical adapter links your PC to a LAN that is set for dynamic IP addressing, setting the network bridge to static addressing will cut communication between your PC and the LAN. You must set the network bridge to dynamic IP addressing if you want to continue communication with a LAN that is set for dynamic IP addressing. To set the network bridge to dynamic IP addressing follow the procedures in Section 2.4.4 and then continue with step 16.



Internet Protocol (TCP/IP) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

○ Obtain an IP address automatically

● Use the following IP address:

IP address: 192 . 168 . 0 . 5

Subnet mask: 255 . 255 . 255 . 0

Default gateway: . . .

○ Obtain DNS server address automatically

● Use the following DNS server addresses:

Preferred DNS server: . . .

Alternate DNS server: . . .

j) Make sure that the circle next to “Use the following IP address“ is checked as shown above.

k) Enter the IP address of the network bridge in the IP address line. Make sure it is 192.168.0.5 as shown above.

l) Press the tab key. This will enter the subnet mask in the Subnet mask line. Make sure it is 255.255.255.0 as shown above.

m) Click the OK button.

n) Click the Close button on the Network Bridge (Network Bridge) 2 Properties window.

o) Restart your computer. This will ensure that all of the network connections are properly reset.

16. Double click on the coLinux shortcut on your desktop.

    A console window will open and you will see the bootup process for the coLinux OS. Since this is the first bootup for coLinux, you will also see a file system check performed.

    Once the bootup and file system check is complete you can log onto the coLinux OS:

    The coLinux login name is : `root`

    The coLinux password is : `root`

    (Remember to use lower case letters as all entries on a Linux system are case sensitive.)

17. Change the keyboard configuration if necessary. By default, coLinux uses the US keyboard configuration. To change the keyboard configuration, execute the following command (xx = language designation).

    ```
    loadkeys xx ↵
    ```

    E.g. for the Italian keyboard configuration the command would be `loadkeys it.`

    You can find the language designation of the language you want to use by looking at the first characters of the file names of the keyboard driver’s translation tables.

    You can list the file names of the keyboard driver’s translation tables by executing the following commands:

    ```
    ls /usr/share/keymaps/i386/qwertz ↵
    ```

    and

```
ls /usr/share/keymaps/i386/qwerty ↵
```

Note: The changed keyboard configuration applies only to the current session of coLinux. During the next start up of coLinux, the US keyboard configuration will be activated again.

18. Check the coLinux network connection by pinging the IP address of some device on your network. If your computer is connected to the Internet, you can try pinging a site on the net.

    If the ping is successful, your coLinux network connect is OK.

    If the ping is not successful, recheck your network connections and make sure that they are set as described above.

19. coLinux installation and setup is complete. You should exit the coLinux console and go on to the GUI interface installation. The correct way to exit the coLinux console is to execute the following command:

    ```
    halt ↵
    ```

    The console will go through a halt process that takes several seconds to finish and then the console window will close.

    Note: Whenever the coLinux console window closes, you will usually see a message that one of your network connections has been lost. This is normal. It happens because the virtual network adapter for coLinux disconnects when the coLinux console closes.

ⓘ The files that you extracted in step 10 of this procedure included everything you need to use the Eclipse IDE, the tool chain used with the IDE, and the Basler code samples. When you install the GUI interface, you will find a shortcut to access Eclipse.

2.4.2.1 Disabling Windows Data Execution Prevention

In some cases, the Windows Data Execution Prevention (DEP) feature will prevent coLinux from running: When starting coLinux with the DEP feature enabled, the console appears briefly and the computer is subsequently halted, displaying a blue screen.

The following instructions describe how to disable the DEP feature ensuring a smooth start of coLinux and preventing a blue screen from being displayed.

To disable DEP:

1. Find the `c:\Boot.ini` file in Windows Explorer.

   If `c:\Boot.ini does not appear` in Windows Explorer:

   a) Click the Tools button in the menu bar of Windows Explorer. A drop down menu opens.
   b) Click Folder Options. The Folder Options window will open.
   c) Click the View tab.
   d) In the list scroll down to find Hidden Files and Folders.
   e) If not already selected, select the option button in front of Show hidden files and folders.
   f) Click the Apply button.
   g) Click OK.
2. Double click on the `Boot.ini` file. The Notepad text editor will start and open the `c:\Boot.ini` file.

   If the Notepad editor does not start use a basic text editor to open and edit the `c:\Boot.ini` file.
3. Move to the line `/noexecute=optin`
4. Change the line to `/noexecute=AlwaysOff`
5. Save the modified `Boot.ini` file and close the text editor.
6. Restart your computer. This is required for the changes to the `Boot.ini` file to become effective.

   The DEP feature is disabled.

After you have changed the `Boot.ini` file we suggest hiding hidden files and folders again if they were hidden initially.

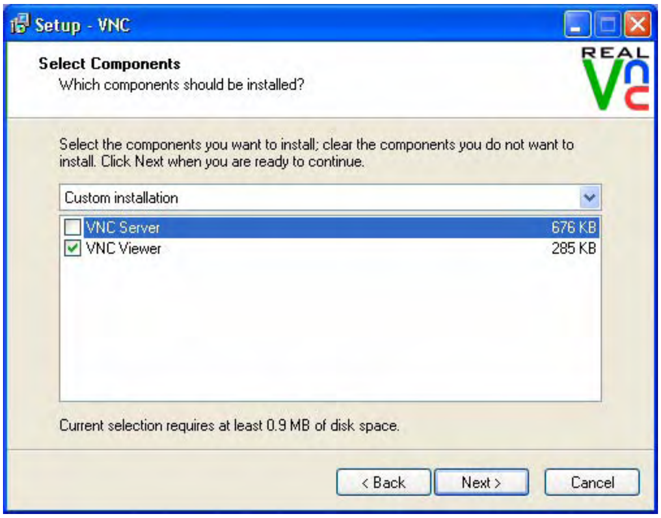
2.4.3 Installing the VNC Viewer

A program called VNC Viewer will be used as a GUI front end for coLinux.

This prodcedure assumes that you know the IP address that is currently assigned to coLinux. For more information on how to determine the currently assigned IP address, see step 1 in section 2.4.4.4.

To install the VNC Viewer:

1. Click Start and click Run.
   a) Click the Browse button, navigate to the `software\windows\` directory on the eXcite CD, and find the “vnc-XXX-x86_win32.exe” file (XXX = the version designation).
   b) Click on the file, click Open, and click OK.
2. An install wizard will open and a Setup window will appear.
   a) Click Next.
3. A License Agreement window will appear.
   a) Check the option button to accept the agreement.
   b) Click Next
4. A Select Destination window will appear.
   a) Set the destination folder as desired and click Next.
5. A Select Components window will appear.
   a) Make sure that “VNC Viewer” is checked and that “VNC Server” **is not** checked as shown below.
   b) Click Next.



Setup - VNC

Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Custom installation

☐ VNC Server 676 KB

☑ VNC Viewer 285 KB

Current selection requires at least 0.9 MB of disk space.

6. A Select Start Menu Folder window will appear.
   a) Set the folder as desired and click Next.
7. A Select Additional Tasks window will appear.
   a) We suggest that you check the selection to create a viewer icon on your desktop.
   b) Click Next.
8. A Ready to Install window will appear.
   a) Check to make sure that the selections are correct and then click Install.

9. The installation will proceed and then an Information window will appear.
   a) Click Next.

10. A Completing window will appear.
    a) Click Finish. The basic VNC Viewer installation is complete.

11. If you don't already know it, determine the resolution of the screen on your PC. To check the resolution:
    a) Right click on your desktop.
    b) Select Properties from the drop down menu.
    c) A Display Properties window will open.
    d) Click the Settings tab and make note of the display resolution.
    e) Click OK.

12. In this step, we will modify a coLinux configuration script so that it contains the screen resolution and color depth information to use with the VNC Viewer. Before starting this step, make sure that you know your screen resolution and the color depth that you want to use. You can use the screen resolution of your screen but 24 bit color depth is the maximum allowed (16 bit color depth is typical). For illustration, we assume that your screen resolution is 1024 x 768 and that you want 16 bit color depth. You should use your actual preferences.

    In this step, we will use the nano text editor to edit the configuration script. For more information on the nano text editor see "Some Basics of the nano Editor" on page 2-9).

ⓘ	We strongly recommend disabling the Windows Data Execution Prevention (DEP) feature before proceeding. Disabling the DEP feature will ensure a smooth start of coLinux and prevent the computer from possibly displaying a blue screen. See Section 2.4.2.1 for more information on disabling the DEP feature.

a) Double click the coLinux icon on your desktop. The coLinux console will open and bootup will begin. When the bootup process is complete, log onto coLinux (login = root, pwd = root).

   Unless the DEP feature is disabled (recommended; see Section 2.4.2.1) double clicking the coLinux icon may have resulted in a blue screen. To correct the situation see Section 2.4.2.1.

b) If you need to change the keyboard configuration, execute the following command (xx = language designation; for more information see step 17 in Section 2.4.2):

```
loadkeys xx ↵
```

c) Execute the following command:

```
nano -w /etc/vnc.conf ↵
```

   This will open the configuration file in the nano text editor.

d) Use the Down Arrow key to move the cursor to the end of the file. Edit the two lines shown below by entering your screen resolution and color depth. The resolution and color depth shown below are given as examples only.

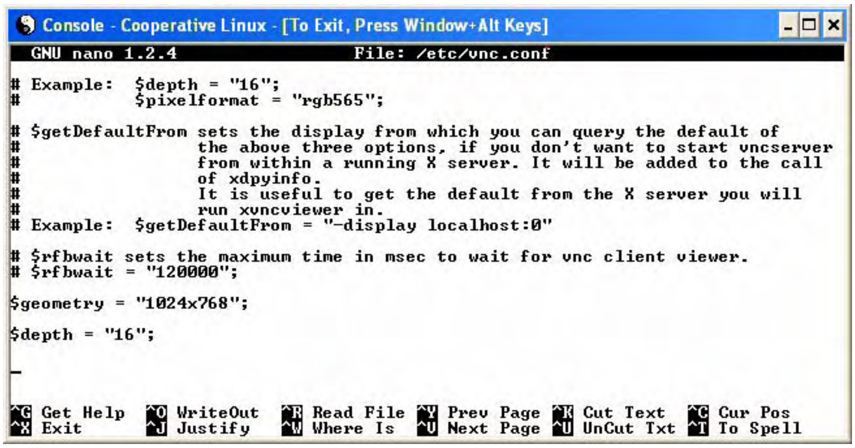
```
$geometry = "1024x768";

$depth = "16";
```

e) Press the Control+X keys.

f) When you are asked to save the modified buffer, execute:

`y ↵`

g) When you are asked the file name to write, press the Enter key. This will close the file and save the changes.



Console - Cooperative Linux - [To Exit, Press Window+Alt Keys]

```
  GNU nano 1.2.4                  File: /etc/vnc.conf

# Example:  $depth = "16";
#           $pixelformat = "rgb565";

# $getDefaultFrom sets the display from which you can query the default of
#                  the above three options, if you don't want to start vncserver
#                  from within a running X server. It will be added to the call
#                  of xdpyinfo.
#                  It is useful to get the default from the X server you will
#                  run xvncviewer in.
# Example:  $getDefaultFrom = "-display localhost:0"

# $rfbwait sets the maximum time in msec to wait for vnc client viewer.
# $rfbwait = "120000";

$geometry = "1024x768";

$depth = "16";

_

^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Txt  ^T To Spell
```

13. Shut down and restart coLinux:

a) Execute the following command:

`halt ↵`

The console will go through a halt process that takes several seconds to finish and then the console window will close.

Note: Whenever the coLinux console window closes, you will usually see a message that one of your network connections has been lost. This is normal. It happens because the virtual network adapter for coLinux disconnects when the coLinux console closes.

b) Double click the coLinux icon on your desktop. The coLinux console will open and bootup will begin. When the bootup process is complete, log onto coLinux.

Unless the DEP feature is disabled (recommended; see Section 2.4.2.1) double clicking the coLinux icon may have resulted in a blue screen. To correct the situation see Section 2.4.2.1.

14. In this step, we will start a “VNC Server” running in coLinux. A VNC Server must be running in coLinux to allow the VNC Viewer to properly connect with coLinux.

a) Change the keyboard configuration if necessary. For further information see step 17 in Section 2.4.2.

b) To start the VNC Server execute the following command in the coLinux console:

`vncserver :0 ↵`

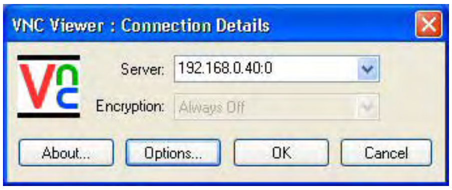
The VNC server software in coLinux will start up and indicate that an X desktop has been created.

15. Minimize the coLinux console (**don’t close it!**).

16. In this step, we will start the VNC Viewer and connect the viewer to coLinux.

a) Double click the VNC Viewer icon on your desktop. A Connection Details window will open.

b) On the Server line, enter the server’s Display ID. The Display ID consists of the IP address that is currently assigned to coLinux and, separated by a colon, the appended display number. Enter `192.168.0.40:0` in accord with the present installation using static IP addressing.



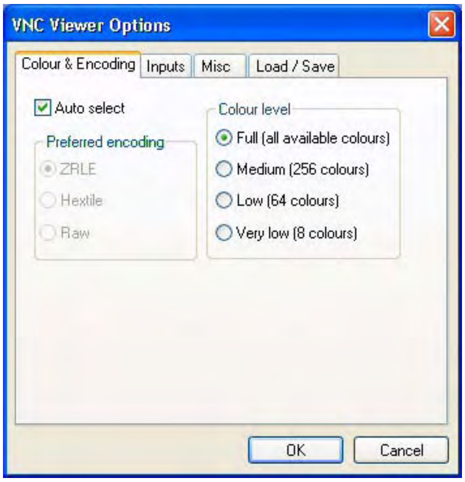
VNC Viewer : Connection Details

Server: 192.168.0.40:0

Encryption: Always Off

c) Click the Options button.

d) On the Colour & Encoding tab, go to the Colour Level section and check Full as shown below.



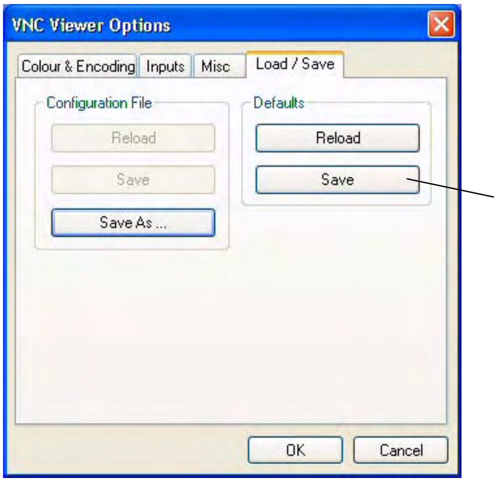
VNC Viewer Options

☑ Auto select

Preferred encoding
- ● ZRLE
- ○ Hextile
- ○ Raw

Colour level
- ● Full (all available colours)
- ○ Medium (256 colours)
- ○ Low (64 colours)
- ○ Very low (8 colours)

e) On the Load/Save tab, go to the defaults section and click the Save button as shown below. (This action saves your entries so that they will be automatically reentered each time you start the viewer.)

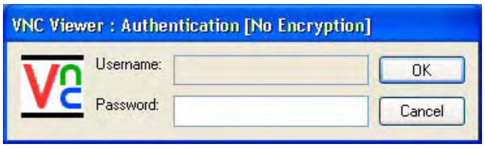


Click this Save button

f) Click the OK button.

17. On the Connection Details window, click the OK button.

18. An Authentication window will appear as shown below.
    a) Type in the password: `excite`
    b) Click OK.



VNC Viewer : Authentication [No Encryption]

19. The VNC Viewer will open, it will connect to the coLinux OS, and it will act as a GUI front end for coLinux.

    To expand the VNC Viewer so that it uses the full screen, press the F8 key and click Full screen in the menu that appears.

You now have full GUI access to coLinux operating system. In essence, you are now using a self-contained Linux PC that resides on your development PC.

Notice that on your desktop, you will find an icon you can use to open the Eclipse IDE.

If you click on the file drawer icon near the top of your screen, it will open a file browser application. If you open the application and look through the available files, you will notice that sample code files and applications that are part of the tool chain were installed along with the base file system.

Recommended Procedure for Closing coLinux and the VNC Viewer

1. If you are running the VNC Viewer in full screen mode, click the F8 key. In the menu that opens, click Full Screen. This will take the viewer out of full screen mode.
2. Click the X in the upper right hand corner of the Viewer window to close the Viewer application.
3. Maximize the coLinux console.
4. Execute the following command in the coLinux console:

   ```
   halt ↵
   ```

   The console will go through a halt process that takes several seconds to finish and then the console window will close.

   Note: Whenever the coLinux console window closes, you will usually see a message that one of your network connections has been lost. This is normal. It happens because the virtual network adapter for coLinux disconnects when the coLinux console closes.

Recommended Procedure for Opening coLinux and the VNC Viewer

ⓘ	If the Windows Data Execution Prevention (DEP) feature is not already disabled, we strongly recommend disabling the DEP feature before proceeding. This wil lensure a smooth start of coLinux and prevent the computer from possibly displaying a blue screen. See Section 2.4.2.1 for more information on disabling the DEP feature.

1. Double click the coLinux icon on your desktop. The coLinux console will open and bootup will begin. When the bootup process is complete, log onto coLinux (login = root, pwd = root).

   Unless the DEP feature is disabled (recommended; see Section 2.4.2.1) double clicking the coLinux icon may have resulted in a blue screen. To correct the situation see Section 2.4.2.1.
2. Execute the following command in the coLinux console:

   ```
   vncserver :0 ↵
   ```
3. The VNC server software in coLinux will start up and indicate that an X desktop has been created. Minimize the coLinux console (**don’t close it!**).
4. Double click the VNC Viewer icon on your desktop. A Connection Details window will open. This window should already show the necessary information needed to connect the viewer with coLinux.
   a) Click the OK button.
5. An Authentication window will appear.
   a) Type in the password: `excite`
   b) Click OK.
6. The VNC Viewer will open, it will connect to the coLinux OS, and it will act as a GUI front end for coLinux.

   To expand the VNC Viewer so that it uses the full screen, press the F8 key. In the menu that opens, click Full Screen.

2.4.4 Configuring for Dynamic IP Addressing

If you chose static IP addressing in accord with the preceding procedures, coLinux, the VNC server, and the network bridge will have the following addresses, each being consistent with a subnet mask of 255.255.255.0:

Conponent	Static IP Address/Display ID
coLinux	192.168.0.40
VNC server	192.168.0.40:0
Network bridge	192.168.0.5

You can use dynamic IP addressing where a DHCP server assigns IP addresses to CoLinux, the VNC server, and the network bridge.

If you want to use dynamic IP addressing the following conditions must all be fulfilled:

- The eXcite must be set to dynamic IP addressing.
- You must set coLinux **and** the network bridge to dynamic IP addressing.
- You must determine the IP address that is currently assigned to coLinux and then set the Display ID of the VNC server. The Display ID consists of the current IP address of coLinux and, separated by a colon, an appended 0 as display number.

The following procedures assume that all settings are for static IP addressing before configuring for dynamic IP addressing.

2.4.4.1 Setting the eXcite to Dynamic IP Addressing

As the default of the factory setting, the eXcite first makes several attempts to connect to a network with dynamic IP addressing (Section 2.1.1.2). Only if these attempts fail, the eXcite sets itself to a static IP address of 192.168.0.173 and a subnet mask of 255.255.255.0.

The default of the factory setting may however have been changed by the user. If you want to use dynamic IP addressing, make sure the eXcite is not set to a static IP address. Check the configuration file of the eXcite following the procedure given in Section 2.1.1.4 and set the eXcite to dynamic IP addressing if necessary.

2.4.4.2 Setting coLinux to Dynamic IP Addressing

To set coLinux to dynamic IP addressing:

Note: In this procedure, you will be using the nano editor to edit the coLinux configuration file. A brief introduction to the editor appears on page 2-9 and you can find many introductions to the nano editor on the web.

> We strongly recommend disabling the Windows Data Execution Prevention (DEP) feature before proceeding. Disabling the DEP feature will ensure a smooth start of coLinux and prevent the computer from possibly displaying a blue screen. See Section 2.4.2.1 for more information on disabling the DEP feature.

1. Double click the coLinux icon on your desktop. The coLinux console will open and bootup will begin. When the bootup process is complete, log onto coLinux (login = root, pwd = root).

   Unless the DEP feature is disabled (recommended; see Section 2.4.2.1) double clicking the coLinux icon may have resulted in a blue screen. To correct the situation see Section 2.4.2.1.

2. Change the keyboard configuration if necessary. For further information see step 17 in Section 2.4.2.

3. Execute the following commands to navigate to the `/etc/network/` directory and make a backup copy of the unmodified `interfaces` file:

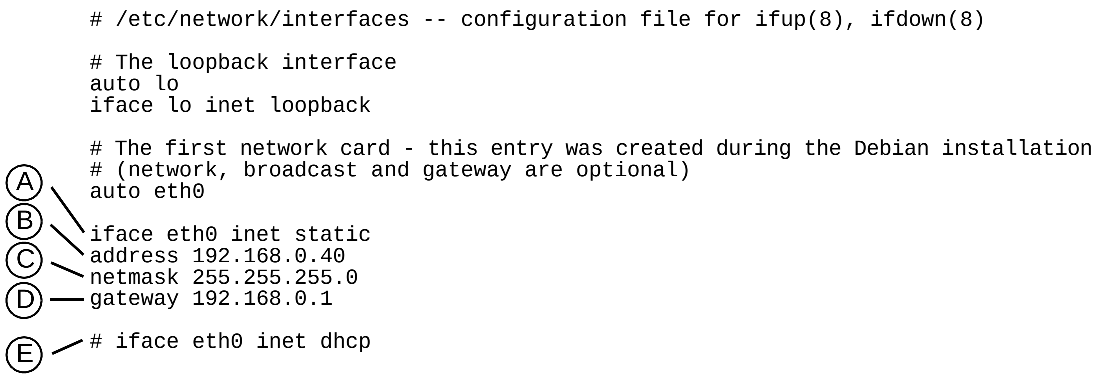
   a) `cd /etc/network` ↵

   b) `cp interfaces interfaces_bu` ↵

4. execute the following command:

   `nano -w interfaces` ↵

5. Put hash marks in front of lines A, B, C, and D of the code shown below to comment them out:



```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

# The first network card - this entry was created during the Debian installation
# (network, broadcast and gateway are optional)
auto eth0

iface eth0 inet static
address 192.168.0.40
netmask 255.255.255.0
gateway 192.168.0.1

# iface eth0 inet dhcp
```

   Remove the hash mark in front of line E.

6. Press the Control+X keys.

7. When you are asked to save the modified buffer, execute:

   `y` ↵

8. Execute the following command in the coLinux console:

   `halt` ↵

   The console will go through a halt process that takes several seconds to finish and then the console window will close.

   Note: Whenever the coLinux console window closes, you will usually see a message that one of your network connections has been lost. This is normal. It happens because the virtual network adapter for coLinux disconnects when the coLinux console closes.

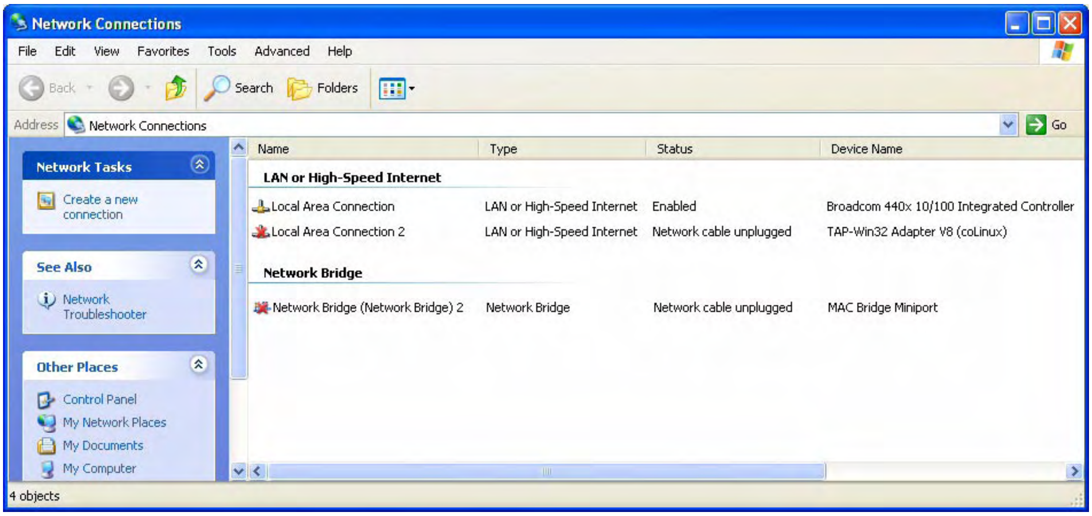
   coLinux is now set to dynamic IP addressing.

2.4.4.3 Setting the Network Bridge to Dynamic IP Addressing

This procedure assumes that you have created a network bridge, bridging the physical network adapter in your PC and the virtual network adapter that was added during the coLinux installation. For more information on creating a network bridge see Section 2.4.2.

To set the network bridge to dynamic IP addressing:

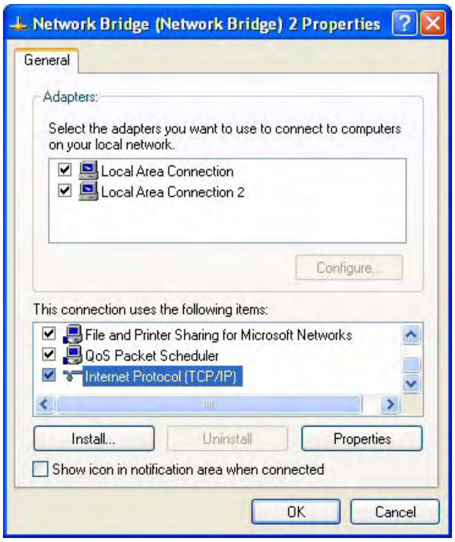
1. Open the Network Connections window. In the Network Connections window, find the name of the network bridge as shown below:



Name	Type	Status	Device Name
LAN or High-Speed Internet			
Local Area Connection	LAN or High-Speed Internet	Enabled	Broadcom 440x 10/100 Integrated Controller
Local Area Connection 2	LAN or High-Speed Internet	Network cable unplugged	TAP-Win32 Adapter V8 (coLinux)
Network Bridge			
Network Bridge (Network Bridge) 2	Network Bridge	Network cable unplugged	MAC Bridge Miniport

4 objects

2. Right click on the name of the network bridge in the Network Connections window. A drop down menu will open.
3. Select Properties. A Network Bridge (Network Bridge) 2 Properties window will open.



Select the adapters you want to use to connect to computers on your local network.

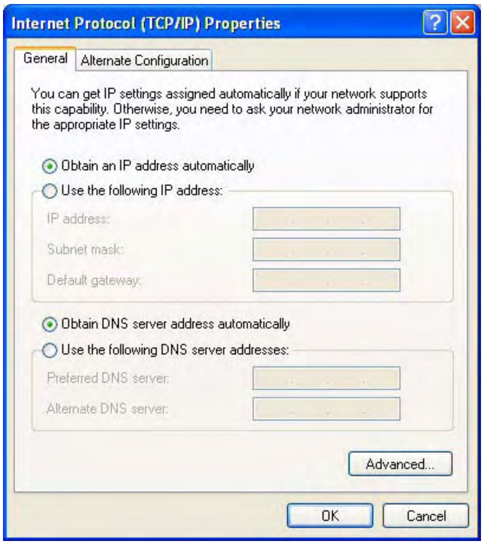
☑ Local Area Connection
☑ Local Area Connection 2

This connection uses the following items:

☑ File and Printer Sharing for Microsoft Networks
☑ QoS Packet Scheduler
☑ Internet Protocol (TCP/IP)

□ Show icon in notification area when connected

4. Make sure the adapters of the bridge are checked in the edit field of the Adapter section as shown above.
5. Select Internet protocol (TCP/IP) in the lower edit field as shown above.
6. Click the Properties button. An Internet Protocol (TCP/IP) Properties window will open.



Internet Protocol (TCP/IP) Properties

General | Alternate Configuration

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

● Obtain an IP address automatically

○ Use the following IP address:

IP address:

Subnet mask:

Default gateway:

● Obtain DNS server address automatically

○ Use the following DNS server addresses:

Preferred DNS server:

Alternate DNS server:

7. Make sure the circle next to “Obtain an IP address automatically“ is checked as shown above.
8. Make sure that the circle next to “Use the following IP address“ is not checked as shown above.
9. Click the OK button.
10. Click the Close button on the Network Bridge (Network Bridge) 2 Properties window.
11. Restart your computer. This will ensure that network connections are properly reset.

2.4.4.4 Adjusting the Display ID of the VNC Server

In the following procedure you will first determine the IP address that is currently assigned to coLinux and then will adjust the Display ID of the VNC server to the current IP address of coLinux. The Display ID of the VNC server must use the IP address currently assigned to coLinux.

To adjust the IP address of the VNC server:

1. In this step, we will determine the IP address that is currently assigned to coLinux:
   a) Double click the coLinux icon on your desktop. The coLinux console will open and bootup will begin. When the bootup process is complete, log onto coLinux (login = root, pwd = root).
   b) If you need to change the keyboard configuration, execute the following command (xx = language designation; for more information see step 17 in Section 2.4.2):

      ```
      loadkeys xx ↵
      ```

   c) Execute the following command in the coLinux console:

      ```
      ifconfig ↵
      ```

      The current settings of coLinux will be displayed in text similar to the one shown below:

```
      eXcite-devel:~# ifconfig
      eth0      Link encap:Ethernet  HWaddr 00:FF:B5:50:00:00
Ⓐ ————————      inet addr:172.16.50.9  Bcast:172.16.255.255  Mask:255.255.0.0
                UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                RX packets:670 errors:0 dropped:0 overruns:0 frame:0
                TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
                collisions:0 txqueuelan:1000
                RX bytes:63395 <61.9 KiB>  TX bytes:2052 <2.0 KiB>
                Interrupt:2
```

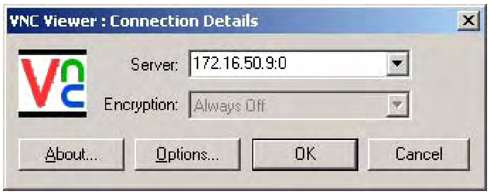
As can be seen from line A of the example shown above, a current IP address of 172.16.50.9 was dynamically assigned to the virtual Ethernet adapter of coLinux.

2. To start the VNC Server execute the following command in the coLinux console:

   `vncserver :0` ↵

   The VNC server software in coLinux will start up and indicate that an X desktop has been created.

3. Minimize the coLinux console (**don't close it!**).
4. In this step, we will start the VNC Viewer and connect the viewer to coLinux.
   a) Double click the VNC Viewer icon on your desktop. A Connection Details window will open.
   b) On the Server line, enter the server's Display ID. The Display ID consists of the current IP address of coLinux that you have determined in step 1 and, separated by a colon, an appended 0 as display number. If, for example, the current IP address of coLinux is 172.16.50.9 you would have to enter: `172.16.50.9:0`.



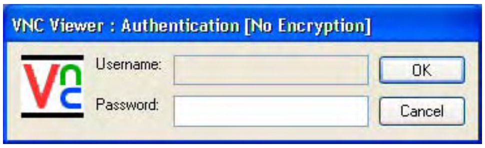
VNC Viewer : Connection Details

Server: 172.16.50.9:0

Encryption: Always Off

   c) Click the OK button.

5. An Authentication window will appear as shown below.
   a) Type in the password: `excite`
   b) Click the OK button.



VNC Viewer : Authentication [No Encryption]

Username:

Password:

The VNC Viewer will open, it will connect to the coLinux OS, and it will act as a GUI front end for coLinux.

3 Learning to Use the eXcite

To best learn about the capabilities of the eXcite and about how to work with the product, we suggest that you do the following:

- Read Sections 4 and 5 of the User's Manual. Section 4 will give you a general understanding of the interface options available on the camera. Section 5 will acquaint you with the features available on the eXcite. If you understand how the eXcite's features work and how they are parameterized, it will be easier to operate the product and to understand our code samples.
- Read the eXcite API overview in the API Reference documentation. This overview will give you an introduction about how the API is used to control the camera section of the eXcite. The API overview and the API reference in general contain the detailed information you will need to know about the eXcite API when you are writing programs to operate the eXcite.
- Work through the examples in Sections 3.2 and 3.3. These will give you some basic practice in building, loading, running, and debugging programs.
- Work with the other code samples provided by Basler. The code samples are the main tool for learning how to work with the eXcite. Section 3.4 provides a description for each code sample along with the source code for the sample's main function. The description explains what the sample is trying to illustrate and the source code lets you see exactly how the sample program is implemented.

  By reading the description for a sample, examining the source code for the sample, and compiling and running the sample, you will see a working example of how to make use of the eXcite's capabilities.

The eXcite is shipped with the Basler SFF Viewer GX software which is designed for use in a Windows 2000 or Windows XP operating system. The Basler SFF Viewer GX software allows you to easily enable and disable the eXcite's features, set the eXcite's parameters, and capture and view images. The Basler SFF Viewer GX software will not operate on a PC with a Linux operating system. For more information on installing and using the Basler SFF Viewer GX software see the SFF Viewer GX Getting Started Guide.

3.1 The Basler eXcite library provides the APIs used to:

- configure the camera section of the eXcite
- control image exposure
- transfer captured images from the camera section of the eXcite to the processor section of the eXcite
- transmit image data from the eXcite to a PC or other device

Applications using the eXcite library must be linked against the eXcite library files. These libraries were installed on your development PC as part of the tool chain installation process. They can be found in `/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/lib`.
The corresponding header files are located in `/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/include`.
(Note that on a Windows development PC, these files are part of the coLinux file system.)

When compiling an application that uses the Basler eXcite library, use the `-I` and `-L` compiler/linker switches to specify the directory where the library and the header files can be found. Here are some examples:

```
mips-linux-gnu-g++ -c -I/opt/excite-tools/mips-linux-gnu/sys-root/opt/
excite/include -o Simplegrab.o simplegrab.cpp
```

```
mips-linux-gnu-g++ -L/opt/excite-tools/mips-linux-gnu/sys-root/opt/
excite/include -o Simplegrab simplegrab.o -lxcam
```

Additional information about the libraries to link against and their dependencies can be found in the eXcite API overview in the API Reference documentation

ⓘ	The code samples supplied with the eXcite are set up as projects for the Eclipse IDE. So if you open the code samples in Eclipse, the library information is automatically included in the IDE. See Section 3.2.4 for more information.

3.2 Building, Loading, & Running a Program

Now that the Eclipse IDE and the tool chain are installed, you can build executables and run them on the MIPS processor in the eXcite. Basler source code samples were also added to your PC as part of the installation procedure. These code samples include a number of files that will configure Eclipse to find the correct libraries and includes and to find the cross-compiler, linker, etc. needed to build each sample. Because these configuration files are included with the samples, building a sample is a good way to learn how to configure and use the Eclipse IDE.

Sections 3.2.1 through 3.2.3 describes how to build, load, and run a release version of the SimpleGrab sample program. You can build, load, and run other programs in a similar fashion.

Section 3.2.4 describe how to build a release version or a debug version of a program and also describes how to target a build for different platforms.

These descriptions assume that you are using a PC with a Linux OS or that you are using a Windows PC with coLinux and the VNC Viewer open.

3.2.1 Building the SimpleGrab Sample Program

To build a release version of the “SimpleGrab” sample program to run on the eXcite.

1. Start the Eclipse Integrated Development Environment (IDE) on your development PC.
2. You may see a Workspace Launcher dialog box similar to the one shown below.

   If you do not see the Workspace Launcher, go on to step 3 now.

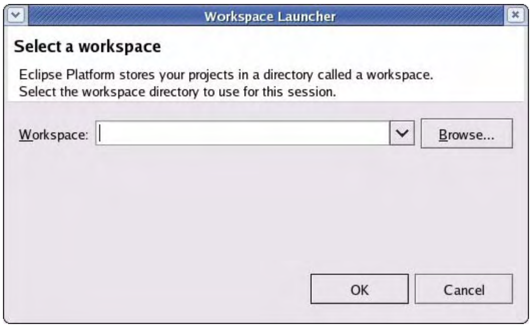
   If you do see the Workspace Launcher:

   a) Click the Browse button in the Workspace Launcher window.

   b) Navigate to the `samples` directory.

   c) Click the OK button.

   d) Go on to step 3.



Select a workspace

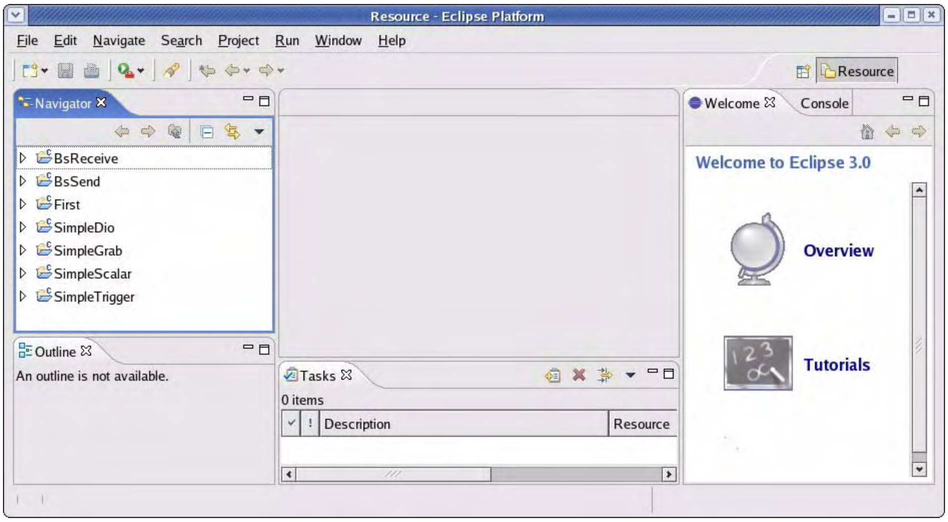
Eclipse Platform stores your projects in a directory called a workspace.
Select the workspace directory to use for this session.

3. Eclipse will open a samples workspace that looks similar to the one shown below. (The look can vary slightly depending on your OS.)

   The Navigator pane at the left side of the workspace should show a list of the sample programs as “projects” included in the workspace. All of the Basler sample programs should be included in the workspace and should be listed in the Navigator pane. Frequently, however, many of the sample programs will not be included.

   If the navigation pane contains a list of the sample programs, and if the SimpleGrab sample is included in the list, go to step 5.

   If the navigation pane does not contain a list of the sample programs or if the list does not include the SimpleGrab program, go to step 4.



An outline is not available.

0 items

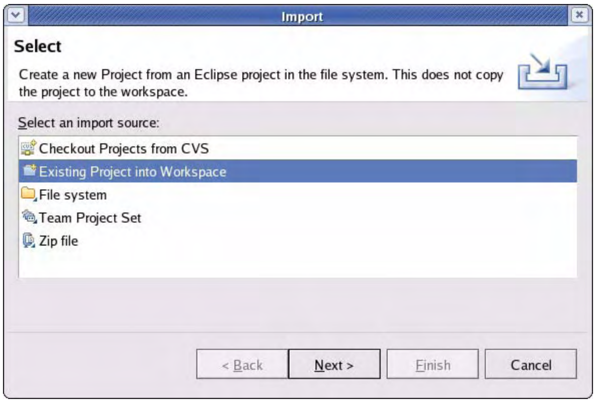
Welcome to Eclipse 3.0

4. If you do not see the Simple Grab sample program in the workspace, you will need to import it. The steps below describe importing the SimpleGrab sample program project into the workspace. The other sample programs can be imported in a similar fashion.

   To import the SimpleGrab sample program:

   a) Click the File menu at the top of the Eclipse window and select Import from the drop down list.

   b) When the Import window opens, select “Existing project into workspace” as shown below and then click Next.

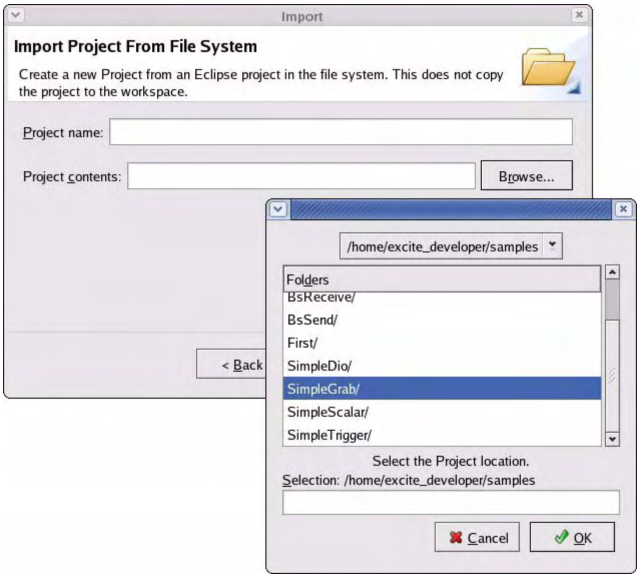


Select

Create a new Project from an Eclipse project in the file system. This does not copy the project to the workspace.

Select an import source: Existing Project into Workspace

c) Click Browse and then navigate to the `samples/SimpleGrab` directory as shown below. Double-click SimpleGrab and then click OK.



Import

Import Project From File System

Create a new Project from an Eclipse project in the file system. This does not copy the project to the workspace.

Project name:

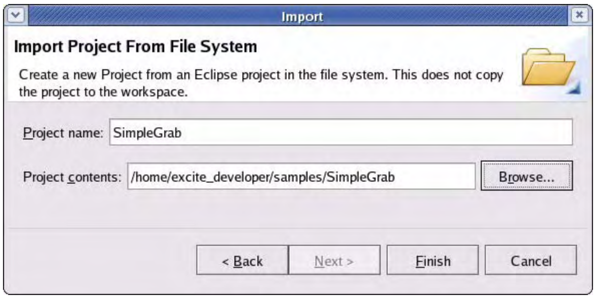
Project contents:

/home/excite_developer/samples

Select the Project location.

Selection: /home/excite_developer/samples

d) The Import window should indicate that it will import the SimpleGrab sample program project as shown below. Click Finish.



Import

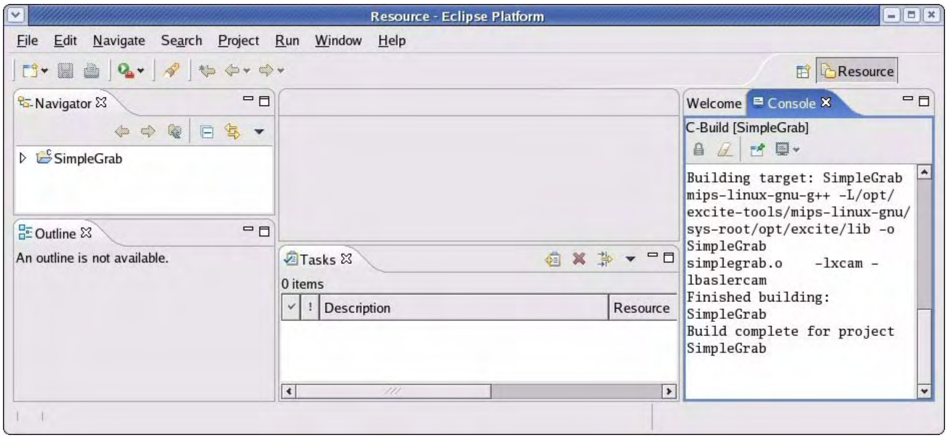
Import Project From File System

Create a new Project from an Eclipse project in the file system. This does not copy the project to the workspace.

Project name: SimpleGrab

Project contents: /home/excite_developer/samples/SimpleGrab

e) The SimpleGrab sample program project will be imported into the workspace. An automatic build will run when the sample is imported. Once the build is complete, you should see the SimpleGrab sample program project in the Navigator pane as shown below.



Resource - Eclipse Platform

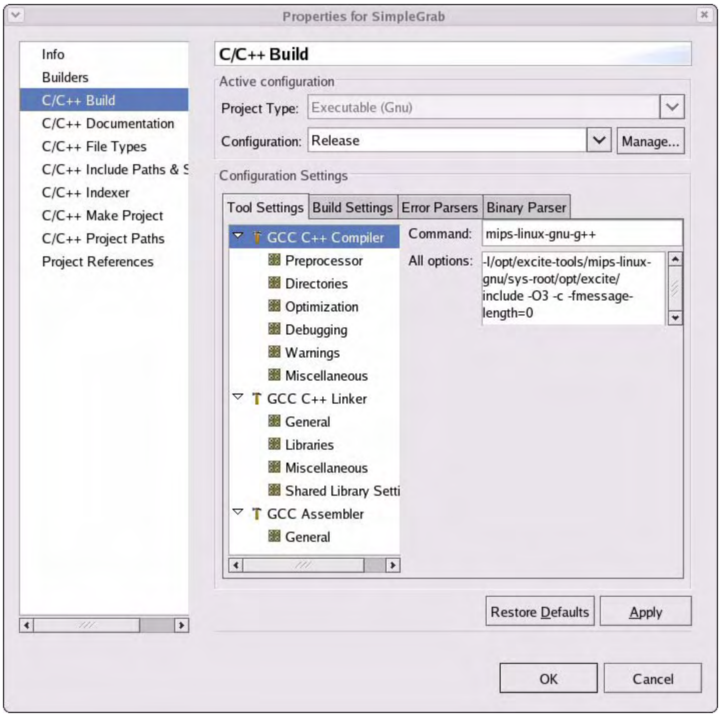
An outline is not available.

0 items

C-Build [SimpleGrab]

```
Building target: SimpleGrab
mips-linux-gnu-g++ -L/opt/
excite-tools/mips-linux-gnu/
sys-root/opt/excite/lib -o
SimpleGrab
simplegrab.o    -lxcam -
lbaslercam
Finished building:
SimpleGrab
Build complete for project
SimpleGrab
```

5. Right-click the SimpleGrab folder in the Navigator pane and select Properties from the drop down list. A Properties for SimpleGrab window will appear.
6. In the Properties for SimpleGrab window:
   a) Select “C/C++ Build” from the list at the left side of the window as shown below.
   b) The settings in the window will update. When the update is complete, select “Release” from the drop down list for the Configuration entry. This will set the compiler to create a release build of the program to run on the eXcite.
   c) Click Apply.
   d) Click OK.



Properties for SimpleGrab

C/C++ Build

Active configuration

Project Type: Executable (Gnu)

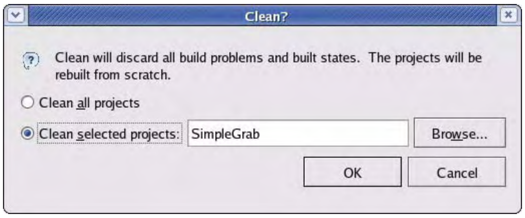
Configuration: Release

Configuration Settings

Command: mips-linux-gnu-g++

All options: -I/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/include -O3 -c -fmessage-length=0

7. Click the Project menu at the top of the Eclipse window and select Clean from the drop down list. When the Clean window appears:
   a) Make sure that “Clean selected projects” is checked as shown below.
   b) Click OK.
   c) The clean and build process will start.



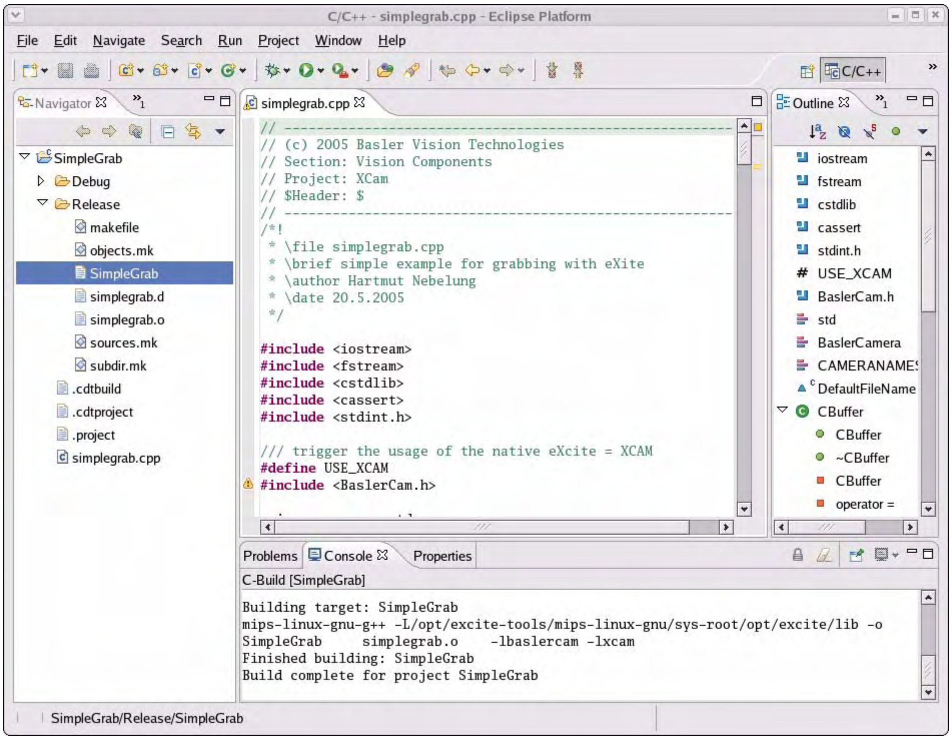
Clean?

Clean will discard all build problems and built states. The projects will be rebuilt from scratch.

○ Clean all projects

● Clean selected projects: SimpleGrab

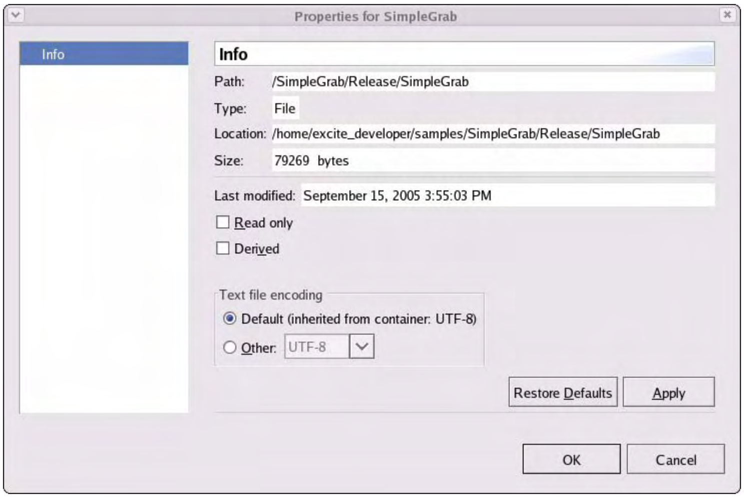
8. When the build is complete, an entry for the executable program created by the build process appears in the Release folder for the SimpleGrab project as shown below.



```
// -----------------------------------------------------------
// (c) 2005 Basler Vision Technologies
// Section: Vision Components
// Project: XCam
// $Header: $
// -----------------------------------------------------------
/*!
 * \file simplegrab.cpp
 * \brief simple example for grabbing with eXite
 * \author Hartmut Nebelung
 * \date 20.5.2005
 */

#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cassert>
#include <stdint.h>

/// trigger the usage of the native eXcite = XCAM
#define USE_XCAM
#include <BaslerCam.h>
```

```
Building target: SimpleGrab
mips-linux-gnu-g++ -L/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/lib -o
SimpleGrab     simplegrab.o    -lbaslercam -lxcam
Finished building: SimpleGrab
Build complete for project SimpleGrab
```

9. To view the properties of the SimpleGrab executable that was created:
   a) Right click on the SimpleGrab entry in the Release folder.
   b) Select properties from the drop down menu.
   c) A Properties window will open as shown below.



Info

Path: /SimpleGrab/Release/SimpleGrab

Type: File

Location: /home/excite_developer/samples/SimpleGrab/Release/SimpleGrab

Size: 79269 bytes

Last modified: September 15, 2005 3:55:03 PM

□ Read only

□ Derived

Text file encoding

● Default (inherited from container: UTF-8)

○ Other: UTF-8

3.2.2 Loading the SimpleGrab Executable onto Your eXcite

You can use a network connection to the eXcite and an FTP session to transfer an executable from your development PC to the file system in the eXcite. In this example, we will transfer the SimpleGrab executable that you created in Section 3.2.1. You can use a similar technique to transfer any file to the eXcite.

To establish an FTP connection to the eXcite, you will need to know the IP address of the eXcite you want to connect to. If you do not know the IP address of the eXcite, use one of the methods described in Section 2.1.1.5 to find it.

Once you have the IP address in hand:

1. From the command line, navigate to the directory on the development PC that contains the SimpleGrab executable file.
2. Execute the following commands to start an FTP session, log onto the eXcite, and upload the file:
   a) `ftp [eXcite IP address]` ↵

   b) `root` ↵ (This is the login for the eXcite's OS.)

   c) `root` ↵ (This is the password for the eXcite's OS.)

   d) `cd [pathname]` ↵

   (Where [pathname] is the absolute pathname to a directory on the eXcite where you want to put the file.)

   e) `put SimpleGrab` ↵ (This transfers the file from the PC to the eXcite.)

   f) `chmod 777 SimpleGrab` ↵ (This sets the access rights on the uploaded file.)
3. Execute the following command to exit the FTP session and log off of the eXcite:
   a) `bye` ↵

3.2.3 Running the SimpleGrab Executable on Your eXcite

You can use a network connection to the eXcite and a telnet session to start an executable stored in the eXcite's file system. In this example, we will run the SimpleGrab executable that you created in Section 3.2.1 and loaded into the eXcite in Section 3.2.2. You can use a similar technique to run any executable on the eXcite.

To establish a telnet connection to the eXcite, you will need to know the IP address of the eXcite you want to connect to. If you do not know the IP address of the eXcite, use one of the methods described in Section 2.1.1.5 to find it.

Once you have the IP address in hand:

1. Execute the following commands to start a telnet session, log onto the eXcite, and upload the file:
   a) `telnet [eXcite IP address]` ↵
   b) `root` ↵ (This is the login for the eXcite's OS.)
   c) `root` ↵ (This is the password for the eXcite's OS.)
   d) `cd [pathname]` ↵

      (where [pathname] is the absolute pathname to the directory on the eXcite where you uploaded the SimpleGrab executable)
   e) `./SimpleGrab` ↵ (This starts the executable.)
2. Execute the following command to exit the telnet session and log off of the eXcite:
   a) `logout` ↵

If the executable produces an output – for example, running the “SimpleGrab” sample program produces an output file called “image.pgm" – you can use an FTP session to transfer the output files from the eXcite to your development PC. For Basler sample programs that create output files, the output files are created in the working directory active when the sample program was started. If you follow the steps shown above, this will be the directory where the executable program is located.

3.2.4 Building Debug or Release Versions & Building for Different Targets

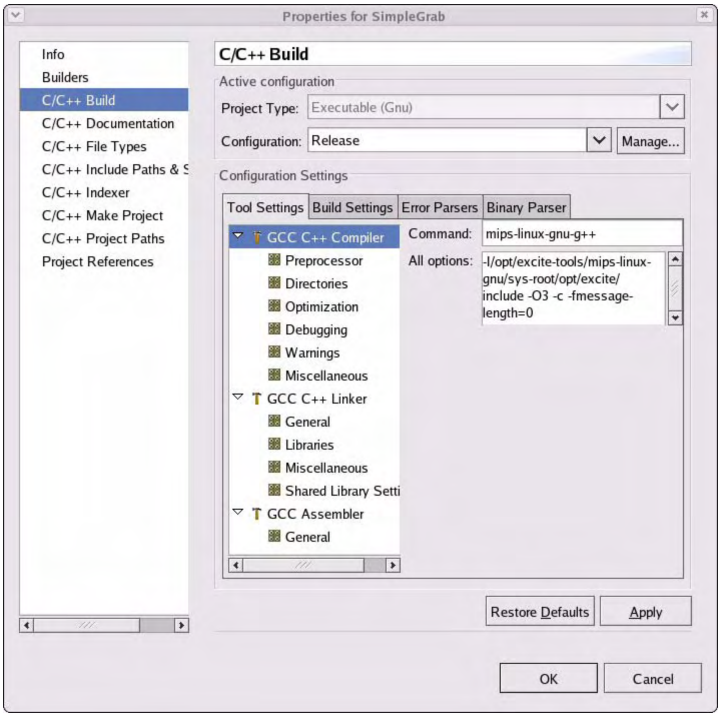
In Section 3.2.1, we built a release version of the SimpleGrab sample. In this section we will look at how to set Eclipse to produce a debug version or to produce a release version of the Simple Grab sample.

We will also use the StreamingClient sample to show how to build releases for different targets.

Building Release and Debug Versions

To select a debug or a release build configuration:

1. Start the Eclipse Integrated Development Environment (IDE) on your development PC.
2. Right-click the SimpleGrab folder in the Navigator pane and select Properties from the drop down list. A Properties for SimpleGrab window will appear.
3. In the Properties for SimpleGrab window:
   a) Select “C/C++ Build” from the list at the left side of the window as shown below.
   b) The settings in the window will update. When the update is complete, select “Release” from the drop down list for the Configuration entry. This will set the compiler to create a release build of the program to run on the eXcite. Take a close look at the compiler settings for the Command entry and for the All Options entry.
   c) Select “Debug” from the drop down list for the Configuration entry. This will set the compiler to create a debug build of the program to run on the eXcite. Take a close look at the compiler settings for the Command entry and for the All Options entry and notice how they are different from the entries for creating a release version.
   d) Notice that you can use the Tool Settings list to see the settings for the Linker and the Assembler.
   e) Finally, note that in order to apply your selected configuration so that it will be used on the next build, you must click Apply and then OK.


Properties for SimpleGrab

C/C++ Build

Active configuration

Project Type: Executable (Gnu)

Configuration: Release

Configuration Settings

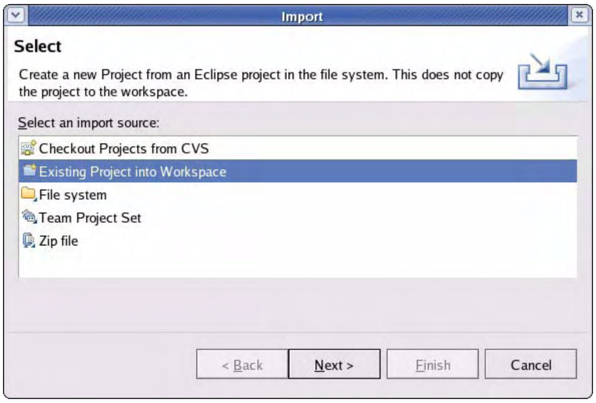
Command: mips-linux-gnu-g++

All options: -I/opt/excite-tools/mips-linux-gnu/sys-root/opt/excite/include -O3 -c -fmessage-length=0

Building for Different Targets

The SimpleGrab sample is designed to run on the eXcite, so its target is always the processor in the eXcite. The StreamingClient sample program can be run either on the eXcite or on your development PC. So here will look at how to configure Eclipse for the two possible targets.

1. If you do not see the StreamingClient sample program in the Navigator pane of the workspace, you will need to import it.
   a) Click the File menu at the top of the Eclipse window and select Import from the drop down list.
   b) When the Import window opens, select “Existing project into workspace” as shown below and then click Next.



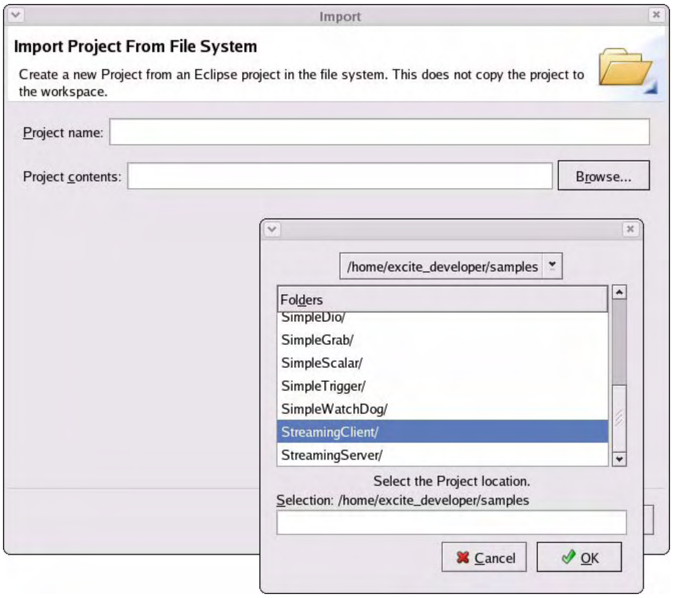
Import

Select

Create a new Project from an Eclipse project in the file system. This does not copy the project to the workspace.

Select an import source:

   c) Click Browse and then navigate to the `samples/StreamingClient` directory as shown below. Double-click StreamingClient and then click OK.



Import

Import Project From File System

Create a new Project from an Eclipse project in the file system. This does not copy the project to the workspace.

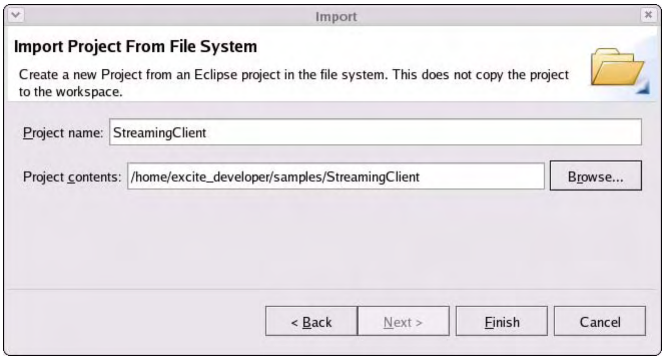
Project name:

Project contents:

Select the Project location.

Selection: /home/excite_developer/samples

d) The Import window should indicate that it will import the StreamingClient sample program project as shown below. Click Finish.



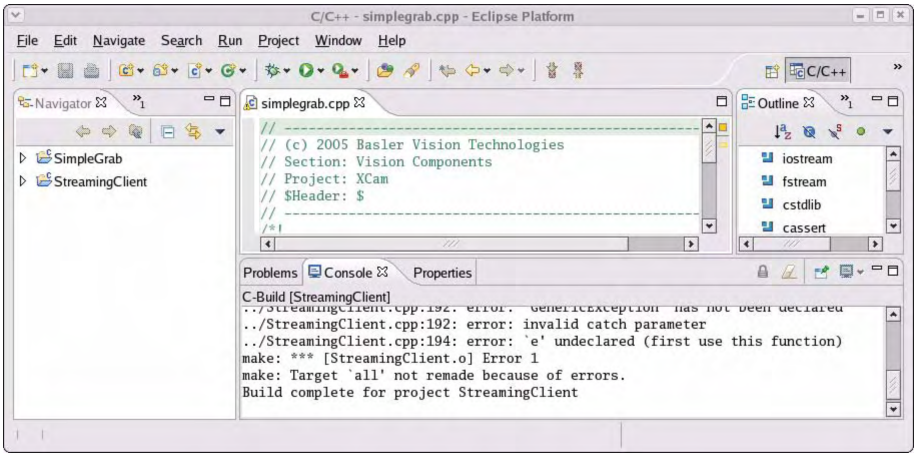
Import Project From File System

Create a new Project from an Eclipse project in the file system. This does not copy the project to the workspace.

Project name: StreamingClient

Project contents: /home/excite_developer/samples/StreamingClient

e) The StreamingClient sample program project will be imported into the workspace. An automatic build will run when the sample is imported. Once the build is complete, you should see the SimpleGrab sample program project in the Navigator pane as shown below.

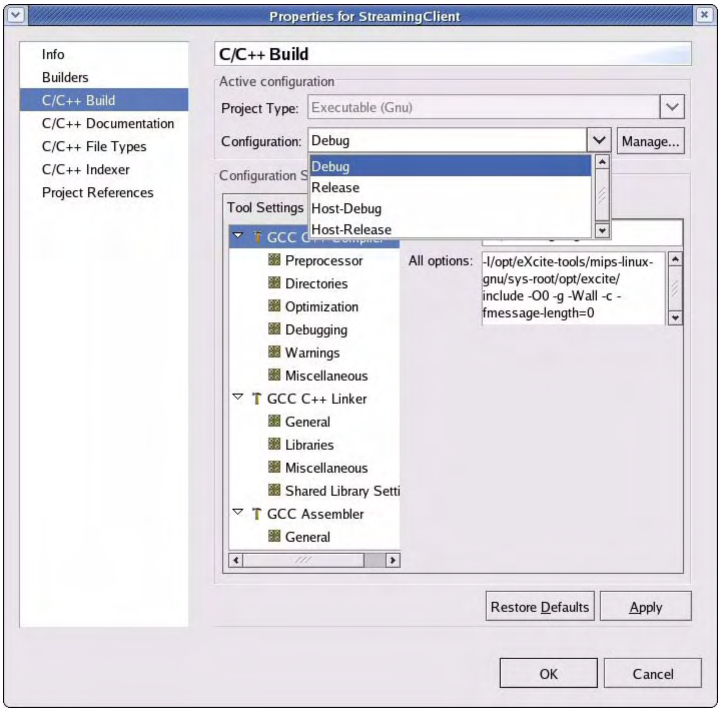


```
// -----------------------------------------------------------
// (c) 2005 Basler Vision Technologies
// Section: Vision Components
// Project: XCam
// $Header: $
// -----------------------------------------------------------
/*!
```

C-Build [StreamingClient]

```
../StreamingClient.cpp:192: error: `GenericException' has not been declared
../StreamingClient.cpp:192: error: invalid catch parameter
../StreamingClient.cpp:194: error: `e' undeclared (first use this function)
make: *** [StreamingClient.o] Error 1
make: Target `all' not remade because of errors.
Build complete for project StreamingClient
```

2. Right-click the StreamingClient folder in the Navigator pane and select Properties from the drop down list. A Properties for SimpleGrab window will appear.
3. In the Properties for StreamingClient window:
   a) Select "C/C++ Build" from the list at the left side of the window as shown below.
   b) The settings in the window will update. When the update is complete, click the Down Arrow next to the Configuration entry. Notice that there are four selections. The "Debug" and the "Release" selections are used to build debug or release versions of the program for use on the eXcite. The "Host Debug" and the "Host Release" selections are used to build debug or release versions of the program for use on your development PC. (The host PC programs are built for a Linux OS. So if you are using a WIndows development PC, you would run the programs under coLinux.)
   c) Select "Release" from the drop down list for the Configuration entry. This will set the compiler to create a release build of the program to run on the eXcite. Take a close look at the compiler settings for the Command entry and for the All Options entry.
   d) Select "Host Release" from the drop down list for the Configuration entry. This will set the compiler to create a release build of the program to run on your development PC. Take a close look at the compiler settings for the Command entry and for the All Options entry and notice how they are different from the entries for creating a release version for the eXcite.
   e) Notice that you can use the Tool Settings list to see the settings for the Linker and the Assembler.
   f) Finally, note that in order to apply your selected configuration so that it will be used on the next build, you must click Apply and then OK.



Project Type: Executable (Gnu)

Configuration: Debug

All options: -I/opt/eXcite-tools/mips-linux-gnu/sys-root/opt/excite/include -O0 -g -Wall -c -fmessage-length=0

4. Close the Properties window.

ⓘ	The compiler, linker, and assembler settings that you looked at in this section are available because they are part of the Eclipse project files included with the Basler code samples. When you write your own programs, you will need to adjust the compiler settings accordingly.

3.3 Debugging the SimpleGrab Application Running on Your eXcite

This section describes how to remotely debug an application running on the eXcite. As an example of remote debugging, we describe how to debug the SimpleGrab sample program using the GNU gdb debugger. On the eXcite, the gdbserver tool is used as a debug agent, i.e., the application to be debugged is run on the eXcite under the control of gdbserver. Gdbserver manages communication with the gdb debugger via TCP/IP.

This section assumes that you are using a development PC with a Linux operating system or that you are you are using a Windows PC with Colinux and a VNC Viewer installed and running (as described in Section 2.4) and with the Eclipse IDE available.

This section first describes the initial steps needed to use the gdb debugger. It then describes how to use the debugger from the command line and how to use the Eclipse IDE as a graphical user front end for the gdb debugger.

ⓘ The examples provided assume that you know the IP address of the eXcite you want to connect to. If you do not know the IP address of the eXcite, use one of the methods described in Section 2.1.1.5 to find it.

3.3.1 Initial Steps

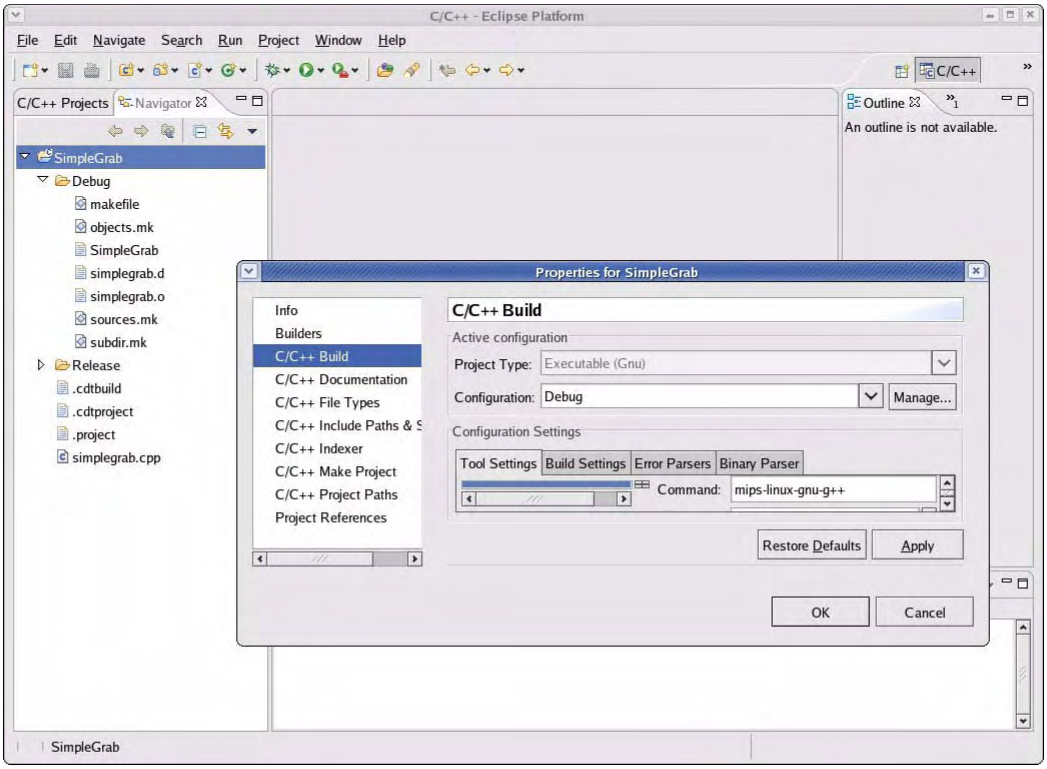
Building the Application with Debug Information

The application to be debugged must contain debug information, so ensure that your application is built with debug information by using the -g -O0 compiler switches.

If you are using Eclipse to build the sample programs:

1. Start Eclipse and open the workspace containing the SimpleGrab example program. (If necessary, import the SimpleGrab sample into the current workspace as described on page 3-4.)
2. In the Eclipse Navigator pane, right-click the SimpleGrab folder and then select Properties from the drop down menu.
3. When the Properties window opens:
   a) Select “C/C++ Build” from the list at the left side of the window as shown on the next page.
   b) The settings in the window will update. When the update is complete, change the setting for the Configuration entry to “Debug”.
   c) Click Apply.
   d) Click OK.
4. Click the Project menu at the top of the Eclipse window and select “Clean” from the drop down list. When the Clean window appears:
   a) Make sure that “Clean selected projects” is checked.
   b) Click OK.
   c) The clean and build process will start.

When the build process is finished, a SimpleGrab executable containing debug information will be located in the `[pathname]/samples/SimpleGrab/Debug` folder. Where [pathname] is the absolute path name to the directory on your development PC where the eXcite sample programs are located.



Project Type: Executable (Gnu)

Configuration: Debug

Command: mips-linux-gnu-g++

Note: You can also rebuild SimpleGrab with debug information from the command line without using Eclipse. This can be done by navigating to the SimpleGrab's Debug folder and launching `make`.

For example, you would execute the following commands to rebuild the SimpleGrab sample program:

1. `cd /[pathname]/samples/SimpleGrab/Debug ↵`

   (where [pathname] is the absolute path name to the directory on your development PC where the eXcite sample programs are located)

2. `make all ↵`

Uploading the Application to the eXcite

You can use an FTP session to copy an application to the eXcite as described in Section 3.2.2. If you want to save space in the eXcite's flash file system, you can strip the debug information from the copy of the executable that will be uploaded to the eXcite. Note that you **must not** strip the debug information from the executable that you will be using on your development system. The executable on the development system must contain the debug information. In the following examples, we assume that you will strip the debug information from the copy of the SimpleGrab executable that will be uploaded to the eXcite and that you name the stripped copy “SimpleGrab_stripped.”

To prepare a stripped copy of the executable, execute the following from the command line on your development PC:

1. `cd [pathname]/samples/SimpleGrab/Debug` ↵

   (Where [pathname] is the absolute path name to the directory on your development PC where the eXcite sample programs are located.)

2. `mips-linux-gnu-strip SimpleGrab -o SimpleGrab_stripped` ↵

To upload the stripped copy of the executable to the eXcite:

1. From the command line on your development PC, navigate to the directory where SimpleGrab_stripped is located.
2. Execute the following commands:

   a) `ftp [eXcite IP address]` ↵

   b) `root` ↵ (This is the login for the eXcite's OS.)

   c) `root` ↵ (This is the password for the eXcite's OS.)

   d) `cd [pathname]` ↵

      (Where [pathname] is the absolute path name to a directory on the eXcite where you want to put the stripped file.)

   e) `put SimpleGrab_stripped` ↵

   f) `chmod 777 SimpleGrab_stripped` ↵ (This sets the access rights on the file.)

3. Execute the following command to exit the FTP session and log off of the eXcite:

   a) `bye` ↵

Starting the Application on the eXcite Under the Control of Gdbserver

The gdb debugger that runs on the development PC and the gdbserver that runs on the eXcite communicate with each other by using a TCP/IP port. In following example, we chose to use the port number 4321 when we start the gdbserver on the eXcite. Other port numbers greater than 1024 could also be used.

To start the gdbserver on the eXcite, execute the following from the command line on your development PC:

1. `telnet [eXcite IP address]` ↵
2. `root` ↵ (This is the login for the eXcite's OS.)
3. `root` ↵ (This is the password for the eXcite's OS.)
4. `cd [pathname]` ↵

   (There [pathname] is the absolute path name to a directory on the eXcite where you loaded the SimpleGrab_stripped executable.)
5. `gdbserver :4321 SimpleGrab_stripped` ↵

   You should now see a “Listening on port 4321” message.

These commands will launch the SimpleGrab_stripped application on the eXcite under the control of the gdbserver tool. The program is stopped by gdbserver before the main function is called. GdbServer remains in this state until a debug connection is established and a command to continue program execution has been sent from the gdb debugger running on the development system to the gdbserver tool running on the eXcite.

The gdbserver terminates itself when the program to be debugged terminates or when the debug session is closed. The gdbserver program on the eXcite cannot be terminated by pressing Control-c. If you must terminate gdbserver for some reason, open a new command shell and execute the following commands:

1. `telnet [eXcite IP address]` ↵
2. `root` ↵ (This is the login for the eXcite's OS.)
3. `root` ↵ (This is the password for the eXcite's OS.)
4. `killall -9 gdbserver` ↵

Creating a Configuration File for the Gdb Debugger

To use a gdb debugger on your development PC, you need a configuration file for the debugger. The basic debugger configuration file is simply a text file containing the following lines:

```
set architecture mips:isa32
set solib-absolute-prefix /opt/excite-tools/mips-linux-gnu/sys-root
set solib-search-path /opt/excite-tools/mips-linux-gnu/lib
```

Use a text editor of your choice to create the file. In the examples on the following pages, we assume that the file is created in the user's home directory and that the file is named “mips-linux-gnu-gdbinit”. Please note, the configuration file is also necessary when using GUI front ends for the gdb such as Eclipse.

For convenience, we recommend that you also define a new gdb `connect` command by adding the following lines to the gdb configuration file:

```
define connect
target remote [eXcite IP address]:4321
end
document connect
Establish connection to remote target
end
```

This new connect command is a short cut used for establishing a connection to the application running on the eXcite. In the lines shown above, you must use the actual IP address currently assigned to the eXcite for `[eXcite IP address]`. We assume that port number 4321 will be used when you start the gdbserver. You should also use the same port number (the portion after the colon) that you used when you started the gdbserver on the eXcite (see page 3-17).

ⓘ	If your eXcite is configured for DHCP network addressing and you disconnect the eXcite from the network or remove power from the eXcite, the eXcite may have a new IP address assigned to it when you reconnect it to your network. Anytime you disconnect the eXcite or switch power off and back on, you should check the IP address and you should change the IP address in the gdb configuration file to match the address currently assigned to your eXcite.

3.3.2 Using the Gdb Debugger from the Command Line

In this section, we will walk through the steps for debugging the SimpleGrab program using the gdb debugger from the command line. We assume that you have built the SimpleGrab program with debug information, that you have created and uploaded a stripped version of the program to the eXcite, that you have started the stripped program on the eXcite under control of the gdbserver, and that you have created a gdb debugger configuration file as described on pages 3-14 through 3-18.

Now you must launch the mips-linux-gnu-gdb debugger on your development PC and step through the program. On the development PC, execute the following from the command line:

1. `cd [pathname]/samples/SimpleGrab/Debug ↵`

   (Where [pathname] is the absolute path name to the directory on your development PC where the eXcite sample programs are located.)

2. `mips-linux-gnu-gdb -x $HOME/mips-linux-gnu-gdbinit SimpleGrab ↵`

   (This starts the gdb debugger and you should see the (gdb) prompt appear.)

3. `connect ↵`

   (This command connects the debugger on the PC to the gdbserver in the eXcite.
   Note that the connect command will only work if it has been defined in the debugger configuration file as described on page 3-18.)

4. `br main ↵`

   (This command sets a breakpoint at the beginning of SimpleGrab's main function.)

5. `cont ↵`

   (This command resumes program execution until the break point is hit.)

6. `next ↵`
7. `next ↵`
8. `next ↵`

   (The three next commands are used to step over the first three statements of the main function.)

9. `cont ↵`

   (This command lets the program run until it terminates itself.)

10. `quit ↵`

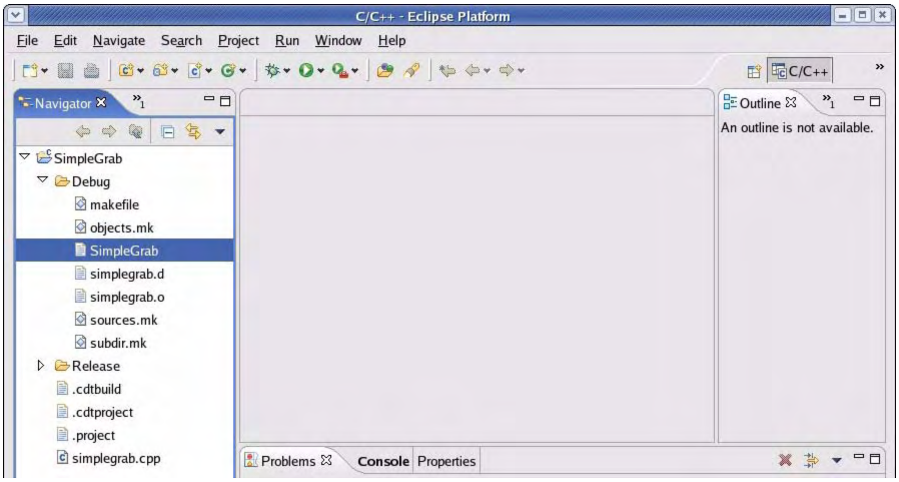
    (This command ends the gdbdebugger program.)

3.3.3 Using the Gdb Debugger with Eclipse

In this section, the necessary settings to perform remote debugging with Eclipse are described. We assume that you have built the SimpleGrab program with debug information, that you have created and uploaded a stripped version of the program to the eXcite, that you have started the stripped program on the eXcite under control of the gdbserver, and that you have created a gdb debugger configuration file as described on pages 3-14 through 3-18.

To use the gdb bugger with Eclipse:

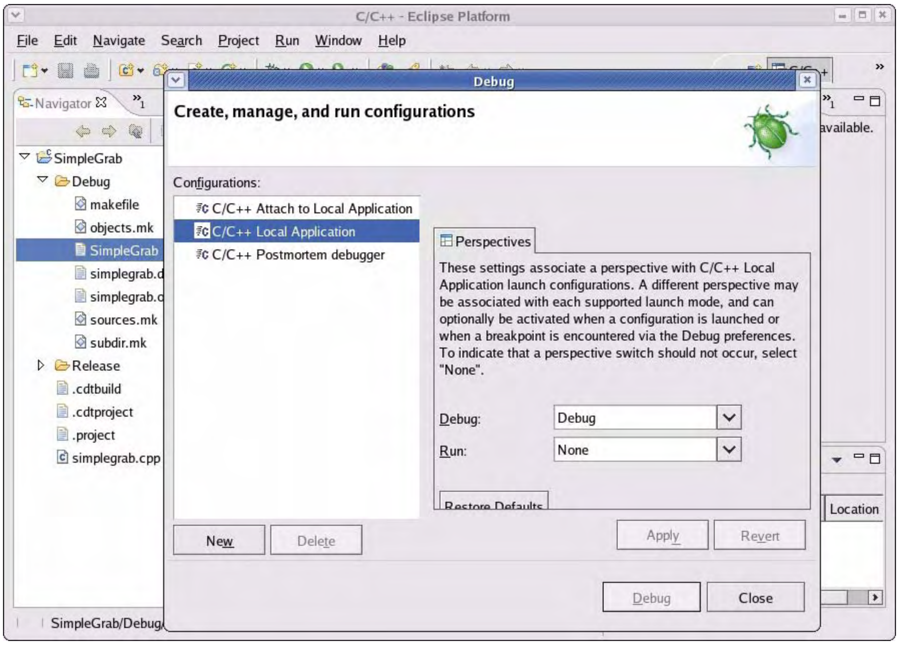
1. Make sure that you are using the C/C++ perspective (click the Window menu at the top of the Eclipse window and select Open Perspective ⇒ C/C++).
2. In Eclipse Navigator pane, right click the SimpleGrab entry beneath the Debug folder as shown below.



An outline is not available.

3. From the drop down menu that appears select Debug ⇒ Debug and a Debug configuration window will open as shown below.

   Select the “C/C++ Local Application” entry, and press the New button to create a new debug configuration.



Create, manage, and run configurations

These settings associate a perspective with C/C++ Local Application launch configurations. A different perspective may be associated with each supported launch mode, and can optionally be activated when a configuration is launched or when a breakpoint is encountered via the Debug preferences. To indicate that a perspective switch should not occur, select "None".

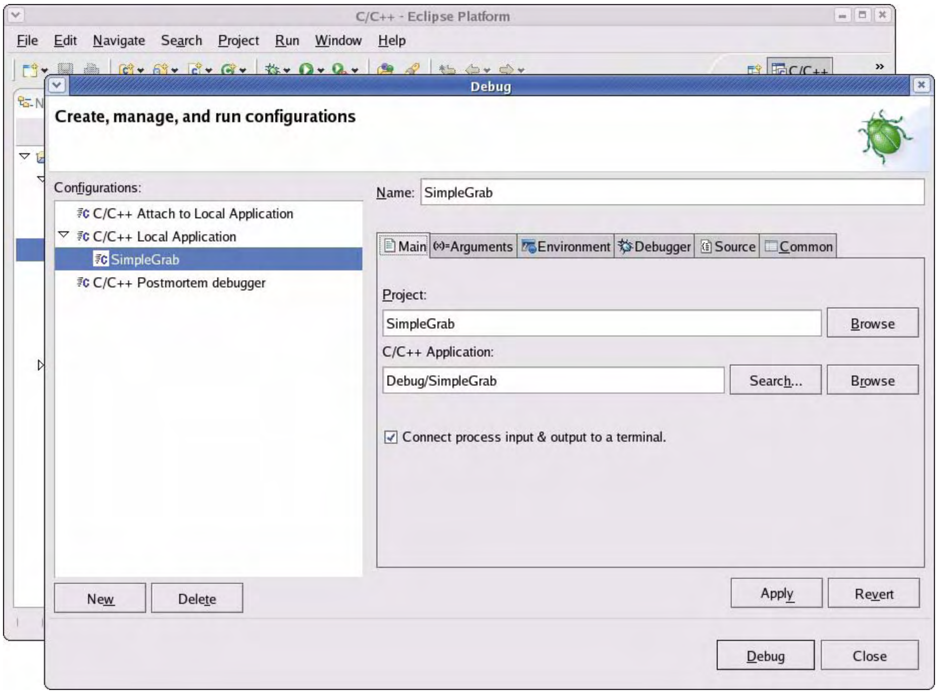
Debug: Debug

Run: None

4. The screen shots below show which settings you must enter on the Main tab and on the Debugger tab for the new debug configuration.

   When you enter the path for the “GDB command file”, make sure that you **enter the path to the folder where the gdbinit file is actually located on your development PC**.

   Make sure that you **enter the actual IP address for your eXcite and the actual port number that you are using for the gdbserver**.



C/C++ - Eclipse Platform

Debug

Create, manage, and run configurations

Configurations:

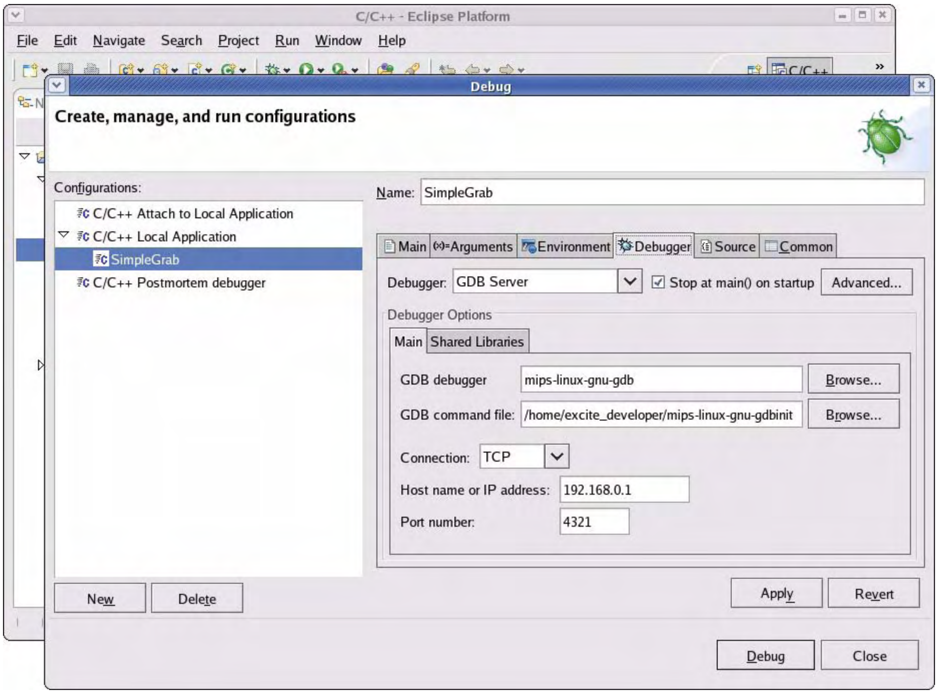
- C/C++ Attach to Local Application
- C/C++ Local Application
  - SimpleGrab
- C/C++ Postmortem debugger

Name: SimpleGrab

Project: SimpleGrab

C/C++ Application: Debug/SimpleGrab

☑ Connect process input & output to a terminal.



C/C++ - Eclipse Platform

Debug

Create, manage, and run configurations

Configurations:

- C/C++ Attach to Local Application
- C/C++ Local Application
  - SimpleGrab
- C/C++ Postmortem debugger

Name: SimpleGrab

Debugger: GDB Server

☑ Stop at main() on startup

Debugger Options

GDB debugger: mips-linux-gnu-gdb

GDB command file: /home/excite_developer/mips-linux-gnu-gdbinit

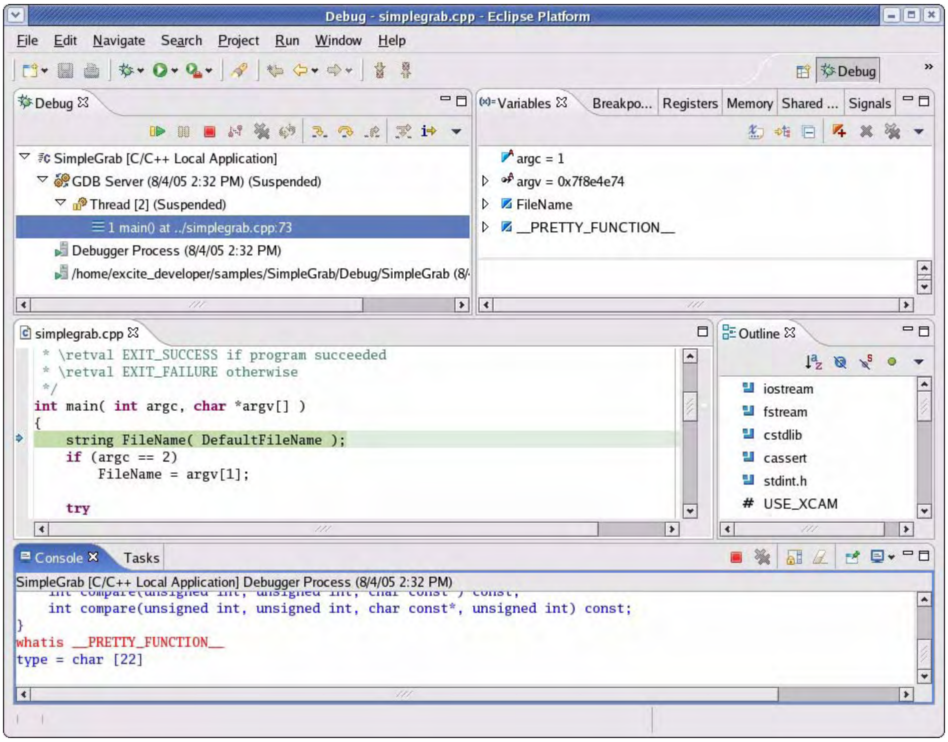
Connection: TCP

Host name or IP address: 192.168.0.1

Port number: 4321

5. When you have finished entering your settings, click Apply and then click Debug.
6. A connection to the gdbserver running on the eXcite will be established and the program execution will be resumed until the main function is reached, as shown below.

   You can now use the Eclipse IDE to debug the remote application in the same way as debugging a local application.



```
▽ SimpleGrab [C/C++ Local Application]
    ▽ GDB Server (8/4/05 2:32 PM) (Suspended)
        ▽ Thread [2] (Suspended)
            1 main() at ../simplegrab.cpp:73
    Debugger Process (8/4/05 2:32 PM)
    /home/excite_developer/samples/SimpleGrab/Debug/SimpleGrab (8/
```

```
  argc = 1
▷ argv = 0x7f8e4e74
▷ FileName
▷ __PRETTY_FUNCTION__
```

```
 * \retval EXIT_SUCCESS if program succeeded
 * \retval EXIT_FAILURE otherwise
 */
int main( int argc, char *argv[] )
{
    string FileName( DefaultFileName );
    if (argc == 2)
        FileName = argv[1];

    try
```

```
iostream
fstream
cstdlib
cassert
stdint.h
# USE_XCAM
```

```
SimpleGrab [C/C++ Local Application] Debugger Process (8/4/05 2:32 PM)
    int compare(unsigned int, unsigned int, char const*, unsigned int) const;
}
whatis __PRETTY_FUNCTION__
type = char [22]
```

Now that you have created a debug configuration for the SimpleGrab program, you can start the debugging simply by clicking the button in the Eclipse tool bar.

If you want to modify the debug configuration file for the SimpleGrab program:

1. In Eclipse Navigator pane, right click the SimpleGrab entry beneath the Debug folder.
2. From the drop down menu that appears select Debug ⇒ Debug and a Debug configuration window will open.
3. In the Configurations column, select SimpleGrab under C/C++ Local Applications.
4. Make changes as desired.
5. Click Apply.

ⓘ	If your eXcite is configured for DHCP network addressing and you disconnect the eXcite from the network or remove power from the eXcite, the eXcite may have a new IP address assigned to it when you reconnect it to your network. Anytime you disconnect the eXcite or switch power off and back on, you should check the IP address and you should change the IP address in the debug configuration file to match the address currently assigned to your eXcite.

3.4 Introduction to the Sample Programs

3.4.1 Assumptions and Basics

Basler supplies a collection of code samples with the eXcite. The code samples can be compiled into working executable programs. Each executable illustrates some basic aspect of using the eXcite.

We assume a system configuration where the code samples supplied by Basler will be compiled on a development PC and that the compiled executable programs will usually be run on the MIPS processor in the eXcite. Compiled programs designed to run on the MIPS processor in the eXcite should be copied to the eXcite processor using an FTP transfer via an Ethernet connection.

Our recommend Integrated Development Environment (IDE) is “Eclipse”. We assume that Eclipse is already installed on your development PC or that you have installed Eclipse as described in Section 2.3 or 2.4. (If you are using a development PC with a Windows operating system, we assume that you have coLinux installed and that you are running the Eclipse IDE under coLinux.)

In addition to the source code, each sample directory contains all of the necessary Eclipse project files. This allows you to open and work with the samples from the Eclipse IDE.

Some samples have one component designed to run on the eXcite and one component designed to run on the development PC. In these cases, one component must be cross-compiled for running on the MIPS processor in the eXcite and the other must be compiled for running on the PC. You should be careful to configure the Eclipse IDE so that the samples will be properly compiled for the system where they will run. For a description of how to build, load, and run sample programs, see Section 3.2.

3.4.2 Overview

Each source code directory contains the source code for one sample program along with all of the necessary Eclipse project files. For each sample, the name of the source code directory is descriptive of what the sample does or is intended for. We will generally refer to each sample by the directory name. The source code directory for each sample contains one *.cpp file – which contains the *main()* function – and may contain one or more additional *.h files specific to the sample.

The samples can divided into five broad types depending on what they illustrate:

- A simple sample called “First” to test building an executable and running it on the eXcite.
- Samples called “SimpleGrab”, “MultiGrab”, “SimpleScalar”, “SimpleTrigger”, “SimpleDio” and “SimpleWatchDog” that illustrate the basics of grabbing an image, setting parameters, using the physical digital input and output ports on the eXcite, and using the watchdog timer.
- Samples called “RsReceive” and “RsSend” that illustrate serial communication.
- Samples called “BsReceive” and “BsSend” that illustrate network communication via and Ethernet using sockets.
- Samples called “StreamingServer” and “StreamingClient” that illustrate how to send image data from the eXcite to another computer

Note: The “StreamingClient” sample program can only be run in a Linux environment. The sample program can not be run under coLinux.

Note that the first three types of samples do not use, and are not intended to illustrate, the Basler eXcite library.

Section 3.5 gives a detailed description of each sample. The detailed description is followed by the source code for the sample. We suggest that you refer to the source code as you are reading the sample description. Once you have finished reading the description and looking through the source code, we suggest that you compile, load, and run the sample.

3.5 The eXcite Sample Programs

3.5.1 The “First” Sample Program

Description

This is a very simple program that only prints a message to stdout and then quits. You can use this sample to test that your IDE is properly configured to build an executable program for the MIPS processor in the eXcite or for your development PC. You can also use it to test that a very simple executable can be loaded onto the eXcite and executed.

The sample comes in two forms – one form is designed to run on the eXcite and one form is designed to run on the development PC.

Each form must be properly compiled for the computer where it will run, i.e., one form should be cross-compiled for running on the MIPS processor in the eXcite and the other should be compiled for running on the PC. You should be careful to configure the Eclipse IDE so that the sample will be properly compiled for the system where it will run. For a description of how to build, load, and run sample programs, see Section 3.2.

To build an executable to run on the PC use either the Host-Debug or the Host-Release configuration. The executable to run on the eXcite must be built using either the Debug or the Release configuration.

If you are using a development PC with a Windows operating system, the component meant to run on the development PC must be compiled and run under Colinux.

Source Code (hello.cpp)

```
#include <iostream>
#include <cstdlib>

int main( int argc, const char *argv[] )
{
   using std::cout;
   using std::endl;

   cout << "Hello, this is really eXciting!" << endl;

   return EXIT_SUCCESS;
}
```

3.5.2 SimpleGrab

Description

This sample shows how to use the eXcite library to make the eXcite device grab an image in “one-shot” mode. In this mode, the camera will capture one (and only one) image at a command from the application program.

The sample shows how to open a “Camera” object, and then how to set some basic camera parameters and capture (grab) the image, all through operations on the Camera' object.

At the head of the simplegrab.cpp source file, note the following:

- The #define USE_XCAM and the #include <BaslerCam.h>.

  (These are explained in the “Include File” section of the API Overview in the API Reference documentation.)

- The “using namespace” statements.

  (These are explained in the “Namespaces” section of the API Overview.)

The *main()* function in simplegrab.cpp is the only place where classes and/or functions from the eXcite library are used in this sample.

Through calls to the eXcite API, the application code instructs the functions/classes in the eXcite library to put the grabbed image into a buffer provided by the application code. This is how the grabbed image becomes available to your application code.

Outside of the *main()* function, the simplegrab.cpp source file further defines a *CBuffer* local helper class with its member functions, plus two related *saveBuffer()* and *fillBuffer()* local helper functions. This class and these functions do not use anything from the eXcite library. They implement a simple, general-purpose buffer for image data as it might be typically coded in an application that uses the eXcite library. All parts of this example user buffer are documented (commented) in the sample source.

The *main()* function in simplegrab.cpp carries out the following sequence of steps:

1. Create and open a Camera object.

   This is explained in detail in the “Standard Open/close Camera Blocks” section of the API Overview.

2. Set a few camera parameters, to configure the camera so that it will perform the correct image grabbing actions.

   In this sample, the Video Mode parameter is set using the *VideoMode* data member of the Camera object and the Color Coding parameter is set so that the camera will send 8 bit images.

   The API functions to access and set the camera parameters are explained in the “Member Objects of Camera” section of the API Overview. See Section 5 of the User's Manual for an explanation of the meaning of these camera parameters. All of the camera parameters are set in a similar way.

3. Grab an image.

   The image grabbing executed in this sample uses one-shot operation (see Section 5.3.1).The image grabbing is performed in the sample by the following function calls and member object accesses:

   pCamera->PrepareGrab();

   pCamera->QueueBuffer(...);

```
pCamera->OneShot = true;
pCamera->WaitForBuffer( ... );
pCamera->FinishGrab();
```

The four member functions of the Camera object used here are described in the “Member Functions of Camera” section of the API Overview in the API Reference documentation. The *OneShot* member object is described in the “Member Objects of Camera” section of the Overview.

The two functions:

```
pCamera->PrepareGrab();
pCamera->FinishGrab();
```

initialize and de-initialize internal data structures and resources in the Camera object used for image grabbing. *PrepareGrab()* must be called before grabbing an image. *FinishGrab()* should be called after you are done grabbing an image to release all of the resources allocated by *PrepareGrab*.

The image grabbing itself is carried out by:

```
pCamera->QueueBuffer( ... );
pCamera->OneShot = true;
pCamera->WaitForBuffer( ... );
```

The address and size of the user buffer are passed to the Camera object by means of the *QueueBuffer()* call. This makes a user buffer available to the eXcite library where it can write the grabbed image. From this point, until the *WaitForBuffer()* call returns successfully, the user buffer is “owned” by the Camera object. During this time, the application code should not access the buffer.

The *pCamera->OneShot = true;* statement right after the *QueueBuffer()* call does two things. First, it tells the eXcite to use one-shot mode. Second, it issues the command to the eXcite to start the actual process of grabbing one image, i.e., it commands the camera to start an exposure.

The call to *WaitForBuffer()* then waits for the image grabbing to complete, i.e., it blocks until the image transfer from the sensor into the user buffer is complete. Note the timeout value passed to the *WaitForBuffer()* call, and note how the return value of the *BufferStatus* type of the *WaitForBuffer()* function is inspected to see whether the image grab was executed successfully (and within the timeout).

On successful completion, the *ppBuffer* parameter of the *WaitForBuffer* function returns the address of the user buffer where the grabbed image was written. The value returned through *ppBuffer* is the same user buffer address value that was passed earlier to the *QueueBuffer()* call.

The *ppUser* parameter of the *WaitForBuffer()* function returns the same *pUser* pointer value that was passed earlier to the *QueueBuffer()* call. The eXcite library only passes through the exact *pUser* pointer value; it doesn't access or modify what the pointer points to in any way.

The functionality of the *pUser* pointer passed into *QueueBuffer()* and returned again by *WaitForBuffer()* is provided by the eXcite library to offer the application programmer a way to attach an arbitrary additional user object to a buffer address passed into *QueueBuffer().*

The user object can be anything. For example, it might be a pointer to a (user-instantiated) string containing some identification that you want to attach to the buffer (or image). This usage possibility is shown in the sample code. Another useful usage possibility is that the API user can pass as the *pUser* value a pointer to an instance of a user-defined class that manages the user buffer. That usage possibility is shown in the MultiGrab sample (see page 3-32).

The application programmer need not use this *pUser* functionality in *QueueBuffer()* and *WaitForBuffer()*. If you don't use it, you can, for example, simply pass a “NULL” pointer value as the *pUser* input parameter to *QueueBuffer()*.

Interspersed between the statements in which these functions from the eXcite library are called, there are statements that instantiate the user-supplied buffer (= the *CBuffer* object) and that initialize its contents (= the *fillBuffer()* call). If the image grab succeeds, then the buffer will be overwritten with the grabbed image. At the end, the *saveBuffer()* call writes the contents of the user buffer to a file as a PGM (Portable Grey Map) image named “image.pgm.” The file is created in (written to) the current working directory on the Linux system on the eXcite.

4. Close and destroy the Camera object. This is explained in the “Standard Open/close Camera Blocks” section of the API Overview in the API Reference documentation.

Note how all of the points in the application code where classes from the eXcite library are used (including instantiation of classes) are put inside of a “try...catch” block. This is explained in the “Exceptions” section in the API Overview.

For a fuller and more general explanation of the eXcite library features used in this sample, please refer to the “Using the DeviceManager” and “Using the Camera Object” sections in the API Overview.

Source Code (simplegrab.cpp)

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cassert>
#include <stdint.h>

/// trigger the usage of the native eXcite = XCAM
#define USE_XCAM
#include <BaslerCam.h>

using namespace std;
using namespace BaslerCamera;
using namespace CAMERANAMESPACE;
// -----------------------------------------------------------
// constants
// -----------------------------------------------------------
/// the default filename
const string DefaultFileName( "image.pgm" );
// -----------------------------------------------------------
// local types
// -----------------------------------------------------------
class CBuffer
{
   public:
      CBuffer( uint32_t width, uint32_t height );
      ~CBuffer();
   private:
      CBuffer( const CBuffer& );
      CBuffer& operator=( const CBuffer& );
   public:
      uint32_t Width() const;
      uint32_t Height() const;
      uint32_t BufferSize() const;
      void* Buffer() const;
      uint8_t* Buffer();
   private:
      uint8_t* m_pBuffer;
      const uint32_t m_Width;
      const uint32_t m_Height;
};
// ---------------------------------------------------------------
// local functions
// ---------------------------------------------------------------
static bool saveBuffer( const string& FileName, const CBuffer& Buffer );
static void fillBuffer( CBuffer& Buffer );

// ---------------------------------------------------------------
// int main( int argc, char *argv[] )
// ---------------------------------------------------------------
/*!
 * \brief main entry point
 * \param argc number of arguments
 * \param argv vector of c-string arguments
 *
 * \retval EXIT_SUCCESS if program succeeded
 * \retval EXIT_FAILURE otherwise
 */
int main( int argc, char *argv[] )
{
   string FileName( DefaultFileName );
   if (argc == 2)
      FileName = argv[1];

   try
   {
      //
      // STEP 1 (Create and open Camera object)
      //

      // create first availble camera
      //
      DeviceManager &dm = DeviceManager::GetInstance();
      DeviceManager::DeviceInfoList_t lstDevices(
         dm.EnumerateDevices( Camera::DeviceTypeId ) );
      // check if we found any devices
      if (0 == lstDevices.size())
         throw RUNTIME_EXCEPTION( "No devices found" );
      // let the device manager create the first device
      Camera *pCamera = dynamic_cast<Camera*>( dm.CreateDevice( *lstDevices.begin() ) );
      if (NULL == pCamera)
         throw RUNTIME_EXCEPTION( "Device isn't a camera" );

      cout << "Using device "
         << pCamera->GetDeviceInfo().GetFriendlyDeviceName() << endl;

      // open camera
      //
      pCamera->Open();

      //
      // STEP 2 (Parameterize camera)
      //
      pCamera->VideoMode.SetValue( CEnumeration_VideoModeEnums::VideoMode_VideoMode0 );
      pCamera->ColorCoding.SetValue( CEnumeration_ColorCodingEnums::ColorCoding_Mono8 );

      //
```

```
// STEP 3 (Grab one image)
//

// prepare for image acquisition
//
pCamera->PrepareGrab();

// allocate memory
const unsigned width_px( pCamera->Width() );
const unsigned height_px( pCamera->Height() );

// create a buffer
CBuffer aBuffer( width_px, height_px );

 // fill the buffer
fillBuffer( aBuffer );

const char info[] = "MyVeryFirstImage";
//  grab an image
pCamera->QueueBuffer( aBuffer.BufferSize(), aBuffer.Buffer(), (void*)&info );
pCamera->OneShot = true;

void *pBuffer=NULL, *pUser=NULL;
IInDataStream::BufferStatus result = pCamera->WaitForBuffer( &pBuffer, &pUser,
2000UL );
switch (result)
{
   case IInDataStream::bsOk:
       clog << "Grabbed buffer" << endl;
       assert( aBuffer.Buffer() == pBuffer );
       assert( info == pUser );
      break;
   case IInDataStream::bsTimeOut:
      cerr << "Timeout occurred" << endl;
      break;
   case IInDataStream::bsCancelled:
      cerr << "Buffer cancelled" << endl;
      break;
   case IInDataStream::bsError:
      cerr << "Got Buffer with error status" << endl;
   default:
      cerr << "WaitForBuffer returned unexpected value"
         << hex << "0x" << (unsigned int) result << endl;
      break;
}

// device is not required anymore
pCamera->FinishGrab();

//
// STEP 4 (Close and destroy Camera object)
//

// close the device
dm.DestroyDevice( pCamera );

// save buffer
if (saveBuffer( FileName, aBuffer ))
{
   cout << "Saved image as " << FileName << endl;
}
else
{
   cerr << "SimpleGrab failed to save image in "
       << FileName << endl;
```

```
        }
    }
    catch( exception &e )
    {
        cerr << "SimpleGrab failed because " << e.what();
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```

3.5.3 MultiGrab

Description

This sample illustrates how to use the eXcite library in an application program to perform continuous image capturing on the eXcite.

When “continuous-shot” operation is initiated by the application program, the eXcite starts a process of continuously exposing images and transmitting them to the frame buffer and from there into the user buffers of the application program. The exposure of the second and subsequent images is started automatically by the camera and the camera will continue exposing images until the application program issues a command to stop continuous-shot operation.

The code for this sample is essentially identical to that of the SimpleGrab sample, except for the following:

- At the point where the SimpleGrab sample issued a one-shot command via the *OneShot* member object of the “Camera" object, the MultiGrab sample instead issues a continuous-shot command via the *ContinuousShot* member object of Camera. In the multigrab.cpp source, this is the statement:

  pCamera->ContinuousShot = true;

  This statement does two things. First, it tells the eXcite to use continuous-shot mode. Second, it issues the command to the eXcite to start exposing the first image.

- After continuous-shot operation has been started this way, the camera portion of the eXcite will continue exposing images until continuous-shot operation is stopped. That is, starting continuous-shot operation causes a stream of images to originate from the camera portion of the eXcite. The receiver of this stream of images is the application program.

  Each image in this stream is transferred from the camera portion of the eXcite into a user buffer instantiated by the application program. Since the image stream runs continuously, to avoid losing image data there must always be a user buffer ready and waiting at the receiving end where the incoming image data can be received. To ensure that this is the case, the application program must enqueue a number of user buffers into the Camera object before the continuos-shot operation is started. The sample shows how to do this. Just before the *pCamera->ContinuousShot = true;* statement, it instantiates 10 user buffers and passes each of them into the Camera' object by means of a call to the *QueueBuffer()* function.

  Passing (enqueueing) a user buffer into the Camera' object by means of the *QueueBuffer()'* function makes the user buffer available to the eXcite library to write received image data. Until the user buffer is returned to the application program by a subsequent *WaitForBuffer()'* call, the enqueued user buffer is "owned" by the Camera' object and the application program should not access it.

- After enqueueing a number of user buffers, continuous-shot operation can be started. Immediately after continuous-shot operation begins, the Camera object starts receiving images into the enqueued user buffers. To retrieve user buffers filled with image data into the application program, the application program calls the *Wait For Buffer()* function.

  The *WaitForBuffer()* call waits (blocks) until one of the buffers enqueued in the Camera object has been filled with image data and then returns that buffer to the application program. This releases the user buffer to the application program again. The application program can now do whatever image processing it wants on the image in the user buffer. When it is finished processing the image and no longer needs the user buffer, the application program should immediately pass (re-enqueue) the buffer into the Camera object so that the Camera object can use the buffer to receive another image.

While continuous-shot operation is running, the application program continuously repeats the following sequence of steps:

pCamera->WaitForBuffer(...); //Retrieve user buffer

... do image processing on image in user buffer ...

p*Camera->QueueBuffer(...);* //Re-enqueue user buffer

In the sample, we see these three steps being executed inside the “do...while" loop. (The sample doesn't actually do any image processing, it merely performs a "dummy" action of printing a minimalistic progress marker to stdout, then immediately re-enqueues the buffer.)

The sample quits the loop after 100 images have been retrieved. Once continuous-shot operation has started, "grabbing" images is, on the side of the application program, a fundamentally passive process: The timing of the individual exposures and the transfer of the images from sensor into user buffers is controlled by the camera and the application program just waits for the consecutive buffers with image data to become available to it.

- After the loop, near the end of the *main()* function, the sample application program tells the eXcite to stop exposing and transmitting images. This is done by the statement

  pCamera->ContinuousShot = false;

  Before exiting, any program that starts continuous-shot operation should ensure that it also stops continuous-shot operation. Otherwise, the camera portion of the eXcite may be left in a state where it is still continuing to expose images. In the sample code, you can find the *pCamera->ContinuousShot = false* statement just before the *FinishGrab()* call. Despite what its name might indicate, the *FinishGrab()* function has nothing to do with stopping the process of image grabbing. The only thing that this function does is to de-initialize the internal data structures and resources used by the Camera object that were initialized by the *PrepareGrab()* call.

The final thing that you should note in the MultiGrab sample code is its use of the *pBuffer* and *pUser* input parameters of the *QueueBuffer()* function and the *ppBuffer* and *ppUser* output parameters of the *WaitForBuffer()* function.

pBuffer is a pointer to a user buffer, and *pUser* is a pointer to any user object.

pBuffer

As illustrated in the multiGrab sample, before the first call to *WaitForBuffer()* is made, you are allowed to call *QueueBuffer()* multiple times with different buffer addresses and *pUser* pointer values. A subsequent sequence of *WaitForBuffer()* calls returns the buffer addresses and pUser values again and it is guaranteed to return them in the same order (over time) as they were passed in via the *QueueBuffer()* calls.

The *pUser* functionality of *QueueBuffer()* and *WaitForBuffer()* is especially useful to application programmers in the multi-buffer case – and above all in multi-threading applications –because it saves the user the trouble of programming his own administration to track which buffer returned by which *WaitForBuffer()* call corresponds to which buffer passed to which *QueueBuffer()* call.

In the SimpleGrab sample, *QueueBuffer()* and *WaitForBuffer()* are each called only once, so in that case it's completely obvious which buffer address and which *pUser* pointer value are going to be returned through the *ppBuffer* and *ppUser* output parameters of *WaitForBuffer()*.

In general, when working with multiple (user) buffers, we recommend that application code always use the buffer address and *pUser* pointer value to access the captured image(s) and should not make assumptions about which user buffer will be returned by which call to *WaitForBuffer()*.

pUser

The Camera object keeps the *pBuffer* and *pUser* values passed into it through one *QueueBuffer()* call together as a pair. The same pair of values is returned in a later *WaitForBuffer()* call via its *ppBuffer* and *ppUser* output parameters.

The eXcite library only passes through the exact *pUser* pointer value. It doesn't access or modify what the pointer points to in any way. The purpose of the *pUser* pointer is to allow the application program to "tag" each user buffer passed (enqueued) into the Camera object with any piece of user data. The SimpleGrab sample showed the *pUser* pointer it used to tag the enqueued user buffer with a string object.

Another, more useful and less trivial usage possibility is shown in the MultiGrab sample. There, the *pUser* value passed to *QueueBuffer()* is a pointer to an instance of the user-defined class *CBuffer* that manages the user buffer. That is, in the *QueueBuffer()* call, the *pUser* pointer value passed into the function points to the object managing the user buffer (= a *CBuffer* object), and the *pBuffer* pointer passed into the function points to the actual buffer (the start of the actual buffer memory area) inside that user object. This us useful because *WaitForBuffer()* will then return not only a pointer to the start of the buffer memory area inside the *CBuffer* object, but also a pointer to the *CBuffer* object itself. This allows the application program to access (operate on) a buffer released by *WaitForBuffer()* via the methods of the user *CBuffer* class.

Source Code (multigrab.cpp)

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cassert>
#include <stdint.h>
#include <vector>


/// trigger the usage of the native eXcite = XCAM
#define USE_XCAM
#include <BaslerCam.h>

using namespace std;
using namespace BaslerCamera;
using namespace CAMERANAMESPACE;
// ------------------------------------------------------------
// constants
// ------------------------------------------------------------
/// the default filename
const string DefaultFileName( "image.pgm" );
// ------------------------------------------------------------
// local types
// ------------------------------------------------------------
class CBuffer
{
   public:
      CBuffer( uint32_t width, uint32_t height );
      ~CBuffer();
      CBuffer( const CBuffer& );
   private:
      CBuffer& operator=( const CBuffer& );
   public:
      uint32_t Width() const;
      uint32_t Height() const;
      uint32_t BufferSize() const;
      void* Buffer() const;
      uint8_t* Buffer();
   private:
      uint8_t* m_pBuffer;
      const uint32_t m_Width;
      const uint32_t m_Height;
};
// ------------------------------------------------------------
// local functions
// ------------------------------------------------------------
static bool saveBuffer( const string& FileName, const CBuffer& Buffer );
static void fillBuffer( CBuffer& Buffer );

// ------------------------------------------------------------
// int main( int argc, char *argv[] )
// ------------------------------------------------------------
/*!
 * \brief main entry point
 * \param argc number of arguments
 * \param argv vector of c-string arguments
 *
 * \retval EXIT_SUCCESS if program succeeded
 * \retval EXIT_FAILURE otherwise
 */
int main( int argc, char *argv[] )
{
   string FileName( DefaultFileName );
   if (argc == 2)
      FileName = argv[1];

   try
   {
      //
      // create first availble camera
      //
      DeviceManager &dm = DeviceManager::GetInstance();
      DeviceManager::DeviceInfoList_t lstDevices(
         dm.EnumerateDevices( Camera::DeviceTypeId ) );
      // check if we found any devices
      if (0 == lstDevices.size())
         throw RUNTIME_EXCEPTION( "No devices found" );
      // let the device manager create the first device
      Camera *pCamera = dynamic_cast<Camera*>( dm.CreateDevice( *lstDevices.begin() ) );
      if (NULL == pCamera)
         throw RUNTIME_EXCEPTION( "Device isn't a camera" );

      cout << "Using device "
         << pCamera->GetDeviceInfo().GetFriendlyDeviceName() << endl;

      //
      // open and parameterize camera
      //
      pCamera->Open();
      pCamera->VideoMode.SetValue( CEnumeration_VideoModeEnums::VideoMode_VideoMode0 );
        pCamera->ColorCoding.SetValue(CEnumeration_ColorCodingEnums::ColorCoding_Mono8 );
      //
      // prepare for image acquisition
      //
      pCamera->PrepareGrab();

      //
      // allocate memory
      //
      const unsigned width_px( pCamera->Width() );
      const unsigned height_px( pCamera->Height() );
      // create ten buffers in a standard container
      vector<CBuffer> vBuffer( 10, CBuffer( width_px, height_px ) );
      // ---------------------------------------------------
      // Continuously grabbing is a process that consists of
      // three parts
      // a) turning the camera on and
      // b) providing an image buffer for storing the image data and
      // c) receiving the buffer, processing and
      // requeueing it.
      //
      // ---------------------------------------------------


      //  queue in the image bufferss (b)
      for (vector<CBuffer>::iterator it = vBuffer.begin(); it!=vBuffer.end(); ++it)
      {
         pCamera->QueueBuffer( it->BufferSize(), it->Buffer(), &(*it) );
      }

      // turn grabbing on (a)
      pCamera->ContinuousShot = true;

      int NumGrabs=0;
      const int MaxGrabs = 100;

      IInDataStream::BufferStatus result;
      do {
         void *pBuffer=NULL, *pUser=NULL;
         result = pCamera->WaitForBuffer( &pBuffer, &pUser, 2000UL );
         switch (result)
         {
            case IInDataStream::bsOk:
                CBuffer *pb = reinterpret_cast< CBuffer* >(pUser);
                clog << ".";
                if (NumGrabs % 50 == 0) clog << endl;
                //
                // do some processing and then
                // requeue the buffer
                pCamera->QueueBuffer( pb->BufferSize(), pb->Buffer(), pUser );
               break;
            case IInDataStream::bsTimeOut:
               cerr << "Timeout occurred" << endl;
               break;
            case IInDataStream::bsCancelled:
               cerr << "Buffer cancelled" << endl;
               break;
            case IInDataStream::bsError:
               cerr << "Got Buffer with error status" << endl;
            default:
               cerr << "WaitForBuffer returned unexpected value"
                  << hex << "0x" << (unsigned int) result << endl;
               break;
         }

      } while (result == IInDataStream::bsOk && ++NumGrabs < MaxGrabs);

      // device is not required anymore
      pCamera->ContinuousShot = false;
      pCamera->FinishGrab();

      // close the device
      dm.DestroyDevice( pCamera );
```

```
   }
   catch( exception &e )
   {
      cerr << "MultiGrab failed because " << e.what();
      return EXIT_FAILURE;
   }

   return EXIT_SUCCESS;
}
```

3.5.4 SimpleScalar

Description

This is a very simple sample program that shows how to use the Basler eXcite library to work with a “scalar” camera parameter, i.e., a parameter that can be set to some value within a defined range of integers.

As you look through source code, pay special attention to the function calls:

pCamera->Shutter.GetMin( );

pCamera->Shutter.GetMax( );

pCamera->Shutter.GetInc( );

pCamera->Shutter.GetValue( );

pCamera->Shutter.GetRepresentation( );

The *GetMin*, *GetMax*, *GetInc*, and *GetValue* calls will return the minimum allowed Shutter parameter setting, the maximum allowed Shutter parameter setting, the increment for the Shutter parameter setting, and the current value for the Shutter parameter setting respectively. The *GetRepresentation* setting will return information about whether this parameter should be represented on a linear scale or a logarithmic scale in a GUI interface.

Also note the:

pCamera->Shutter.SetValue( );

function call. This call is used to set the shutter parameter to a new value.

Source Code (simplescalar.cpp)

```
/// trigger the usage of the native eXcite = XCAM
#define USE_XCAM
#include <BaslerCam.h>
using namespace BaslerCamera;
using namespace CAMERANAMESPACE;
using namespace GenApi;



// ------------------------------------------------------------
// int main( int argc, char *argv[] )
// ------------------------------------------------------------
/*!
 * \brief main entry point
 * \param argc number of arguments
 * \param argv vector of c-string arguments
 *
 * \retval EXIT_SUCCESS if program succeeded
 * \retval EXIT_FAILURE otherwise
 */
int main( int argc, char *argv[] )
{
   if (argc > 1)
      clog << "Ignoring parameters" << endl;

   try
   {
      //
      // create first availble camera
      //
      DeviceManager &dm = DeviceManager::GetInstance();
      DeviceManager::DeviceInfoList_t lstDevices(
          dm.EnumerateDevices( Camera::DeviceTypeId ) );
      // check if we found any devices
      if (0 == lstDevices.size())
         throw RUNTIME_EXCEPTION( "No devices found" );
      // let the device manager create the first device
      Camera *pCamera = dynamic_cast<Camera*>( dm.CreateDevice( *lstDevices.begin() ) );
      if (NULL == pCamera)
         throw RUNTIME_EXCEPTION( "Device isn't a camera" );

      cout << "Using device "
      << pCamera->GetDeviceInfo().GetFriendlyDeviceName() << endl;

      //
      // open and parameterize camera
      //
      pCamera->Open();

      //
      // short version
      //
      const int64_t shutter = pCamera->Shutter();
      pCamera->Shutter = shutter;


      //
      // elaborate version
      //

      // get current value
      const int64_t minimum = pCamera->Shutter.GetMin();
      const int64_t maximum = pCamera->Shutter.GetMax();
      const int64_t increment = pCamera->Shutter.GetInc();
      const int64_t value = pCamera->Shutter.GetValue();
      ERepresentation  representation =  pCamera->Shutter.GetRepresentation();

      // set new value
      const int64_t newvalue = minimum + (maximum - minimum) / 5;
      pCamera->Shutter.SetValue( newvalue );


      cout << "Current shutter value is " << value << endl;
      cout << "Range is [" << minimum << " - " << maximum<< "]"
      << " with an Increment of " << increment << endl
      << " use representation " << representation << endl;


   }
   catch (exception &e)
   {
      cerr << "Exception occurred:" << endl
      << e.what() << endl;
      exit( EXIT_FAILURE );
   }

   return EXIT_SUCCESS;
}
```

3.5.5 SimpleTrigger

Description

This sample shows how to use the Basler eXcite library to make the eXcite grab an image in “external trigger/one-shot” mode. In this mode, a one-shot function call prepares the camera to react to an external trigger signal and the camera will start grabbing (exposing) an image when the external trigger signal changes state.

The sample uses techniques to open a “Camera” object and to create and manage buffers that are very similar to those used in the SimpleGrab sample (see Section 3.5.2).

As you look through source code, pay special attention to the four function calls:

pCamera->TriggerMode = ... ;

pCamera->TriggerPolarity = ... ;

pCamera->TriggerSource = ...;

pCamera->TriggerEnable = ...;

that appear after the ENABLING EXTERNAL TRIGGER ON INPUT PORT 0 comment. These four calls enable external triggering and set the eXcite to use a high active external sync signal applied to input port 0.

Also note the:

pCamera->OneShot = ...;

function call that appears immediately after the four calls mentioned above. This one-shot call sets up the camera to react to a change in the external trigger signal. Once this call is issued, the camera will begin grabbing an image the next time the external trigger signal on input port 0 goes high.

Source Code (simpletrigger.cpp)

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cassert>
#include <string>
#include <stdint.h>

#define USE_XCAM
#include <BaslerCam.h>
using namespace BaslerCamera;
using namespace CAMERANAMESPACE;
using namespace std;

// ------------------------------------------------------------
// constants
// ------------------------------------------------------------
/// the default filename
const string DefaultFileName( "image.pgm" );
// ------------------------------------------------------------
// local types
// ------------------------------------------------------------
class CBuffer
{
```

```
public:
   CBuffer( uint32_t width, uint32_t height );
   ~CBuffer();
private:
   CBuffer( const CBuffer& );
   CBuffer& operator=( const CBuffer& );
public:
   uint32_t Width() const;
   uint32_t Height() const;
   uint32_t BufferSize() const;
   void* Buffer() const;
   uint8_t* Buffer();
private:
   uint8_t* m_pBuffer;
   const uint32_t m_Width;
   const uint32_t m_Height;
};

// ------------------------------------------------------------
// local functions
// ------------------------------------------------------------
static bool saveBuffer( const string& FileName, const CBuffer& Buffer );
static void fillBuffer( CBuffer& Buffer );


// ------------------------------------------------------------
// int main( int argc, char *argv[] )
// ------------------------------------------------------------
/*!
 * \brief main entry point
 * \param argc number of arguments
 * \param argv vector of c-string arguments
 *
 * \retval EXIT_SUCCESS if program succeeded
 * \retval EXIT_FAILURE otherwise
 */
int main( int argc, char *argv[] )
{
   if (argc > 1)
      clog << "Ignoring parameters" << endl;
   string FileName( DefaultFileName );

   try
   {
      //
      // create first availble camera
      //
      DeviceManager &dm = DeviceManager::GetInstance();
      DeviceManager::DeviceInfoList_t lstDevices(
          dm.EnumerateDevices( Camera::DeviceTypeId ) );
      // check if we found any devices
      if (0 == lstDevices.size())
         throw RUNTIME_EXCEPTION( "No devices found" );
      // let the device manager create the first device
      Camera *pCamera = dynamic_cast<Camera*>( dm.CreateDevice( *lstDevices.begin() ) );
      if (NULL == pCamera)
         throw RUNTIME_EXCEPTION( "Device isn't a camera" );

      cout << "Using device "
      << pCamera->GetDeviceInfo().GetFriendlyDeviceName() << endl;
```

```
//
// open and parameterize camera
//
pCamera->Open();

//
// prepare for image acquisition
//
pCamera->PrepareGrab();

//
// allocate memory
const unsigned width_px( pCamera->Width() );
const unsigned height_px( pCamera->Height() );
// create a buffer
CBuffer aBuffer( width_px, height_px );

// fill the buffer
fillBuffer( aBuffer );

const char info[] = "MyVeryFirstImage";
//  grab an image
pCamera->QueueBuffer( aBuffer.BufferSize(), aBuffer.Buffer(), (void*)&info );

//
// enable external trigger on port 0
//
pCamera->TriggerMode = CEnumeration_TriggerModeEnums::TriggerMode_TriggerMode0;
pCamera->TriggerPolarity =
CEnumeration_TriggerPolarityEnums::TriggerPolarity_HighActive;
pCamera->TriggerSource
CEnumeration_TriggerSourceEnums::TriggerSource_ExTrigPort0 ;
pCamera->TriggerEnable = true;

// grab a single image
pCamera->OneShot = true;

// wait up to 2 seconds for the result
void *pBuffer=NULL, *pUser=NULL;
IInDataStream::BufferStatus result = pCamera->WaitForBuffer( &pBuffer, &pUser,
2000UL );
switch (result)
{
case IInDataStream::bsOk:
   clog << "Grabbed buffer" << endl;
   assert( aBuffer.Buffer() == pBuffer );
   assert( info == pUser );
   break;
case IInDataStream::bsTimeOut:
   cerr << "Timeout occurred" << endl;
   break;
case IInDataStream::bsCancelled:
   cerr << "Buffer cancelled" << endl;
   break;
case IInDataStream::bsError:
   cerr << "Got Buffer with error status" << endl;
default:
   cerr << "WaitForBuffer returned unexpected value"
   << hex << "0x" << (unsigned int) result << endl;
   break;
}

// device is not required anymore
pCamera->FinishGrab();
```

```
      // optional: disable external trigger
      pCamera->TriggerEnable = false;

      // close the device
      dm.DestroyDevice( pCamera );

      // save buffer
      if (result == IInDataStream::bsOk)
      {
         if (saveBuffer( FileName, aBuffer ))
         {
            cout << "Saved image as " << FileName << endl;
         }
         else
         {
            cerr << "SimpleTrigger failed to save image in "
            << FileName << endl;
         }

      }

   }
   catch (exception &e)
   {
      cerr << "Exception occurred:" << endl
      << e.what() << endl;
      exit( EXIT_FAILURE );
   }

   return EXIT_SUCCESS;
}
```

3.5.6 SimpleDio

Description

This sample illustrates how to use the eXcite API to operate the four digital input ports and the four digital output ports on the eXcite. For a detailed description of the I/O ports, see Sections 4.4, 5.9.8, and 5.9.9.1.

The code in the source file for the SimpleDio sample contains only a *main()* function. The structure of the body of the *main()* function is similar to that in the SimpleGrab sample. First, we let the “DeviceManager” create and open a “Camera” object for us. Second, there are follow-up operations on that Camera object. And finally, the Camera object is closed and destroyed. Also similar to SimpleGrab is the way in which all instantiations and accesses to API classes are enclosed inside of a “try...catch” block.

Inside of the *main()* function, after a Camera object is opened, the four output ports are each operated individually. Each port is operated in a different way to illustrate the different ways that the output ports can be used. All the API calls that operate the input or output ports are calls to methods of member objects of the Camera object, such as *PioOut1Src*, *PioOut1Invert*, and *PioInput*.

See the API Reference documentation for a listing of the member objects of Camera and for a full description of the methods and possible values of each of the member objects.

In the SimpleDio sample, two output ports (number 0 and 1) are set to hard values from the application program. Note how these two output ports must first be designated as “user settable” before they can be set to a hard value. The outputs are designated as user settable through the *pCamera->PioOut0Src* member object (for output port 0) and the *pCamera->PioOut1Src* member object (for output port 1). After being designated as user settable, output ports 0 and 1 are then operated as follows:

- Using the *pCamera->PioOut0Setting* member object, output port number 0 is set a to logical one. (Note that the preceding *pCamera->PioOut0Invert* member object was set to “False.” So in this case, positive logic is used.)
- Using the *pCamera->PioOut1Setting* member object, output port number 1 is set to a logical one. (Note that the preceding *pCamera->PioOut1Invert* member object was set to “True.” So in this case, negative logic is used.)

After the values of output ports 0 and 1 are set, the current value of these two output ports is read by accessing the *pCamera->PioOut0Monitor* member object (for output port 0) and the *pCamera->PioOut1Monitor* member object (for output port 1).

Port 2 is then assigned to a different internal output signal of the eXcite as follows:

- Using the *pCamera->PioOut2Src* member object, port 2 is assigned to the eXcite’s "Integrate Enabled" output signal.

Port 3 is left unassigned.

After the output ports are assigned, the current value for each port is read by accessing the *pCamera->PioOut2Monitor* member object (for output port 2) and the *pCamera->PioOut3Monitor* member object (for output port 3).

Finally, a single *pCamera->PioInput()* call is used to read the state of the eXcite’s four input ports. Note that:

pCamera->PioInput()

is not a call to a *PioInput()* function (there is no such member function in Camera). Instead it is a call to the *operator()(void)* method of the *PioInput* member object of Camera.

This *operator()(void)* function is simply an alternative to the *GetValue(void)* function and operates identically, namely, it returns the value of the parameter. In this case, that value is an integer in which the four least significant bits represent the state of the four input ports (see Section 5.9.8).

Source Code (simpledio.cpp)

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cassert>
#include <stdint.h>

/// trigger the usage of the native eXcite = XCAM
#define USE_XCAM
#include <BaslerCam.h>

// use members w/o scope operator
using namespace std;
using namespace BaslerCamera;
using namespace CAMERANAMESPACE;
using namespace GenApi;

int main( int argc, char *argv[] )
{
   try
   {
      //
      // create first availble camera
      //
      DeviceManager &dm = DeviceManager::GetInstance();
      DeviceManager::DeviceInfoList_t lstDevices(
          dm.EnumerateDevices( Camera::DeviceTypeId ) );
      // check if we found any devices
      if (0 == lstDevices.size())
         throw RUNTIME_EXCEPTION( "No devices found" );
      // let the device manager create the first device
      Camera *pCamera = dynamic_cast<Camera*>( dm.CreateDevice( *lstDevices.begin() ) );
      if (NULL == pCamera)
         throw RUNTIME_EXCEPTION( "Device isn't a camera" );

      cout << "Using device "
      << pCamera->GetDeviceInfo().GetFriendlyDeviceName() << endl;

      //
      // open and parameterize camera
      //
      pCamera->Open();


      //
      // read the all output lines
      //
      cout << "Current output state is 0x" << hex << pCamera->PioOutput() << endl;

      //
      // setup line 0
      //
      // enable user defined output
      pCamera->PioOut0Src = CEnumeration_PioOut0SrcEnums::PioOut0Src_UserSet;
      // use positive logic
      pCamera->PioOut0Invert = false;
      // set to logical 1
      pCamera->PioOut0Setting= CEnumeration_PioOut0SettingEnums::PioOut0Setting_High;
      // check the current outvalue
      cout << "Current output of line 0 is " << pCamera->PioOut0Monitor.ToString() << endl;

      //
      // setup line 1
      //
      // enable user defined output
      pCamera->PioOut1Src = CEnumeration_PioOut1SrcEnums::PioOut1Src_UserSet;
      // use negative logic
      pCamera->PioOut1Invert = true;
      // set to logical 1
      pCamera->PioOut1Setting = CEnumeration_PioOut1SettingEnums::PioOut1Setting_High;
      // check the current output value
      cout << "Current output of line 1 is " << pCamera->PioOut1Monitor.ToString() << endl;

      //
      // setup line 2
      //
      // link output to integration signal
      pCamera->PioOut2Src = CEnumeration_PioOut2SrcEnums::PioOut2Src_IntegrationEnable;
      // check the current output value
      cout << "Current output of line 2 is " << pCamera->PioOut2Monitor.ToString() << endl;

      //
      // read input lines
      //
      cout << "Current input state is 0x" << hex << pCamera->PioInput() << endl;

   }
   catch( exception &e )
   {
      cerr << "SimpleDio failed because " << e.what();
      return EXIT_FAILURE;
   }

   return EXIT_SUCCESS;
}
```

3.5.7 SimpleWatchDog

Description

This sample illustrates how to use the “Watchdog” function of the eXcite. The watchdog is a hardware circuit that resets the processor when a timeout expires.

The sample uses the Linux *<linux/watchdog.h>* and *<sys/ioctl.h>* libraries to operate the watchdog function of the processor.

The points (function calls) in the sample that should be noted are marked in the source code by the numbers (1)...(6) in the comments. These points are:

1. Open the watchdog UNIX device file. The watchdog driver is operated through its UNIX device file. The device file is opened simply with the UNIX *open()* call.
2. Check to see if the last boot was initiated by the watchdog. This is done via the *ioct(...,WDIOC_GETBOOTSTATUS,...)* function call.
3. Configure the watchdog.

   The sample shows how to first query the capabilities of the watchdog. This is done with the *ioctl(...,WDIOC_GETSUPPORT,...)* function call. The sample shows how to interpret the information that this function call returns.

   Set the watchdog timeout to 5 seconds. This is done with the *ioctl(...,WDIOC_SETTIMEOUT,...)* function call. This function call also turns on the watchdog.

After the watchdog has been configured and started, the sample code then enters a loop. The loop illustrates how the watchdog is "kept alive" by the application program repeatedly resetting it at time intervals shorter than the watchdog timeout time. The loop is iterated over 100 times. In each iteration of the loop, the program does the following:

Sleep for a duration shorter than the watchdog timeout time.

Reset the watchdog timer, thus preventing watchdog timeout.

The sample illustrates two alternative ways of resetting the watchdog timer. Application programs need only use one of these (4a or 4b):

(4a) by writing an empty string

(4b) by using the *WDIOC_KEEPALIVE ioctl* call

Inside of the loop, the watchdog is continually reset before its timeout runs out, so this loop will not trigger the watchdog.

After the loop, the sample shows the following additional operations on the watchdog:

5a. Stop the watchdog. This is done by writing the “V” "magic character" to the watchdog device file. This is interpreted by the watchdog driver as the command to stop the watchdog timer and disable the watchdog.

Normally, each application program that uses (initializes and sets into action the watchdog) should take care to stop and disable the watchdog before the program exits. By setting the *disableWatchdogOnTermination* local variable to false, the user can experiment with having the watchdog timer expire and thus having the watchdog reset the eXcite’s processor.

6. Close the device file using the standard close function.

Note: The sample uses the R*UNTIME_EXCEPTION()* macro from the Basler eXcite library. This is the only feature from the eXcite library that is used in this sample. The macro is merely a handy way to create a exception class.

Source Code (simplewatchdog.cpp)

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cassert>
#include <stdint.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/errno.h>
#include <sys/ioctl.h>
#include <linux/watchdog.h>

/// trigger the usage of the native eXcite = XCAM
#define USE_XCAM
#include <BaslerCam.h>// RUNTIME_EXCEPTION

using namespace std;

// ------------------------------------------------------------
// constants
// ------------------------------------------------------------
const string WatchdogDeviceName( "/dev/watchdog" );
// ------------------------------------------------------------
// local types
// ------------------------------------------------------------
// ------------------------------------------------------------
// local functions
// ------------------------------------------------------------

// ------------------------------------------------------------
// int main( int argc, char *argv[] )
// ------------------------------------------------------------
/*!
 * \brief main entry point
 * \param argc number of arguments
 * \param argv vector of c-string arguments
 *
 * \retval EXIT_SUCCESS if program succeeded
 * \retval EXIT_FAILURE otherwise
 */
int main( int argc, char *argv[] )
{
   int watchdogFd = -1;
   bool disableWatchdogOnTermination = true;

   if (argc > 2)
   {
      const char *pDelim = strrchr( argv[0], '/' );
      cerr << "Usage " << (char*)(pDelim ? pDelim+1 : argv[0]) << " [timeout]" << endl
      << "    timeout value in seconds,default is 5" << endl
      << endl;
      exit( EXIT_FAILURE );
   }

   try
   {
      //
      // open the watchdog (1)
      //
      watchdogFd = open( WatchdogDeviceName.c_str(), O_RDWR );
      if (watchdogFd < 0)
      {
         throw RUNTIME_EXCEPTION( "open watchdog failed: %s", strerror( errno ) );
      }
```

```
//
// check if the last boot was initiated by the watchdog (2)
//
int boot_status;
if ( ioctl( watchdogFd, WDIOC_GETBOOTSTATUS, &boot_status ) < 0 )
{
   throw RUNTIME_EXCEPTION( "get watchdog boot status: %s", strerror( errno ) );
}
cout << "boot status: last boot " << ( boot_status == WDIOF_CARDRESET ? "was" :
"wasn't" ) << " caused by watchdog" << endl;

//
// configure watchdog (3)
//

// query support
watchdog_info wdInfo;
if (ioctl( watchdogFd, WDIOC_GETSUPPORT, &wdInfo ) < 0)
{
   throw RUNTIME_EXCEPTION( "query support info failed: %s", strerror( errno ) );
}

cout << "Watchdog '" << wdInfo.identity << "':" << endl
<< "SetTimeout : " << (bool) (wdInfo.options & WDIOF_SETTIMEOUT) << endl
<< "Overheat   : " << (bool) (wdInfo.options & WDIOF_OVERHEAT) << endl
<< "Fan fault  : " << (bool) (wdInfo.options & WDIOF_FANFAULT) << endl
<< "External 1 : " << (bool) (wdInfo.options & WDIOF_EXTERN1) << endl
<< "External 2 : " << (bool) (wdInfo.options & WDIOF_EXTERN2) << endl
<< "Power fault: " << (bool) (wdInfo.options & WDIOF_POWERUNDER) << endl
<< "Card resest: " << (bool) (wdInfo.options & WDIOF_CARDRESET) << endl
<< "Over power : " << (bool) (wdInfo.options & WDIOF_POWEROVER) << endl
<< "Magic close: " << (bool) (wdInfo.options & WDIOF_MAGICCLOSE) << endl
<< "Keepalive  : " << (bool) (wdInfo.options & WDIOF_KEEPALIVEPING) << endl;

// set watchdog timeout to 5 seconds
const int timeout_s = argc <= 1 ? 5 : atoi( argv[1] );
if (ioctl( watchdogFd, WDIOC_SETTIMEOUT, &timeout_s ) < 0)
{
   throw RUNTIME_EXCEPTION( "set watchdog timeout: %s", strerror( errno ) );
}
else
{
   int s;
   if (ioctl( watchdogFd, WDIOC_GETTIMEOUT, &s ) < 0)
   {
      throw RUNTIME_EXCEPTION( "get watchdog timeout: %s", strerror( errno ) );
   }
   cout << "new timeout value is " << s << " [sec] " << endl;
}


int numIterations = 100;
do
{
   //
   // do something (...)
   //
   sleep( timeout_s/2 );
   if (numIterations % 50 == 0) clog << endl;
   clog << "." ;
```

```
            // keep alive ping (4a)
            if (write( watchdogFd, "\0", 0 ) < 0)
            {
               throw RUNTIME_EXCEPTION( "watchdog write failed : %s", strerror( errno ) );
            }

            // or using the ioctl api (4b)
            if (ioctl( watchdogFd, WDIOC_KEEPALIVE, NULL ) < 0)
            {
               throw RUNTIME_EXCEPTION( "watchdog KEEPALIVE failed : %s", strerror( errno ) );
            }

         }
         while (--numIterations > 0);


         if (disableWatchdogOnTermination)
         {
            //
            // release the watchdog by writing the magic character 'V' (5a)
            //
            if (write( watchdogFd, "V", 1 ) < 0)
            {
               throw RUNTIME_EXCEPTION( "watchdog write magic failed : %s", strerror( errno
) );
            }
         }
         //
         // and close (6)
         //
         if (close( watchdogFd ) < 0)
         {
            throw RUNTIME_EXCEPTION( "watchdog close failed : %s", strerror( errno ) );
         }


      }
      catch( exception &e )
      {
         cerr << "SimpleWatchDog failed because " << e.what();
         return EXIT_FAILURE;
      }

      return EXIT_SUCCESS;
   }
```

3.5.8 RsReceive and RsSend

Description

This set of two samples illustrates communication via the RS-232 connector.

RsReceive is a simple server program and RsSend is a client that communicates with that server.

To run these sample programs, the RS-232 port on the eXcite must be connected to the RS-232 port of your development PC.

Run either the server or the client on the processor of the eXcite and run the other component (client or server, respectively) on the PC connected with the eXcite. Each component must be properly compiled for the computer where it will run, i.e., one component should be cross-compiled for running on the MIPS processor in the eXcite and the other should be compiled for running on the PC.You should be careful to configure the Eclipse IDE so that each component will be properly compiled for the system where it will run. For a description of how to build, load, and run sample programs, see Section 3.2.

If you are using a development PC with a Windows operating system, the component meant to run on the development PC must be compiled and run under coLinux.

If your development PC has a Linux operating system, you can actually use these samples to send data between the client and the server. If your development PC has a Windows operating system, you will not be able to send data because the coLinux installation on the PC can't access the PC's physical serial port.

By default, the RS-232 serial port on the eXcite is set up to transmit console output from the Linux OS on the eXcite's processor and to accept input from your keyboard. You can't run the compiled RsReceive/RsSend sample code with the serial port configured this way. Before running the compiled sample code, you must reconfigure the port as descrbed in Section 5.9.5

Source Code (server.cpp)

```
#include <iostream>
#include <sstream>
#include <cstring>
#include <cstdlib>
#include <sys/types.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>

using namespace std;

// ---------------------------------------------------------------------------
// local functions
int main( int argc, char *argv[] );
static void configPort( int fd );
static int openPort( const char port[] );
static void echo( int fd );

// ---------------------------------------------------------------------------
// int main( int argc, char *argv[] )
// ---------------------------------------------------------------------------
/**
```

```
 * \brief main entry point of RsReceive
 * \param argc number of arguments
 * \param argv vector of string arguments
 *
 * Opens a serial line (default /dev/ttyS0) and
 * echos the input until CTLR-D is received.
 *
 */
int main( int argc, char *argv[] )
{
   if (argc > 2)
   {
      cerr << "Usage: RsReceive [port]" << endl;
      exit( EXIT_FAILURE );
   }

   const char *portNum = (argc == 1 ? "0" : argv[1]);
   int pd = openPort( portNum );
   if (pd < 0)
      exit( EXIT_FAILURE );

   configPort( pd );

   echo( pd );

    close( pd );
    clog << "closed port" << endl;

   return EXIT_SUCCESS;
}

// ---------------------------------------------------------------------------
//int openPort( const char number[] )
// ---------------------------------------------------------------------------
/**
 * \brief open a serial line
 * \param number the port/device number
 * \return The file descriptor is returned if device was opened successfully.
 */

int openPort( const char number[] )
{
   stringstream ss;
   ss << "/dev/ttyS" << number << ends;
   clog << "trying to open '" << ss.str() << "'" << endl;
   int fd = open( ss.str().c_str(), O_RDWR | O_NOCTTY );
   if (fd < 0)
   {
      int e = errno;
      cerr <<  "openPort : "  << strerror( e ) << endl;
      exit( EXIT_FAILURE );
   }

   return fd;
}

// ----------------------------------------------------------------------------
//void configPort( int fd )
// ----------------------------------------------------------------------------
/**
 * \brief configure the serial port
 * \param fd the file descriptor
 * Turns character handling off.
 * Reading is to block until a character is availble.
 */
```

```
void configPort( int fd )
{
   termios options;

   // read options
   if (tcgetattr( fd, &options ) != 0)
   {
      int e = errno;
      cerr << "configPort : " << strerror( e ) << endl;
      exit( EXIT_FAILURE );
   }

#if 0
    // control characters
   options.c_cc[VMIN]= 1; // wakeup after 1 character
   options.c_cc[VTIME]= 0; // infinite
   // inputflags to clear
   options.c_iflag &= ~( BRKINT |// ignore break
         IGNPAR | INPCK |// ignore parity
         ISTRIP |// don't mask
         INLCR | IGNCR | ICRNL | // ignore <cr> and <lf>
         IXON);   // ignore stop character
   // inputflags to set
   options.c_iflag |= IGNBRK;     // ignore break

   // output flags to clear
   options.c_oflag &= ~(OPOST);// no output processing

   // local flags to clear
   options.c_lflag &= ~(ECHO | // no echo
         ICANON);            // no line processing

   // control flags to clear
   options.c_cflag &= ~(CSIZE |// size flags
         CSTOPB |            // stop bit
         HUPCL |              // hangup on close
         PARENB);            // parity bit
   // control flags to set
   options.c_cflag |= (CLOCAL |// no modem
         CREAD |              // enable input
         CS8);             // eight bit data
#else
   cfmakeraw( &options );
#endif
   // input and output speed
   cfsetispeed( &options, B57600 );
   cfsetospeed( &options, B57600 );

   // clear input queue
   tcflush( fd, TCIFLUSH );
   // set configuration
   if (tcsetattr( fd, TCSANOW, &options ) != 0)
   {
      int e = errno;
      cerr << "configPort - tcsetattr failed : " << strerror( e ) << endl;
   }
}

// ----------------------------------------------------------------------------
// void echo( int fd )
// ----------------------------------------------------------------------------
/**
 * \brief echo the input
 * \param fd file descriptor for read and write operation
 *
 * Read characterwise from the input und CTL-D is read.
 * Lower case characters are converted to upper-case.
 *
 */
void echo( int fd )
{
   int in, out;
   char ch;

   do {
      // read a single character
      in = read( fd, &ch, sizeof ch );
      if (in < 0)
      {
         int e = errno;
         cerr << "echo - read failed" << strerror( e ) << endl;
         exit( EXIT_FAILURE );
      }

      // process data : here conversion to uppercase
      if (ch >= 'a' && ch <= 'z') ch += 'A'-'a';

      // write data back
      out = write( fd, &ch, sizeof ch );
      if (out < 0)
      {
         int e = errno;
         cerr << "echo - write failed" << strerror( e ) << endl;
         exit( EXIT_FAILURE );
      }

   } while( ch != '\004'); // break on ctrl-D
}
```

Source Code (client.cpp)

```
#include <cstdio> // Standard IO definitions
#include <cstring>   // String function definitions
#include <sstream> // String stream
#include <iostream>  // Standard IO stream
#include <unistd.h>  // Unix standard definitions
#include <fcntl.h>   // File controle definitions
#include <errno.h>   // Error number definitions
#include <termios.h> // POSIX terminal control definitions
#include <sys/signal.h>

using namespace std;

// ---------------------------------------------------------------------------
// local functions
static int configPort( int fd );
static int openPort( const char PortNum[] );
static int readMsg( int fd, char msg[], size_t maxlen );
static int writeMsg( int fd, const char msg[], size_t msglen );



// ---------------------------------------------------------------------------
// int main( int argc, char *argv[] )
// ---------------------------------------------------------------------------
/**
 * \brief main entry point of RsSend
 * \param argc number of arguments
 * \param argv vector of string arguments
 * \retval EXIT_SUCCESS if successful
 * \retval EXIT_FAILURE otherwise
 *
 * Opens the serial port (default /dev/ttyS0)
 * sends a short message and reads the answer.
 * If no answer is available the program terminates.
 *
 */
int main( int argc, char *argv[] )
{
   if (argc > 2)
   {
      clog << "usage RsSend [port]" << endl;
      exit( EXIT_FAILURE );
   }

   const char *port = (argc == 1 ? "0" : argv[1]);
   cout << "Trying to open port" << port << endl;

   // open the serial port
   const int pd = openPort( port );
   if (pd < 0)
   {
      exit( EXIT_FAILURE );
   }

    // configure serial port
    configPort( pd );

    //
    // write a message
    //
   char msg[] = "I'm an eXcite client" ;
   writeMsg( pd, msg, sizeof msg ) < 0;
   cout << "Sent     : " << msg << endl;

   //
   // read the answer(s) if available
   //
   char buf[ 128 ];
   int cnt;
   while (cnt = readMsg( pd, buf, sizeof buf ))
   {

      buf[cnt] = '\0';
      cout << "Received : " << buf << endl;
   }

   //
   // close the serial port
   //
   close( pd );
   cout << "Closed port" << endl;

   return EXIT_SUCCESS;
}

// ------------------------------------------------------------------------------
// int openPort( const char PortNum[] )
// ------------------------------------------------------------------------------
/**
 * \brief Open serial port
 * \param Port number of the port
 * \return Returns the file handle
 */
int openPort( const char PortNum[] )
{
   int fd;
   std::stringstream ss;
   ss << "/dev/ttyS" << PortNum;

   // O_RDWR open for read/write access
   // O_NOCTTY not controlling terminal
   fd = open( ss.str().c_str(), O_RDWR | O_NOCTTY );
   if (fd < 0)
   {   // error occurred
      int e = errno;
      cerr << "openPort : " << strerror( e ) << endl;
      exit( EXIT_FAILURE );
   }

   return fd;
}

// ----------------------------------------------------------------------------
// int configPort( int fd )
// ----------------------------------------------------------------------------
/**
 * \brief configure the serial line
 * \param fd the file descriptor
 *
 * Turns character handling off.
 * Reading times out after 0.1 seconds
 */
int configPort( int fd )
{
   termios options;

   // get current options
   if (tcgetattr( fd, &options ) != 0)
   {
      int e = errno;
      cerr << "configPort - tcgetattr failed :" << strerror( e );
      exit( EXIT_FAILURE );
   }

   cfmakeraw( &options );        // raw mode, no character handling
    options.c_cc[VMIN]      = 0;  // return even without data
    options.c_cc[VTIME]     = 1;  // timeout is 0.1 seconds

    // set input and output speed to 57600 baud
    cfsetispeed( &options, B57600 );
    cfsetospeed( &options, B57600 );

    // clear input queue
    tcflush( fd, TCIFLUSH );

   // set new options, NOW
   if (tcsetattr( fd, TCSANOW, &options ) != 0)
    {
    int e = errno;
    cerr << "configPort - tcsetattr failed : " << strerror( e );
       exit( EXIT_FAILURE );
   }

   return 0;
}
```

```
// ---------------------------------------------------------------------------
// int writeMsg( int fd, char msg[], size_t msglen )
// ---------------------------------------------------------------------------
/**
 * \brief write a message
 * \param fd the file descriptor
 * \param msg the message to write
 * \param msglen length of the message
 * \retval If successful, the number of written characters is returned.
 *
 */
int writeMsg( int fd, const char msg[], size_t msglen )
{
   int n = write( fd, msg, msglen );
   if (n < 0)
   {
      int e = errno;
      cerr << "writeMsg failed" << strerror( e ) << endl;
      exit( EXIT_FAILURE );
   }

   return n;
}

// ---------------------------------------------------------------------
// readMsg( int fd, char msg[], size_t maxlen )
// ---------------------------------------------------------------------
/**
 * \brief read a message
 * \param fd input file descriptor
 * \param msg buffer for the message
 * \param maxlen maximum length of message
 * \return the length of the message
 */
int readMsg( int fd, char msg[], size_t maxlen )
{
   int avl = maxlen-1;
   char* pmsg = msg;
   while (avl)
   {
      // read bytewise
      int cnt = read( fd, pmsg, 1 );
      if (cnt < 0)
      {   // error occurred
         int e = errno;
         cerr << "readMsg failed : " << strerror( e ) << endl;
         exit( EXIT_FAILURE );
      }
      else if (0 == cnt)
      {  // no more data available
         return pmsg-msg;
      }
       --avl;
       ++pmsg;
   }

   return maxlen;

}
```

3.5.9 BsReceive and BsSend

Description

This set of two samples illustrates communication via an Ethernet network connection between your eXcite and another computer in the network using sockets. BsReceive is a simple server program and BsSend is a client that connects to the server. Server and client are communicating with each other by implementing a very simple protocol – the server simply echoes a message sent by the client.

To run these sample programs, the eXcite must be plugged into an Ethernet network and your development PC must have access to the same network. Run either the server or the client on the processor of the eXcite and run the other component (client or server, respectively) on the PC connected with the eXcite.

Each component must be properly compiled for the computer where it will run, i.e., one component should be cross-compiled for running on the MIPS processor in the eXcite and the other should be compiled for running on the PC. You should be careful to configure the Eclipse IDE so that the sample component will be properly compiled for the system where it will run. For a description of how to build, load, and run sample programs, see Section 3.2.

To build an executable to be run on the PC use either the "Host-Debug" or the "Host-Release" configuration. The executable to be run on the eXcite must be built using either the "Debug" or the "Release" configuration.

If you are using a development PC with a Windows operating system, the component meant to run on the development PC must be compiled and run under coLinux.

BsReceive

BsReceive is the server component and it passes a message received from the BsSend client back to the client.

The Server.cpp file contains the main function of BsReceive. The main function is implemented as described in the steps below. The numbers in parentheses are markers that can be found in the source code.

1. Setup a socket(1) for an inter-machine connection and then bind (2) the socket to the agreed port, BsPort, allowing clients to connect. To enable communication across machine boundaries, BsSockAdrFamily is set to AF_INET. The socket is used in connection-oriented style, using TCP.
2. Prepare to receive connection requests (3) by calling the listen function.
3. Accept (4) return a new socket for each connection request.
4. With this socket, the communication starts. First a line of text is read (5a) from the client and then it is returned (5b).
5. Finally the sockets are closed (6ab) and the program terminates.

BsReceive Usage

Call \code BsReceive \endcode

The BsReceive program doesn't expect any parameters. BsReceive must be started first, before executing the BsSend client program.

When the message from a connected client has been exchanged, BsReceive terminates.

BsSend

BsSend is the client component and it establishes a connection to the BsReceive server. BsSend sends a message string to BsReceive, which will echo (i.e., send back) the string to the BsSend client.

The Client.cpp file contains the main function of BsSend. The main function is implemented as described in the steps below. The numbers in parentheses are markers that can be found in the source code.

1. Create a socket for a connection-oriented, inter-machine communication. Therefore, BsSockAdrFamily is set to AF_INET and BsSockType to SOCK_STREAM.
2. A connection to the arranged BsPort on the server machine is requested (2).
3. After establishing the connection, the client starts the session (3) by writing a text message and waiting for an answer (4).
4. The socket is closed (5) and the program terminates.

BsSend Usage

Call \code BsSend [server]\endcode

[server] stands either for the host-name or IP address of the system the BsRecive server program is running on.

Examples:

code BsSend eXcite-004A \endcode

code BsSend 192.168.0.173 \endcode

BsSend connects to the BsReceive program and terminates after a message has been exchanged.

For more information on the networking API used in the samples, see textbooks on C++ systems programming and also see the Linux "man" pages of the functions called in the sample.

Source Code (bs.h)

```
#ifndef _BS_H_
#define _BS_H_

/// Adressing family is internet
const int BsSockAdrFamily = AF_INET;
/// Socket type is stream, i. e. we are using TCP
const int BsSockType  = SOCK_STREAM;
/// Socket protocol is default
const int BsSockProtocol  = 0;
/// Using some non-reserved port above 1024.
const int BsPort= 1066;

#endif //_BS_H_
```

Source Code (server.cpp)

```
#include <cstdlib>
#include <cstdio>
#include <cassert>
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include "Bs.h"

using namespace std;

// local functions
static void echo( int fdin, int fdout );

// -----------------------------------------------------------------------
// int main()
// -----------------------------------------------------------------------
/*!
 * \brief main entry point
 * \retval EXIT_SUCCESS if everything works smoothly
 * \retval EXIT_FAILURE otherwise
 *
 * Creates a socket and binds it to a port.
 * Accepts one incoming connection and simply echos the data.
 * Then the connection is closed.
 */
int main()
{
   //
   // create a socket (1)
   //
   clog << "creating socket" << endl;
   int sd = socket( BsSockAdrFamily, BsSockType, BsSockProtocol );
   if (sd < 0)
   {
      perror( "Creating socket" );
      exit( EXIT_FAILURE );
   }

   //
   // bind the socket to a port (2)
   //

   // initialize socket address structure
   sockaddr_in socket;
   // accept any input device
   socket.sin_addr.s_addr = htonl( INADDR_ANY );
   // familiy is internet
   socket.sin_family= BsSockAdrFamily;
   // our port
   socket.sin_port= htons( BsPort );
   // bind
   if (bind( sd, (sockaddr *)&socket, sizeof socket ) < 0)
   {
      perror( "binding socket" );
      exit( EXIT_FAILURE );
   }
   clog << "Socket bound to port " << BsPort << endl;
   //
   // wait for connection
   //

   const int numConnections = 1;
   // prepare for connection requests(3)
   listen( sd, numConnections );
   do
   {
      // description of client socket, filled by accept
      sockaddr_in client;
      // initialize with structure size
      socklen_t size = sizeof client;
        clog << "waiting for a connection..." << endl;
      // open an incoming connection request (4)
      int fd = accept( sd, (sockaddr *)&client, &size );
      if (fd < 0)
      {
         perror( "accepting connection" );
         exit( EXIT_FAILURE );
      }

      clog << "Accepted connection from "
          << inet_ntoa( client.sin_addr ) << endl;
      //
      // use connection: here echo
      //
      echo( fd, fd );

      // close connection (6a)
      close( fd );
      clog << "Closed connection"  << endl;
   } while (false);

   // close socket (6b)
   close( sd );
   clog << "Closed socket" << endl;

   return EXIT_SUCCESS;
}

void echo( int fdin, int fdout )
{
   char data[ 128 ];
   int cnt;

   //
   // read data
   //

   do {
      // try reading data (5a)
      cnt = read( fdin, data, sizeof data );
      // if failure was not a simple interrupt then quit
      if (cnt < 0 && errno != EINTR)
      {
         perror( "reading data" );
         exit( EXIT_FAILURE );
      }
      // until read w/o error
   } while (cnt < 0);

   data[cnt] = '\0';
   clog << "Read : " << data << endl;
   //
   // return the data to output
   //

   // prefix
   char msg[ 128];
    sprintf( msg, "echo : %s", data );
    // send answer (5b)
   cnt = write( fdout, msg, strlen( msg ) );
   if (cnt < 0)
   {
      perror( "writing data" );
      exit( EXIT_FAILURE );
   }
   clog << "Sent message" << endl;
   assert( cnt == (int) strlen( msg ) );
   }
```

Source Code (client.cpp)

```
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include "Bs.h"

/// using members of standard C++ library without qualification
using namespace std;
// constants

// local functions

// ------------------------------------------------------------------------------
// int main()
// ------------------------------------------------------------------------------
/*!
 * \brief main entry point
 * \retval EXIT_SUCCESS on success
 **/

int main( int argc, char *argv[] )
{

   //
   // (1) create socket
   //
   int sd = socket( BsSockAdrFamily, BsSockType, BsSockProtocol );
   if (sd < 0)
   {
      perror( "create socket" );
      exit( EXIT_FAILURE );
   }

   //
   // (2) connect with server port
   //

   const char * const serverName = argc == 2 ? argv[1] : "localhost";
```

```
    // get host info
    const hostent* hostInfo = gethostbyname( serverName );
    if (NULL == hostInfo)
    {
       cerr <<  "Unknown host " << serverName << endl;
       exit( EXIT_FAILURE );
    }
    // setup server address
    sockaddr_in server;
    server.sin_family= hostInfo->h_addrtype;
    memcpy( (void*) &server.sin_addr, hostInfo->h_addr, hostInfo->h_length );
    server.sin_port= htons( BsPort );
    // connect
    if (connect( sd, (sockaddr *) &server, sizeof server ) < 0)
    {
       perror( "connecting with server" );
       exit( EXIT_FAILURE );
    }
    clog << "Connected to server: " << serverName << endl;

    //
    // (3) write to server
    //
    const size_t bufferSize = 80;
    const char msg[ bufferSize ] = "I am an eXcited Client";
    char buffer[ bufferSize ] = "";
    if (write( sd, msg, strlen( msg ) ) < 0)
    {
       perror( "writing message" );
       exit( EXIT_FAILURE );
    }
    clog << "Sent message: " << msg << endl;

    //
    // (4) read from server
    //
    int cnt = read( sd, (void*)buffer, sizeof buffer );
    if (cnt < 0)
    {
       perror( "reading message" );
       exit( EXIT_FAILURE );
    }
    // ensure terminating null character
    buffer[cnt] = '\0';
    cout << "Recieved message: "
         << buffer << endl;


    //
    // (5) close connection
    //
    close( sd );
    clog << "Closed connection" << endl;

    return EXIT_SUCCESS;
}
```

3.5.10 StreamingClient and StreamingServer

Description

This set of sample application programs shows how to use the eXcite library to transfer images from the eXcite to a PC. The use of the *GxClient* and *GxStreamServer* class is demonstrated. The details of setting up a data transmission between an application running on the eXcite and an application running on another PC are describe in the "Transferring Images from an eXcite to the Outside" section of the API Overview in the API Reference documentation.

StreamingClient (StreamingClient.cpp) is the client application that receives a stream of images from the StreamingServer server application (StreamingServer.cpp). The server doesn't grab images from the eXcite, instead it sends computed test images to the client.

StreamingServer

The StreamingServer program implements the server part of the application level protocol described in the "Transferring Images from an eXcite to the Outside" section in the API Reference. The application demonstrates the usage of the *GxStreamServer* class to send a stream of images. In the main function of StreamingServer.cpp, the following steps are performed:

1. Create an instance of the *GxStreamServer* class and "open" the server. A client now can connect to the server. At the time of server creation, the image size and format must be defined. Note that for its whole lifetime, the server sends images having a fixed size and format!
2. Wait for a specified period for a client to connect and to open a stream channel used for image data transmission. If the timeout expires, the application terminates.
3. Wait for a specified period for the client to give the signal to send images. If the timeout expires, the application terminates.
4. As long the client hasn’t signalled to stop the image data transmission, images are sent to the client. The constructor of the *CBuffer* class used in this example generates some test image data.
5. When the client tells the server to stop image data transmission, the server waits for a specified period for the client to close the streaming channel.
6. Close the server so that no other clients can see the server or connect to it.

StreamingServer Usage

Build the StreamingServer using the cross-compiler for the MIPS (in Eclipse use either the Debug or the Release configuration) and launch the StreamingServer program on the eXcite. Then start the StreamingClient program on your Linux PC or on your Windows PC running coLinux.

StreamingClient

The StreamingClient program implements the client part of the application level protocol described in the "Transferring Images from an eXcite to the Outside" in the API Reference documentation. The application illustrates how to use the *GxClient* class to receive images from an application using the *GxStreamServer* class. In the main function of StreamingClient.cpp the following steps are performed:

1. Use the DeviceManager to return a list with all servers found.
2. Use the first item in the list to create a *GxClient* object "bound" to the server that the list item is representing.
3. Open the connection by calling the *GxClient::Open()* method.
4. Ask the server for the format and size of the images the server will send.
5. Open the connection used for image data transmission (i.e., the "streaming channel") by calling *GxClient::PrepareGrab()*.
6. Allocate some buffers and pass them to the *GxClient* object to be filled with image data received from the connected server.
7. In a loop, the filled buffers are received and passed back to the *GxClient* object to be filled again.
8. If the desired number of images have been received, tell the server to stop image data transmission.
9. Close the streaming channel.
10. Close the connection between the server and the client by closing the *GxClient* object.

The "StreamingClient" program is structured in a fashion similar to the MultiGrab sample program. The main differences are:

- A different identifier is passed to the *DeviceManager::EnumerateDevices()* method.
- Instead of using the *Camera::ContinuousShot* member to start and stop the image transmission, the *GxClient::EnableStreaming* is used.

StreamingClient Usage

The StreamingClient program is designed to run on a Linux PC or a Windows PC running coLinux. Build the StreamingClient application using the compiler on your Linux or coLinux development system (in Eclipse use either the "Host-Debug" or "Host-Release" configuration) and launch the StreamingClient program after starting the StreamingServer program on the eXcite.

Configuring the Port Used for the Connection Between Client and Server

WIth no modification, the StreamingClient and StreamingServer programs use the default port number compiled into the *GxClient* and *GxStreamServer* classes for the network connection. The *DeviceManager::EnumerateDevices()* will return a list containing information about all server GxStreamServer applications using the default port number.

To establish the connection using a different port number, set the preprocessor define *USE_DEFAULT_PORT* to 0 both in StreamingServer.cpp and StreamingClient.cpp. In this case, the server and the client are configured to use the port number specified by the *PORT* preprocessor defines in StreamingClient.cpp and StreamingServer.cpp. Please ensure, that the values assigned to *PORT* are the same in both source files.

When *USE_DEFAULT_PORT* is set to 0, StreamingServer.cpp demonstrates how the port for the server is specified when creating the *GxStreamServer* object. StreamingClient.cpp shows how to create and initialize a *PropertySet* object used to pass the desired port number to the *DeviceManager::EnumerateDevices()* method. The *DeviceManager::EnumerateDevices()* will return a list that only contains information about the servers using the port number specified in the property set.

Source Code (streamingclient.cpp)

```
#include <cstdlib>
#include <iostream>
#include <vector>
#include <device/DeviceManager.h>
#include <gxdevice/GxClient.h>


// Server and client are communicating using a UDP port. Set the following define to 0 if
you want
// to choose a port number differing from the default one.
#define USE_DEFAULT_PORT 1

#if ! USE_DEFAULT_PORT
# define PORT "4242"
#endif


using namespace std;
using namespace BaslerCamera;


class Buffer
{
private:
   unsigned char* pBuffer;
   unsigned long size;
   Buffer( const Buffer& );
   Buffer& operator=(const Buffer&);

public:
   Buffer( size_t nBytes )
   {
      pBuffer = new unsigned char[ nBytes ];
      size = nBytes;
   }
   unsigned char* getBuffer()
   {
      return pBuffer;
   }

   unsigned long getSize()
   {
      return size;
   }

   ~Buffer()
   {
      delete[] pBuffer;
   }
};

int main(int argc, char* argv[])
{
   try
   {

      //-------------------------------------------------------------------------
      //  attach to first available server
      //-------------------------------------------------------------------------

      // get the device manager
      DeviceManager & dm = DeviceManager::GetInstance();

      // let the device manager enumerate all servers
#if USE_DEFAULT_PORT
      DeviceManager::DeviceInfoList_t lstDevices(dm.EnumerateDevices(
GxClient::DeviceTypeId ));
#else
      // when using the non-default port, we have to pass a property set
      // containing a "PortNumber" property
      // to the device manager's enumerate method.
      PropertySet propset;
      propset.AddProperty("PortNumber", PORT);
      DeviceManager::DeviceInfoList_t lstDevices(dm.EnumerateDevices(
      GxClient::DeviceTypeId, &propset));
#endif

      // check if we found any servers
      if ( 0 == lstDevices.size() )
         throw RUNTIME_EXCEPTION("No servers found.\n");

      // let the device manager create a device bound to the first server found
      GxClient* pClient = dynamic_cast<GxClient*>( dm.CreateDevice
      ( *lstDevices.begin() ) );

      // check if we got what we expected
      if ( NULL == pClient )
         throw RUNTIME_EXCEPTION( "Device isn't a GxClient device" );

      pClient->Open();

      cout << "Connected to server " << pClient->GetDeviceInfo().GetFriendlyDeviceName()
      << endl;

      const unsigned int nFrames = 15;   // number of frames we want to grab
      const unsigned int nBuffers = 3;   // number of buffers we want to use to grab

      //-------------------------------------------------------------------------
      //  Grab images from the server's image data stream
      //-------------------------------------------------------------------------

      const unsigned long width = (unsigned long) pClient->Width();
      const unsigned long height = (unsigned long) pClient->Height();
      const unsigned long dataDepth = (unsigned long) pClient->DataDepth();
      assert( dataDepth  == 1);  // we are expecting 8 bit image data
      cout << "Frame size: " << width << "x" << height << " pixel" << endl;

      // open the stream
      pClient->PrepareGrab();

      // allocate memory
      vector<Buffer*> buffers(nBuffers);
      for ( unsigned int i = 0; i < nBuffers; ++i )
      {
         buffers[i] = new Buffer( pClient->TotalBytes() );
      }
```

```
// enqueue buffers to be filled with image data
for ( unsigned int i = 0; i < nBuffers; ++i )
{
pClient->QueueBuffer( buffers[i]->getSize(), buffers[i]->getBuffer(), (void*) i );
}

// enable the stream, i.e., tell the server that we are ready to receive the data data
pClient->EnableStreaming = 1;

for ( unsigned int n = 0; n < nFrames;  )
{
   // wait for the next buffer to be filled
   void* pBuffer;
   void* pUser;
   IInDataStream::BufferStatus result = pClient->WaitForBuffer( &pBuffer, &pUser, (
   unsigned int ) 1000 );
   switch ( result )
   {
   case IInDataStream::bsOk:
   {
      unsigned int nr = (unsigned int) pUser; // index of the buffer
      cout << "Captured buffer nr. " << nr << endl;
      /// process the buffer
      /// ...
      /// pass it to the device to be filled again
      if ( n++  < nFrames - nBuffers )
      {
         cout << "   Enqueue buffer " << nr << " again" << endl;
         pClient->QueueBuffer( buffers[nr]->getSize(), pBuffer, (void*) nr);
      }
      break;
   }
   case IInDataStream::bsTimeOut:
      cerr << "TimeOut" << endl;
      break;
   case IInDataStream::bsCancelled:
      cerr << "Buffer cancelled" << endl;
      break;
   case IInDataStream::bsError:
      cerr << "Got Buffer with error status" << endl;
      break;
   default:
      cerr << "WaitForBuffer returned unexpected value" << endl;
      break;
   }
}
// tell the server that we no longer want to receive data
pClient->EnableStreaming = 0;

// close the stream
pClient->FinishGrab();

//-----------------------------------------------------------------------
//  Clean-up
//-----------------------------------------------------------------------
// close the device
pClient->Close();

// let the device manager destroy the device
dm.DestroyDevice( pClient );
```

```
      // free the memory
      for ( unsigned int i = 0; i < nBuffers; ++i )
      {
         delete buffers[i];
      }

   }
   catch ( GenApi::GenericException & e )
   {
      cerr << "Error occurred: " << e.GetDescription() << endl;
      cerr << "(" << e.GetSourceFileName() << ":" << e.GetSourceLine() << endl;
      return EXIT_FAILURE;
   }
   return EXIT_SUCCESS;
}
```

Source Code (streamingserver.cpp)

```
#include <iostream>
#include <gxdevice/GxStreamServer.h>

using namespace std;
using namespace BaslerCamera;

// Server and client are communicating using a UDP port.
// Set the following define to 0 if you want
// to choose a port number differing from the default one.
#define USE_DEFAULT_PORT 1

#if ! USE_DEFAULT_PORT
# define PORT 4242
#endif

class CBuffer
{
public:
   CBuffer( uint32_t width, uint32_t height );
   ~CBuffer();
private:
   CBuffer( const CBuffer& );
   CBuffer& operator=( const CBuffer& );
public:
   uint32_t Width() const;
   uint32_t Height() const;
   uint32_t BufferSize() const;
   uint8_t* Buffer() const;
   uint8_t* Buffer();
private:
   uint8_t* m_pBuffer;
   const uint32_t m_Width;
   const uint32_t m_Height;
};

int main( int argc, char* argv[] )
{
   const unsigned long width = 640;  // width of the buffers to be sent
   const unsigned long height = 480; // heigth of the buffers to be sent
   const unsigned long datadepth = 1; // we want to send 1 byte per pixel

   try
   {
      cout << endl << "Press ^C to terminate" << endl;
      //-------------------------------------------------------------------------
      // Create Server
      //-------------------------------------------------------------------------
      GxStreamServer Server(width, height, datadepth);
#if USE_DEFAULT_PORT
      Server.Open( "MyServer" );
#else
      Server.Open( "MyServer", PORT );
#endif

      //-------------------------------------------------------------------------
      // Wait up to a minute for the client to open the stream
      //-------------------------------------------------------------------------
      cout << "Waiting for a client to open the stream" << endl;
      if ( ! Server.WaitForStreamOpen( 60000 ) )
      {
         cerr << "Timeout expired. No client has opened the stream" << endl;
         Server.Close();
         return EXIT_FAILURE;
      }
      cout << "Client has opened stream" << endl;

      //-------------------------------------------------------------------------
      // Wait up to 10 seconds for the client's request to start sending data
      //-------------------------------------------------------------------------
      if ( ! Server.WaitForStreamingEnabled( 10000 ) )
      {
         cerr << "Timeout expired. Client hasn't enabled the streaming" << endl;
         Server.Close();
         return EXIT_FAILURE;
      }
      cout << "Client has requested to start sending of data" << endl;

      //-------------------------------------------------------------------------
      // Send image buffers until the client disables the streaming
      //-------------------------------------------------------------------------
      while ( Server.IsStreamingEnabled() )
      {
         CBuffer buffer( width, height );  // creates buffer and fills it with a pattern
         Server.QueueBuffer( width * height, buffer.Buffer(), NULL );  // starts sending
                                                                   // of the buffer
         Server.WaitForBuffer( NULL );            // waits until the buffer has been sent
      }

      cout << "Client has requested to stop sending of data" << endl;

      //-------------------------------------------------------------------------
      // Shut-down the server
      //-------------------------------------------------------------------------
      // first we have to wait until the client closed the stream
      if ( ! Server.WaitForStreamClose( 10000 ) )  // wait up to 10 seconds for
                                                // the client to close the stream
      {
         cerr << "Timeout expired. Client hasn't closed the stream" << endl;
         Server.Close();
         return EXIT_FAILURE;
      }

      cout << "Client has closed the stream" << endl;
      // allow the client to disconnect
      sleep(1);
      Server.Close();  // closes the server
   }
   catch ( GenApi::GenericException & e )
   {
      cerr << "Error occurred: ";
      cerr << e.GetDescription() << endl;
      cerr << "(" << e.GetSourceFileName() << ":" << e.GetSourceLine() << endl;
      return EXIT_FAILURE;
   }

   return EXIT_SUCCESS;
}

/// Construct a buffer of 8-bit depth
CBuffer :: CBuffer( uint32_t width, uint32_t height )
      :
      m_pBuffer( new uint8_t[ width * height ] ),
      m_Width( width ),
      m_Height( height )
{
   // fill buffer with a fancy pattern
   const uint32_t cr = Height() / 2;
   const uint32_t cc = Width() / 2;
   uint8_t *p = Buffer();

   for ( uint32_t r = 0; r < Height(); ++r )
   {
      for ( uint32_t c = 0; c < Width(); ++c )
      {
         *p++ = ( uint8_t ) ( ( r - cr ) * ( r - cr ) + ( c - cc ) * ( c - cc ) );
      }
   }
}

/// Destroy the buffer
CBuffer :: ~CBuffer()
{
   delete [] m_pBuffer;
}

/// Return the width
uint32_t CBuffer::Width() const
{
   return m_Width;
}

/// Return the height
uint32_t CBuffer::Height() const
{
   return m_Height;
}

/// Return the buffer size in bytes
uint32_t CBuffer::BufferSize() const
{
   return m_Width * m_Height;
}

/// Get the Buffer
uint8_t* CBuffer::Buffer() const
{
   return m_pBuffer;
}

/// Get the Buffer
uint8_t* CBuffer::Buffer()
{
   return m_pBuffer;
}
```

4 eXcite Interface

4.1 Connections

4.1.1 General Description

The eXcite is interfaced to external circuitry via five connectors located on the back of the housing:

- two standard 4-pin, USB type A connectors used to provide USB access to the processor.
- an 8-pin, RJ-45 jack used to provide a 10/100/1000 Mbps Ethernet connection to the processor.
- a 10-pin receptacle used to receive input power for the eXcite and for accessing an RS-232 serial connection to the processor.
- a 12-pin receptacle used to provide access to the eXcite's four digital input ports and four digital output ports.

Two status LEDs are located on the back of the eXcite. One is used to indicate power present and the other is used to indicate the state of the Ethernet connection. Figure 4-1 shows the connectors and the LEDs.

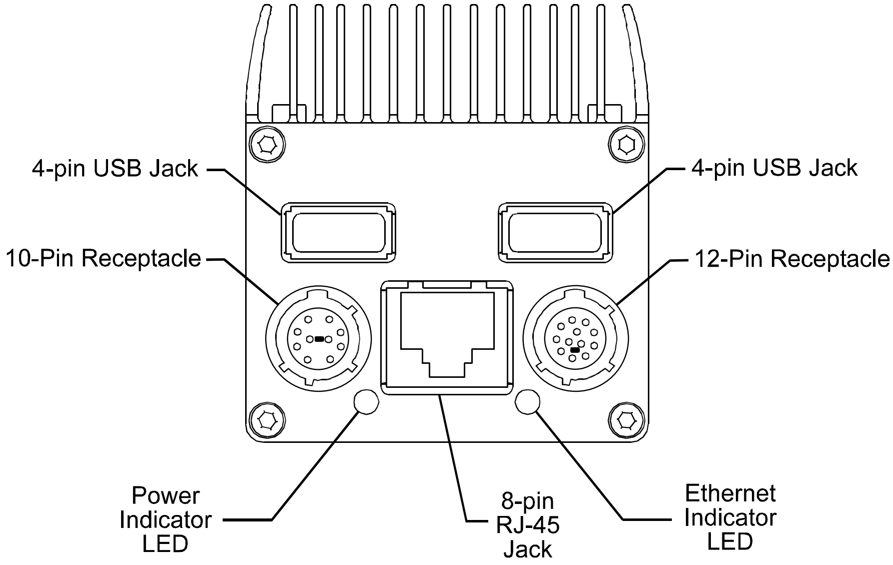


Figure 4-1: Connectors and LED

4.1.2 Pin Assignments and Numbering

4-pin USB Jacks

The two 4-pin jacks provide USB access to the processor. They are USB Type A connectors. Pin numbering and assignment adhere to the USB 2.0 standard.

8-pin RJ-45 Jack

The 8-pin, RJ-45 jack provides Ethernet access to the processor. Pin numbering and assignment adhere to the Ethernet standard.

10-pin Receptacle

The 10-pin receptacle is used to receive input power for the eXcite and for accessing an RS-232 serial connection to the processor. The pin assignments for the receptacle are as shown in Table 4-1 and the pin numbering is as shown in Figure 4-2.

Pin	Assignment
1	+12 VDC Input Power *
2	+12 VDC Input Power *
3	+12 VDC Input Power *
4	Input Power Gnd **
5	Input Power Gnd **
6	Input Power Gnd **
7	RS-232 RxD (Receive) (This is an input to the eXcite.)
8	RS-232 CTS (Clear to Send) (This is an input to the eXcite.)
9	RS-232 TxD (Transmit) (This is an output from the eXcite.)
10	RS-232 RTS (Request to Send) (This is an output from the eXcite.)

Table 4-1: Pin Assignments for the 10-pin Receptacle

* There must be a separate wire from the +12 VDC input power source to pin 1, from the +12 VDC input power source to pin 2, and from the +12 VDC input power source to pin 3.

** There must be a separate wire from the input power source ground to pin 4, from the input power source ground to pin 5, and from the input power source ground to pin 6.

Warning!

For input power, the camera has overvoltage protection up to 30 VDC. An input voltage higher than 30 VDC will cause damage leaving the camera nonoperational.

12-pin Receptacle

The 12-pin receptacle provides access to the eXcite's four digital input ports and four digital output ports. The pin assignments for the receptacle are as shown in Table 4-2 and the pin numbering is as shown in Figure 4-2.

Pin	Assignment
1	Input Port 0
2	Input Port 1
3	Input Port 2
4	Input Port 3
5	Gnd
6	Output 3
7	Output 2
8	Output 1
9	Output 0
10	VCC (+10 to + 30 VDC)
11	Not Connected
12	Not Connected

Table 4-2: Pin Assignments for the 12-pin Receptacle

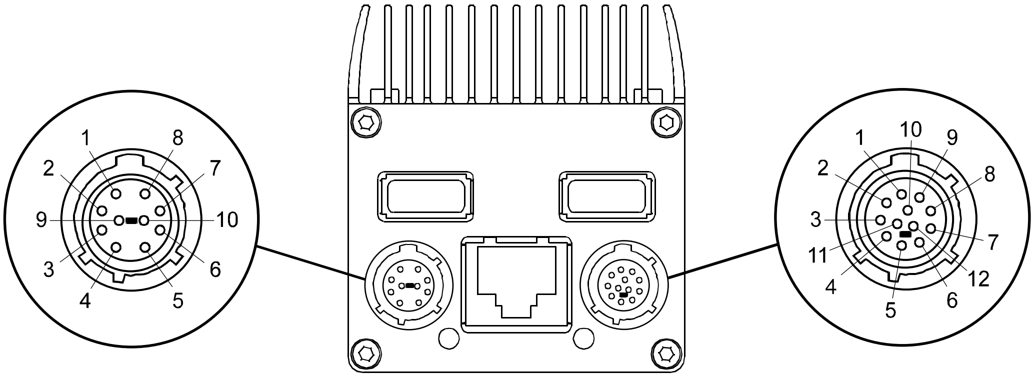


Figure 4-2: Pin Numbering for the 10-pin and 12-pin Receptacles

4.1.3 Connector Types

4-pin USB Jacks

The 4-pin jacks for eXcite’s two USB ports are standard USB Type A connectors.

8-pin RJ-45 Jack

The 8-pin jack for eXcite’s Ethernet port is a standard RJ-45 connector.

10-pin Receptacle

The 10-pin power/RS-232 connector on the eXcite is a Hirose micro receptacle (part # HR10A-10R-10P) or the equivalent.

The recommended mating connector is the Hirose micro plug (part # HR10A-10P-10S).

12-pin Receptacle

The 12-pin I/O connector on the eXcite is a Hirose micro receptacle (part # HR10A-10R-12P) or the equivalent.

The recommended mating connector is the Hirose micro plug (part # HR10A-10P-12S).

4.2 LED Indicators

The eXcite is equipped with two LED indicators as shown in Figure 4-1.

The LED on the right side of the RJ-45 connector indicates the condition of the Ethernet connection:

- Solid green means that a 1000 Mb Ethernet connection is present, but no data transfer is taking place.
- Solid red means that a 100 Mb Ethernet connection is present, but no data transfer is taking place.
- If the LED is not lit, two conditions are possible:

  No Ethernet connection is present.

  A 10 Mb connection is present, but no data transfer is taking place.
- A blinking LED indicates that an Ethernet connection is present and that data transfer is taking place.

The LED on the left side of the RJ-45 jack is used to indicate the condition of the eXcite's internal power:

- If the LED is green, the input power to the eXcite is correct and the eXcite is properly generating all internal voltages.
- If the LED is not lit, the eXcite is not generating internal voltages.

  An unlit LED can occur because the input power to the eXcite is missing or incorrect.

  An unlit LED can also occur if the input power to the eXcite is correct, but the eXcite is in an overtemperature condition. See Section 5.9.13.2 for more information.

4.3 Input Power

The required input voltage for the eXcite is +12.0 (+/- 0.5) VDC. Typical power consumption at 12.0 VDC is 14.0 W for CMOS camera models and 16.0 W for CCD camera models. The power consumption was determined for operation at max. frame capture rate at full resolution combined with high processor load.

If the input voltage drops below approximately 10 VDC, the camera will enter an undervoltage lockout condition (see Section 5.9.13.4).

⚠ **Warning!**

For input power, the camera has overvoltage protection up to 30 VDC. An input voltage higher than 30 VDC will cause damage leaving the camera nonoperational.

ⓘ The input current for the eXcite must not exceed 1.8 A.

Keep in mind that when you plug a bus-powered USB device into the eXcite, the input current to the eXcite will increase. Bus-powered USB devices that draw more than 100 mA **must not** be used with the eXcite.

Also keep in mind that a 1000 Mb Ethernet connection will draw more current than a 10 or a 100 Mb connection. So the input current to the eXcite will be higher when you use a 1000 Mb Ethernet connection.

Cabling

As shown in Figure 4-3, your power supply can be connected directly to the eXcite. The output cable on the power supply must be terminated with a Hirose plug as Figure 4-3 shows. A wiring diagram for the power supply output cable appears in Figure 4-4.

When a power supply is connected directly to the 10 pin connector on the eXcite, you will not have physical access to the eXcite's RS-232 port. To have access to the RS-232 port, you must use a Y cable between the power supply and the eXcite as shown in Figure 4-5. Please note that you will also need to use a null modem cable between the D connector on the Y cable and the serial port on your PC. The Y cable must be terminated with the proper connectors as shown in Figure 4-5. A wiring diagram for the Y cable is shown in Figure 4-6.

A power supply with the proper plug on the output cable is available from Basler. A Y cable terminated with the proper connectors is also available. Please contact your Basler sales representative to order power supplies or Y cables.

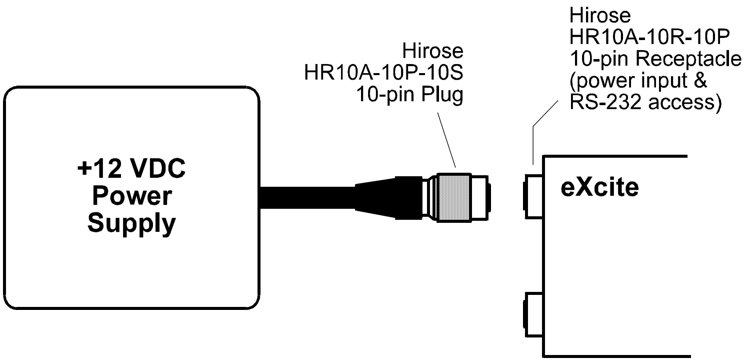


Figure 4-3: Power Supply Connected Directly to an eXcite

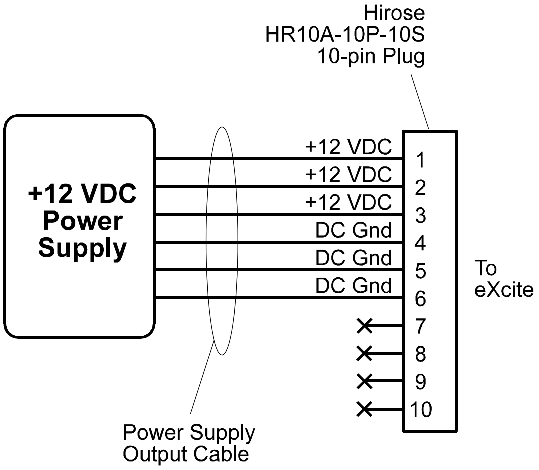


Figure 4-4: Cable Diagram for a Power Supply

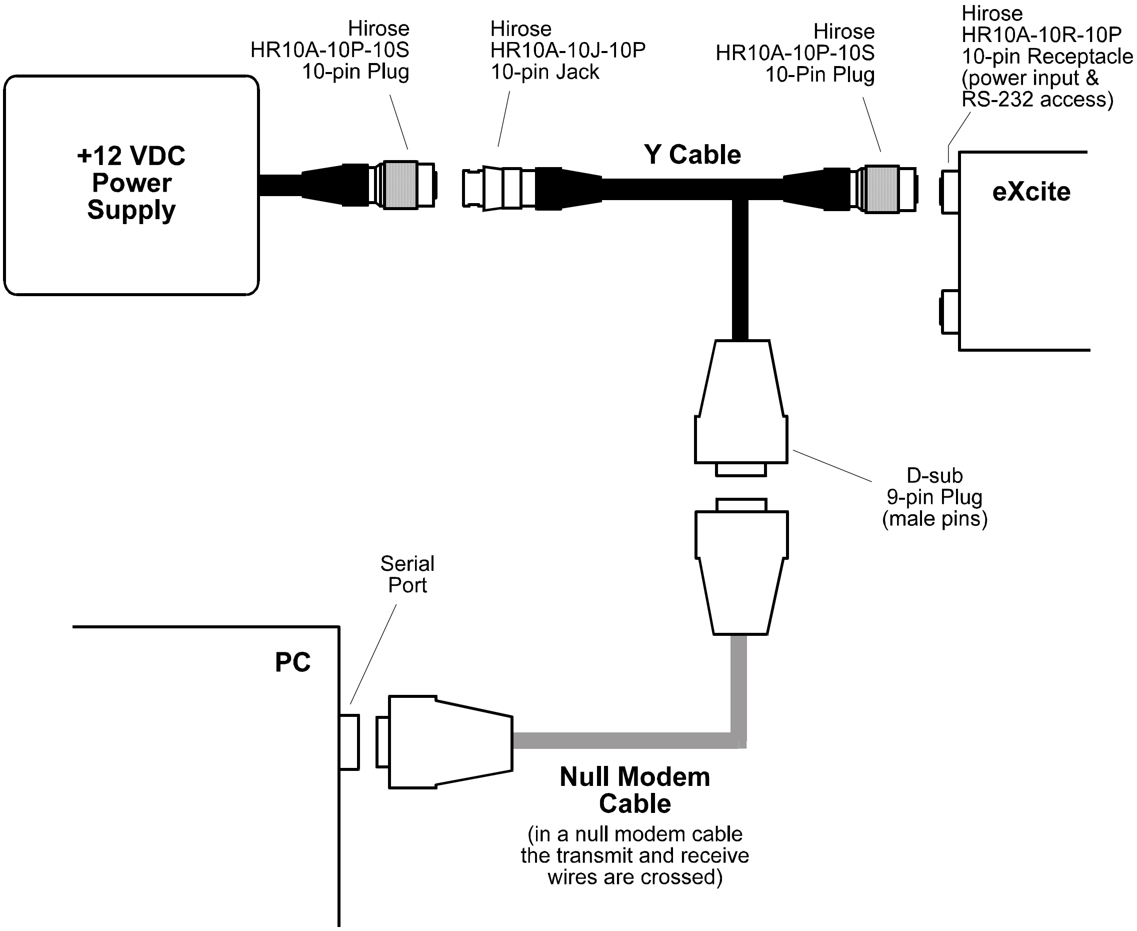


Figure 4-5: Power Supply Connected to an eXcite Through a Y Cable

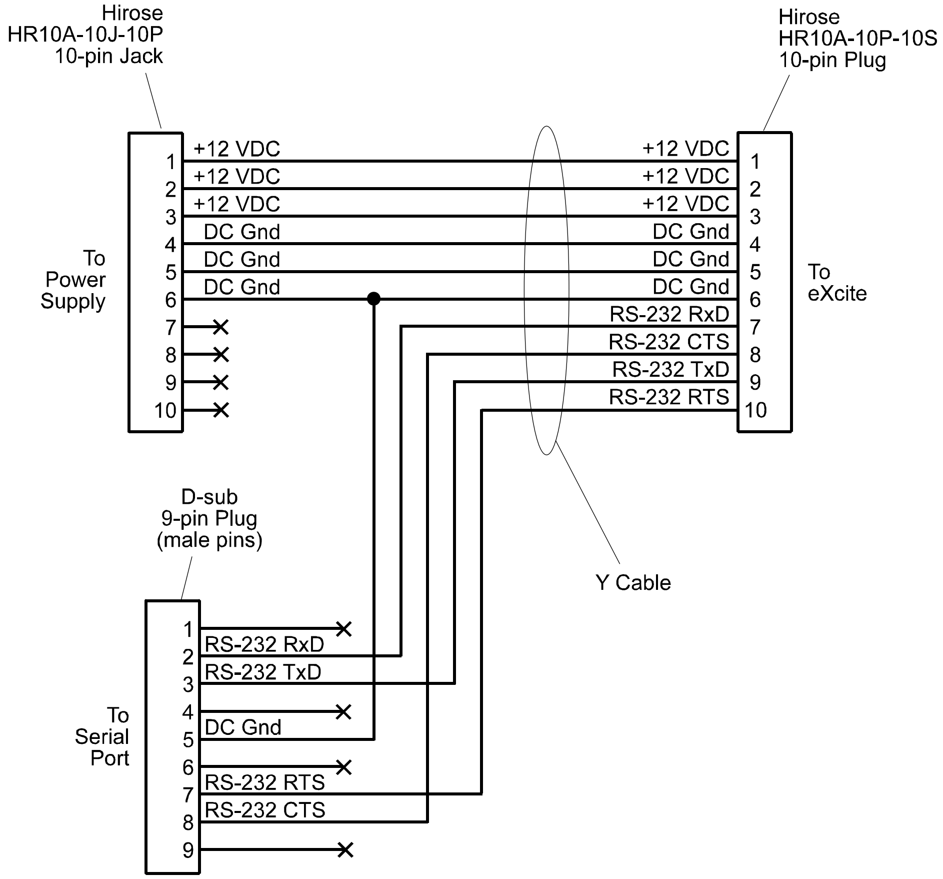


Figure 4-6: Cable Diagram for a Y Cable

4.4 Ports

4.4.1 RS-232 Serial Port

The eXcite's processor is equipped with one RS-232 serial port. The port is accessed via four pins in the 10-pin micro-miniature receptacle on the back of the eXcite. The pin assignments for the receptacle are as shown in Table 4-1.

You will note that the wiring for handshaking (“clear to send” and “request to send”) is present on the port. Although the wiring is present, handshaking has not yet been implemented.

The serial port is available for use by programs that you design to run on the eXcite. However, by default the serial port has been assigned to transmit console output from the processor and to accept input from a keyboard. Before you can access the serial port from your programs, you must reconfigure the eXcite as described in Section 5.9.5.

Cabling

The 10-pin connector on the back of the eXcite is used both for input power and for access to the RS-232 serial port. If a power supply is attached directly to the 10-pin connector as shown in Figure 4-3, you will not have physical access to the port. To have access to the RS-232 port, you must use a Y cable between the power supply and the eXcite as shown in Figure 4-5. Please note that you will also need to use a null modem cable between the D-sub connector on the Y cable and the serial port on your PC. The Y cable must be terminated with the proper connectors as shown in Figure 4-5. A wiring diagram for the Y cable is shown in Figure 4-6.

A Y cable terminated with the proper connectors is available from Basler. Please contact your Basler sales representative to order Y cables.

4.4.2 USB Ports

The eXcite's processor is equipped with two USB 2.0 ports. Each port is accessed via a standard USB Type A connector on the back of the eXcite. The Linux kernel in the eXcite processor is equipped with a driver to provide basic USB port functionality for mass storage devices. When any other USB device is used with the eXcite, you will typically be required to install a Linux driver for the device.

Bus-powered USB devices that draw more than 100 mA **must not** be used with the eXcite.

Cabling

Use high-quality USB cables. To avoid EMI, the cables must be shielded.

4.4.3 Ethernet Port

The eXcite's processor is equipped with one standard 10/100/1000 Ethernet port. See Section 5.9.7 for more information about using the Ethernet port.

Cabling

Use high-quality Ethernet cables. To avoid EMI, the cables must be shielded.

If you are connecting the eXcite to a local area network (LAN), use a straight-through Ethernet cable.

If you are using the eXcite in a peer-to-peer Ethernet network, e.g., connecting the eXcite directly to an Ethernet card in a computer, you must use a crossover Ethernet cable with a 10/100 Mbps connection.

In a peer-to-peer Ethernet cable with a 1000 Mbps connection, you can use either a straight-through or a crossover Ethernet cable.

4.4.4 Input and Output Ports

A schematic of the I/O ports is shown in the figure below. The I/O ports are described in greater detail in the subsequent sections.

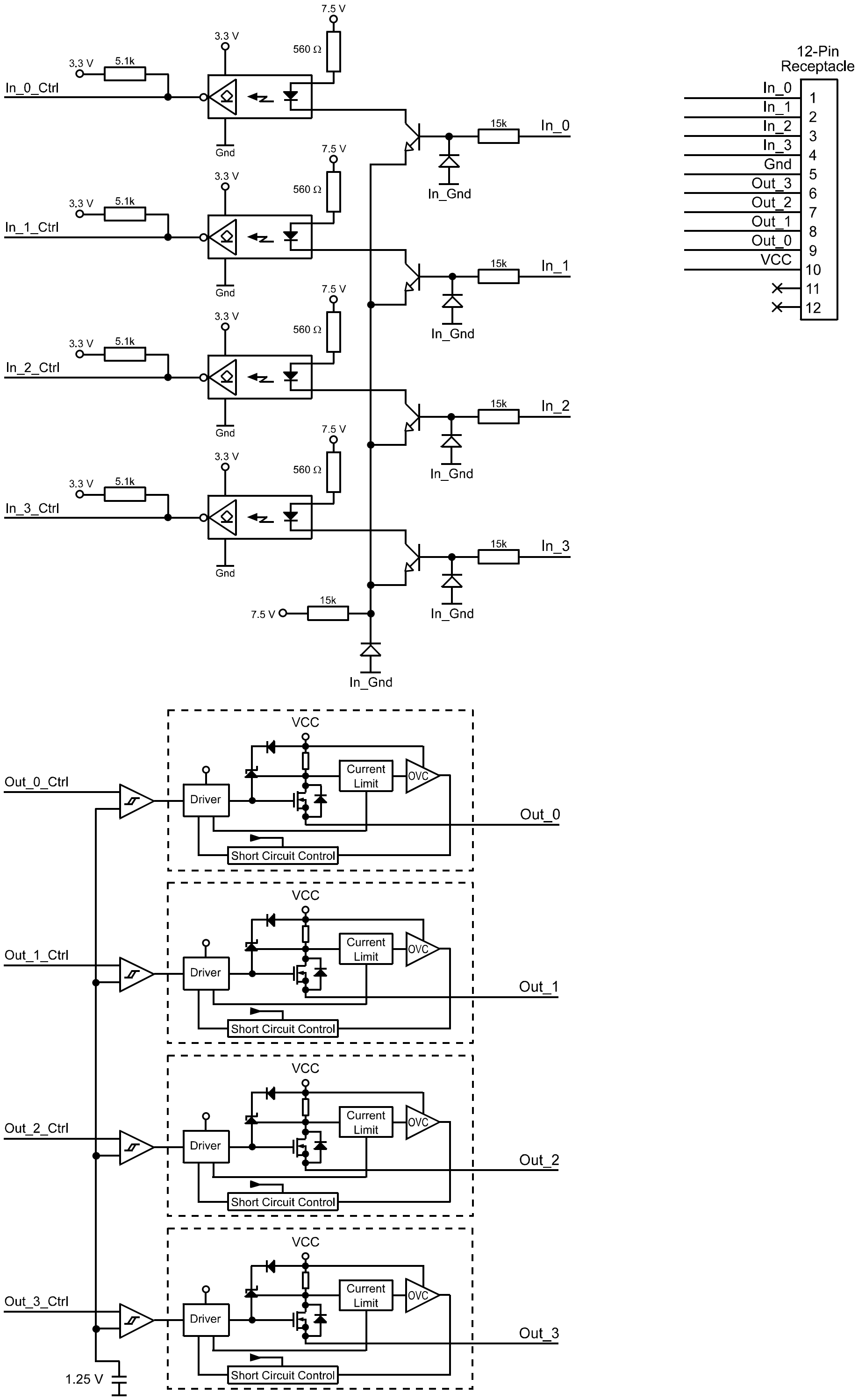


Figure 4-7: I/O Port Schematic

4.4.4.1 Input Ports

The eXcite equipped with four physical input ports designated as Input Port 0, Input Port 1, Input Port 2, and Input Port 3. The input ports are accessed via the 12 pin micro-miniature connector on the back of the eXcite. See Table 4-2 and Figure 4-2 for input port pin assignments and pin numbering.

As shown in the schematic in Figure 4-7, each input port is opto-isolated. For each port, the minimum input voltage to indicate a logical one is +5.2 VDC and the maximum is +30 VDC. An input voltage less than +5.2 VDC means a logical zero.

Figure 4-8 shows an example of a typical circuit you can use to input a signal into the eXcite.

By default, Input Port 0 is assigned to receive an external trigger (ExTrig) signal that can be used to control the start of exposure. For more information about how to use an ExTrig signal to control exposure start and for information about assigning the ExTrig signal to a different input port, see Section 5.3.3.

For more information on reading the state of the input ports, see Section 5.9.8.

The input ports on the eXcite are not compatible with TTL voltage levels.

As shown in Figure 4-8, a voltage (VCC) between +10 and +30 VDC must be present on pin 10 and a DC ground must be present on pin 5. If this voltage and ground are not present, the input ports and the output ports will not operate.

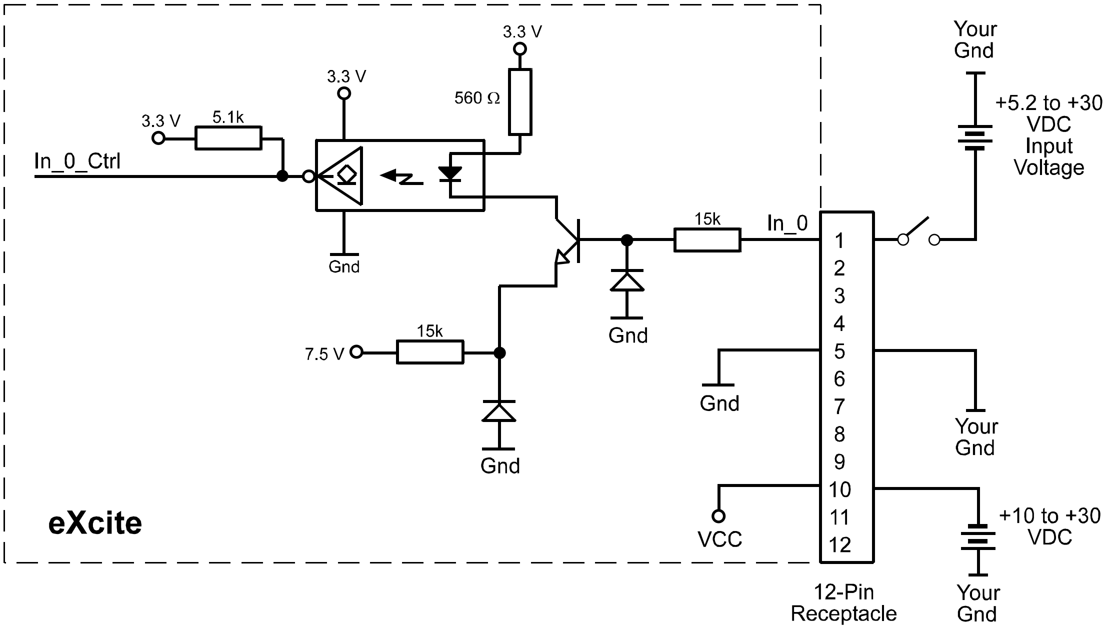


Figure 4-8: Typical Input Circuit

4.4.4.2 Output Ports

The eXcite is equipped with four physical output ports designated as Output Port 0, Output Port 1, Output Port 2, and Output Port 3. The output ports are accessed via the 12 pin micro-miniature connector on the back of the eXcite. See Table 4-2 and Figure 4-2 for output port pin assignments and pin numbering.

As shown in the schematic in Figure 4-7, each output port is implemented as a high side switch. The outputs are short circuit protected. A conducting output circuit means a logical one and a non-conducting output circuit means a logical zero.

Figure 4-9 shows a typical circuit you can use to monitor an output port with a voltage signal. The circuit in Figure 4-9 is monitoring output port 2.

Figure 4-10 shows a typical circuit you can use to monitor an output port with a LED or an opto-coupler. In this example, the voltage for the external circuit is +24 VDC. Current in the circuit is limited by an external resistor. The circuit in Figure 4-10 is monitoring output port 3.

Any device attached to an eXcite output port must draw a minimum of 7 mA. Each output can deliver up to 500 mA, however, the total current draw for all of the output ports combined must not exceed 500 mA at any time.

By default, Output Port 0 is assigned to transmit an integration enabled (IntEn) signal that indicates when exposure is taking place. For more information about the IntEn signal, see Section 5.3.7.

By default, Output Port 1 is assigned to transmit a trigger ready (TrigRdy) signal that goes high to indicate the earliest point at which exposure start for the next frame can be triggered. For more information about the TrigRdy signal, see Section 5.3.6.

The assignment of output signals to physical output ports can be changed by the user. See Sections 5.9.8 and 5.9.9.1 for more information about configuring output ports.

ⓘ	As shown in Figures 4-9 and 4-10, a voltage (VCC) between +10 and +30 VDC must be present on pin 10 and a DC ground must be present on pin 5. If this voltage and ground are not present, the input ports and the output ports will not operate. By default, output ports 0, 1, and 2 are set to a low state after power on. Output port 3 is initially set to low but will go high approximately 100 to 300 ms after power on. Output port 3 will remain high for approximately 750 ms and will then reset to low.

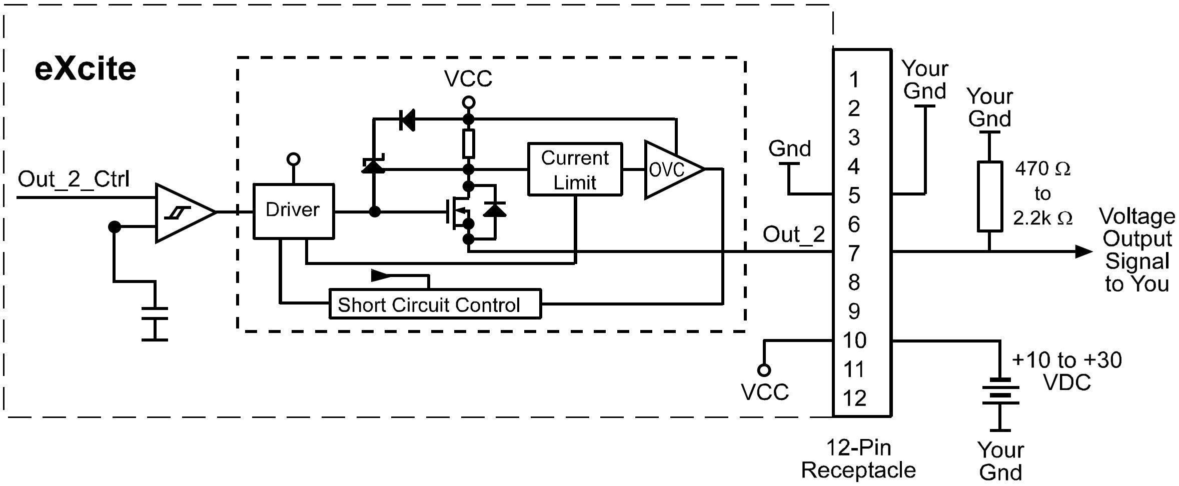


Figure 4-9: Typical Voltage Output Circuit

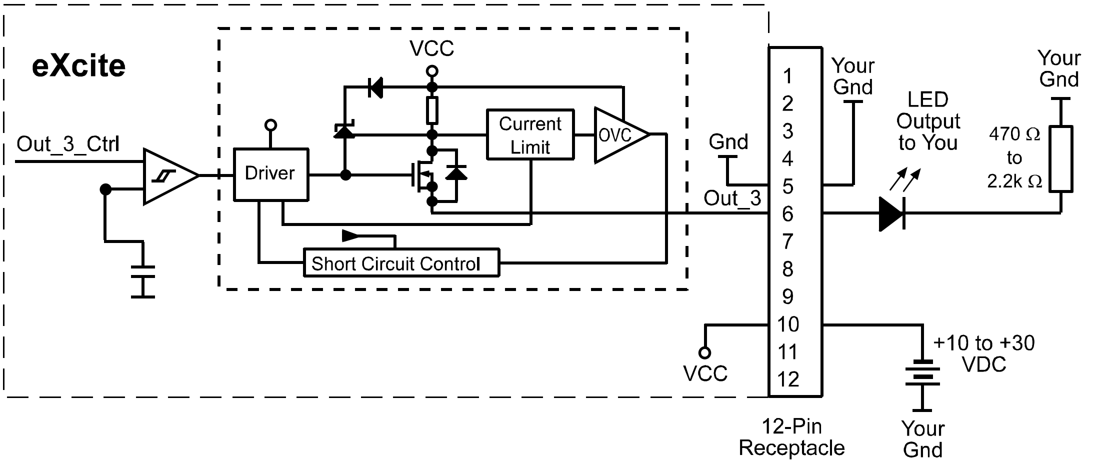


Figure 4-10: Typical LED Output Signal

4.4.4.3 I/O Port Cabling

The end of the I/O cable that connects to the eXcite must be terminated with a 12-pin Hirose HR10A-10P-12S micro-miniature locking plug (or the equivalent). The cable must be wired to conform with the pin assignments shown in Table 4-2.

The maximum length of the I/O cable is at least 10 meters. The cable must be shielded and must be constructed with twisted pair wire. Close proximity to strong magnetic fields should be avoided.

The required 12-pin Hirose plug is available from Basler. Basler also offers an I/O cable assembly that is terminated with a 12-pin Hirose plug on one end and is unterminated on the other end. Please contact your Basler sales representative to order connectors or I/O cables.

5 Operation and Features

5.1 Terminology

In this part of the documentation:

- The term “camera” means the camera section of the eXcite.
- The term “processor” means the processor section of the eXcite.
- The term “eXcite” means the product as a whole.

5.2 Functional Description

5.2.1 CMOS Camera Models

The CMOS camera models employ a CMOS sensor chip that provides features such as a full frame shutter and electronic exposure time control.

Normally, exposure time and charge readout are controlled by setting parameter values for the camera. Parameters are available to set exposure time and frame rate and to set the camera for single frame capture or continuous frame capture.

Exposure can also be controlled via an externally generated trigger (ExTrig) signal. The ExTrig signal facilitates periodic or non-periodic start of exposure. When exposure start is controlled by a rising ExTrig signal and the camera is set for the programmable exposure mode, exposure begins when the trigger signal goes high and continues for a pre-programmed period of time. Accumulated charges are read out when the programmed exposure time ends.

At readout, accumulated charges are transported from each of the sensor’s light-sensitive elements (pixels) to a pixel memory (see Figure 5-1). As the charges are moved out of the pixels and into the pixel memories, they are converted to voltages. There is a separate memory for each pixel. Because the sensor has memories that are separate from the pixels, exposure of the next image can begin while the sensor is reading out data from the previously captured image.

The pixel memories can be connected to a bus and there is one bus per vertical column. For readout, the pixel memories are addressed row-wise by closing a switch that connects each pixel memory in the addressed row to the column buses. As the voltages leave the column buses, they are amplified, an offset is applied, and they are digitized by the ADCs. A variable gain control and a 10 bit, analog-to-digital converter (ADC) are attached to the end of each column bus.

From the column buses, the digitized signals enter a horizontal output register. The 10 bit digital image data is then clocked out of the output register, through an FPGA, and into an image buffer.

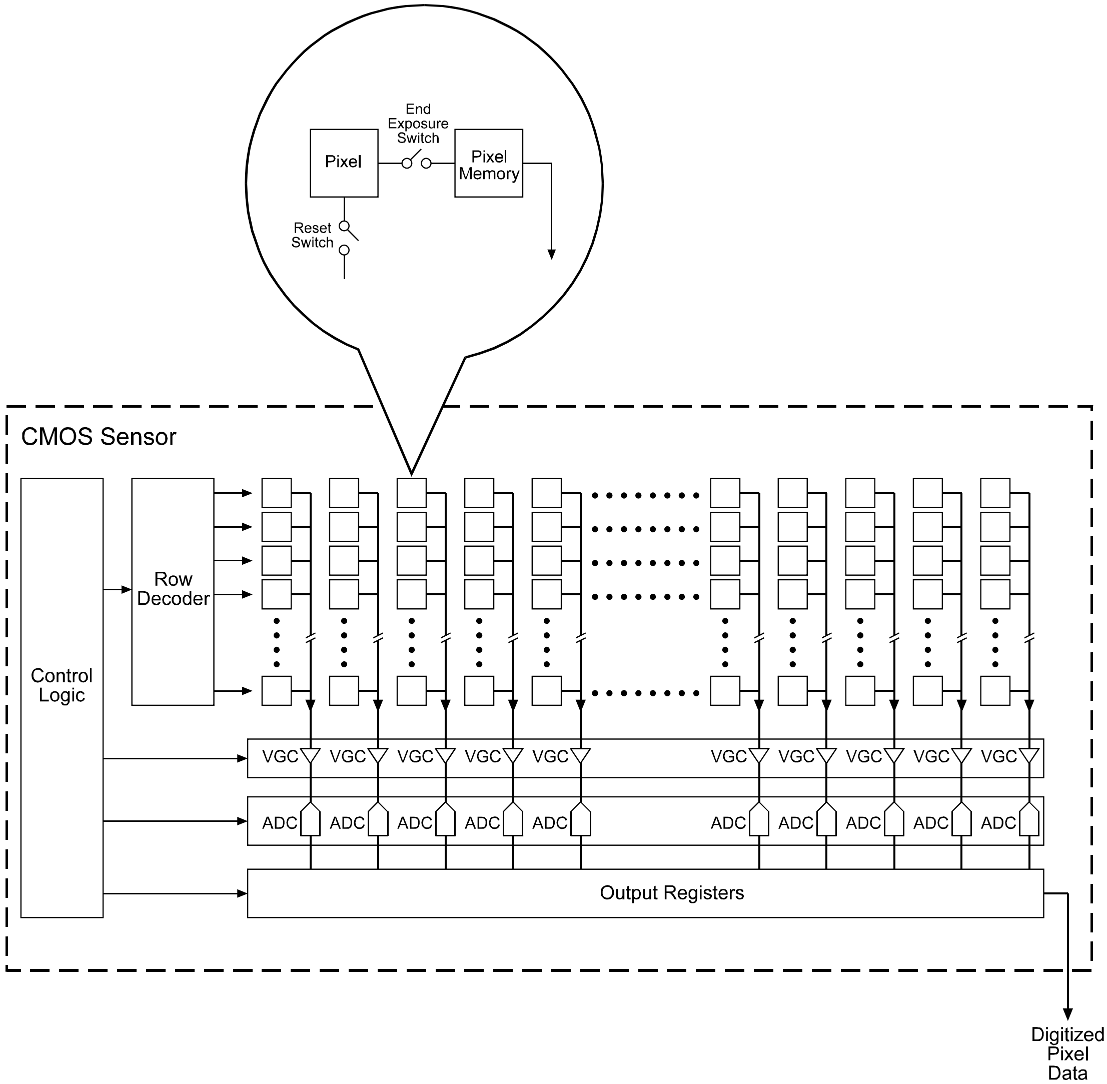


Figure 5-1: CMOS Sensor Architecture

When sensor readout is complete, the image data leaves the image buffer, passes back through the FPGA and is then transferred over a GPI bus to the MIPS processor.

The image buffer between the sensor and processor allows data to be read out of the sensor at a rate that is independent of the of the data transfer between the camera and the processor. This ensures that the data transfer rate has no influence on image quality.

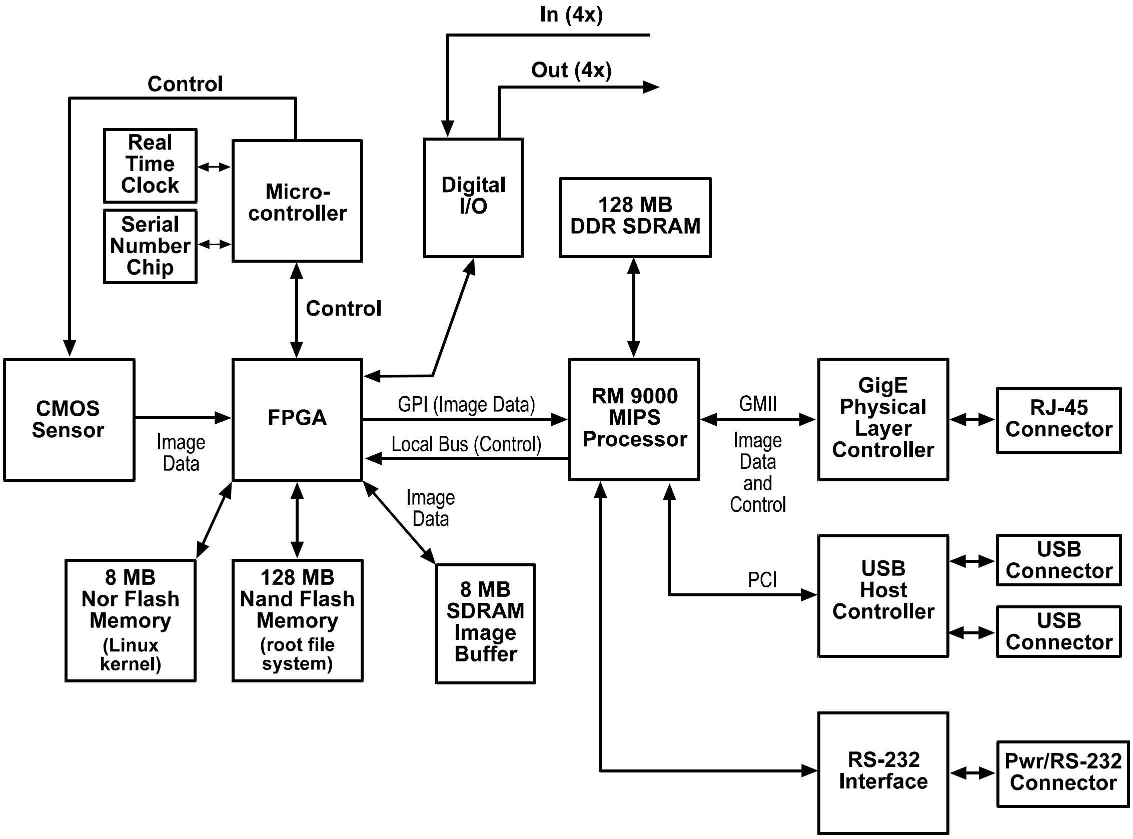


Figure 5-2: Block Diagram

5.2.2 CCD Camera Models

The CCD camera models employ a CCD sensor chip that provides features such as a full frame shutter and electronic exposure time control.

Normally, exposure time and charge readout are controlled by setting parameter values for the camera. Parameters are available to set exposure time and frame rate and to set the camera for single frame capture or continuous frame capture.

Exposure can also be controlled via an externally generated trigger (ExTrig) signal. The ExTrig signal facilitates periodic or non-periodic start of exposure. When exposure start is controlled by a rising ExTrig signal and the camera is set for the programmable exposure mode, exposure begins when the trigger signal goes high and continues for a pre-programmed period of time. Accumulated charges are read out when the programmed exposure time ends.

At readout, accumulated charges are transported from each of the sensor's light-sensitive elements (pixels) to the vertical shift registers (see Figure 5-3). The charges from the bottom line of pixels in the array are then moved into a horizontal shift register. Next, the charges are shifted out of the horizontal register. As the charges move out of the horizontal shift register, they are converted to voltages that are proportional to the size of each charge. Each voltage is then amplified by a Variable Gain Control (VGC) and digitized by a 12 bit, Analog-to-Digital converter (ADC). For optimal digitization, gain and brightness can be programmed by setting parameters in the camera. After each voltage has been amplified an digitized, it passes through an FPGA and into an image buffer. All shifting is clocked according to the camera's internal data rate. Shifting continues in a linewise fashion until all image data has been read out of the sensor.

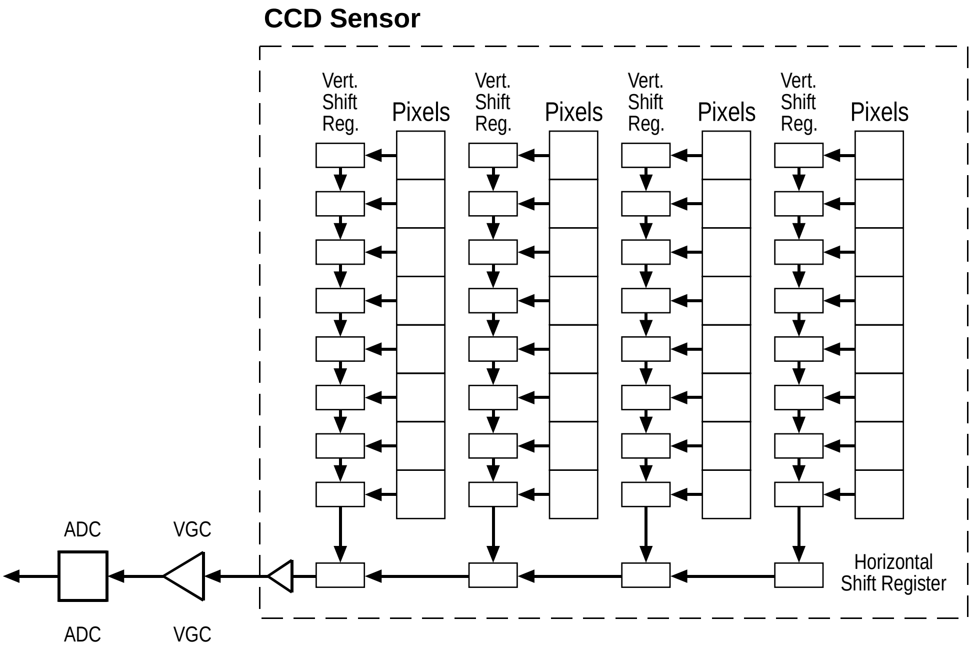


Figure 5-3: CCD Sensor Architecture

When sensor readout is complete, the image data leaves the image buffer, passes back through the FPGA and is then transferred over a GPI bus to the MIPS processor.

The image buffer between the sensor and processor allows data to be read out of the sensor at a rate that is independent of the of the data transfer between the camera and the processor. This ensures that the data transfer rate has no influence on image quality.

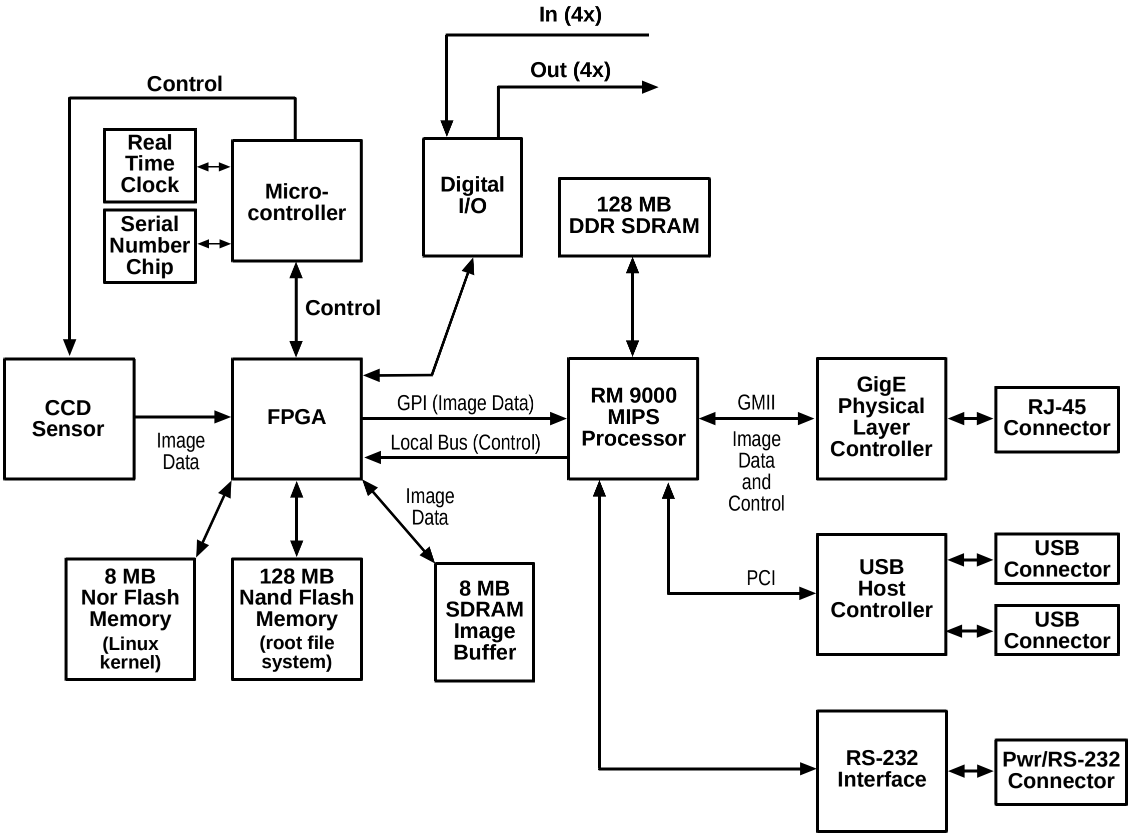


Figure 5-4: Block Diagram

5.3 Exposure Control

5.3.1 Controlling Exposure with Shot Parameters Only (No Triggering)

You can configure the eXcite so that exposure start will be controlled by simply setting the value of the camera's "one shot" parameter or "continuous shot" parameter via the eXcite API. When the eXcite is configured to start exposure based on shot parameter values only, a software trigger or an external trigger (ExTrig) signal is not required for exposure start.

Switching Off Triggering

If you want to control exposure start based on shot parameters alone, you must make sure that the camera's triggering feature is switched off. Triggering will be switched off when the value of the Trigger Enable parameter is set to false. To set the Trigger Enable parameter value, access the eXcite API and use the SetValue method for the TriggerEnable object. (For more information on using the API, refer to the API Reference documentation.)

Controlling Exposure with the One Shot Parameter

In "one-shot" operation, the camera exposes a single image and transfers it to the MIPS processor. Exposure begins when the value of the camera's One Shot parameter is set to true. Exposure time is determined by the value of the camera's Shutter parameter (see Section 5.3.4). The value of the One Shot parameter is automatically reset to false after transfer of the image data.

To set the One Shot parameter value, access the eXcite API and use the SetValue method for the OneShot object. (For more information on using the API, refer to the API Reference documentation.)

When using the one-shot method to start each exposure, you must not capture frames at a rate that exceeds the maximum allowed for the current settings. See Section 5.4 for more information about the maximum frame capture rate.

Controlling Exposure with the Continuous Shot Parameter

In "continuous-shot" operation, the camera continuously captures images and transfers them to the MIPS processor. The exposure of the first image begins when the value of the camera's Continuous Shot parameter is set to true. The exposure time for each image is determined by the value of the camera's Shutter parameter (see Section 5.3.4). The start of exposure on the second and subsequent images is automatically controlled by the camera. Image exposure and transmission will stop when the value of the Continuous Shot parameter is set to false.

To set the Continuous Shot parameter value, access the eXcite API and use the SetValue method for the ContinuousShot object. (For more information on using the API, refer to the API Reference documentation.)

In continuous-shot mode, the camera will capture frames at the maximum rate allowed for the current settings. See Section 5.4 for more information about adjusting the maximum frame capture rate.

ⓘ The explanations above are intended to give you a basic idea of how the One Shot and the Continuous Shot parameters can be used to trigger exposure start. For a more complete description, refer to the source code for the "SimpleGrab" and the "MultiGrab" sample programs in Section 3.5.

5.3.2 Controlling Exposure with a Software Trigger

You can configure the eXcite so that exposure will be controlled by setting the value of the camera’s Software Trigger parameter via the eXcite API. If you are controlling exposure with a software trigger, only the programmable exposure mode is available. In programmable mode, exposure starts when the Software Trigger parameter is set to true. The exposure time for each image is determined by the value of the camera’s Shutter parameter (see Section 5.3.4). The Software Trigger parameter will self clear (return to false) shortly after exposure start. Figure 5-5 illustrates programmable exposure with a software trigger.

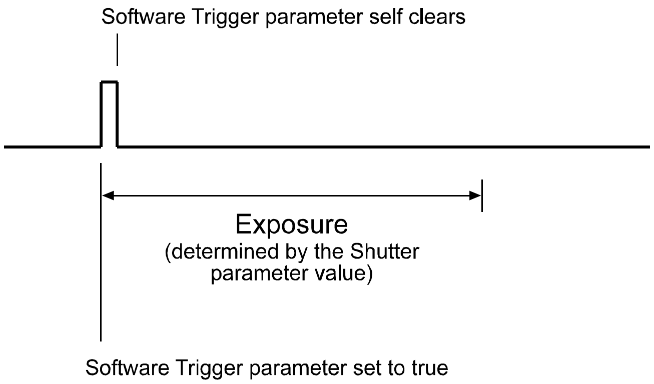

Software Trigger parameter self clears
Exposure
(determined by the Shutter parameter value)
Software Trigger parameter set to true

Figure 5-5: Programmable Exposure with a Software Trigger

Enabling the Software Trigger Feature

To enable the software trigger feature:

- Enable triggering by setting the Trigger Enable parameter to true. To set the Trigger Enable parameter value, access the eXcite API and use the SetValue method for the TriggerEnable object. (For more information on using the API, refer to the API Reference documentation.)
- Select the trigger source by setting the value of the Trigger Source parameter to TriggerSource_SoftTrig. To set the Trigger Source parameter value, access the API and use the SetValue method for the TriggerSource object.
- Set the mode for triggering to programmable by setting the value of the Trigger Mode parameter to TriggerMode_TriggerMode0 (setting the value to mode 0 selects the programmable mode). To set the Trigger Mode parameter value, access the API and use the SetValue method for the TriggerMode object.

Software Trigger / One-shot Operation

In Software Trigger/One-shot operation, setting the camera for ”one-shot“ prepares the camera to capture a single image. Exposure will begin when the Software Trigger parameter is set to true. To use this operating method, follow this sequence:

1. Use the shutter settings described in Section 5.3.4 to set your desired exposure time.
2. Set the value of the camera’s One Shot parameter to true. This prepares the camera to capture a single image. To set the One Shot parameter value, access the eXcite API and use

the SetValue method for the OneShot object. (For more information on using the API, refer to the API Reference documentation.)

3. Check the Software Trigger parameter value.
   a) If the parameter value is set to false, set the parameter value to true. Exposure will begin.
   b) If the parameter value is set to true, wait until the parameter value self-clears to false and then set it to true. Exposure will begin.

   To set the Software Trigger parameter value, access the eXcite API and use the SetValue method of the Software Trigger object. (For more information on using the API, refer to the API Reference documentation.)
   Note that the Software Trigger parameter self-clears to false after the exposure of the image has ended.
4. Exposure will continue for the length of time you specified in step 1.
5. At the end of the specified exposure time, readout and transfer of the captured image will take place.
6. To capture another image wait until the Software Trigger parameter has self-cleared to false.

Software Trigger / Continuous-shot Operation

In Software Trigger/Continuous-shot operation, setting the camera for “continuous-shot” prepares the camera to respond to software triggering. With this method of operation, an exposure will begin each time the Software Trigger parameter is set to true. To use this operating method, follow this sequence:

1. Use the shutter settings described in Section 5.3.4 to set your desired exposure time.
2. Set the value of the camera’s Continuous Shot parameter to true. To set the Continuous Shot parameter value, access the eXcite API and use the SetValue method for the ContinuousShot object. (For more information on using the API, refer to the API Reference documentation.)
3. Check the Software Trigger parameter value.
   a) If the parameter value is set to false, set the parameter value to true. Exposure will begin.
   b) If the parameter value is set to true, wait until the parameter value self-clears to false and then set it to true. Exposure will begin.

   To set the Software Trigger parameter value, access the eXcite API and use the SetValue method of the Software Trigger object. (For more information on using the API, refer to the API Reference documentation.)
   Note that the Software Trigger parameter self-clears to false after the exposure of the image has ended.
4. Exposure will continue for the length of time you specified in step 1.
5. At the end of the specified exposure time, readout and transfer of the captured image will take place.
6. Repeat steps 3 and 4 each time you want to begin exposure and capture an image.
7. To disable continuous-shot operation, set the value of the camera’s Continuous Shot parameter to false.

When you are triggering exposure start with a software trigger, you must not capture frames at a rate that exceeds the maximum allowed for the current settings. See Section 5.4 for more information about the maximum frame capture rate.

ⓘ The explanation above is intended to give you a basic idea of how Software Trigger/ Continuous Shot operation works. To actually operate the camera, you will need to use techniques similar to those shown in the source code for the “MultiGrab” and the “SimpleTrigger” sample programs in Section 3.5. (The SimpleTrigger sample actually illustrates using an external hardware trigger, but looking through this sample will give you a general idea of how triggering is done.)

Why Use the Software Trigger?

At first glance, using the software trigger feature to start image exposure appears to be equivalent to just issuing a one-shot command as described in Section 5.3.1. The difference is in the way the camera reacts to each method. With a one-shot command only, there will be some delay between the One Shot parameter being set to true and the actual start of exposure time. This delay is required so that the camera can be properly set up to react to the change in the parameter. With the software trigger method, there is no delay between the Software Trigger parameter being set to true and the start of exposure. After the parameter value is set, exposure begins immediately. So the advantage of the software trigger feature is that it gives you more precise control of exposure start.

5.3.3 Controlling Exposure with an ExTrig Signal

You can configure the eXcite so that an external trigger (ExTrig) signal applied to one of the input ports will control exposure. A rising edge or a falling edge of the ExTrig signal can be used to trigger exposure start.

The ExTrig signal can be periodic or non-periodic. When the camera is operating under control of an ExTrig signal, the period of the ExTrig signal will determine the rate at which the camera is capturing frames:

$$\frac{1}{\text{ExTrig period in seconds}} = \text{frame capture rate}$$

For example, if you are operating a camera with an ExTrig signal period of 20 ms (0.020 s):

$$\frac{1}{0.020} = 50\text{ fps}$$

So in this case, the frame rate is 50 fps.

The minimum high time for a rising edge trigger (or low time for a falling edge trigger) is 1 µs.

When you are triggering exposure start with an ExTrig signal, you must not capture frames at a rate that exceeds the maximum allowed for the current settings. See Section 5.4 for more information about the maximum frame capture rate.

Exposure Modes

If you are triggering exposure start with an ExTrig signal, two exposure modes are available, programmable mode and level controlled mode.

Programmable Exposure Mode

When programmable mode is selected, the exposure time for each image is determined by the value of the camera's Shutter parameter (see Section 5.3.4). If the camera is set for rising edge triggering, exposure starts when the ExTrig signal rises. If the camera is set for falling edge triggering, exposure starts when the ExTrig signal falls. Figure 5-6 illustrates programmable exposure with the camera set for rising edge triggering.

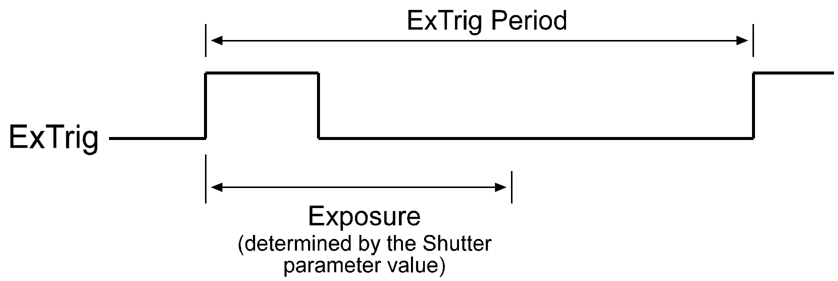


Figure 5-6: Programmable Exposure with Rising Edge Triggering

Level Controlled Exposure Mode

When level controlled mode is selected, the length of the exposure will be controlled by the ExTrig signal. If the camera is set for rising edge triggering, exposure begins when the ExTrig signal rises and continues until the ExTrig signal falls. If the camera is set for falling edge triggering, exposure begins when the ExTrig signal falls and continues until the ExTrig signal rises. Figure 5-7 illustrates level controlled exposure with the camera set for rising edge triggering.

Level controlled exposure is especially useful if you intend to vary the length of the exposure time for each captured image.

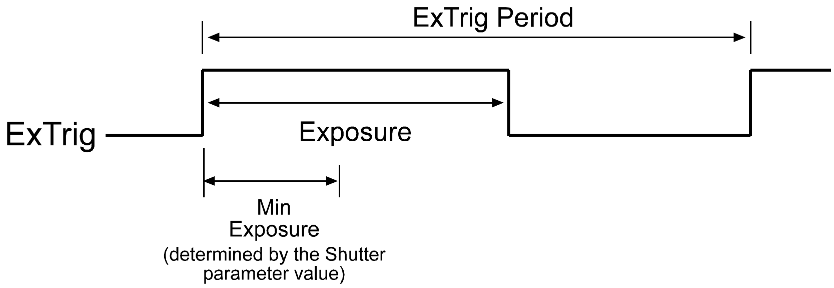


Figure 5-7: Level Controlled Exposure with Rising Edge Triggering

When you operate the camera in level controlled mode, you must use the Shutter parameter setting (see Section 5.3.4) to set a minimum exposure time. In level controlled operation, the exposure time determined by the Shutter parameter setting represents an absolute minimum time for each exposure. For each full cycle of the ExTrig signal:

- If the exposure time as controlled by the ExTrig signal is less than or equal to the minimum, the camera will use the minimum exposure time.
- If the exposure time as controlled by the ExTrig signal is greater than the minimum, the camera will use the exposure time determined by the ExTrig signal.

Example 1: Assume that you set your camera for level controlled exposure with rising edge triggering and that you set the Shutter parameter for a 2000 µs minimum exposure time. Next assume that a rising edge has triggered an exposure and that the ExTrig signal remained high for 1500 µs. In this case, the camera would have a 2000 µs exposure time because the ExTrig high time was less than the minimum exposure setting.

Example 2: Assume that you set your camera for ExTrig level controlled exposure with rising edge triggering and that you set the Shutter parameter for a 2000 µs minimum exposure time. Next assume that a rising edge has triggered an exposure and that the ExTrig signal remained high for 3500 µs. In this case, the camera would have a 3500 µs exposure time because the ExTrig high time was greater than the minimum exposure setting.

When using level controlled exposure, you should adjust the Shutter parameter setting to represent the shortest exposure time you intend to use. For example, assume that you will be using level controlled exposure and that you intend to use the ExTrig signal to vary the exposure time in a range from 3000 µs to 5500 µs. In this case you would use the Shutter parameter setting to set the minimum exposure time to 3000 µs.

Setting the eXcite for External Triggering

To set the eXcite for external triggering:

- Enable triggering by setting the value of the Trigger Enable parameter to true.

  To set the Trigger Enable parameter value, access the eXcite API and use the SetValue method for the TriggerEnable object. (For more information on using the API, refer to the API Reference documentation.)

- Set the value of the Trigger Polarity parameter to TriggerPolarity_LowActive to select falling edge triggering or TriggerPolarity_HighActive to select rising edge triggering.

  To set the Trigger Polarity parameter value, access the API and use the SetValue method for the TriggerPolarity object.

- Set the value of the Trigger Mode parameter to TriggerMode_TriggerMode0 to select the programmable exposure mode or to TriggerMode_TriggerMode1 to select the level controlled exposure mode.

  To set the Trigger Mode parameter value, access the API and use the SetValue method for the TriggerMode object.

- Select the physical input port that will be used to receive the ExTrig signal:

  - To select physical input port 0, set the value of the Trigger Source parameter to TriggerSource_ExTrigPort0.
  - To select physical input port 1, set the value of the Trigger Source parameter to TriggerSource_ExTrigPort1.
  - To select physical input port 2, set the value of the Trigger Source parameter to TriggerSource_ExTrigPort2.
  - To select physical input port 3, set the value of the Trigger Source parameter to TriggerSource_ExTrigPort3.

  To set the Trigger Source parameter value, access the API and use the SetValue method for the TriggerSource object.

> ⓘ Physical input port 0 is the default port for receiving the ExTrig signal. The electrical characteristics of the ExTrig input signal that you apply to the eXcite must be as described in Section 4.4.4.1.

Using External Triggering

To operate the camera properly, the ExTrig signal must be used in combination with changes to the camera's One Shot or Continuous Shot parameters. You should also monitor the Trigger Ready signal and you should base the timing of the ExTrig signal on the state of the Trigger Ready signal (see Section 5.3.6).

The following descriptions assume that the camera is set for a rising edge ExTrig signal and for the programmable exposure mode.

ExTrig / One Shot Operation

In ExTrig/One shot operation, a change to the value of the camera's One Shot parameter setting prepares the camera to capture a single image. When the ExTrig signal rises, exposure of the image will begin.

To use this operating method, follow this sequence:

1. Set the value of the Shutter parameter as described in Section 5.3.4 for your desired exposure time.
2. Set the value of the camera's One Shot parameter to true. This will prepare the camera to react to the ExTrig signal.

   To set the One Shot parameter value, access the eXcite API and use the SetValue method for the OneShot object. (For more information on using the API, refer to the API Reference documentation.)
3. Check the state of the TrigRdy signal:
   a) If TrigRdy is high, you can toggle ExTrig when desired.
   b) If TrigRdy is low, wait until TrigRdy goes high and then toggle ExTrig when desired.
4. When ExTrig rises, exposure will begin. Exposure will continue for the length of time you specified in step 1.
5. At the end of the specified exposure time, the captured image will be read out of the sensor and transferred to the processor.
6. The value of the One Shot parameter will automatically reset to false after image transfer.
7. To capture another image, repeat steps 1 through 5.

ExTrig / Continuous Shot Operation

In ExTrig/Continuous shot operation, a change to the value of the camera's Continuous Shot parameter setting prepares the camera to capture multiple images. With this method of operation, exposure will begin on each rising edge of the ExTrig signal.

To use this operating method, follow this sequence:

1. Set the value of the Shutter parameter as described in Section 5.3.4 for your desired exposure time.
2. Set the value of the camera's Continuous Shot parameter to true. This will prepare the camera to react to the ExTrig signal.

   To set the Continuous Shot parameter value, access the eXcite API and use the SetValue method for the ContinuousShot object. (For more information on using the API, refer to the API Reference documentation.)
3. Check the state of the TrigRdy signal:
   a) If TrigRdy is high, you can toggle ExTrig when desired.
   b) If TrigRdy is low, wait until TrigRdy goes high and then toggle ExTrig when desired.

4. When ExTrig rises, exposure will begin. Exposure will continue for the length of time you specified in step 1.
5. At the end of the specified exposure time, the captured image will be read out of the sensor and transferred to the processor.
6. Repeat steps 3 and 4 each time you want to begin exposure and capture an image.
7. To disable continuous-shot operation, set the value of the camera’s Continuous Shot parameter to false. This camera will no longer react to the ExTrig signal.

ⓘ	These descriptions of external triggering are included to give you a basic idea of how the ExTrig signal can be used along with the shot parameters to trigger exposure start. For a more complete description, refer to the source code for the “Simple Trigger” sample program in Section 3.5.

5.3.4 Setting the Exposure Time

Many of the eXcite’s exposure control modes require you to specify an exposure time. For those modes, the exposure time will be determined by a combination of two elements. The first element is the value of the Shutter parameter, and the second element is the shutter time base. The exposure time is determined by the product of these two elements:

Exposure Time = (Shutter Parameter Value) x (Shutter Time Base)

The shutter time base is fixed at 20 µs by default. The exposure time is normally adjusted by changing the value of the Shutter parameter. The Shutter parameter value can range from 1 to 4095 (decimal). So if the value is set to 100, for example, the exposure time will be 100 x 20 µs or 2000 µs.

To set the Shutter parameter value, access the eXcite API and use the SetValue method for the Shutter object. (For more information on using the API, refer to the API Reference documentation.)

As mentioned above, the exposure time base is normally fixed at 20 µs and the exposure time is normally adjusted by changing the Shutter parameter value setting only. However, if you require a exposure time that is shorter or longer than what you can achieve by changing the Shutter parameter value alone, the shutter time base can also be changed. The Shutter Time Base smart feature can be used to change the shutter time base. For more information on the shutter time base and how to change it, see Section 5.10.2.2.

5.3.5 Overlapped and Non-overlapped Exposure

The image capture process on the eXcite includes two distinct parts. The first part is the exposure of the pixels in the sensor. Once exposure is complete, the second part of the process – readout of the pixel values from the sensor – takes place.

In regard to this image capture process, there are two common ways for the camera to operate, with “non-overlapped” exposure and with “overlapped” exposure. In the non-overlapped mode of operation, each time an image is captured, the camera completes the entire exposure/readout process before capture of the next image is triggered. This situation is illustrated in Figure 5-8.

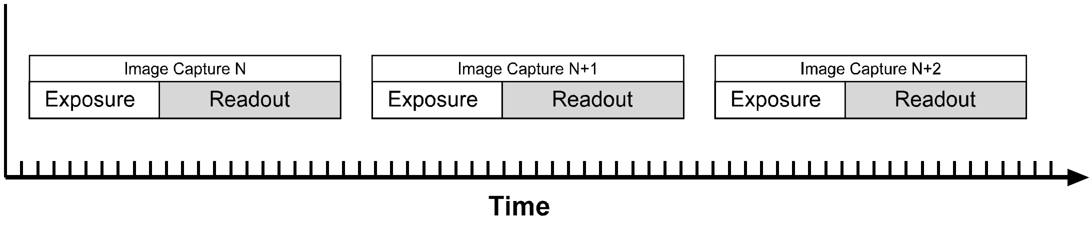


Figure 5-8: Non-overlapped Exposure

While operating in a non-overlapped fashion is perfectly normal and is appropriate for many situations, it is not the most efficient way to operate the camera in terms of frame rate. On the eXcite, it is allowable to begin exposing a new image while a previously captured image is being read out. This situation is illustrated in Figure 5-9 and is known as operating the camera with “overlapped” exposure.

As you can see, running the camera with readout and exposure overlapped can allow higher frame rates because the camera is performing two processes at once.

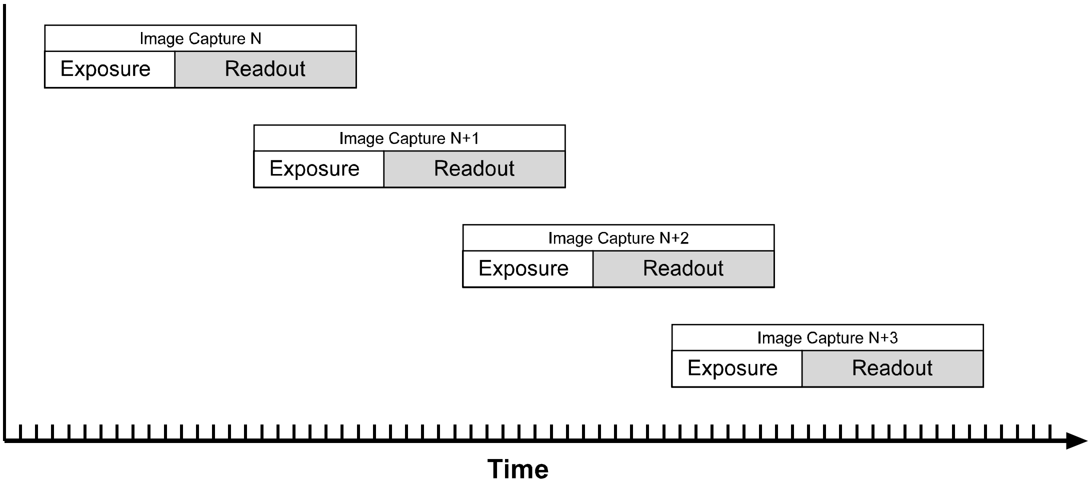


Figure 5-9: Overlapped Exposure

Determining whether your camera is operating with overlapped or non-overlapped exposures is not a matter of writing to a register or switching a setting on or off. Rather the way that you operate the camera will determine whether the exposures are overlapped or not overlapped. If we define the "frame period" as the time from the start of exposure for one image capture to the start of exposure for the next image capture, then:

- Exposure will overlap when: Frame Period ≤ Exposure Time + Readout Time
- Exposure will not overlap when: Frame Period > Exposure Time + Readout Time

You can calculate the readout time for a captured image by using the formula on page 5-20.

Guidelines for Overlapped Operation

If you will be operating the camera with overlapped exposures, there are two **very important** guidelines to keep in mind:

- You must not begin the exposure of a new frame while the exposure of the previous frame is in progress.
- You must not end the exposure of a new frame until readout of the previous frame is complete.

If either of these guidelines is violated, the camera will produce unacceptable images or may stop capturing images and require a reset.

When you are operating the camera with overlapped exposures, you could use the camera's Shutter parameter setting and the timing numbers shown in Section 5.3.9 to calculate when it is safe to begin and end each exposure. However, there is a much more convenient way to know when it is safe to begin and end overlapped exposures. The eXcite supplies a "Trigger Ready" signal that is specifically designed to let you perform overlapped exposures safely and efficiently. For more information on using the Trigger Ready signal, please see Section 5.3.6.

5.3.6 Trigger Ready Signal

As described in Section 5.3.5, the camera can operate in an "overlapped" exposure fashion. When the camera is operated in this manner, it is especially important that:

- exposure of a new image not start until exposure of the previous image has ended, and
- exposure of the new image not end until readout of the previous image is complete.

The eXcite supplies a "Trigger Ready" (TrigRdy) signal you can use to ensure that these conditions are met when you are using a hardware trigger signal to trigger image capture. When you are capturing images, the camera automatically calculates the earliest moment it is safe to trigger each new capture. The trigger ready signal will go high when it is safe to trigger a capture, will go low when the capture has started, and will go high again when it is safe to trigger the next capture (see Figure 5-10). The camera calculates the rise of the trigger ready signal based on the current Shutter parameter setting, the current size of the area of interest, and the time it will take to readout the captured pixel values from the sensor.

The trigger ready signal is especially useful if you want to run the camera at the maximum frame capture rate for the current conditions. If you monitor the trigger ready signal and you begin the capture of each new image immediately after the signal goes high, you will be sure that the camera is operating at the maximum frame capture rate for the current conditions.

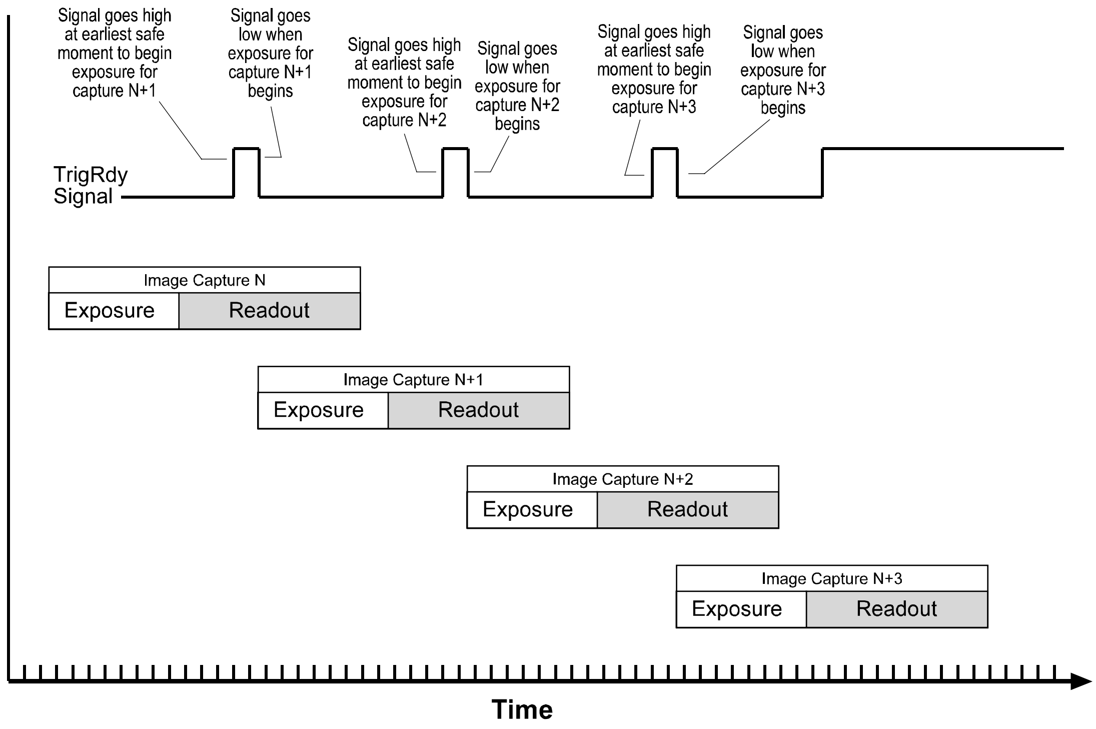

Signal goes high at earliest safe moment to begin exposure for capture N+1
Signal goes low when exposure for capture N+1 begins
Signal goes high at earliest safe moment to begin exposure for capture N+2
Signal goes low when exposure for capture N+2 begins
Signal goes high at earliest safe moment to begin exposure for capture N+3
Signal goes low when exposure for capture N+3 begins
TrigRdy Signal
Image Capture N
Exposure
Readout
Image Capture N+1
Exposure
Readout
Image Capture N+2
Exposure
Readout
Image Capture N+3
Exposure
Readout
Time

Figure 5-10: Trigger Ready Signal

By default, the TrigRdy signal is assigned to physical output port 1 on the camera. See Section 4.4.4.2 for a description of the electrical characteristics of the camera's physical output ports.

The assignment of the TrigRdy signal to a physical output port can be changed. See Section 5.9.9.1 for more information on changing the assignment of camera output signals to physical output ports.

ⓘ	If you signal the camera to start an exposure when trigger ready is low, the camera will simply ignore the signal. If the camera is in continuous shot mode and external triggering is disabled, the trigger ready output signal will not be present.

5.3.7 Integrate Enabled Signal

The camera’s “Integrate Enabled” (IntEn) signal goes high when each exposure begins and goes low when each exposure ends (see Figure 5-11). This signal can be used as a flash trigger and is also useful when you are operating a system where either the camera or the object being imaged is movable. For example, assume that the camera is mounted on an arm mechanism and that the mechanism can move the camera to view different portions of a product assembly. Typically, you do not want the camera to move during exposure. In this case, you can monitor the IntEn signal to know when exposure is taking place and thus know when to avoid moving the camera.

By default, the IntEn signal is assigned to physical output port 0 on the camera. See Section 4.4.4.2 for a description of the electrical characteristics of the camera’s physical output ports.

The assignment of the IntEn signal to a physical output port can be changed. See Section 5.9.9.1 for more information on changing the assignment of camera output signals to physical output ports.

ⓘ	When you use the integrate enabled signal, be aware that there is a delay in the rise and the fall of the signal in relation to the start and the end of exposure. See Figure 5-11 for details.

5.3.8 A Recommended Method for Controlling Exposure

ⓘ The camera can be programmed to begin exposure on a rising edge or on a falling edge of an ExTrig signal. Also, two modes of exposure control are available: programmable and level controlled (see Section 5.3.3). For this example, we are assuming that a rising edge trigger and the programmable exposure mode are used.

If you require close control of exposure start, there are several general guidelines that must be followed:

- The camera should be set for continuous shot operation and an external trigger (ExTrig) signal must be used to start exposure.
- You must monitor the trigger ready (TrigRdy) signal.
- A rising edge of the ExTrig signal must only occur when the TrigRdy signal is high.

Assuming that these general guidelines are followed, the reaction of the camera to a rising external trigger signal will be as shown in Figure 5-11:

- The start of exposure will typically occur 17 µs after the rise of the ExTrig signal.
- The integrate enabled (IntEn) signal will rise between 10. and 14 µs after the start of exposure.
- The actual length of exposure will be equal to the programmed exposure time.
- The IntEn signal will fall between 11 and 15 µs after the end of exposure.

5.3.9 Exposure Timing Charts

5.3.9.1 CMOS Camera Models

As shown in Figure 5-11, after each image is captured, the camera begins reading out the captured image data from the CMOS sensor into a buffer in the camera. Once readout is complete, the captured frame is packetized, transferred from the buffer to the MIPS processor, and de-packetized. This buffering technique is an important element in achieving the highest possible frame rate with the best image quality.

The **frame readout time** is the amount of time it takes to read a captured image out of the CMOS sensor and into the image buffer.

The **frame transfer time** is the amount of time it takes to transfer the captured image from the camera section of the eXcite to the MIPS processor.

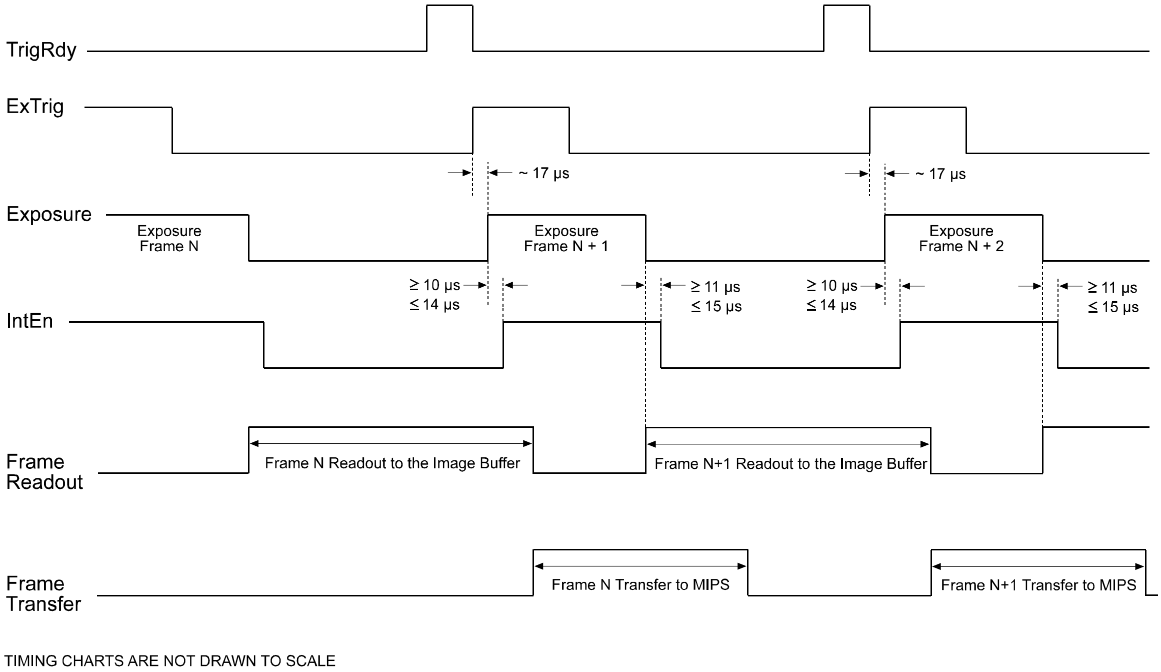


Figure 5-11: Exposure Controlled with an ExTrig Signal

You can calculate the frame readout time with this formula:

Frame Readout Time = [(AOI Height + 2) x C µs] + C µs

Where: C = 11.46 for exA640-180m/c models

C = 15.28 for exA640-120m/c or exA640-60m/c models

You can calculate the frame transfer time with this formula:

Frame Transfer Time = Packets transferred per frame x 42 µs

To determine the number of packets transferred per frame, check the value of the Packet Number parameter. The Packet Number parameter indicates the number of packets needed to transfer a captured frame from the camera section to the MIPS processor.

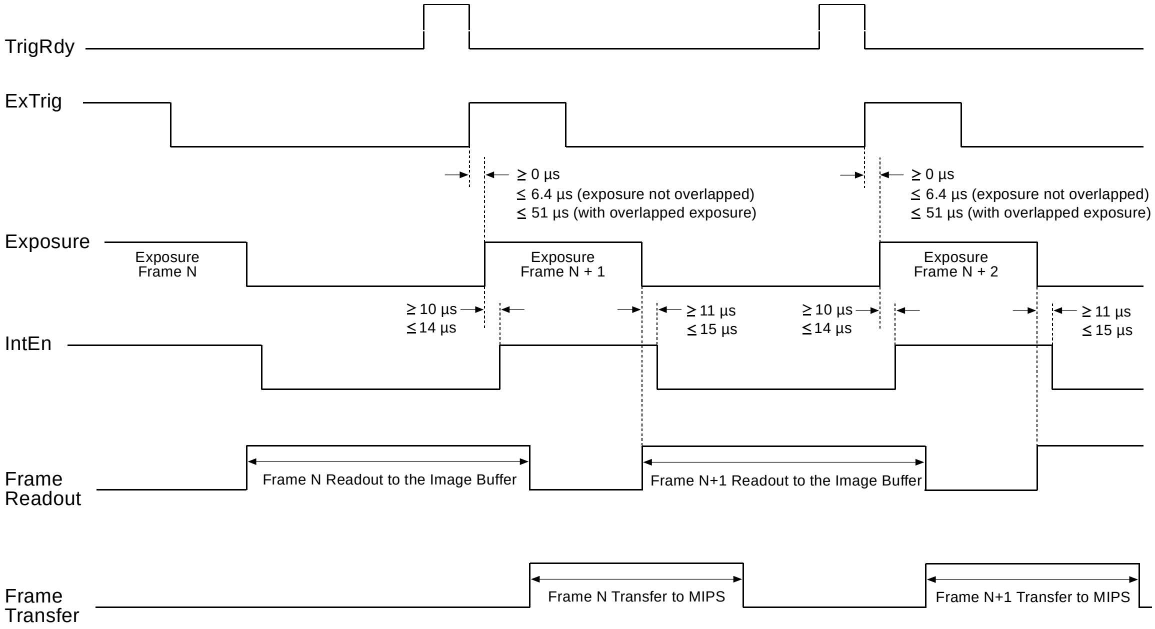
To check the value of the Packet Number parameter access the eXcite API and use the GetValue method for the PacketNumber object. (For more information on using the API, refer to the API Reference documentation.)

5.3.9.2 CCD Camera Models

As shown in Figures 5-12 and 5-13 after each image is captured, the camera begins reading out the captured image data from the CCD sensor into a buffer in the camera. Once readout is complete, the captured frame is packetized, transferred from the buffer to the MIPS processor, and de-packetized. This buffering technique is an important element in achieving the highest possible frame rate with the best image quality.

The **frame readout time** is the amount of time it takes to read a captured image out of the CCD sensor and into the image buffer.

The **frame transfer time** is the amount of time it takes to transfer the captured image from the camera section of the eXcite to the MIPS processor.


TrigRdy
ExTrig
≥ 0 µs
≤ 6.4 µs (exposure not overlapped)
≤ 51 µs (with overlapped exposure)
≥ 0 µs
≤ 6.4 µs (exposure not overlapped)
≤ 51 µs (with overlapped exposure)
Exposure
Exposure Frame N
Exposure Frame N + 1
Exposure Frame N + 2
≥ 10 µs
≤ 14 µs
≥ 11 µs
≤ 15 µs
≥ 10 µs
≤ 14 µs
≥ 11 µs
≤ 15 µs
IntEn
Frame Readout
Frame N Readout to the Image Buffer
Frame N+1 Readout to the Image Buffer
Frame Transfer
Frame N Transfer to MIPS
Frame N+1 Transfer to MIPS

TIMING CHARTS ARE NOT DRAWN TO SCALE

Figure 5-12: Exposure Controlled with an ExTrig Signal for exA1390-19m/c Models

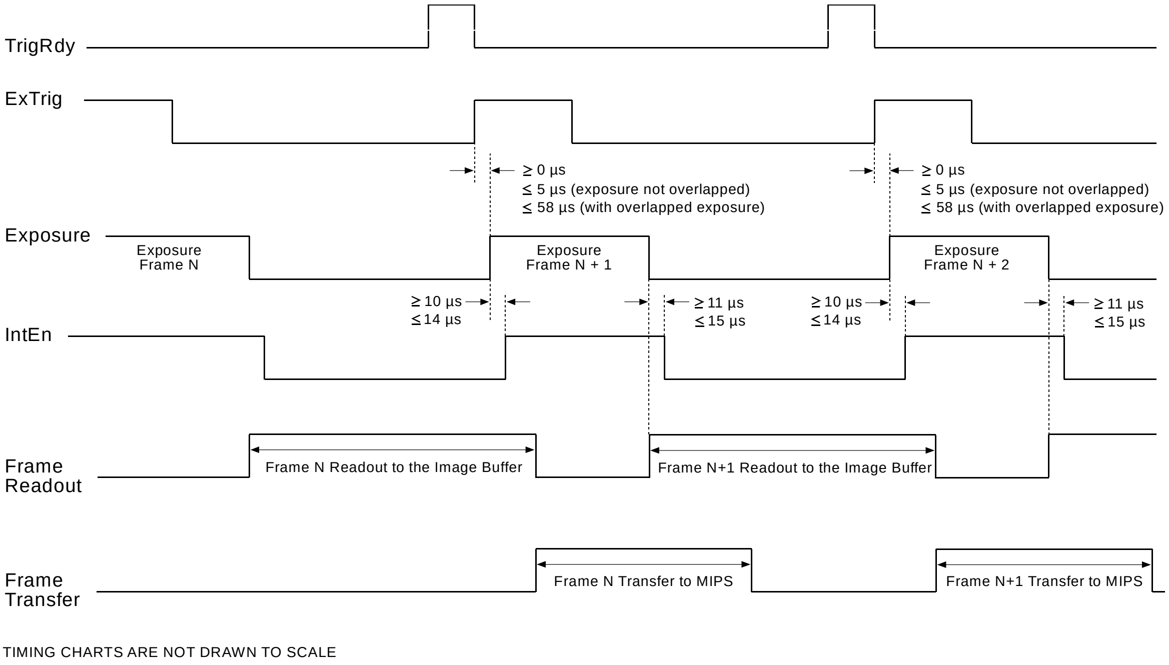

TrigRdy
ExTrig
≥ 0 µs
≤ 5 µs (exposure not overlapped)
≤ 58 µs (with overlapped exposure)
≥ 0 µs
≤ 5 µs (exposure not overlapped)
≤ 58 µs (with overlapped exposure)
Exposure
Exposure Frame N
Exposure Frame N + 1
Exposure Frame N + 2
≥ 10 µs
≤ 14 µs
≥ 11 µs
≤ 15 µs
≥ 10 µs
≤ 14 µs
≥ 11 µs
≤ 15 µs
IntEn
Frame Readout
Frame N Readout to the Image Buffer
Frame N+1 Readout to the Image Buffer
Frame Transfer
Frame N Transfer to MIPS
Frame N+1 Transfer to MIPS
TIMING CHARTS ARE NOT DRAWN TO SCALE

Figure 5-13: Exposure Controlled with an ExTrig Signal for exA1600-14m/c Models

You can calculate the frame readout time with this formula:

Frame Readout Time = [AOI Height x E µs] + C µs

Where: C = 7029.11 for exA1390-19m

C = 7073.67 for exA1390-19c

C = 6557.8 for exA1600-14m

C = 6610.35 for exA1600-14c

E = 44.56 for exA1390-19m/c models

E = 52.55 for exA1600-14m/c models

You can calculate the frame transfer time with this formula:

Frame Transfer Time = Packets transferred per frame x 88 µs

To determine the number of packets transferred per frame, check the value of the Packet Number parameter. The Packet Number parameter indicates the number of packets needed to transfer a captured frame from the camera section to the MIPS processor.

To check the value of the Packet Number parameter access the eXcite API and use the GetValue method for the PacketNumber object. (For more information on using the API, refer to the API Reference documentation.)

5.4 Maximum Allowed Frame Capture Rate

5.4.1 On exA640-180m/c and exA640-120m/c Models

In general, the maximum allowed frame capture rate for the exA640-180m/c and the exA640-120m/c models of the eXcite can be limited by three factors:

- The amount of time it takes to read out a captured image from the image sensor to the frame buffer. This time varies with the height of the area of interest (see Section 5.9.4). Shorter AOIs take less time to read out of the sensor.
- The number of packets needed to transfer a captured image from the eXcite's camera section to the MIPS processor. This can vary depending on the Bandwidth setting (see Section 5.4.5).
- The exposure time (see Sections 5.3.1 through 5.3.4) for a captured frame.

  If you use very long exposure times, you can capture fewer images per second.

To determine the maximum allowed frame capture rate with your current camera settings, calculate a result for each of the three formulas below. The formula that returns the lowest value will determine the maximum frame capture rate with the camera at its current settings. (In other words, the factor that restricts the rate the most will determine the maximum allowed frame capture rate.)

Formula 1 calculates the maximum frame capture rate based on the sensor readout time:

$$\text{Max. Frames/s} = \frac{1}{[(\text{AOI Height} + 2) \times \text{C µs}] + \text{C µs}}$$

Where: AOI Height = the height of the area of interest.

C = 11.46 for exA640-180m/c models

C = 15.28 for exA640-120m/c models

Formula 2 calculates the maximum frame capture rate based on the number of packets needed to transfer a frame:

$$\text{Max. Frames/s} = \frac{1}{\text{Packets transferred per frame} \times 42\ \text{µs}}$$

You can determine the number of packets needed to transfer a frame by checking the Packet Number parameter (see Section 5.4.5).

Formula 3 calculates the maximum frame capture rate based on the current exposure time:

$$\text{Max. Frames/s} = \frac{1}{\text{Exposure time in µs} + 20.8\ \text{µs}}$$

Example of Calculating the Maximum Allowed Frame Capture Rate

Assume that your exA640-180m camera is set for an area of interest of 200 columns wide and 210 rows high and that your exposure time is set for 2000 µs. Also assume that after making all parameter settings, you check the current value of the Packet Number parameter. You find that the packet number (packets transferred per frame) with the current settings is 11.

Formula 1:

$$\text{Max. Frames/s} = \frac{1}{[(210 + 2) \times 11.46\ \mu s] + 11.46\ \mu s}$$

Max. Frames/s = 409.6

Formula 2:

$$\text{Max. Frames/s} = \frac{1}{11 \times 42\ \mu s}$$

Max. Frames/s = 2164.5

Formula 3:

$$\text{Max. Frames/s} = \frac{1}{2000\ \mu s + 20.8\ \mu s}$$

Max. Frames/s = 494.8

Formula one returns the lowest value. So in this case, the limiting factor is the AOI size and the maximum allowed frame capture rate would be 409.6 frames per second.

5.4.2 On exA640-60m/c Models

The absolute maximum allowed frame capture rate for the exA640-60m/c model of the eXcite is 60 frames per second and this limit does not change when you change the size of your AOI.

The maximum allowed frame capture rate for the exA640-60m/c model can also be limited by two other factors:

- The number of packets needed to transfer a captured image from the eXcite's camera section to the MIPS processor. This can vary depending on the Bandwidth setting (see Section 5.4.5).
- The exposure time (see Sections 5.3.1 through 5.3.4).

  If you use very long exposure times, you can capture fewer images per second.

To determine the maximum allowed frame capture rate, calculate a result for each of the two formulas below. The maximum allowed frame capture rate will be lowest of 60 frames per second, the result of formula 1, or the result of formula 2.

Formula 1 calculates the maximum frame capture rate based on the number of packets needed to transfer a frame:

$$\text{Max. Frames/s} = \frac{1}{\text{Packets transferred per frame} \times 42\ \mu\text{s}}$$

You can determine the number of packets needed to transfer a frame by checking the Packet Number parameter (see Section 5.4.5).

Formula 2 calculates the maximum frame capture rate based on the current exposure time:

$$\text{Max. Frames/s} = \frac{1}{\text{Exposure time in } \mu\text{s} + 20.8\ \mu\text{s}}$$

Example of Calculating the Maximum Allowed Frame Capture Rate

Assume that you are using an exA640-60m set for an exposure time of 2000 µs. Also assume that after making all parameter settings, you check the current value of the Packet Number parameter. You find that the packet number (packets transferred per frame) with the current settings is 79.

Formula 1:

$$\text{Max. Frames/s} = \frac{1}{79 \times 42\ \mu s}$$

Max. Frames/s = 301.4

Formula 2:

$$\text{Max. Frames/s} = \frac{1}{2000\ \mu s + 20.8\ \mu s}$$

Max. Frames/s = 494.8

In this case, the result of formula 1 and the result of formula 2 are both greater than 60. But since the absolute maximum frame capture rate for the exA640-60m/c is 60, then our maximum allowed frame capture rate in this case is 60.

(If one of the formulas returned a value lower than 60 then that result would determine the maximum frame rate. For example, if formula 2 returned a value of 48, then 48 would be your maximum allowed frame capture rate.)

5.4.3 On exA1390-19m/c and exA1600-14m/c Models

In general, the maximum allowed frame capture rate for the exA1390-19m/c and the exA1600-14m/c models of the eXcite can be limited by three factors:

- The amount of time it takes to read out a captured image from the image sensor to the frame buffer. This time varies with the height of the area of interest (see Section 5.9.4). Shorter AOIs take less time to read out of the sensor.
- The number of packets needed to transfer a captured image from the eXcite's camera section to the MIPS processor. This can vary depending on the Bandwidth setting (see Section 5.4.5).
- The exposure time (see Sections 5.3.1 through 5.3.4) for a captured frame.

  If you use very long exposure times, you can capture fewer images per second.

To determine the maximum allowed frame capture rate with your current camera settings, calculate a result for each of the three formulas below. The formula that returns the lowest value will determine the maximum frame capture rate with the camera at its current settings. (In other words, the factor that restricts the rate the most will determine the maximum allowed frame capture rate.)

Formula 1 calculates the maximum frame capture rate based on the sensor readout time:

$$\text{Max. Frames/s} = \frac{1}{[\text{AOI Height} \times \text{E } \mu\text{s}] + \text{C } \mu\text{s}}$$

Where: AOI Height = the height of the area of interest.

C = 7029.11 for exA1390-19m

C = 7073.67 for exA1390-19c

C = 6557.8 for exA1600-14m

C = 6610.35 for exA1600-14c

E = 44.56 for exA1390-19m/c models

E = 52.55 for exA1600-14m/c models

Formula 2 calculates the maximum frame capture rate based on the number of packets needed to transfer a frame:

$$\text{Max. Frames/s} = \frac{1}{\text{Packets transferred per frame} \times 88\ \mu\text{s}}$$

You can determine the number of packets needed to transfer a frame by checking the Packet Number parameter (see Section 5.4.5).

Formula 3 calculates the maximum frame capture rate based on the current exposure time:

$$\text{Max. Frames/s} = \frac{1}{\text{Exposure time in µs} + 157.0\ \text{µs}}$$

Example of Calculating the Maximum Allowed Frame Capture Rate

Assume that your exA1390-19m camera is set for an area of interest of 200 columns wide and 210 rows high and that your exposure time is set for 2000 µs. Also assume that after making all parameter settings, you check the current value of the Packet Number parameter. You find that the packet number (packets transferred per frame) with the current settings is 11.

Formula 1:

$$\text{Max. Frames/s} = \frac{1}{[210 \times 44.56\ \text{µs}] + 7029.11\ \text{µs}}$$

Max. Frames/s = 61.0

Formula 2:

$$\text{Max. Frames/s} = \frac{1}{11 \times 88\ \text{µs}}$$

Max. Frames/s = 1033.0

Formula 3:

$$\text{Max. Frames/s} = \frac{1}{2000\ \text{µs} + 157\ \text{µs}}$$

Max. Frames/s = 463.6

Formula one returns the lowest value. So in this case, the limiting factor is the AOI size and the maximum allowed frame capture rate would be 61.0 frames per second.

5.4.4 What Does the Max Allowed Frame Capture Rate Mean to Me?

If you are starting image exposure by using a software trigger or an ExTrig signal (see Sections 5.3.2 and 5.3.3), or if you are starting image capture by using a series of one-shot commands (see Section 5.3.1), the maximum allowed frame capture rate represents the maximum number of exposures (frames) that you are allowed to start per second.

If you are running the camera in continuous shot mode and not using triggering (see Section 5.3.1), the camera will always capture frames (begin exposures) at the maximum allowed frame capture rate.

What if I Need a Higher or a Lower Maximum Allowed Frame Capture Rate?

Raising the Maximum Allowed Rate

You may find that you would like to capture frames at a rate higher than the maximum allowed with the camera's current settings. In this case, you must determine which of the factors (as determined by the formulas on page 5-23, 5-25, or 5-27) is restricting the frame rate the most and you must try to make that factor less restrictive:

- You will often find that the sensor readout time is most restrictive factor. Decreasing the height of the camera's area of interest (see Section 5.9.4) will decrease the sensor readout time and will make this factor less restrictive.

  This does not apply to exA640-60m/c models. On these models, changing the height of the AOI has no effect on the frame rate.
- If you are using normal exposure times and you are using an area of interest that includes the full sensor size, your exposure time will not normally be the most restrictive factor on the frame rate. However, if you are using long exposure times or small areas of interest, it is quite possible to find that your exposure time is the most restrictive factor on the frame rate. In this case, you should lower your exposure time. (You may need to compensate for a lower exposure time by using a brighter light source or increasing the opening of your lens aperature.)
- You should check the setting of the camera's Bandwidth parameter and make sure that it is set to the maximum. If the Bandwidth parameter is set to a low value, this will make the packets transferred per frame higher and it could restrict the frame capture rate. See Section 5.4.5 for more information on the Bandwidth parameter.

Lowering the Maximum Allowed Rate

Ordinarily, you would not want to lower the maximum allowed frame capture rate. However, if you are running the camera in continuous shot mode and not using triggering, the camera will always capture frames at the maximum allowed rate. In this case, you may want to lower the maximum allowed rate and thus lower the rate at which the camera is capturing frames. To lower the maximum allowed frame capture rate, you must make one of the rate restricting factors more restrictive:

- If the height of the area of interest is not already at the maximum, then increasing the AOI height will increase the sensor readout time and will make this factor more restrictive. In some cases, increasing the size of the AOI may not be the best option because it also increases the amount of image data that you must process.

  This does not apply to exA640-60m/c models. On these modes, changing the height of the AOI has no effect on the frame rate.
- Increasing the exposure time to a high value will restrict the camera's maximum frame rate. This is not usually a viable option for restricting the frame rate and should be avoided.

- Increasing the packets transferred per frame is the most viable way of restricting the maximum frame rate. You can increase the packets transferred per frame by decreasing the value of the camera’s Bandwidth parameter (see Section 5.4.5).

5.4.5 Using the Bandwidth Parameter

As mentioned in Section 5.2, after an eXcite captures an image, the image data is read out from the sensor into a buffer. Once the entire image has been read out to the buffer, the image data is packetized and transferred across a GPI bus to the MIPS processor in the eXcite.

A parameter called the Bandwidth determines the number of bytes of data that will be included in each packet transferred across the GPI bus. The minimum value for the Bandwidth parameter is 1 and the maximum value is 4096. Normally, the value of the Bandwidth parameter is set to the maximum and at maximum, the Bandwidth parameter has no noticeable effect on the operation of the eXcite.

If you lower the value of the Bandwidth parameter, the amount of image data included in each packet transferred across the GPI bus will be lower. This means that it will take more packets to transfer each image and since the cycle time of the GPI bus is fixed, it also means that it will take more time to transfer each image. When you lower the Bandwidth parameter enough, the slower data transfer rate can begin to affect the maximum allowed frame capture rate of your eXcite. If you look at Sections 5.4.1, 5.4.2, and 5.4.3 you will notice that one of the factors that can limit the maximum allowed frame capture rate is the number of packets needed to transfer a frame. The number of packets per frame is directly related to the Bandwidth setting.

You can see the effect of changing the Bandwidth parameter by looking at the read only parameter called Packet Number. The Packet Number parameter indicates the number of packets needed to transfer a frame with the current camera settings. If you decrease the setting for the Bandwidth parameter, you will notice that the value of the Packet Number parameter will increase.

If you are running the camera in continuous shot mode and not using triggering, the camera will always capture frames at the maximum allowed frame capture rate. A convenient way to lower the maximum allowed frame capture rate in this case is to lower the setting for the Bandwidth parameter. (You will usually find that you need to significantly lower the value of the Bandwidth parameter before the parameter will actually begin to restrict the frame rate.)

To set the value of the Bandwidth parameter, access the eXcite API and use the SetValue method for the Bandwidth object. To check the value of the Packet Number parameter access the eXcite API and use the GetValue method for the PacketNumber object. (For more information on using the API, refer to the API Reference documentation.)

5.5 Image Data Output Formats (on Monochrome Models)

Two image data output formats, Mono 8 and Mono 16, are available on the exA640-120m, exA640-60m, exA1390-19m, and the exA1600-14m models of the eXcite.

On the exA640-180m model, only Mono 8 is available.

Mono 8

When a monochrome eXcite is operating in the mono 8 output format, the camera transfers the data for each pixel in a captured image at 8 bit depth.

For detailed information on the format and range of mono 8 image data, see Sections 6.2.1 and 6.3.1.

Mono 16

When a monochrome eXcite is operating in the mono 16 output format, the camera transfers the data for each pixel in a captured image at 16 bit depth. However, only 10 of the bits are effective in the exA640-120m and exA640-60m models and only 12 of the bits are effective in the exA1390-19m and exA1600-14m models.

For detailed information on the format and range of mono 16 image data, see Sections 6.2.2 and 6.3.2.

> ⓘ When the camera is set for the mono 16 format, the camera outputs 16 bits per pixel. However, only 10 of the bits are effective in the exA640-120m and exA640-60m models and only 12 of the bits are effective in the exA1390-19m and exA1600-14m models. The effective pixel data fills from the LSB and the unused bits are filled with zeros. When the camera is set for 16 bit output, bytes are placed into any image buffers created in the MIPS processor in **big endian format**.

Setting the Image Data Output Format

The value of the Color Coding parameter determines the image data output format as shown in the table below.

Color Coding Parameter Setting	Image Data Output Format
ColorCoding_Mono8	Mono 8
ColorCoding_Mono16	Mono 16

To set the Color Coding parameter value, access the eXcite API and use the SetValue method for the ColorCoding object. (For more information on using the API, refer to the API Reference documentation. You can also refer to the source code for the “Simple Grab” sample program to see how the Color Coding object is used.)

5.6 Image Data Output Formats (on Color Models)

5.6.1 The “Bayer” Color Filter

The CMOS and CCD sensors used in color models of the eXcite are equipped with an additive color separation filter known as a Bayer filter. With the Bayer filter, each individual pixel is covered by a micro-lens that allows light of only one color to strike the pixel. As Figure 5-14 illustrates, within each block of four pixels, one pixel sees only red light (R), one sees only blue light (B), and two pixels see only green light (G). (This combination mimics the human eye’s sensitivity to color.)

The alignment of the Bayer filter used in the CMOS camera models of the eXcite is shown in Figure 5-14 where the first row starts with the sequence G, B, etc. Accordingly, the second row starts with the sequence R, G, etc. In the CCD camera models the first row starts with the sequence R, G, etc. Accordingly, the second row starts with the sequence G, B, etc.

Due to the repetitive nature of the Bayer filter this information is all you need to determine the order of the pixel colors with your current AOI settings. For more information on setting the AOI on a color eXcite see Section 5.9.4.

The image data output formats available on color cameras are related to the Bayer pattern. The output formats are explained in the next section.

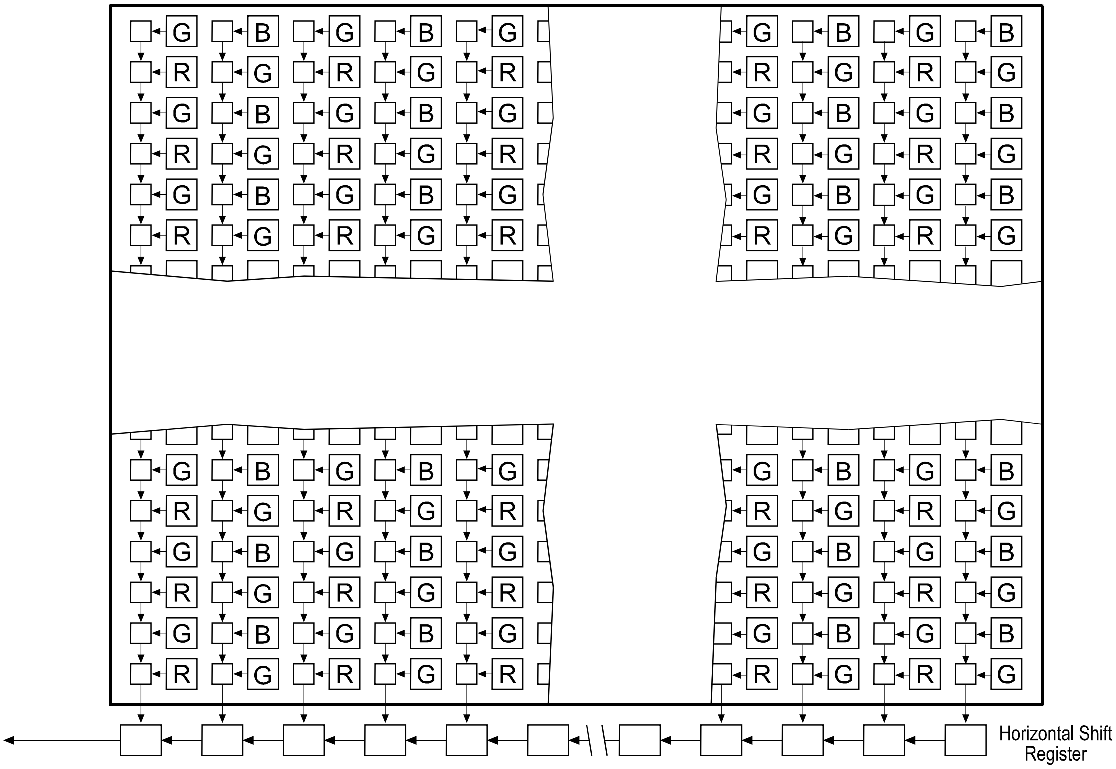


Figure 5-14: Bayer Filter Pattern on the eXcite’s Sensor

5.6.2 Output Formats

Five image data output formats, Raw 8, Raw 16, YUV 4:2:2, Mono 8, and Mono 16 are available on the exA640-120c, exA640-60c, exA1390-19c, and the exA1600-14c models of the eXcite.

On the exA640-180c model, only the Raw 8 and Mono 8 are available.

Raw 8

With raw 8 output, the data for each pixel in the captured image is output at 8 bit depth and the pixel data is not processed in any way. So for each pixel covered with a red lens, you get 8 bits of red data. For each pixel covered with a blue lens, you get 8 bits of blue data. And for each pixel covered with a green lens, you get 8 bits of green data. (This type of pixel data output is sometimes referred to as “Bayer 8”.)

For detailed information on the format and range of raw 8 image data, see Sections 6.2.3 and 6.3.1.

Raw 16

With raw 16 output, the data for each pixel in the captured image is output at 16 bit depth, however, with a reduced number of bits effective, and the pixel data is not processed in any way. Only 10 of the bits are effective in the exA640-120m and exA640-60m models and only 12 of the bits are effective in the exA1390-19m and exA1600-14m models. So for each pixel covered with a red lens, you get 16 bits of red data with the reduced number of bits effective. For each pixel covered with a blue lens, you get 16 bits of blue data with the reduced number of bits effective. And for each pixel covered with a green lens, you get 16 bits of green data with the reduced number of bits effective. (This type of pixel data output is sometimes referred to as “Bayer 16”.)

For detailed information on the format and range of raw 16 image data, see Sections 6.2.4 and 6.3.2.

ⓘ	When the camera is set for the raw 16 format, the camera outputs 16 bits per pixel. However, only 10 of the bits are effective in the exA640-120m and exA640-60m models and only 12 of the bits are effective in the exA1390-19m and exA1600-14m models. The effective pixel data fills from the LSB and the unused bits are filled with zeros. When the camera is set for 16 bit output, bytes are placed into any image buffers created in the MIPS processor in **big endian format**.

YUV 4:2:2

With YUV 4:2:2 output, each pixel in the captured image goes through a two step conversion process as it exits the sensor and passes through the camera’s electronics. This process yields full Y, U, and V color information for each pixel.

In the first step of the process, an interpolation algorithm is performed to get full RGB data for each pixel. (Because each individual pixel gathers information for only one color, an interpolation must be made from the surrounding pixels to get full RGB data for an individual pixel.)

The second step of the process is to convert the RGB information to the YUV color model. The conversion algorithm uses the following formulas:

$$Y = 0.30\,R + 0.59\,G + 0.11\,B$$

$$U = -\,0.17\,R - 0.33\,G + 0.50\,B$$

$$V = 0.50\,R - 0.41\,G - 0.09\,B$$

Once the conversion to a YUV color model is complete, pixels are transferred from the camera to the MIPS processor in the YUV 4:2:2 format.

For detailed information on the format and range of YUV 4:2:2 image data, see Sections 6.2.5 and 6.3.3.

The values for U and for V normally range from -128 to +127. Because the eXcite transfers U values and V values with unsigned integers, 128 is added to each U value and to each V value before the values are transferred from the camera. This process allows the values to be transferred on a scale that ranges from 0 to 255 (see Section 6.3.3).

Mono 8

When a color eXcite is operating in the Mono 8 output format, the pixel values in each captured image are first interpolated and converted to the YUV color model as described on the previous page. The camera then transfers the 8 bit Y value for each pixel to the MIPS processor. In the YUV color model, the Y component for each pixel represents a brightness value. This brightness value can be considered as equivalent to the value that be would be sent from a pixel in a monochrome camera. In essence, when a color eXcite camera is set for Mono 8, it outputs an 8 bit monochrome image. (This type of output is sometimes referred to as “Y Mono 8”.)

For detailed information on the format and range of mono 8 data, see Sections 6.2.1 and 6.3.1.

Mono 16

When a monochrome eXcite is operating in the mono 16 output format, the camera transfers the data for each pixel in a captured image at 16 bit depth. However, only 10 of the bits are effective in the exA640-120c and exA640-60c models and only 12 of the bits are effective in the exA1390-19c and exA1600-14c models.

For detailed information on the format and range of mono 16 image data, see Sections 6.2.2 and 6.3.2.

When the camera is set for the mono 16 format, the camera outputs 16 bits per pixel. However, only 10 of the bits are effective in the exA640-120c and exA640-60c models and only 12 of the bits are effective in the exA1390-19c and exA1600-14c models. The effective pixel data fills from the LSB and the unused bits are filled with zeros. When the camera is set for 16 bit output, bytes are placed into any image buffers created in the MIPS processor in **big endian format**.

Setting the Image Data Output Format

The value of the Color Coding parameter determines the image data output format as shown in the table below.

Color Coding Parameter Setting	Image Data Output Format
ColorCoding_Mono8	Mono 8
ColorCoding_Mono16	Mono 16
ColorCoding_YUV8_4_2_2	YUV 4:2:2
ColorCoding_Raw8	Raw 8
ColorCoding_Raw16	Raw 16

To set the Color Coding parameter value, access the eXcite API and use the SetValue method for the ColorCoding object. (For more information on using the API, refer to the API Reference documentation. You can also refer to the source code for the “Simple Grab” sample program to see how the Color Coding object is used.)

5.7 Video Modes

The eXcite currently has only one video mode available, i.e., video mode 0. The video mode feature has mainly been included on the eXcite as a platform for future use.

The value of the Video Mode parameter determines the Video Mode setting. To set the Video Mode parameter value, access the eXcite API and use the SetValue method for the VideoMode object. The parameter should always be set to VideoMode_VideoMode0.

For more information on using the API, refer to the API Reference documentation. You can also refer to the source code for the “Simple Grab” sample program to see how the VideoMode object is used.

5.8 Image Information Parameters

The eXcite includes a set of read only parameters that provide some basic information about the captured images that will be transferred from the camera section to the MIPS processor. Before checking these image information parameters, you should set the AOI (see Section 5.9.4), you should select an image data output format (see Sections 5.5 and 5.6).

The image information parameters include:

- A **Total Bytes** parameter that indicates the total number of bytes for each captured image.

  To read the Total Bytes parameter value, access the eXcite API and use the GetValue method for the TotalBytes object. (For more information on using the API, refer to the API Reference documentation.)

- A **Pixel Number** parameter that indicates the total number of pixels included in each captured image.

  To read the Pixel Number parameter value, access the API and use the GetValue method for the PixelNumber object.

- A **Data Depth** parameter that indicates the effective data depth of the pixels in the captured images. The effective data depth depends on which image data format is currently selected. The table below shows the effective data depth that will be indicated for each image data format.

  To read the Data Depth parameter value, access the API and use the GetValue method for the DataDepth object.

Image Data Output Format	Indicated Effective Data Depth	Note
Mono 8	8	8 bits are effective
Mono 16	10 or 12	16 bits are transferred but only 10 or 12 bits, depending on camera model, are effective (see Sections 5.5 and 5.6.2)
YUV 4:2:2	8	In the YUV 4:2:2 format, the information for each component is at 8 bit depth and the information is transferred at an average of 16 bits/pixel. See Section 6.2.5 for more information.
Raw 8	8	8 bits are effective
Raw 16	10 or 12	16 bits are transferred but only 10 or 12 bits, depending on camera model, are effective (see Section 5.6.2)

- The **Packet Number** parameter that indicates the number of packets that will be needed to transfer the captured image across the data bus between the camera section and the processor section of the eXcite.

  To read the Packet Number parameter value, access the eXcite API and use the GetValue method for the PacketNumber object.

- If you are running the eXcite in continuous shot mode without triggering (see Section 5.3.1) the **Frame Interval** parameter will indicate the time in seconds that it takes to capture each image. (The “frame interval” is also commonly referred to as the “frame period.”)

  To read the Frame Interval parameter value, access the eXcite API and use the GetValue method for the FrameInterval object.

5.9 Standard Features

5.9.1 Gain and Brightness

The gain and brightness features operate differently on CMOS cameras and CCD cameras. Section 5.9.1.1 describes gain and brightness on CMOS models of the eXcite. Section 5.9.1.2 describes gain and brightness on CCD models of the eXcite.

5.9.1.1 CMOS Camera Models

Gain Basics

On eXcite cameras, the output from the camera's sensor is digital and the gain and brightness functions are accomplished by manipulation of the sensor's digital output signal.

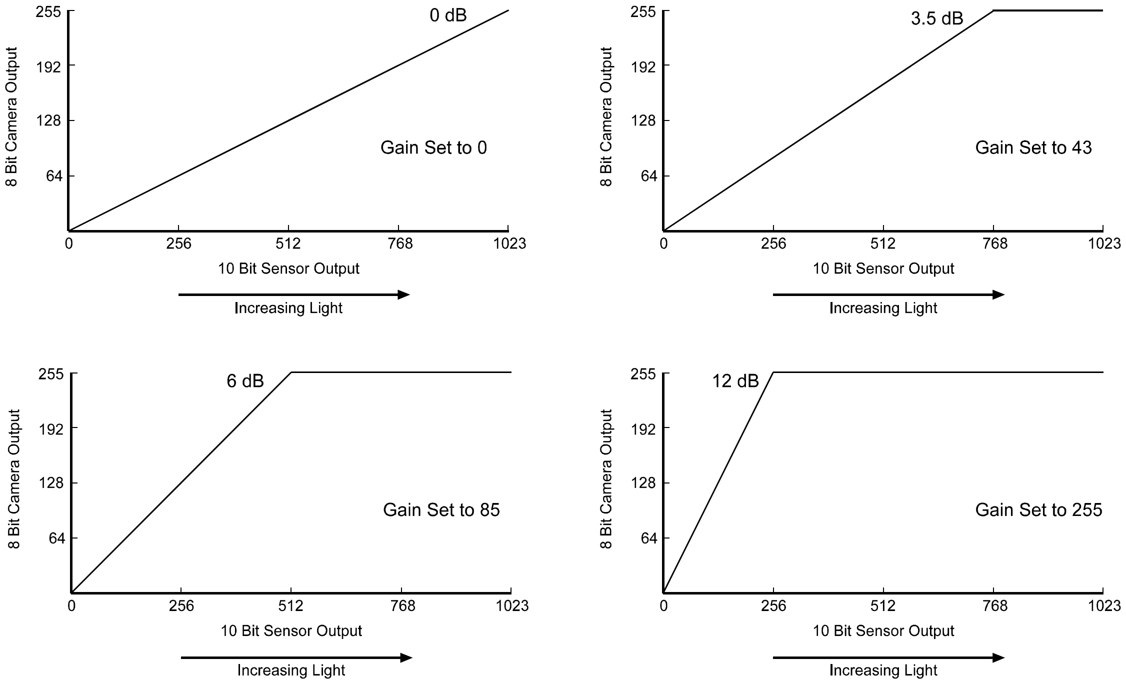


Figure 5-15: Mapping at Various Gain Settings

On eXcite CMOS cameras, As shown in the top left graph in Figure 5-15, when the gain is set to 0, the full 10 bit output range of the camera's CMOS sensor is mapped directly to the 8 bit output range of the camera. In this situation, a gray value of 0 is output from the camera when the pixels in the sensor are exposed to no light and a gray value of 255 is output when the pixels are exposed to very bright light. This condition is defined as 0 dB of system gain for the camera.

As shown in the three other graphs, increasing the gain setting to a value greater than 0 maps a smaller portion of the sensor's 10 bit range to the camera's 8 bit output. When a smaller portion of the sensor's range is mapped to the camera's output, the camera's response to a change in light level is increased.

This can be useful when at your brightest exposure, a gray value of less than 255 is achieved. For example, if gray values no higher than 127 were achieved with bright light, you could increase the gain setting so that the camera is operating at 6 dB (an amplification factor of 2) and see an increase in gray values to 254.

Brightness Basics

As shown in the top graph in Figure 5-16, setting the brightness higher than the default value of 725 moves the response curve to the left. This would increase the 8 bit value output from the camera for any given 10 bit value output from the sensor and thus increase the apparent brightness of the image.

As shown in the bottom graph, setting the brightness lower than the default value of 725 moves the response curve to the right. This would decrease the 8 bit value output from the camera for any given 10 bit value output from the sensor and thus decrease the apparent brightness of the image.

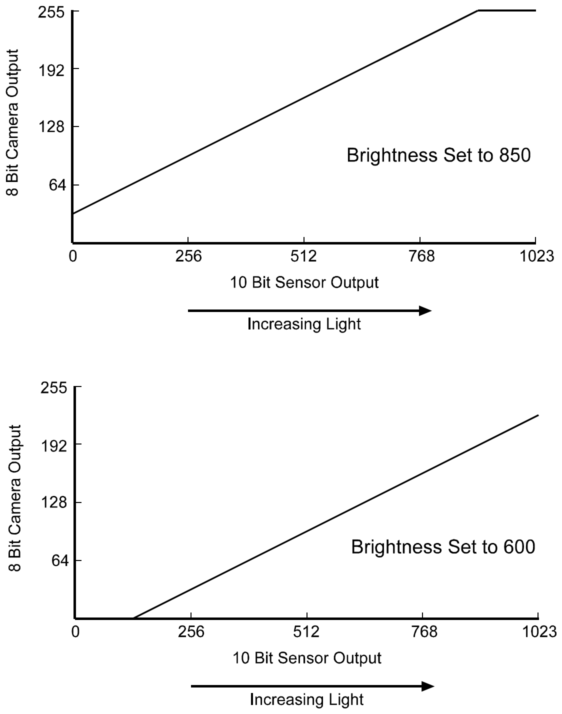


Figure 5-16: Brightness Setting Changes Mapping

Setting the Gain

The camera’s gain can be adjusted by changing the value of the Gain parameter. The Gain parameter value can range from 0 to 255 (decimal). To set the Gain parameter value, access the eXcite API and use the SetValue method for the Gain object. (For more information on using the API, refer to the API Reference documentation. You can also refer to the source code for the “SimpleScalar” sample program to see how “scalar” parameters such as the gain are set.)

Typical Gain parameter settings and the resulting amplifications are shown in Table 5-1.

Decimal	dB	Amplification Factor
0	0.0	x 1.0
28	2.5	x 1.3
43	3.5	x 1.5
85	6.0	x 2.0
128	8.0	x 2.5
170	9.5	x 3.0
213	10.9	x 3.5
255	12.0	x 4.0

Table 5-1: Gain Settings

ⓘ Because the sensor used in eXcite CMOS cameras has a direct digital output, the implementation of the gain settings on eXcite cameras is different from the implementation on eXcite CCD cameras and other Basler cameras. This means that you can not directly compare the response of an eXcite CMOS camera to another Basler camera that has the same gain setting. For example, if you compare the response of an eXcite with the gain set to 100 and an A301f with the gain set to 100, you will see a significant difference. This happens because the gain scales on the two cameras are implemented differently and are not directly comparable.

Setting the Brightness

The image brightness can be adjusted by changing the value of the Brightness parameter. The Brightness parameter value can range from 0 to 1023 (decimal). The default is typically 725, but may vary slightly from camera to camera. Settings below the default decrease the brightness and settings above the default increase the brightness.

To set the Brightness parameter value, access the eXcite API and use the SetValue method for the Brightness object. (For more information on using the API, refer to the API Reference documentation.)

The effect of a change in the brightness setting varies depending on the gain setting. With the gain set to 0, changing the brightness setting by 4 results in a change of 1 in the digital values output by the camera. With the gain set to 255, changing the brightness setting by 1 results in a change of 1 in the digital values output by the camera.

5.9.1.2 CCD Camera Models

Gain and Brightness Basics

The major components in the electronics of eXcite CCD cameras include: a CCD sensor, one VGC (Variable Gain Control), and one ADC (Analog to Digital Converter). The pixels in the CCD sensor output voltage signals when they are exposed to light. These voltages are amplified by the VGC and transferred to the ADC, which converts the voltages to digital output signals.

Two parameters, gain and brightness are associated with the VGC. As shown in Figure 5-17, and Figure 5-18, increasing or decreasing the gain increases or decreases the amplitude of the signal that is input to the ADC. Increasing or decreasing the brightness moves the signal up or down the measurement scale but does not change the signal amplitude.

For most applications, black should have a gray value of 1 and white should have a gray value of 255 (in modes that output 8 bits per pixel) or 4095 (in modes that output 12 effective bits per pixel).

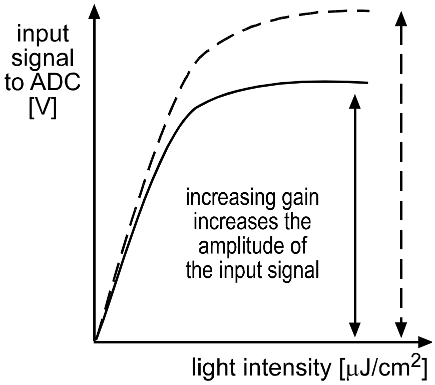


Figure 5-17: Gain

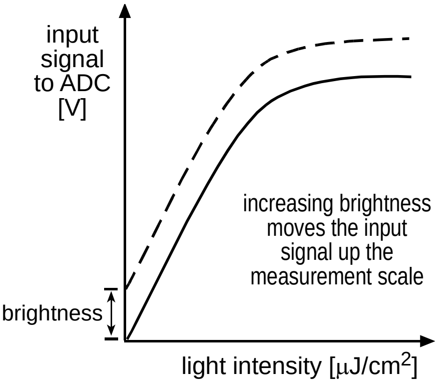


Figure 5-18: Brightness

Because increasing gain increases both signal and noise, the signal to noise ratio does not change significantly when gain is increased.

Setting the Gain

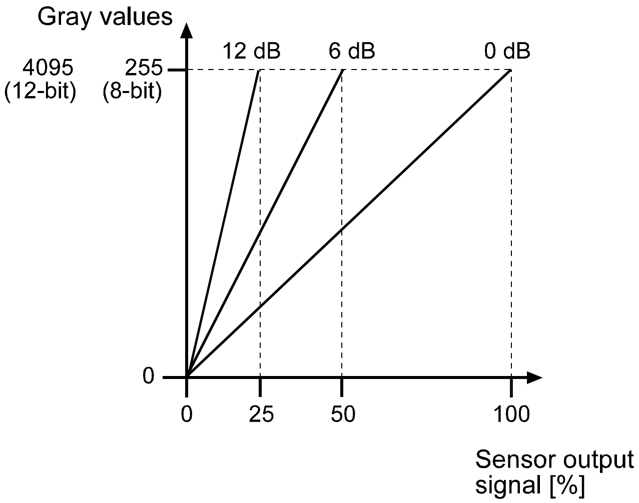


Figure 5-19: Gain Settings in dB

When the gain is set to 0 dB, the sensor's output range directly matches the input voltage range of the ADC. Thus, with a gain of 0 dB, a gray value of 1 is produced when the pixels are exposed to no light and a gray value of 255 (in modes that output 8 bits) or 4095 (in modes that output 12 effective bits) is produced when the pixels are exposed to bright light.

0 dB of gain is achieved when the gain setting is programmed to a decimal value of 350. Increasing the gain setting to more than 350 maps a smaller portion of the sensor's linear output range to the ADC's input.

Increasing the gain is useful when at your brightest exposure, a gray value lower than 255 (in modes that output 8 bits) or 4095 (in modes that output 12 effective bits) is reached. For example, if you found that in the brightest areas of your captured images your gray values were no higher than 127 (8 bit mode), you could increase the gain to 6 dB (amplification factor of 2) and thus reach gray values of 254 (see Figure 5-19).

If you know the decimal number (DN) setting for the gain on your camera, the equivalent decibel value can be calculated using one of the following equations:

When DN setting = 350 to 511:

$$\text{dB} = 20 \times \log_{10}\left(\frac{658 + \text{DN}}{658 - \text{DN}}\right) - 10.298$$

When DN setting = 512 to 1023:

$$\text{dB} = 0.0354 \times \text{DN} - 10.298$$

Please note that the relationship between the digital number settings and the amount of amplification is not linear.

With a camera set for mono 8, raw 8, or YUV 4:2:2 output, the entire gain range from 350 to 1023 can be used to set the value of the Gain parameter.

With a camera set for mono 16 or raw 16 output, only settings from 350 to 511 are valid. Settings above 511 should not be used with a camera set for mono 16 or raw 16 output.

The default is 370.

In normal operation, gain settings lower than 350 should not be used. When the gain is set lower than 350, the sensor output signal will not be properly mapped to the ADC input.

The camera’s gain can be adjusted by changing the value of the Gain parameter. To set the Gain parameter value, access the eXcite API and use the SetValue method for the Gain object. (For more information on using the API, refer to the API Reference documentation. You can also refer to the source code for the “SimpleScalar” sample program to see how “scalar” parameters such as the gain are set.)

Table 5-2 shows some typical Gain parameter settings and the amplification that will result:

Decimal Number (DN)	Hexadecimal	dB	Factor
350	0x15E	0	×1
484	0x1E4	6	×2
630	0x276	12	×4
800	0x320	18	×8
969	0x3C9	24	×16
1023	0x3FF	25.9	×20

Table 5-2: Examples of Gain Settings

Setting the Brightness

The image brightness can be adjusted by changing the value of the Brightness parameter. The Brightness parameter value can range from 0 to 255 (decimal). The default is typically 16, but may vary slightly from camera to camera. Settings below the default decrease the brightness and settings above the default increase the brightness.

With a camera set for mono 8, raw 8, or YUV 4:2:2 output, a brightness setting of 16 (decimal) will result in an offset of 1 in the digital values output for the pixels. An increase of 16 (decimal) in the brightness setting will result in a positive offset of 1 in the digital values output for the pixels. For example, a brightness setting of around 32 (16 + 16, decimal) would be required to reach a positive offset of 2. A brightness setting of around 48 (16 + 16 + 16, decimal) would be required to reach a positive offset of 3, and so on.

With a camera set for mono 16 or raw 16 output, a brightness setting of around 16 (decimal) will result in an offset of 1 in the digital values output for the pixels. Each increase of 1 (decimal) in the brightness setting will result in a positive offset of 1 in the digital values output for the pixels.

To set the Brightness parameter value, access the eXcite API and use the SetValue method for the Brightness object. (For more information on using the API, refer to the API Reference documentation.)

5.9.2 White Balance (on Color Models)

White balance capability has been implemented on eXcite color cameras. White balancing can be used to adjust the color balance of the images transferred from the camera when the camera is operating in YUV 4:2:2 output mode. With white balancing, correction factors are applied to the interpolated R, G, and B values that are used to calculate the Y, U, and V values for each pixel.

With the white balancing scheme used on the eXcite, blue and red are adjustable and green is not. Green has a fixed value of 64 (0x40) which corresponds to a correction factor of 1.0.

The White Balance UB parameter can be used to change the blue correction factor. The usable range of settings for this parameter is from 64 (0x40) to 255 (0xFF). If the parameter is set to 64 (0x40), blue will have the same 1.0 correction factor as green. If the parameter is set to a higher value, blue will have a higher correction factor and the image will be more blue. The default setting for the White Balance UB parameter is 112 (0x70).

To set the White Balance UB parameter value, access the eXcite API and use the SetValue method for the WhiteBalanceUB method. (For more information on using the API, refer to the API Reference documentation.)

To determine the correction factor for blue, use this formula:

$$\text{Blue Correction Factor} = \frac{\text{White Balance UB parameter setting}}{64}$$

The White Balance VR parameter can be used to change the red correction factor. The usable range of settings for the this parameter is from 64 (0x40) to 255 (0xFF). If the parameter is set to 64 (0x40), red will have the same 1.0 correction factor as green. If the parameter is set to a higher value, red will have a higher correction factor and the image will be more red. The default setting for the White Balance VR parameter is 64 (0x40).

To set the White Balance VR parameter, access the eXcite API and use the SetValue method for the WhiteBalanceVR method.

To determine the correction factor for red, use this formula:

$$\text{Red Correction Factor} = \frac{\text{White Balance VR parameter setting}}{64}$$

To make your images appear less green, raise the correction factor for both blue and red. To make your images appear more green, lower the correction factor for both blue and red.

ⓘ	The actual range of valid settings for the White Balance UB parameter is from 16 (0x10) to 255 (0xFF), however, only the settings from 64 (0x40) to 255 (0xFF) are useful. If you set the parameter value lower than 64 (0x40), the camera will continue to operate, but you will see unacceptable changes in the color balance. The actual range of valid settings for the White Balance VR parameter is from 16 (0x10) to 255 (0xFF), however, only the settings from 64 (0x40) to 255 (0xFF) are useful. If you set the parameter value lower than 64 (0x40), the camera will continue to operate, but you will see unacceptable changes in the color balance.

5.9.3 Integrated IR Cut Filter (on Color Models)

Color eXcite models are equipped with an IR cut filter as standard equipment. The filter is mounted in the lens adapter. Cameras without an IR cut filter are available on request.

Caution!

The location of the filter limits the thread length of the lens that can be used on an eXcite. The thread length on your lens must be less than 7.5 mm. If a lens with a longer thread length is used, the eXcite will be damaged and will no longer operate. See Section 1.4.3 for more details.

5.9.4 Area of Interest (AOI)

The area of interest (AOI) feature lets you specify a portion of the sensor array and after each image is captured, only the pixel information from the specified portion of the array is transferred to the MIPS processor.

The area of interest is referenced to the top left corner of the array. The top left corner is designated as column 0 and row 0 as shown in Figure 5-20.

The location and size of the area of interest is defined by declaring a left-most column, a width, a top row, and a height. For example, suppose that you specify the left column as 10, the width as 16, the top row as 4, and the height as 10. The area of the array that is bounded by these settings is shown in Figure 5-20.

The camera will only transfer pixel data from within the area defined by your settings. Information from the pixels outside of the area of interest is discarded.

One of the main advantages of the AOI feature is that decreasing the height of the AOI can increase the camera's maximum allowed frame rate (does not apply to exA640-60m/c models). See Section 5.4 for more information.

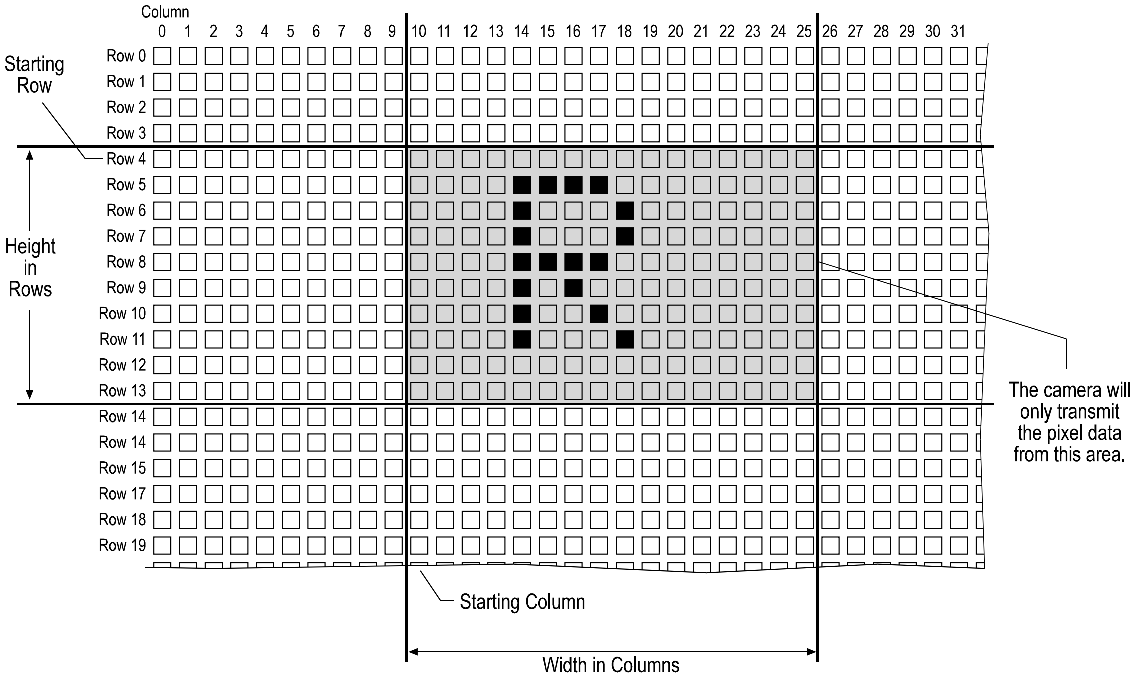


Figure 5-20: Area of Interest

By default, the AOI on the camera is set to use the full resolution of the sensor. You can change the size and the position of the AOI by changing the value of the camera's X Position, Y Position, Width, and Height parameters AOI parameters).

- The value of the X Position parameter determines the starting column for the area of interest. (The columns are numbered starting with 0.)
- The value of the Y Position parameter determines the starting row for the area of interest. (The rows are numbered starting with 0.)
- The value of the Width parameter determines the width of the area of interest.

- The value of the Height parameter determines the height of the area of interest.

For minimum and maximum values of the AOI parameters see Tables 5-3 and 5-4.

AOI Parameters	**exA640-60**				**exA640-120**				**exA640-180**			
	mono		**color**		**mono**		**color**		**mono**		**color**	
	min	**max**	**min**	**max**	**min**	**max**	**min**	**max**	**min**	**max**	**min**	**max**
X position	0	655	0	655	0	655	0	655	0	655	0	655
Y position	0	490	0	489	0	490	0	489	0	490	0	489
Width	1	656	1	656	1	656	1	656	1	656	1	656
Height	1	491	1	490	1	491	1	490	1	491	1	490

Table 5-3: Ranges of AOI Parameters: CMOS Camera Models

AOI Parameters	**exA1390-19**				**exA1600-14**			
	mono		**color**		**mono**		**color**	
	min	**max**	**min**	**max**	**min**	**max**	**min**	**max**
X position	0	1391	0	1387	0	1623	0	1623
Y position	0	1039	0	1037	0	1235	0	1233
Width	1	1392	1	1388	1	1624	1	1624
Height	1	1040	1	1038	1	1236	1	1234

Table 5-4: Ranges of AOI Parameters: CCD Camera Models

To set the X Position, Y Position, Width, and Height parameter values, access the eXcite API and use the SetValue method for the XPosition object, YPosition object, Width object, and Height object respectively. (For more information on using the API, refer to the API Reference documentation.)

> The sum of the X Position parameter value plus the Width parameter value must not exceed 656 on CMOS models, 1392 on the exA1390-19m, 1388 on the exA1390-19c, and 1624 on the exA1600-14m/c.
>
> The sum Y Position parameter value plus the Height parameter value must not exceed 491 on monochrome CMOS models, 490 on color CMOS models, 1040 on the exA1390-19m, 1038 on the exA1390-19c, 1236 on the exA1600-14m, and 1234 on the exA1600-14c.
>
> When you are setting the AOI on a color eXcite:
>
> - The setting for the *Width* must be divisible by 2.
> - The setting for the *Height* must be divisible by 2.
> - The setting for the *X Position* must be zero or be divisible by 2.
> - The setting for the *Y Position* must be zero or be divisible by 2.
>
> Decreasing the height of the AOI can increase the camera's maximum allowed frame rate. This does not apply to exA640-60m/c models. On these models, changing the height of the AOI has no effect on the frame rate. See Section 5.4 for more information.

For information on the order of pixel colors with the current AOI settings of a color eXcite see Section 5.6.1.

5.9.4.1 Changing AOI Parameters “On-the-Fly”

Making AOI parameter changes “on-the-fly” means making the parameter changes while the camera is capturing images continuously. On-the-fly changes are only allowed for the parameters that determine the position of the AOI, i.e., the parameters for X Position and Y Position. Changes to the AOI size are not allowed on-the-fly.

The camera's response to an on-the-fly change in the AOI position will vary depending on the way that you are operating the camera:

- If the exposure time is ≥ 100 µs, the changes will take effect on the next trigger after the changes are received by the camera.
- If the exposure time is < 100 µs and the camera is running with non-overlapped exposure (see Section 5.3.5) the changes will take effect on the next trigger after the changes are received by the camera.
- If the exposure time is < 100 µs and the camera is running with overlapped exposure, when the changes are received by the camera, the camera will delay the triggering of the next image until transmission of the current image is complete. When transmission of the current image is complete, the camera will change the AOI position, will trigger the next image, and will resume running in overlapped mode.

5.9.5 RS-232 Serial Port

The eXcite is equipped with one standard RS-232 serial port. This port provides access to serial port one on the eXcite processor.

For details of the physical and electrical characteristics of the RS-232 port, see Section 4.4.1. You will note that the wiring for handshaking (“clear to send” and “request to send”) is present on the port. Although the wiring is present, handshaking has not yet been implemented.

The serial port on the processor is available for use by programs that you design to run on the eXcite. However, by default the serial port has been assigned to transmit console output from the processor and to accept input from a keyboard. Before you can access the serial port from your programs, you must reconfigure the eXcite as described below.

Configuring an eXcite to Make the Serial Port Available

Caution!

You must follow the procedure below exactly as written. Failure to follow the procedure could cause the camera to become inoperable and it can be very difficult to recover. Please call Basler technical support if you have questions or need assistance.

If you decide to return the serial port to its default condition, i.e., console output is directed to the serial port and the port accepts keyboard input, you must have a working Ethernet connection to the eXcite. **If you do not have a working Ethernet connection to the eXcite available, you can not return the serial port to the default**.

This procedure assumes that you will use the nano editor commonly used on Linux systems. A brief guide to using the nano editor appears on page 2-9.

To make the serial port available for your use:

1. Install a Y connector between the eXcite and its power supply as shown in Figure 2-1.
2. Use a null modem cable to connect the 9-pin connector on the Y cable to a serial port on your development PC.
3. Start a terminal emulation program such as Minicom (Linux) or Hyperterminal (Windows OS) on your development PC. The emulator should have the following settings:

    Bps = 57600

    Data bits = 8

    Parity = none

    Stop bits = 1

    Flow control = none

    Emulation = VT100

    Com Port = the serial port on your PC you will use to connect with the eXcite

4. Remove power from the eXcite, wait several seconds, and then reapply power.

5. Check the emulator screen. You will see the eXcite begin its bootup process.
6. When you see:

   ```
   Executing boot script in 5 seconds - enter ^C to abort
   ```

   Press Ctrl +C to break the boot process.

In steps 7 through 15, you will be modifying the boot loader script so that console output from the processor will no longer be directed to the serial port.

7. Type in:

   ```
   fconfig
   ```

   And then press the Enter key.
8. When you see:

   ```
   Run script at boot: true
   ```

   Press the Enter key.
9. Type in: `fi lo linux`

   And then press the Enter key.
10. Type in:

    ```
    exec -c "console=ttyS1,57600n8 root=/dev/mtdblock0 rootfstype=jffs2 rw"
    ```

    And then press the Enter key.
11. Press the Enter key to enter an empty line.

We most strongly recommend that you follow step 12 as written. (If you set the script timeout value to 0, the boot loader will not execute properly and the Linux OS in the eXcite will not start.)

12. When you see:

    ```
    Boot script timeout (1000 ms resolution): 5
    ```

    Press the Enter key.
13. When you see:

    ```
    Console baud rate: 57600
    ```

    Press the Enter key.
14. When you see:

    ```
    Update RedBoot non-volatile configuration - continue (y/n)?
    ```

    Type in:

    ```
    y
    ```

    And then press the Enter key.
15. Wait several seconds for the boot configuration script to be saved.
16. Remove power from the eXcite, wait several seconds, and then reapply power. This will cause the eXcite to reboot. You can monitor the reboot process on your emulator screen.
17. At the login prompt on your emulator screen, type in:

    ```
    root
    ```

    And press the Enter key.
18. At the password prompt, type in:

    ```
    root
    ```

    And Press the Enter key.

In the next steps you will be using the nano editor to modify the “inittab” file. This file contains a single line that begins with “TTYS0:: respawn”. Your goal is to use the editor to comment out the line by placing a hash mark (#) at the beginning of the line. This will modify the file so that input from the keyboard is no longer directed to the serial port.

19. Type in:

    ```
    nano -w /etc/inittab
    ```

    And press the Enter key. This action will start the nano editor and open the inittab file.
20. The inittab file contains just one line. You should use the editor to comment out this line by placing a hash mark (#) at the beginning of the line.
21. To quit the nano editor:
    a) press the Control+X keys
    b) when you are asked to save the modified buffer, execute y ↵.
22. Remove power from the eXcite, wait several seconds, and then reapply power.

    If you watch the terminal emulator screen, you will notice that you only see the initial boot process of the eXcite and not the console output. You will also note that the port will no longer accept input from the keyboard.

    The port is now free for your use.

Returning the Serial Port to Default Configuration

Caution!

You must follow the procedure below exactly as written. Failure to follow the procedure could cause the camera to become inoperable and it can be very difficult to recover. Please call Basler technical support if you have questions or need assistance.

To complete this procedure, you must have a working Ethernet connection to the eXcite.

This procedure assumes that you will use the nano editor commonly used on Linux systems. A brief guide to using the nano editor appears on page 2-9.

To complete the procedure you will need to know the IP address of the eXcite. If you don’t know the eXcite’s IP address, use the methods described in Section 2.1.1.5 to find it.

To return the serial port to default configuration:

1. Install a Y connector between the eXcite and its power supply as shown in Figure 2-1.
2. Use a null modem cable to connect the 9-pin connector on the Y cable to a serial port on your development PC.
3. Start a terminal emulation program such as Minicom (Linux) or Hyperterminal (Windows OS) on your development PC. The emulator should have the following settings:

    Bps = 57600

    Data bits = 8

    Parity = none

    Stop bits = 1

    Flow control = none

    Emulation = VT100

    Com Port = the serial port on your PC you will use to connect with the eXcite

4. Remove power from the eXcite, wait several seconds, and then reapply power.

   Wait about 30 seconds to allow the eXcite to complete its boot process.

5. Open a command shell (Linux) or a command prompt window (Windows).
6. From the command line, type in:

   ```
   telnet [eXcite IP address]
   ```

   And press the Enter key.
7. At the login prompt, type in:

   ```
   root
   ```

   And press the Enter key.
8. A the Password prompt, type in:

   ```
   root
   ```

   And Press the Enter key.

In the next steps you will be using the nano editor to modify the “inittab” file. This file contains a single line that begins with “TTYS0:: respawn”. Your goal is to use the editor to uncomment the line by removing the hash mark (#) at the beginning of the line. This will modify the file so that input from the keyboard will be directed to the serial port.

9. Type in:

   ```
   nano -w /etc/inittab
   ```

   And press the Enter key. This action will start the nano editor and open the inittab file.
10. The inittab file contains just one line. You should use the editor to uncomment this line by removing the hash mark (#) at the beginning of the line.
11. To quit the nano editor:

    a) press the Control+X keys

    b) when you are asked to save the modified buffer, execute y ↵.
12. Type in:

    ```
    logout
    ```

    And press the Enter key.
13. Close the terminal window or the command prompt window.
14. Remove power from the eXcite, wait several seconds, and then reapply power.
15. Check the emulator screen. You will see the eXcite begin its bootup process.
16. When you see:

    ```
    Executing boot script in 5 seconds - enter ^C to abort
    ```

    Press Ctrl +C to break the boot process.

In steps 17 through 25, you will be modifying the boot loader script so that console output from the processor will be directed to the serial port.

17. Type in:

    ```
    fconfig
    ```

    And then press the Enter key.
18. When you see:

    ```
    Run script at boot: true
    ```

    Press the Enter key.
19. Type in: `fi lo linux`

    And then press the Enter key.

20. Type in:

    ```
    exec -c "console=ttyS0,57600n8 root=/dev/mtdblock0 rootfstype=jffs2 rw"
    ```

    And then press the Enter key.
21. Press the Enter key to enter an empty line.

We most strongly recommend that you follow step 22 as written. (If you set the script timeout value to 0, the boot loader will not execute properly and the Linux OS in the eXcite will not start.)

22. When you see:

    ```
    Boot script timeout (1000ms resolution): 5
    ```

    Press the Enter key.
23. When you see:

    ```
    Console baud rate: 57600
    ```

    Press the Enter key.
24. When you see:

    ```
    Update RedBoot non-volatile configuration - continue (y/n)?
    ```

    Type in:

    y

    And then press the Enter key.
25. Wait several seconds for the boot configuration script to be saved.
26. Remove power from the eXcite, wait several seconds, and then reapply power. This will cause the eXcite to reboot. You can monitor the reboot process on your emulator screen

    The port is now returned to its default configuration. Console output from the processor will be directed to the port and the port will accept input from your keyboard.

5.9.6 USB Ports

The eXcite's processor is equipped with two USB 2.0 ports. Each port is accessed via a standard USB Type A connector on the back of the eXcite. The Linux kernel in the eXcite processor is equipped with a driver to provide basic USB port functionality for mass storage devices. When any other USB device is used with the eXcite, you will typically be required to install a Linux driver for the device.

ⓘ	Bus-powered USB devices that draw more than 100 mA must not be used with the eXcite.

5.9.7 Ethernet Port

The eXcite's processor is equipped with one standard 10/100/1000 Ethernet port. The most common use of the Ethernet port is to transmit data from the eXcite to a PC or other device in your Ethernet network.

To transmit data from the eXcite to an external device, you must initiate a client application on the eXcite and a server application on the device. The overview section in the API Reference documentation contains some basic information on establishing and using a client/server. The streaming client and streaming server sample code in Section 3.5.10 provides an example of how to use a client/server to transmit data from the eXcite to an external device.

When you are transmitting data from your eXcite, keep in mind that you can transmit a maximum of 100 MBits per second of image data or other data on a 100 MBits second network. On a 1000 MBits per second network, you can transmit up to 1000 MBits per second.

5.9.8 Input Ports

The eXcite is equipped with four physical input ports designated as Input Port 0, Input Port 1, Input Port 2, and Input Port 3. For a detailed description of the input port physical and electrical characteristics, see Sections 4.1.2, 4.1.3, and 4.4.4.

Assigning an Input Port to a Standard Camera Input Signal

Currently, there is only one standard input signal for the camera - the External Trigger (ExTrig) input signal. The ExTrig input signal can be used to control exposure as described in Section 5.3.3.

You can use the Trigger Source parameter to assign one of the camera's input ports to receive an ExTrig input signal. To set the Trigger Source parameter value, access the API and use the SetValue method for the TriggerSource object. (For more information on using the API, refer to the API Reference documentation.)

The value of the Trigger Source parameter determines the port assigned to receive the ExTrig signal as shown in the table below.

Trigger Source Parameter Setting	Port Assigned to ExSync
TriggerSource_ExTrigPort0	Input Port 0
TriggerSource_ExTrigPort1	Input Port 1
TriggerSource_ExTrigPort2	Input Port 2
TriggerSource_ExTrigPort3	Input Port 3

By default, physical input port 0 is assigned to receive the ExTrig signal. You can assign only one port to receive the ExTrig input signal.

Using Input Ports to Receive User Defined Signals

You can use the eXcite's input ports to receive your own, user-designed input signals. The electrical characteristics of your input signals must meet the requirements shown in Section 4.4.4.1.

You can use the parallel input port read feature (see page 5-56), to monitor the state of the input ports and you can design the software that you run on the Excite's processor to react to the state of the inputs.

The port assigned to receive the ExTrig input signal can't be used to receive user-designed input signals.

Parallel Input Port Read

You can determine the state of the four physical input ports on the eXcite by reading the current value for the PIO Input parameter. The PIO Input parameter is a 32 bit value, but only the four LSBs of the value are used. Each of the four LSBs is assigned to an input port as shown in Figure 5-21. When you read the parameter value, the four LSBs of the returned value indicate the current state of the inputs. If a bit is 1, the camera input port associated with that bit is currently high. If a bit is 0, the camera input port associated with that bit is currently low.

To read the PIO Input parameter value, access the eXcite API and use the GetValue method for the PioInput object. (For more information on using the API, refer to the API Reference documentation.)

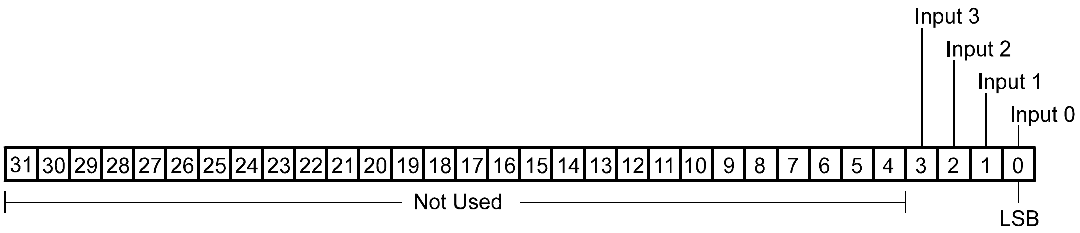


Figure 5-21: PIO Input Parameter Bit to Port Assignments

For some examples of how to work with the input ports, refer to the source code for the “SimpleDio” sample program in Section 3.5.6.

5.9.9 Output Ports

The eXcite is equipped with four physical output ports designated as Output Port 0, Output Port 1, Output Port 2, and Output Port 3. For a detailed description of the output port physical and electrical characteristics, see Sections 4.1.2, 4.1.3, and 4.4.4.

5.9.9.1 Individual Output Port Control

The individual output port control feature lets you individually control the assignment of a standard camera output signal (such as Integrate Enabled and Trigger Ready) to the physical output ports. It also lets you assign ports as having a “user settable” output signal and to set the state of any port so assigned.

Each physical output port can be unassigned or it can have one and only one camera output signal assigned to it.

You can assign a camera generated signal such as Integrate Enabled or Trigger Ready to more than one physical output port. For example, the Trigger Ready signal could be assigned to both physical Output Port 0 and physical Output Port 1.

If you designate an output port as having a “user settable” output signal, you can individually set the state of that port as you desire.

You can enable an invert function on any output port. If the invert function is enabled, the assigned signal will be inverted before it is applied to the output port.

Configuring an Output Port

Setting the Source Signal for a Port

Setting the value of the Output 0 Port Source Select parameter will assign a signal to port 0 as shown below:

Parameter Value	Result
PioOut0Src_IntegrationEnabled	Integrate enabled signal (see Section 5.3.7) is assigned to port 0
PioOut0Src_ReadyforTrigger	Trigger Ready signal (see Section 5.3.6) is assigned to port 0
PioOut0Src_UserSet	Port 0 is assigned as user settable

To set the Output Port 0 Source Select parameter value, access the eXcite API and use the SetValue method for the PioOut0Src object. (For more information on using the API, refer to the API Reference documentation.)

The source signal for output port 1, 2, and 3 is set in a similar fashion to port 0.

The PioOut1Src object is used to set the Output Port 1 Source Select parameter value.

The PioOut2Src object is used to set the Output Port 2 Source Select parameter value.

The PioOut3Src object is used to set the Output Port 3 Source Select parameter value.

ⓘ	By default, the Integrate Enabled signal is assigned to physical Output Port 0 and the Trigger Ready Signal is assigned to physical Output Port 1.

Setting a Port For Invert

Setting the value of an Output Port Invert parameter will set a port to invert the output signal as shown below.

Parameter Value	Result
false	Do not invert
true	Invert

To set the Output Port 0 Invert parameter value, access the eXcite API and use the SetValue method for the PioOut0Invert object. (For more information on using the API, refer to the API Reference documentation.)

The PioOut1Invert object is used to set the Output Port 1 Invert parameter value.

The PioOut2Invert object is used to set the Output Port 2 Invert parameter value.

The PioOut3Invert object is used to set the Output Port 3 Invert parameter value.

Setting the State of a “User Settable” Port

If you have designated the source signal for an output port as “user settable”, then you can change the state of that port at will. Assuming that you have designated Output Port 0 as user settable, you can use the Output Port 0 Setting parameter to set the state of Output Port 0. Setting the value of the parameter will set the state of a user settable port as shown below.

Parameter Value	Result
PioOut0Setting_Low	Low (non-conducting)
PioOut0Setting_High	High (conducting)

To set the Output Port 0 Setting parameter value, access the eXcite API and use the SetValue method for the PioOut0Setting object. (For more information on using the API, refer to the API Reference documentation.)

The PioOut1Setting object is used to set the Output Port 1 Setting parameter value.

The PioOut2Setting object is used to set the Output Port 2 Setting parameter value.

The PioOut3Setting object is used to set the Output Port 3 Setting parameter value.

If you have the invert function enabled on a user settable port, the user setting sets the state before the inverter.

Checking the Current State of a Port

Reading the value of the Output Port 0 Monitor parameter will indicate the current state of Output Port 0 as shown below.

Parameter Value	Result
PioOut0Monitor_Low	Low (non-conducting)
PioOut0Monitor_High	High (conducting)

To read the Output Port 0 Monitor parameter value, access the eXcite API and use the GetValue method for the PioOut0Monitor object. (For more information on using the API, refer to the API Reference documentation.

The PioOut1Monitor object is used to read the Output Port 1 Monitor parameter value.

The PioOut2Monitor object is used to read the Output Port 2 Monitor parameter value.

The PioOut3Monitor object is used to read the Output Port 3 Monitor parameter value.

> For some examples of how to work with the output ports, refer to the source code for the “SimpleDio” sample program in Section 3.5.6.

5.9.9.2 Parallel Output Port Control

A feature for controlling the state of the output ports in parallel is available on eXcite cameras. This feature lets you use a single parameter to set the state of the four physical output ports on the camera.

Setting the value of the PIO Output parameter will set the state of the four physical output ports on the eXcite. The PIO Output parameter is a 32 bit value, but only the four LSBs of the value are used. Each of the four LSBs is assigned to an output port as shown in Figure 5-22. If you set the value of the PIO Output parameter so that a used bit is equal to 1, the camera output port associated with that bit will be set to high. If you set the value of the PIO Output parameter so that a used bit is equal to 0, the camera output port associated with that bit will be set to low.

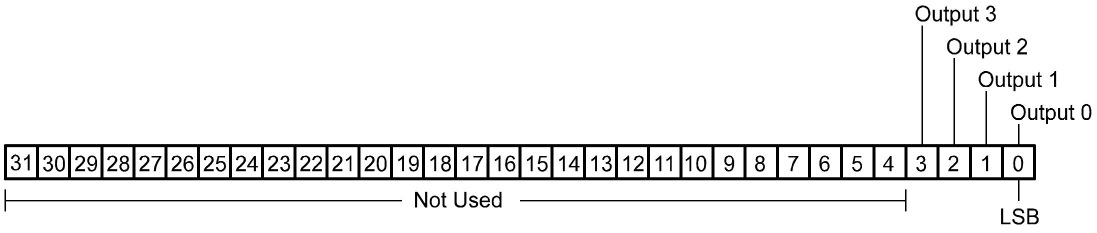


Figure 5-22: PIO Output Parameter Bit to Port Assignments

To set the PIO Output parameter value, access the eXcite API and use the SetValue method for the PioOutput object. To read the PIO Output parameter value, access the API and use the GetValue method for the PioOutput object. (For more information on using the API, refer to the API Reference documentation.)

> Setting the PIO Output parameter value will only set the state of physical output ports that are configured as “User settable.” For any output ports not configured as user settable, the parameter value will be ignored. See Section 5.9.9.1 for information on configuring physical output ports.

You can also use the parallel output port control feature to read the current state of the four physical output ports. When you read the PIO Output parameter value, the four LSBs of the returned value indicate the current state of the outputs. Each of the four LSBs is assigned to an output port as shown in Figure 5-22. If a bit is 1, the camera output port associated with that bit is currently high. If a bit is 0, the camera output port associated with that bit is currently low.

ⓘ For some examples of how to work with the output ports, refer to the source code for the “SimpleDio” sample program in Section 3.5.6.

5.9.10 I/O Port Response Times

Reading the state of an input port or an output port typically requires 250 µs. This means that you can poll the ports up to four times per millisecond.

Writing to an output port (i.e., setting the state of an output port) typically requires 200 µs.

If you will be changing the state of an output port based on a change in state of an input port, the typical cycle time from input port read to the completion of output port write is less than 1 millisecond.

5.9.11 The Watchdog Timer

The Linux kernel on the eXcite includes a watchdog timer function. The watchdog timer is available for use by application programs you design to run on the eXcite. The timer includes calls that let you set the timeout value, start the timer, reset the timer, and stop the timer.

If you are making use of the timer within one of your applications running on the eXcite and the timer times out, the eXcite will automatically reboot.

The SimpleWatchdog sample code in Section 3.5.7 illustrates how to use the watchdog time with your application programs.

5.9.12 Configuration Sets and Memory Channels

A configuration set is a group of values that contains all of the parameter settings needed to control the eXcite. There are two basic types of configuration sets: the work configuration set and the factory configuration set.

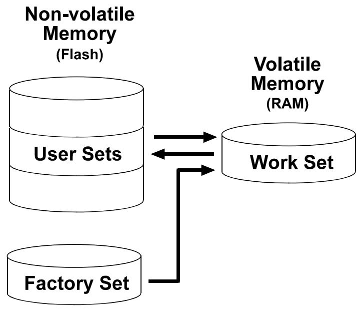


Figure 5-23: Configuration Sets

Work Configuration Set

The work configuration set contains the eXcite's current parameter settings and thus determines the real-time performance of the eXcite. When you change a parameter value using one of the SetValue methods in the eXcite API, you are making changes to the work configuration set. The work configuration set is located in the camera's volatile memory and the settings are lost if the camera is reset or if power is switched off. The work configuration set is usually just called the "work set" for short.

Factory Configuration Set

When an eXcite is manufactured, a test setup is performed on the camera and an optimized configuration is determined. The factory configuration set contains the eXcite's factory optimized configuration. The factory set is saved in a permanent file in the eXcite's non-volatile memory. The factory set can not be altered and since it is stored in non-volatile memory, it is not lost when the eXcite is reset or switched off. The factory configuration set is usually just called the "factory set" for short.

Saving Configuration Sets

As mentioned above, the work configuration set is stored in volatile memory and the settings are lost if the eXcite is reset or if power is switched off. The eXcite can save the current work set values in the volatile memory to reserved areas in the non-volatile memory called "memory channels." Configuration sets saved to memory channels in the non-volatile memory are not lost at reset or power off. There are three memory channels available for saving configuration sets: channel 1, channel 2 and channel 3. A configuration set saved in a memory channel is commonly referred to as a "user configuration set" or "user set."

Saving the current work set to one of the memory channels is a three step process:

1. Make changes to the eXcite's parameters until the eXcite is operating in a manner that you would like to save.
2. Set the value of the Memory Save Channel parameter to 1, 2, or 3. This will select the channel were the configuration set will be saved.

   To set the Memory Save Channel parameter value, access the eXcite API and use the SetValue method for the SaveMemoryCh object. (For more information on using the API, refer to the API Reference documentation.)
3. Set the value of the Memory Save parameter to true. When you set this value to true, the camera will save the current register settings to the designated memory channel.

   To set the Memory Save parameter, access the API and use the SetValue method for the SaveMemory object.

Saving a configuration set to one of the memory channels will overwrite any set that was previously saved to the selected channel.

Designating a Startup Memory Channel

Whenever an eXcite is powered on or is reset, by default, it copies the factory set settings in memory channel 0 into the work set. The eXcite has a “startup channel” feature that lets you change this behavior. The startup channel feature designates which memory channel will be used at power on or reset. For example, if the startup channel is designated as memory channel 2, the settings in memory channel 2 will be copied into the work set at power on or reset.

The value of the Startup Memory Channel parameter determines which memory channel will be copied into the work set at startup. The parameter can be set to 0 for the factory set or to 1, 2, or 3 to select memory channel 1, memory channel 2, or memory channel 3 respectively.

To set the Startup Memory Channel parameter, access the eXcite API and use the SetValue method for the StartupMemoryCh object. (For more information on using the API, refer to the API Reference documentation.)

5.9.13 Error Monitoring and Error Indicators

5.9.13.1 Temperature Sensors

The eXcite contains two temperature sensors. One sensor monitors the temperature of the MIPS processor (the CPU) and the other monitors the temperature of the eXcite’s processing board. The CPU Temperature parameter indicates the temperature of the processor in degrees C. The Board Temperature parameter indicates the temperature of the processing board in degrees C.

To read the CPU Temperature parameter value, access the eXcite API and use the GetValue method for the CpuTemperature object. To read the Board Temperature parameter value, access the API and use the GetValue method for the BoardTemperature object. (For more information on using the API, refer to the API Reference documentation.)

> The eXcite constantly monitors the value of the two internal temperature sensors and if the temperature is too high, the eXcite will enter an overtemperature error condition. See Section 5.9.13.2 for more information.

5.9.13.2 Overtemperature Condition

As mentioned in Section 5.9.13.1, the eXcite contains sensors that monitor the internal temperature of the unit. If the internal temperature of an eXcite reaches 70º C, the unit will enter an overtemperature condition. In an overtemperature condition, all internal power generation is switched off with the exception of the temperature monitoring circuits. To recover from an overtemperature error, you must remove input power from the eXcite, you must allow the eXcite to cool, and then you must reapply input power.

> If an eXcite enters an overtemperature condition, its power indicator LED (see Figure 4-1) will switch off.

5.9.13.3 LEDs

The eXcite has two LED indicators that display the state of the internal voltages generated by the camera and the state of the network connection. Figure 4-1 shows the location of the LEDs and Section 4.2 explains how the LEDs are used.

5.9.13.4 Undervoltage Lockout Condition

If the input power to an eXcite drops below approximately 10.0 VDC, the unit will enter an undervoltage lockout condition. In a lockout condition, the eXcite will stop operating and the power indicator LED (see Figure 4.1) will be unlit. When the input voltage is returned to approximately 11.5 VDC, the eXcite will automatically restart, i.e., the eXcite will go through the same start up process as it would if power was switched off and back on.

5.9.13.5 Error Flags

The eXcite monitors a variety of camera parameters and will set an error flag if the parameters are set incorrectly. The following error flags are available on the eXcite:

Error Flag 1

Error Flag 1 monitors the X Position, Width, Y Position, Height, and Color Coding parameters. If any of these parameters is set out of range or if the parameters are set in conflict, Error Flag 1 will be set to 1. If the settings for all of these parameters are OK, the flag will be set to 0. (See Sections 5.9.4, 5.5, and 5.6 for more information on these parameters.)

To read the current value of Error Flag 1, access the eXcite API and use the GetValue method for the ErrorFlag1 object. (For more information on using the API, refer to the API Reference documentation.)

Error Flag 2

Error Flag 2 monitors the Bandwidth parameter (see Section 5.4). If the Bandwidth parameter is in range, Error Flag 2 will be set to 0. If the Bandwidth parameter is out of range, Error Flag 2 will be set to 1.

To read the current value of Error Flag 2, access the eXcite API and use the GetValue method for the ErrorFlag2 object.

Shutter Error

The Shutter Error Flag monitors the Shutter parameter (see Section 5.3.4). If the Shutter parameter is in range, the Shutter Error Flag will be set to false. If the Shutter parameter is out of range, the Shutter Error Flag will be set to true.

To read the current value of the Shutter Error Flag, access the eXcite API and use the GetValue method for the ErrorShutter object.

Gain Error

The Gain Error Flag monitors the Gain parameter (see Section 5.9.1). If the Gain parameter is in range, the Gain Error Flag will be set to false. If the Gain parameter is out of range, the Gain Error Flag will be set to true.

To read the current value of the Gain Error Flag, access the eXcite API and use the GetValue method for the ErrorGain object.

Brightness Error

The Brightness Error Flag monitors the Brightness parameter (see Section). If the Brightness parameter is in range, the Brightness Error Flag will be set to false. If the Brightness parameter is out of range, the Brightness Error Flag will be set to true.

To read the current value of the Brightness Error Flag, access the eXcite API and use the GetValue method for the ErrorBrightness object.

Trigger Error

The Trigger Error Flag monitors the Trigger Polarity, Trigger Source, and Trigger Mode parameters. If the settings for all of these parameters are OK, the flag will be set to false. If any of these parameters is out of range or if the parameters are in conflict, the Trigger Error Flag will be set to true. (See Sections 5.3.2 and 5.3.3 for more information on these parameters.)

To read the current value of Trigger Error Flag, access the eXcite API and use the GetValue method for the ErrorTrigger object. (For more information on using the API, refer to the API Reference documentation.)

White Balance Error

The White Balance Error Flag monitors the White Balance UB and the White Balance VR parameters (see Section 5.9.2). If both of these parameters are in range, the White Balance Error Flag will be set to false. If either or both of these parameters is out of range, the White Balance Error Flag will be set to true.

To read the current value of the White Balance Error Flag, access the eXcite API and use the GetValue method for the ErrorWhiteBalance object.

5.10Smart Features

5.10.1 What are Smart Features

Smart features are features unique to Basler cameras. Test Images or the trigger counter and trigger flag feature are examples of Basler smart features.

Enabling a smart feature will simply change the behavior of the camera. The Test Image feature is a good example of this type of smart feature. When the Test Image feature is enabled, the camera outputs a test image rather than a captured image. This type of smart feature is referred to as a “non-reporting” smart feature.

5.10.2 Non-reporting Smart Features on the eXcite

5.10.2.1 Test Images

eXcite cameras include a test image mode as a smart feature. The test image mode is used to check the camera’s basic functionality and its ability to transfer an image to the processor. The test image mode can be used for service purposes and for failure diagnostics. In test mode, the image is generated with a software program and the camera’s digital devices and does not use the optics, the sensor’s pixel array, or the ADCs. Three test images are available on eXcite cameras.

When the test image feature is enabled, the gain, brightness, and exposure time have no effect on the image.

When the test image feature is enabled, you must trigger the start of image capture as you normally would. Each time image capture is triggered, the camera section will transfer a test image rather than an actual captured image.

Setting the value of the Test Image parameter will enable the feature and select a test image as shown below.

Parameter Value	Result
TestImage_Disabled	Test images disabled
TestImage_TestImage1	Test image 1 enabled
TestImage_TestImage2	Test image 2 enabled
TestImage_TestImage3	Test image 3 enabled

To set the Test Image parameter value, access the eXcite API and use the SetValue method for the TestImage object. (For more information on using the API, refer to the API Reference documentation.

Test Image one

As shown in Figure 5-24, test image one consists of rows with several gray scale gradients ranging from 0 to 255. Assuming that the camera is operating at full resolution and is set for a monochrome, 8 bit output mode, when the test images are generated:

- row 0 starts with a gray value of 1 for the first pixel,
- row 1 starts with a gray value of 2 for the first pixel,
- row 2 starts with a gray value of 3 for the first pixel, and so on.

A test image is generated displaying inclined stripes. Each stripe shows one gray scale gradient between its edges. The number of inclined stripes depends the number of pixels involved in generating the test image. Therefore, at full resolution, the number of stripes depends on the sensor of your camera model.

(If the camera is operating at a lower resolution when the test images are generated, the basic appearance of the test pattern will be similar to Figure 5-24, but the starting pixel gray values on each row will not be as described above.)

The mathematical expression for test image one is:

$$\text{Gray value} = [\, x + y + 1 \,] \text{ MOD } 256$$

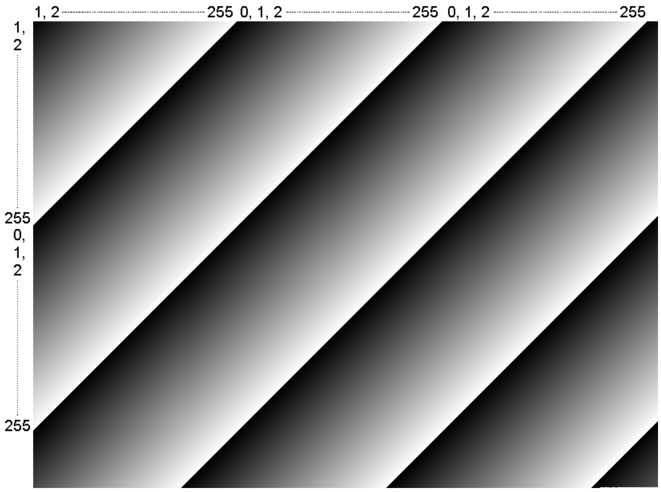


Figure 5-24: Test Image One (Example)

Test Image Two

As shown in Figure 5-25, test image two consists of rows with several gray scale gradients ranging from 0 to 255. Assuming that the camera is operating at full 656 x 491 resolution and is set for a monochrome, 8 bit output mode, when the test images are generated:

- rows 0, 1, and 2 start with a gray value of 0 for the first pixel,
- rows 3, 4, 5, and 6 start with a gray value of 1 for the first pixel,
- rows 7, 8, 9, and 10 start with a gray value of 2 on the first pixel, and so on.

(If the camera is operating at a lower resolution when the test images are generated, the basic appearance of the test pattern will be similar to Figure 5-25, but the staring pixel values on each row will not be as described above.)

The mathematical expression for test image two is:

$$\text{Gray value} = \frac{[\,x + y + 1\,]}{4} \text{ MOD 256, } \quad \text{round off all values}$$

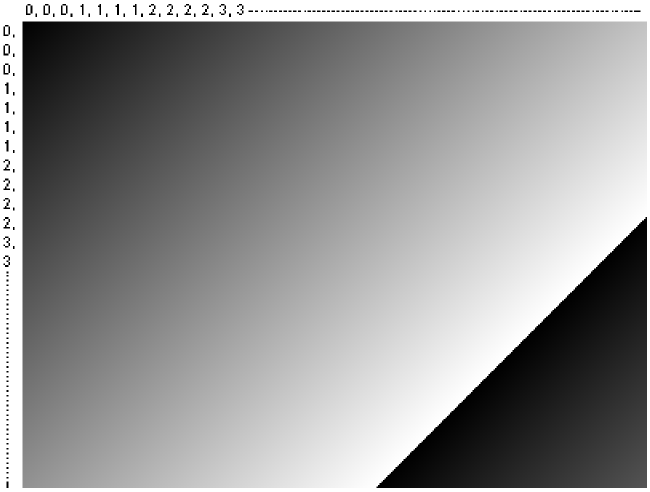


Figure 5-25: Test Image Two

Test Image Three

Test image three is similar to test image one but it is not stationary. The image moves by 1 pixel from right to left whenever an image is captured by the camera.

5.10.2.2 Shutter Time Base

The shutter time base smart feature can be used to change the time base for the exposure time feature (see Section 5.3.4).

Enabling the Shutter Time Base

By default, the shutter time base smart feature is disabled and the shutter time base is fixed at 20 µs. To enable the shutter time base smart feature you must use the Shutter Time Base Enable parameter (disabled = 0, enabled = 1, default = 0).

To set the Shutter Time Base Enable parameter, access the eXcite API and use the SetValue method for the Shutter Time Base Enable object. (For more information on using the API, refer to the API Reference documentation.)

Changing the Shutter Time Base

As described in Section 5.3.4 the exposure time is determined by a combination of two values. The first is the value of the Shutter parameter. The second is the value of the Shutter Time Base parameter. The exposure time is determined as follows:

Exposure Time = (Shutter Parameter value) x (Shutter Time Base parameter value)

The Shutter Time Base is fixed at 20 µs by default and the exposure time is normally adjusted by changing the value of the Shutter parameter only. However, if you require an exposure time that is shorter or longer than what you can achieve by changing the Shutter parameter value alone, the shutter time base can also be changed. The Shutter Time Base smart feature can be used to change the shutter time base.

The Shutter Time Base parameter value can range from 0.000015 second to 0.001221 second in increments of 0.000001 second (decimal). So if the value is set to 0.000040 second and if the value of the Shutter parameter is set to 100, for example, the exposure time will be 100 x 40 µs or 4000 µs.

To set the Shutter Time Base parameter, access the eXcite API and use the SetValue method for the Shutter Time Base object. (For more information on using the API, refer to the API Reference documentation.)

5.10.2.3 Trigger Counter and Trigger Flag

The eXcite includes a trigger counter feature and a trigger flag feature.

The trigger counter increments by one each time an image capture is triggered regardless of whether the trigger is internal (one shot or continuous shot commands only) or is external (hardware or software trigger). Triggers that occur when the camera is not ready are discarded and not counted. The trigger counter wraps to zero after 65535 is reached.

You can read the value of the Trigger Counter parameter to determine the current trigger count. To read the Trigger Counter parameter value, access the eXcite API and use the GetValue method for the TriggerCounter object. (For more information on using the API, refer to the API Reference documentation.)

Writes to the Trigger Counter parameter are ignored.

The trigger flag indicates whether one or more triggers has been detected since the last time the value of the Trigger Flag parameter was checked. If one or more triggers has been detected since the last time the Trigger Flag parameter was read, the trigger flag will be true. Reading the Trigger Flag parameter clears the value of the flag to false.

To read the Trigger Flag parameter value, access the camera API and use the GetValue method for the TriggerFlag object.

Writes to the Trigger Flag parameter are ignored.

5.10.2.4 Extended Version Information

eXcite cameras include a register that contains version numbers for the camera’s internal software. For troubleshooting purposes, Basler technical support may ask you to read this register and to supply the results. (For more information on contacting Basler technical support, see Section 7.)

To read the extended version information, access the eXcite API and use the Get Value method for the ExtVerInfo object. (For more information on using the API, refer to the API Reference documentation. The Get Value method will return a string containing the version information.)

6 Image Data Formats and Ranges

6.1 Image Data Basics

6.1.1 Pixel Transfer Sequence

The eXcite pixel data is transferred from the camera section to the processor section in the following sequence where

m = maximum number of pixels of the sensor in horizontal direction

n = maximum number of pixels of the sensor in vertical direction:

Row 0/Pixel 0, Row 0/Pixel 1, Row 0/Pixel 2 ... Row 0/Pixel m-2, Row 0/Pixel m-1

Row 1/Pixel 0, Row 1/Pixel 1, Row 1/Pixel 2 ... Row 1/Pixel m-2, Row 1/Pixel m-1

Row 2/Pixel 0, Row 2/Pixel 1, Row 2/Pixel 2 ... Row 2/Pixel m-2, Row 2/Pixel m-1

•

•

Row n-3/Pixel 0, Row n-3/Pixel 1, Row n-3/Pixel 2 ... Row n-3/ Pixel m-2, Row n-3/Pixel m-1

Row n-2/Pixel 0, Row n-2/Pixel 1, Row n-2/Pixel 2 ... Row n-2/ Pixel m-2, Row n-2/Pixel m-1

Row n-1/Pixel 0, Row n-1/Pixel 1, Row n-1/Pixel 2 ... Row n-1/ Pixel m-2, Row n-1/Pixel m-1

This sequence assumes that the camera is set for full resolution

6.2 Image Data Formats

6.2.1 Data Format with the Camera Set for Mono 8 Output

The table below describes how the data for a captured image will be ordered after it has been transferred to an image buffer in the MIPS processor.

The following standards are used in the table:

P_0 = the first pixel transmitted by the camera

P_n = the last pixel transmitted by the camera

B_0 = the first byte in the buffer

B_m = the last byte in the buffer

Byte	Data
B_0	Y value for P_0
B_1	Y value for P_1
B_2	Y value for P_2
B_3	Y value for P_3
B_4	Y value for P_4
B_5	Y value for P_5
B_6	Y value for P_6
B_7	Y value for P_7
•	•
•	•
•	•
B_{m-3}	Y value for P_{n-3}
B_{m-2}	Y value for P_{n-2}
B_{m-1}	Y value for P_{n-1}
B_m	Y value for P_n

6.2.2 Data Format with the Camera Set for Mono 16 Output

The table below describes how the data for a captured image will be ordered after it has been transferred to an image buffer in the MIPS processor.

The following standards are used in the table:

P_0 = the first pixel transmitted by the camera

P_n = the last pixel transmitted by the camera

B_0 = the first byte in the buffer

B_m = the last byte in the buffer

Byte	Data
B_0	High byte of Y value for P_0
B_1	Low byte of Y value for P_0
B_2	High byte of Y value for P_1
B_3	Low byte of Y value for P_1
B_4	High byte of Y value for P_2
B_5	Low byte of Y value for P_2
B_6	High byte of Y value for P_3
B_7	Low byte of Y value for P_3
B_8	High byte of Y value for P_4
B_9	Low byte of Y value for P_4
B_{10}	High byte of Y value for P_5
B_{11}	Low byte of Y value for P_5
•	•
•	•
•	•
B_{m-7}	High byte of Y value for P_{n-3}
B_{m-6}	Low byte of Y value for P_{n-3}
B_{m-5}	High byte of Y value for P_{n-2}
B_{m-4}	Low byte of Y value for P_{n-2}
B_{m-3}	High byte of Y value for P_{n-1}
B_{m-2}	Low byte of Y value for P_{n-1}
B_{m-1}	High byte of Y value for P_n
B_m	Low byte of Y value for P_n

ⓘ As shown in the table above, when the camera is set for 16 bit output, bytes are placed in an image buffer in **big endian format**.

When the camera is set for the mono 16 format, the camera outputs 16 bits per pixel. However, only 10 of the bits are effective in the exA640-120m/c and exA640-60m/c models and only 12 of the bits are effective in the exA1390-19m/c and exA1600-14m/c models (see Section 5.5).

6.2.3 Data Format with the Camera Set for Raw 8 Output

The table below describes how the data for a captured image will be ordered after it has been transferred to an image buffer in the MIPS processor.

The following standards are used in the tables:

P_0 = the first pixel transmitted by the camera for a line

P_n = the last pixel transmitted by the camera a line

B_0 = the first byte of data for a line

B_m = the last byte of data for a line

Even Lines	
Byte	**Data**
B_0	Red value for P_0
B_1	Green value for P_1
B_2	Red value for P_2
B_3	Green value for P_3
B_4	Red value for P_4
B_5	Green value for P_5
•	•
•	•
•	•
B_{m-5}	Red value for P_{n-5}
B_{m-4}	Green value for P_{n-4}
B_{m-3}	Red value for P_{n-3}
B_{m-2}	Green value for P_{n-2}
B_{m-1}	Red value for P_{n-1}
B_m	Green value for P_n

Odd Lines	
Byte	**Data**
B_0	Green value for P_0
B_1	Blue value for P_1
B_2	Green value for P_2
B_3	Blue value for P_3
B_4	Green value for P_4
B_5	Blue value for P_5
•	•
•	•
•	•
B_{m-5}	Green value for P_{n-5}
B_{m-4}	Blue value for P_{n-4}
B_{m-3}	Green value for P_{n-3}
B_{m-2}	Blue value for P_{n-2}
B_{m-1}	Green value for P_{n-1}
B_m	Blue value for P_n

6.2.4 Data Format with the Camera Set for Raw 16 Output

The table below describes how the data for a captured image will be ordered after it has been transferred to an image buffer in the MIPS processor.

The following standards are used in the tables:

P_0 = the first pixel transmitted by the camera for a line

P_n = the last pixel transmitted by the camera a line

B_0 = the first byte of data for a line

B_m = the last byte of data for a line

Even Lines	
Byte	**Data**
B_0	High byte of red value for P_0
B_1	Low byte of red value for P_0
B_2	High byte of green value for P_1
B_3	Low byte of green value for P_1
B_4	High byte of red value for P_2
B_5	Low byte of red value for P_2
B_6	High byte of green value for P_3
B_7	Low byte of green value for P_3
•	•
•	•
•	•
B_{m-7}	High byte of red value for P_{n-3}
B_{m-6}	Low byte of red value for P_{n-3}
B_{m-5}	High byte of green value for P_{n-2}
B_{m-4}	Low byte of green value for P_{n-2}
B_{m-3}	High byte of red value for P_{n-1}
B_{m-2}	Low byte of red value for P_{n-1}
B_{m-1}	High byte of green value for P_n
B_m	Low byte of green value for P_n

Odd Lines	
Byte	**Data**
B_0	High byte of green value for P_0
B_1	Low byte of green value for P_0
B_2	High byte of blue value for P_1
B_3	Low byte of blue value for P_1
B_4	High byte of green value for P_2
B_5	Low byte of green value for P_2
B_6	High byte of blue value for P_3
B_7	Low byte of blue value for P_3
•	•
•	•
•	•
B_{m-7}	High byte of green value for P_{n-3}
B_{m-6}	Low byte of green value for P_{n-3}
B_{m-5}	High byte of blue value for P_{n-2}
B_{m-4}	Low byte of blue value for P_{n-2}
B_{m-3}	High byte of green value for P_{n-1}
B_{m-2}	Low byte of green value for P_{n-1}
B_{m-1}	High byte of blue value for P_n
B_m	Low byte of blue value for P_n

As shown in the table above, when the camera is set for raw 16 bit output, bytes are placed in an image buffer in **big endian format**.

When the camera is set for the mono 16 format, the camera outputs 16 bits per pixel. However, only 10 of the bits are effective in the exA640-120c and exA640-60c models and only 12 of the bits are effective in the exA1390-19c and exA1600-14c models (see Section 5.6).

6.2.5 Data Format with the Camera Set for YUV 4:2:2 Output

The table below describes how the data for a captured image will be ordered after it has been transferred to an image buffer in the MIPS processor.

The following standards are used in the table:

P_0 = the first pixel transmitted by the camera

P_n = the last pixel transmitted by the camera

B_0 = the first byte in the buffer

B_m = the last byte in the buffer

Byte	**Data**
B_0	U value for P_0
B_1	Y value for P_0
B_2	V Value for P_0
B_3	Y value for P_1
B_4	U value for P_2
B_5	Y value for P_2
B_6	V Value for P_2
B_7	Y value for P_3
B_8	U value for P_4
B_9	Y value for P_4
B_{10}	V Value for P_4
B_{11}	Y value for P_5
•	•
•	•
•	•
B_{m-7}	U value for P_{n-3}
B_{m-6}	Y value for P_{n-3}
B_{m-5}	V Value for P_{n-3}
B_{m-4}	Y value for P_{n-2}
B_{m-3}	U value for P_{n-1}
B_{m-2}	Y value for P_{n-1}
B_{m-1}	V Value for P_{n-1}
B_m	Y value for P_n

6.3 Image Data Ranges

6.3.1 Data Range for a Mono 8 or a Raw 8 Component

The data output for a mono 8 or a raw 8 component is 8 bit data of the “unsigned char” type. The range of data values for a Y mono component and the corresponding indicated signal levels are shown below.

This Data Value (Hexadecimal)	Indicates This Signal Level (Decimal)
0xFF	255
0xFE	254
●	●
●	●
●	●
0x01	1
0x00	0

6.3.2 Data Range for a Mono 16 or a Raw 16 Component

The data output for a mono 16 or a raw 16 component is 16 bit data of the “unsigned short (little endian)” type. The range of data values for a Y mono component and the corresponding indicated signal levels are shown below.

CMOS Camera Models

This Data Value (Hexadecimal)	Indicates This Signal Level (Decimal)
0x03FF	1023
0x03FE	1022
●	●
●	●
●	●
0x0001	1
0x0000	0

CCD Camera Models

This Data Value (Hexadecimal)	**Indicates This Signal Level (Decimal)**
0x0FFF	4095
0x0FFE	4094
●	●
●	●
●	●
0x0001	1
0x0000	0

Normally, the data values for a 16 bit component would range from 0x0000 to 0xFFFF. However, when an eXcite CMOS camera is set for 16 bit output, only 10 bits are effective. Therefore, the highest data value you will see is 0x03FF indicating a signal level of 1023. When an eXcite CCD camera is set for 16 bit output, only 12 bits are effective. Therefore, the highest data value you will see is 0x0FFF indicating a signal level of 4095.

6.3.3 Data Range for a U or a V Component

The data output for a U or a V component is 8 bit data of the “straight binary” type. The range of data values for a U or a V component and the corresponding indicated signal levels are shown below.

This Data Value (Hexadecimal)	**Indicates This Signal Level (Decimal)**
0xFF	127
0xFE	126
●	●
●	●
●	●
0x81	1
0x80	0
0x7F	-1
●	●
●	●
●	●
0x01	-127
0x00	-128

The signal level of a U component or a V component can range from -128 to +127 (decimal). Notice that the data values have been arranged to represent the full signal level range.

7 Troubleshooting and Support

This section outlines the resources available to you if you need help working with your camera.

7.1 Technical Support Resources

If you need advice about your camera or if you need assistance troubleshooting a problem with your camera, you can contact the Basler technical support team for your area. Technical support contact information is located in the front pages of this manual.

You will also find helpful information such as frequently asked questions, downloads, and technical notes at our website: www.basler-vc.com.

If you do decide to contact Basler technical support, please take a look at the form that appears on the last two pages of this section before you call. Filling out this form will help make sure that you have all of the information the Basler technical support team needs to help you with your problem.

7.2 Before Contacting Basler Technical Support

To help you as quickly and efficiently as possible when you have a problem with a Basler camera, it is important that you collect several pieces of information before you contact Basler technical support.

Copy the form that appears on this and the next page (or download it from the support section of www.basler-vc.com), fill it out, and fax the pages to your local dealer or to your nearest Basler support center. Or, you can write an e-mail listing the requested pieces of information and with the requested files attached. Our technical support contact numbers are shown in the front section of this manual.

1 The camera’s product ID: ______________________

2 The camera’s serial number: ______________________

3 The operating system of your PC: ______________________

4 If you use a PC with a Linux operating system: ______________________

The kind of Linux distribution: ______________________

The kernel version: ______________________

5 If you use a PC with a Windows operating system: ______________________
____

The SFF Viewer GX version: ______________________

6 Describe the problem in as much detail as possible: ______________________
____

(If you need more space, use an extra sheet of paper.) ______________________
____
____
____

7 If known, what’s the cause of the problem? ______________________
____
____

8 When did the problem occur? □ After start. □ While running.

□ After a certain action (e.g., a change of parameters):

____
____
____
____

9 How often did/does the problem occur?

□ Once. □ Every time.

□ Regularly when:

____

□ Occasionally when:

____

____

10 How severe is the problem?

□ Camera can still be used.

□ Camera can be used after I take this action:

____

□ Camera can no longer be used.

11 Did your application ever run without problems?

□ Yes □ No

12 Parameter set

It is very important for Basler technical support to know the exact camera parameters that you were using when the problem occurred. Please try to state the following parameter settings:

□ Data format ______________________________

□ Exposure time control: ______________________________

□ Exposure time: ______________________________

□ Gain: ______________________________

□ Brightness: ______________________________

13 Live image/test image

If you are having an image problem, try to generate and save live images that show the problem. Also generate and save test images. Please save the images in BMP format, zip them, and send them to Basler technical support.

See Section 3.5.10 for sample programs for transferring images from the eXcite to a PC if you are working in a Linux environment. If you use a PC equipped with a Windows 2000 or Windows XP operating system, you can use the Basler SFF Viewer GX software to generate and save life and test images.

Appendix A
License Information

A.1 GPL and GenICam Licensing

The Linux kernel modules on the eXcite are General Public License (GPL) software. If you modify the code for the kernel modules, you must adhere to the GPL terms. A copy of the GPL is available from the licensing section of the Free Software Foundation website at: www.fsf.org.

The API for the eXcite is intended to be compliant with the GenICam™ standard. If you would like to obtain the source code for the API and make changes to it, you must register with the GenICam standards group and you must agree to adhere to the terms of the GenICam license agreement. For more information and to obtain a copy of the license, please visit the GenICam website at: www.genicam.org

Any application software you write for use on the eXcite that simply interfaces with the Linux kernel modules and with the eXcite API **is not** required to adhere to the terms of the GPL or the GenICam license.

Revision History

Doc. ID Number	Date	Changes
DA00074501	21 June 2005	Initial release of the eXcite User's Manual (for prototype cameras only).
DA00074502	22 June 2005	Revised version of the User's Manual for prototypes with updated install procedures.
DA00074503	30 Sept 2005	First release of the User's Manual for production cameras.
DA00074504	02 May 2006	Updated Singapore Address. Added information throughout the manual specific to the CCD models exA1390-19 m/c and exA1600-14 m/c. Made small changes throughout the manual to ensure compatibility with CCD cameras. Added Mono 16 * to exA640-60m/c and exA640-120m/c and changed power consumption to 14 W in Table 1-2 and Section 4.3. Added information on electromagnetic compatibility in Sections 1.2.1 and 1.7. Replaced the VI editor by the nano editor in Sections 2.1.1.4 and 5.9.5. Modified step 4 in Section 2.1.1.4. Deleted steps 5 to 7 in Section 2.1.1.4. Updated Figure 2-2 (configuration file). Added Sections 2.1.1.6 "Configuring Network Speed and Duplex Mode" and 2.1.1.7 "Shutting the Camera Down". Deleted Windows 2000 from Section 2.4.1. Modified required free space on hard drive in Section 2.4.1. Replaced steps 14 to 16 by steps 14 and 15 (network bridge) in Section 2.4.2 . Added loadkeys command in Sections 2.4.2, 2.4.3, and 2.4.4.4. Added Section 2.4.2.1 "Disabling Windows Data Execution Prevention". Modified step 3 in Section 2.4.3. Modified introductory note in step 12 in Section 2.4.3. Changed IP address of the VNC Server in step 16 in Section 2.4.3. Modified step 2 in Section "Recommended Procedure for Closing coLinux and the VNC Viewer". Added note in Section 3.4.2 to run StreamingClient only in a Linux environment. Added information on the Shutter Time Base smart feature in Section 5.3.4. Added Mono 16 output format to Section 5.6.2. Corrected response of LED to overtemperature condition in Section 5.9.13.2.

Doc. ID Number	Date	Changes
DA00074504	02 May 2006	Corrected status indicated by LED in Section 5.9.13.3. Generalized explanation of test image one in Section 5.10.2.1. Added Section 5.10.2.2 ”Shutter Time Base”. Generalized pixel transfer sequence in Section 6.1.1.
DA00074507	15 May 2007	Updated Basler address in the U.S.A. Integrated the CP camera variants. Added input voltage safety warnings in Sections 1.7, 4.1.2, and 4.3. Made minor modifications to mechanical dimensions in Figure 1-11. Added Section 1.4.4 ”Mechanical Stress Test Results”. Added environmental requirements for storage in section 1.6.1. Made former step 16 (”Restart your computer“) part of step 15 in Section 2.4.2. Modified description and added reboot in Section 2.4.2.1. Modified step 16 in Section 2.4.3 for using the current IP address of colinux. Added recommendation to disable the DEP feature in Sections 2.4.3 and 2.4.4.2. Added note related to the Basler SFF Viewer GX software in Section 3. Replaced ”offset“ by ”brightness“ in Section 5.9.1. Added ”Software Trigger / One-shot Operation” in Section 5.3.2. Modified ”Software Trigger / Continuous-shot Operation” in Section 5.3.2. Removed strobe control (deleted sections 5.9.11 and 5.10.5.3, modified sections 3.5.6 and 5.9.9). Removed reporting smart features (deleted sections 5.10.2, 5.10.3, and 5.10.4). Added Section 7 ”Troubleshooting and Support”

Feedback

Your feedback will help us improve our documentation. Please click the link below to access an online feedback form. Your input is greatly appreciated.

http://www.baslerweb.com/umfrage/survey.html

Index

A

address
 see IP address
AOI
 see area of interest
API 3-1, 3-2
area of interest
 changing parameters on-the-fly 5-48
 explained 5-46
 order of pixel colors 5-32
autonegotiation 2-12

B

bandwidth 5-30
bandwidth parameter 5-29, 5-30, 5-64
bayer filter 5-32
 alignment 5-32
block diagram 5-3
blue screen 2-29
board temperature parameter 5-63
brightness
 CMOS cameras 5-38
brightness error 5-65
brightness parameter 5-65
 CCD cameras 5-43
 CMOS cameras 5-40
bsreceive sample program 3-58
bssend sample program 3-58
building programs
 debug or release versions 3-10
 explained 3-3
 for different targets 3-10

C

cables
 power supply 4-6
 Y 4-8
camera models 1-1
camera shutdown 2-14
CCD cameras
 functional description 5-4
cleaning the camera and sensor 1-20
CMOS cameras
 functional description 5-1
code samples
 installing on a Linux PC 2-19
 installing on a Windows PC 2-21
coLinux
 closing procedure 2-35
 installing on a Windows PC 2-21
 opening procedure 2-35
color coding parameter 3-26, 5-31, 5-35, 5-64
color filter 5-32
 alignment 5-32
compiling programs
 see building programs
configuration
 network 2-1–2-8
 RS-232 serial port 5-49
configuration sets
 explained 5-61
 factory set 5-61
 startup memory channel 5-62
 user sets 5-61
 work set 5-61
connections
 general description 4-1
connector types 4-4
contact plate 1-1
continuous shot operation
 with a software trigger 5-8
 with external triggering 5-13
 with no triggering 5-6
continuous shot parameter 5-6, 5-13
cooling fins 1-1
CP housing variant 1-1
CPU
 speed 1-4
 type 1-4
CPU temperature parameter 5-63

D

data depth parameter 5-37
data formats
 see image data formats
debugging programs
 introduction 3-14
 with the Eclipse IDE 3-20
 with the gdb debugger 3-19
DEP
 see Windows Data Execution Prevention
dimensions 1-13
Display ID 2-39
display ID 2-32
duplex mode 2-12
dust 1-19
dynamic IP address 2-3

E

Eclipse IDE
 basics 3-3
 building debug or release versions 3-10
 building programs for different targets 3-10
 installing on a Linux PC 2-16
 installing on a Windows PC 2-21
electromagnetic compatibility 1-20
 noise suppression 1-2, 1-3
environmental requirements 1-18

error conditions
- brightness . . . 5-65
- error flag 1 . . . 5-64
- error flag 2 . . . 5-64
- gain . . . 5-64
- overtemperature . . . 5-63
- shutter error . . . 5-64
- trigger . . . 5-65
- undervoltage lockout . . . 5-63
- white balance . . . 5-65

error monitoring . . . 5-63
Ethernet port
- cable . . . 4-11
- connector . . . 4-4
- pin numbering and assignments . . . 4-2
- type . . . 4-11
- usage . . . 5-54

executables
- loading onto an eXcite . . . 3-8
- running on the eXcite . . . 3-9

executables, building
- see building programs

exposure
- controlling with a software trigger . . . 5-7
- controlling with an ExTrig signal . . . 5-9
- controlling with shot commands . . . 5-6, 5-7
- recommended method for controlling . . . 5-19

exposure time
- setting . . . 5-14

exposure time parameters . . . 5-14, 5-69
extended version information . . . 5-70
external trigger signal
- controlling exposure with . . . 5-9
- min high/low time . . . 5-9

F

factory configuration set . . . 5-61
factory settings
- see configuration sets

finding the IP address . . . 2-10
first sample program . . . 3-25
frame capture rate
- and AOI size . . . 5-23
- basic specification . . . 1-2, 1-3
- CCD cameras . . . 5-27
- changing . . . 5-29
- CMOS cameras . . . 5-23, 5-25
- maximum allowed . . . 5-23–5-28

frame interval parameter . . . 5-37
frame rate
- see frame capture rate

functional description
- CCD cameras . . . 5-4
- CMOS cameras . . . 5-1

functionality test . . . 2-15

G

gain
- CCD cameras . . . 5-41
- CMOS cameras . . . 5-38

gain error . . . 5-64
gain parameter . . . 5-64
- CCD cameras . . . 5-42
- CMOS cameras . . . 5-40

GenICam license . . . A-1
GPL license . . . A-1

H

hardware installation . . . 2-1
heat . . . 1-19
heat dissipation . . . 1-18
height parameter . . . 5-46, 5-64
housing size . . . 1-2, 1-3
housing variant
- CP . . . 1-1

humidity requirements . . . 1-18

I

IDE
- see Eclipse IDE

image data
- output format details . . . 6-2
- output formats on color cameras . . . 5-33
- output formats on mono cameras . . . 5-31

image data ranges . . . 6-7
image information parameters . . . 5-37
input ports . . . 4-13
- assigning to a standard input signal . . . 5-55
- assigning to a user defined signal . . . 5-55
- cabling . . . 4-16
- connector . . . 4-4
- electrical characteristics . . . 4-13
- parallel read . . . 5-56
- pin assignments and pin numbering . . . 4-2
- read response time . . . 5-60

input power . . . 4-6
input voltage . . . 1-19
installing
- code samples on a Linux PC . . . 2-19
- code samples on a Windows PC . . . 2-21
- coLinux on a Windows PC . . . 2-21
- Eclipse IDE on a Linux PC . . . 2-16
- Eclipse IDE on a Windows PC . . . 2-21
- hardware . . . 2-1
- tool chain on a Linux PC . . . 2-18
- tool chain on a Windows PC . . . 2-21
- VNC viewer on a Windows PC . . . 2-30

integrate enabled signal . . . 5-18
interface
- specifications . . . 1-4

IP address
dynamic 2-3
finding 2-10
setting 2-1–2-8
static 2-2
IR cut filter 1-6, 1-10, 1-15, 5-45

K

keyboard configuration
change. 2-27
language designation. 2-27
keyboard driver. 2-27

L

LED indicators 4-5
lens adapters 1-2, 1-3
lens thread length. 1-15, 1-19
level controlled exposure mode 5-11
library 3-2
licensing A-1
loading programs onto an eXcite 3-8

M

maximum frame capture rate 5-23
mechanical stress test 1-16
memory channels
see configuration sets
memory save channel parameter. 5-61
memory save parameter. 5-61
models, camera 1-1
mono 16 data range 6-7
mono 16 output format 5-31, 5-34, 6-3
mono 8 data range 6-7
mono 8 output format 5-31, 5-34, 6-2
mounting points 1-13
multigrab sample program 3-32

N

nano editor basics 2-9
network bridge
creating 2-24
dynamic IP address 2-38
static IP address 2-27
network configuration 2-5
network connections. 2-1–2-8
network speed 2-12
non-overlapped exposure. 5-15

O

one shot operation
with a software trigger 5-7
with external triggering. 5-13
with no triggering 5-6
one shot parameter 5-6, 5-13

output formats for image data
color cameras. 5-33
mono cameras 5-31
output port invert parameters 5-58
output port monitor parameters. 5-59
output port setting parameters 5-58
output port source select parameters 5-57
output ports 4-14
cabling 4-16
checking the state 5-58
configuring 5-57
connector 4-4
electrical characteristics 4-14
individual control. 5-57
inverting 5-58
parallel read or write. 5-59
pin assignments and pin numbering. 4-2
read and write response times 5-60
user settable. 5-58
overlapped exposure. 5-15
overtemperature condition 5-63
overvoltage 1-19, 4-2, 4-6

P

packet number parameter
. . . . 5-20, 5-22, 5-23, 5-25, 5-27, 5-30, 5-37
parameter sets, saving
see configuration sets
peer-to-peer network connection 2-4
pin assignments and numbering 4-2
PIO input parameter 5-56
PIO output parameter 5-59
pixel number parameter 5-37
pixel size 1-2, 1-3
pixel transfer sequence 6-1
position parameter 5-64
positioning accuracy
sensor. 1-11, 1-12
power requirements 1-2, 1-3, 4-6
power supply 4-6
precautions 1-19
processor
specifications 1-4
programmable exposure mode
with a software trigger 5-7
with an external trigger signal. 5-10
programs
loading onto an eXcite 3-8
running on an eXcite 3-9
programs, building
see building programs

R

raw 16 data range 6-7
raw 16 output format 5-33, 6-5
raw 8 data range 6-7
raw 8 output format 5-33, 6-4

readout time, frame
CCD cameras 5-21
CMOS cameras 5-20
requirements
environmental 1-18
heat dissipation 1-18
user 1-17
RS-232 serial port
cable 4-10
configuring 5-49
connector 4-4
physical and electrical characteristics 4-10
pin numbering and assignments 4-2
usage 5-49
rsreceive sample program 3-51
rssend sample program 3-51
running programs on an eXcite 3-9

S

sample code
see sample programs
sample programs
bsreceive 3-58
bssend 3-58
first 3-25
introduction 3-23
multigrab 3-32
rsreceive 3-51
rssend 3-51
simpledio 3-44
simplegrab 3-26
simplescalar 3-38
simpletrigger 3-40
simplewatchdog 3-47
streamingclient. 3-64
streamingserver 3-64
saving parameter sets
see configuration sets
security 1-19
sensor
pixel size 1-2, 1-3
positioning accuracy 1-11, 1-12
size 1-2, 1-3
type 1-2, 1-3
sensor readout time
CCD cameras 5-21
CMOS cameras 5-20
serial port
see RS-232 serial port
SFF Viewer GX. 3-1
shutter. 5-14
shutter error 5-64
shutter parameter . . . 5-6, 5-7, 5-10, 5-11, 5-14, 5-64
shutter time base 5-14, 5-69
shutter time base parameter. 5-69
simpledio sample program 3-44
simplegrab sample program 3-26
simplescalar sample program. 3-38
simpletrigger sample program 3-40
simplewatchdog sample program 3-47
smart features
defined 5-66
software licensing A-1
software trigger
controlling exposure start with 5-7
software trigger parameter 5-7
specifications
camera section 1-2
interface 1-4
processor section 1-4
spectral response 1-5
CCD cameras. 1-7
CMOS cameras 1-5
color cameras. 1-6, 1-9
monochrome cameras 1-5
startup memory channel
see configuration sets
startup memory channel parameter 5-62
static IP address 2-2
storage 1-18
streamingclient sample program. 3-64
streamingserver sample program 3-64
stress test 1-16
support 7-1

T

technical support. 7-1
temperature requirements. 1-18
temperature sensors 5-63
terminology 5-1
test image parameter 5-66
test images 5-66
timing chart
CCD cameras. 5-21
CMOS cameras 5-20
tool chain
installing on a Linux PC 2-18
installing on a Windows PC 2-21
total bytes parameter 5-37
transfer time, frame
CCD cameras. 5-21
CMOS cameras 5-20
trigger counter 5-70
trigger counter parameter 5-70
trigger enable parameter. 5-7, 5-12
trigger flag 5-70
trigger flag parameter 5-70
trigger mode parameter. 5-7, 5-12, 5-65
trigger polarity parameter 5-12, 5-65
trigger ready signal 5-16
trigger source parameter. 5-7, 5-12, 5-55, 5-65
troubleshooting 7-1

U

undervoltage lockout 5-63
USB ports
  cables 4-10
  connector 4-4
  physical characteristics 4-10
  pin numbering and assignments 4-2
  usage 5-54
user configuration sets 5-61
user requirements. 1-17

V

VI editor 2-9
video mode 5-36
video mode parameter 3-26, 5-36
VNC server
  adjusting Display ID 2-39
VNC viewer
  closing procedure 2-35
  installing on a Windows PC 2-30
  opening procedure 2-35

W

watchdog timer 5-60
weight 1-2, 1-3
white balance 5-44
white balance error conditions 5-65
white balance UB parameter 5-44, 5-65
white balance VR parameter 5-44, 5-65
width parameter 5-46, 5-64
Windows Data Execution Prevention 2-29
work configuration set. 5-61

X

X position parameter 5-46, 5-64

Y

Y cable 2-6, 4-8
Y position parameter 5-46
YUV 422 data range 6-9
YUV 422 output format 5-33, 6-6