

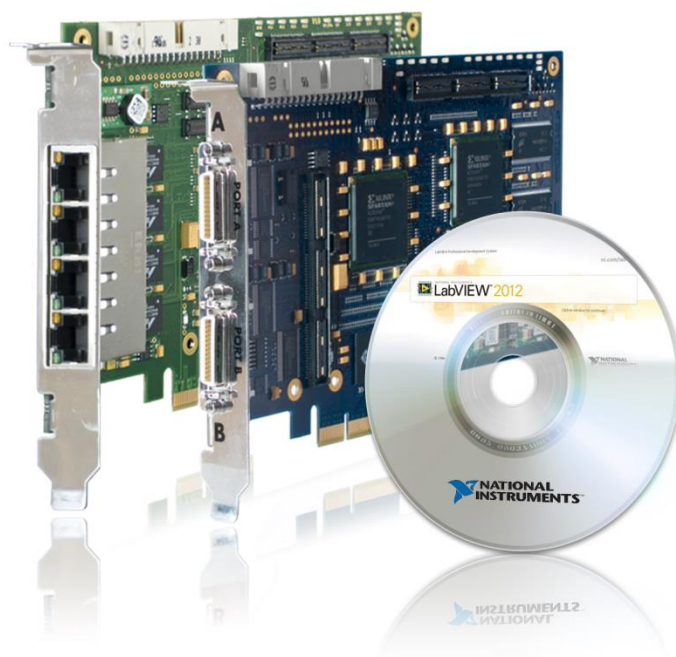
Silicon Software

Interface Library for

NI LabVIEW

Installation and User Guide

Version 2.1



Imprint

Silicon Software GmbH
Steubenstraße 46
68163 Mannheim, Germany
Tel.: +49 (0) 621 789507 0
Fax: +49 (0) 621 789507 10

© Copyright 2014 Silicon Software GmbH. All rights reserved.

Document Version: 1.0
Document Language: en (US)

Last Change: October 2014

Content

1	Introduction.....	5
1.1	Scope of the Silicon Software Interface Library for NI LabVIEW	5
1.2	Software Architecture.....	5
1.3	Concept.....	6
2	Installation.....	7
2.1	System Requirements	7
2.2	Supported Frame Grabber Models	7
2.3	Installation	8
3	How to use the Interface Library.....	10
3.1	Sample Program: SiSoLabviewInterface_Example_Gray.vi for Acquisition of Camera Link Gray Scale Images.....	11
3.1.1	Graphical User Interface.....	11
3.1.2	How to Use this Sample Program for Image Acquisition.....	16
3.1.3	Designing the Sample Program in LabVIEW (Block Diagrams)	19
	The Silicon Software Labview Interface DLL File	19
	Automatic readout of installation path via module FindWrapperDll.vi	21
	Initialization of the frame grabber	21
	Parameterization	23
	Retrieving parameter ID by parameter name	24
	Memory Allocation	27
	Image Acquisition	28
	Blocking and Non-Blocking Mode	29
	Fetching Image Data out of the LabView Interface into the LabView Application	30
	Image Data Array	32
	Freeing up Resources	35
3.2	Sample Program: SiSoLabviewInterface_Example_Color24.vi for Acquisition of Camera Link RGB Color Images	37
	Fetching Image Data out of the LabView Interface into the LabView Application	38

3.3	Sample Program: <i>SiSoLabviewInterface_Example_Gray_GigE.vi</i> for image acquisition with a GigE Vision Frame Grabber	40
3.3.1	Graphical User Interface	41
3.3.2	Programming for GigE Vision Frame Grabbers.....	42
3.4	Sample Program: <i>SiSoLabviewInterface_Example_Gray_Ext_NonBlo_UserMem_TruePointer.vi</i> ..	44
	Memory Allocation	44
	The Labview Code Interface Node(CIN)	46
	Image Acquisition with User-Controlled Memory Management	48
	Freeing the Memory with User-Controlled Memory Management	50

1 Introduction

1.1 Scope of the Silicon Software Interface Library for NI LabVIEW

The **Silicon Software Interface Library for NI LabVIEW** is an interface connecting the program **NI LabVIEW** with the **Silicon Software Runtime SDK Library**. Using the Interface Library, functions of the Runtime SDK Library can be called directly within LabVIEW. The Interface Library makes it easy for LabVIEW developers to handle the Runtime SDK Library and allows a fast integration of Silicon Software products into LabVIEW applications.

With the *Interface Library*, all frame grabbers of the Silicon Software product series can be implemented, and applets for image acquisition and image processing can be used.

1.2 Software Architecture

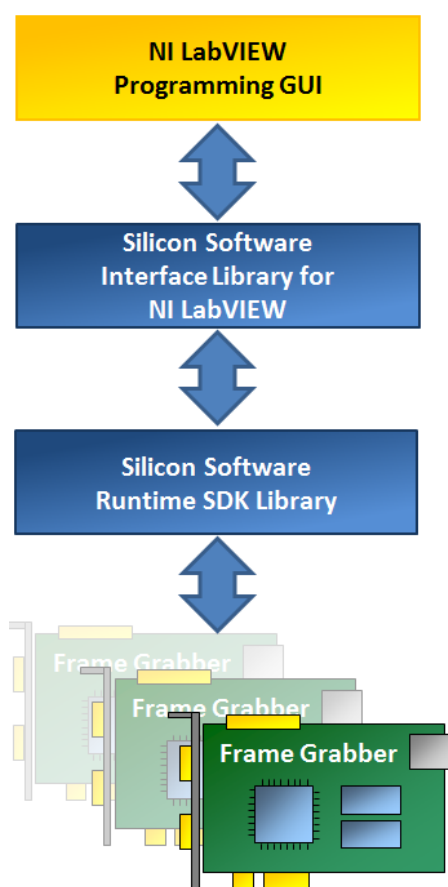




Figure 1: Software Architecture

1.3 Concept

In the *Runtime SDK Library*, many relevant objects (e.g., instance of frame grabber, DMA channel, camera, etc.) are defined by complex data structures which are provided in form of pointers.

	<p>Pointers</p> <p>Complex data types are handed over by the <i>Runtime SDK Library</i> as pointers, and the receiving application only has to manage these pointers.</p>
---	--

In LabVIEW, these objects cannot be used directly. Therefore, the *Interface Library* is mapping the functions of the *Runtime SDK Library* to a LabVIEW compatible format: In the *Interface Library*, all pointers pointing to complex data types of the *SDK Runtime Library* are defined as 64-bit integer values and processed accordingly.

	<p>Note</p> <p>In case of 32-bit applications, the according 32-bit value is provided in the 64-bit variable.</p>
---	--

2 Installation

2.1 System Requirements

- Operating System: MS Windows, 32-bit or 64-bit version
- Silicon Software Runtime
The Silicon Software runtime should be fully installed in the target system. We recommend runtime version 5.2.1 or higher.
- NI LabVIEW 12 Development System & Application Builder

2.2 Supported Frame Grabber Models

The *Interface Library* supports the following Silicon Software frame grabbers:

Camera Link, A Series:

- microEnable IV AS1-CL
- microEnable IV AD1-CL
- microEnable IV AD4-CL

Camera Link, V Series:

- microEnable IV VD1-CL
- microEnable IV VD4-CL

GigE Vision, A Series:

- microEnable IV AQ4-GE

GigE Vision, V Series:

- microEnable IV VQ4-GE

2.3 Installation

The *Interface Library* installation package contains the following files:

File Name		Function
SiSo_LabviewInterface.dll		Implementation of interfaces as shared library(.dll)
SiSo_LabviewInterface.llb		Contains all functional modules as VIs.
SiSo_LabviewInterface.h		Declaration of module prototypes
Guide_SiSO_LabviewInterface.pdf		Technical guide/description of concept
SiSo_LabviewInterface_Prototyp.html		Function reference
SiSoLabviewInterface_Example_Gray_Blocking.vi		Example of a gray image acquisition program in blocking mode(Camera Link 8bit/16bit)
SiSoLabviewInterface_Example_Gray_Non_Blocking.vi		Example of a gray image acquisition program in non-blocking mode(Camera Link 8bit/16bit)
	ExampeConfig_Gray_8bit.mcf	Corresponding sample configuration files – for frame grabber initialization with configuration file (see section Initialization of the frame grabber)
	ExampeConfig_Gray_16bit.mcf	
SiSoLabviewInterface_Example_RGB24_Non_Blocking.vi		Example of an 24bit RGB image acquisition program in non-blocking mode(Camera Link)
	ExampeConfig_Color_24bit.mcf	Corresponding sample configuration file – for frame grabber initialization with configuration file (see section Initialization of the frame grabber)
SiSoLabviewInterface_Example_Gray_GigE_Non_Blocking.vi		Example of a gray image acquisition program in non-blocking mode (GigE Vision)
	ExampeConfig_Gray_GigE_8bit.mcf	Corresponding sample configuration files – for frame grabber initialization with configuration file (see section Initialization of the frame grabber)
	ExampeConfig_Gray_GigE_16bit.mcf	

File Name	Function
SiSoLabviewInterface_Example_CISer.vi	Example for Camera Link Serial Interface
SiSoLabviewInterface_Example_Gray_Extension_Blocking.vi	New examples with advanced LabView interface moduls, for a gray image acquisition program in blocking mode (Camera Link)
SiSoLabviewInterface_Example_Gray_Extension_Non_Blocking.vi	New examples with advanced LabView interface moduls, for a gray image acquisition program in non-blocking mode (Camera Link)
SiSoLabviewInterface_Example_RGB_Extension_Non_Blocking.vi	New examples with advanced LabView interface moduls, for an RGB image acquisition program in non-blocking mode (Camera Link)
SiSoLabviewInterface_Example_Gray_Extension_GigE.vi	New examples with advanced LabView interface moduls, for a gray image acquisition program in non-blocking mode (GigE Vision)
SiSoLabviewInterface_Example_Gray_Ext_Non_Blo_UserMem_TruePointer.vi	New example to use user-controlled memory management, in this case the program delivers the image data as a true pointer


Table 1: Files contained in the installation package of the Silicon Software Interface Library for NI LabVIEW

3 How to use the Interface Library

For illustration of how to use the *Interface Library*, you find eight sample programs in the installation package:

- SiSoLabviewInterface_Example_Gray_Blocking.vi (Camera Link gray scale images)
- SiSoLabviewInterface_Example_Gray_Non_Blocking.vi
- SiSoLabviewInterface_Example_RGB24_Non_Blocking.vi (Camera Link RGB color images)
- SiSoLabviewInterface_Example_Gray_GigE_Non_Blocking.vi (GigE Vision gray scale images)
- SiSoLabviewInterface_Example_Gray_Extension_Blocking.vi
- SiSoLabviewInterface_Example_Gray_Extension_Non_Blocking.vi
- SiSoLabviewInterface_Example_RGB_Extension_Non_Blocking.vi
- SiSoLabviewInterface_Example_Gray_Extension_GigE.vi
- SiSoLabviewInterface_Example_Gray_Ext_NonBlo_UserMem_TruePointer.vi
- SiSoLabviewInterface_Example_CISer.vi

All eight sample programs are designed in LabVIEW, using the *Interface Library* for calling functions from the *Runtime SDK Library*.

	<p>Hands-on Examples</p> <p>For introducing you to the usage of the <i>Interface Library</i>, you find a detailed description on how the first of the eight programs was created in section 3.1.</p> <p>In sections 3.2 and section 3.3, you find information on how you can design other programs on the basis of the first example.</p>
---	--

Just open the programs and switch between GUI and diagram view to follow our explanations.

3.1 Sample Program: SiSoLabviewInterface_Example_Gray.vi for Acquisition of Camera Link Gray Scale Images

The sample program described in this section is designed for image acquisition with a Camera Link frame grabber. It is able to output 8-bit as well as 16-bit images.

We will describe the program in three steps:

Step 1: You are introduced to the graphical user interface of the program and the functions of the individual GUI elements as they are used by the program user (see section [3.1.1](#)).

Step 2: You get all information a user needs to use the program for acquiring images with a Camera Link frame grabber (see section [3.1.2](#)).

Step 3: You get detailed information on how the program was designed in LabVIEW (see section [3.1.3](#)).

3.1.1 Graphical User Interface

The graphical user interface of the sample program looks as follows:

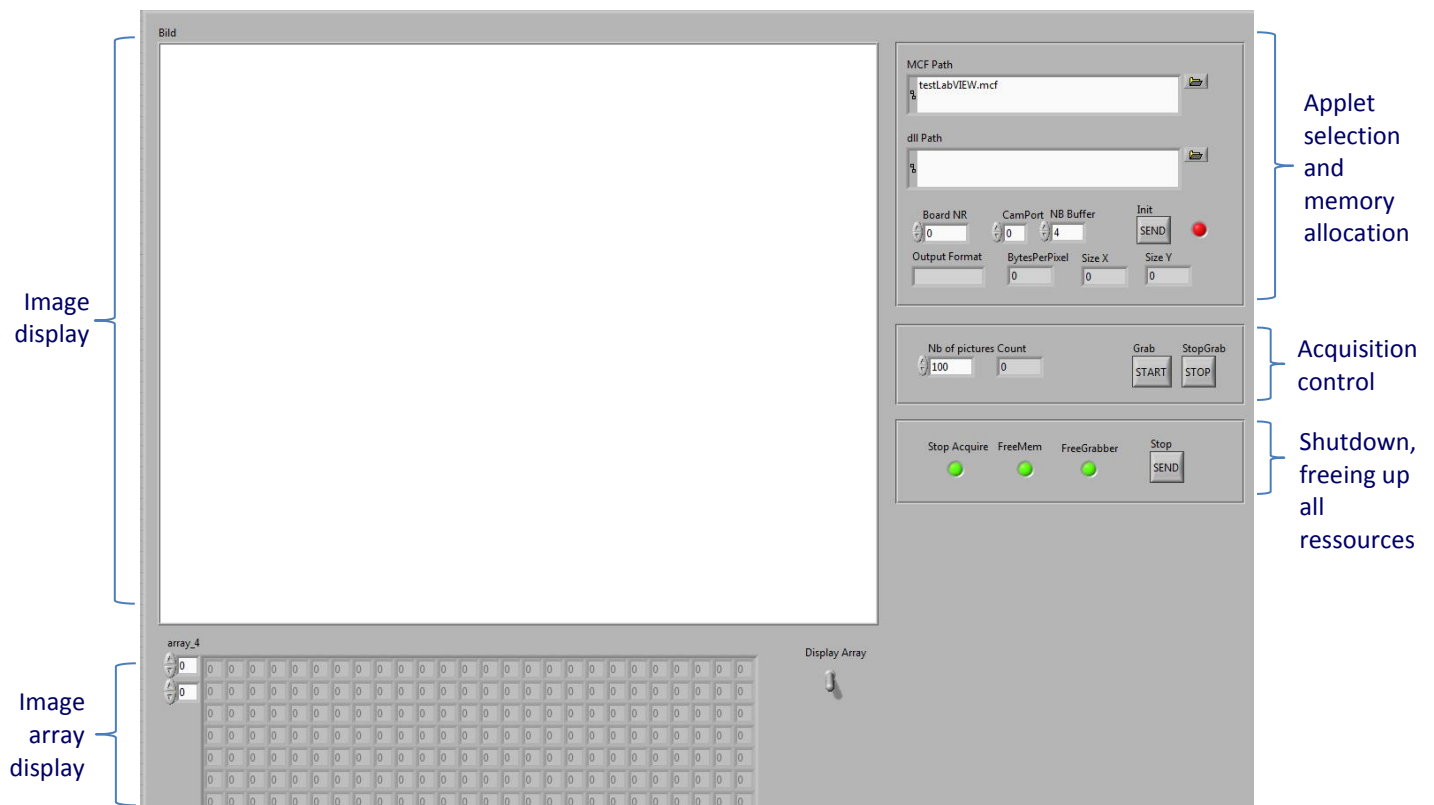


Figure 2: Graphical User Interface

The functions of the individual text fields, buttons and indicators are the following:

Selecting the applet, specifying the DLL path and allocating memory (1st panel right upper corner):

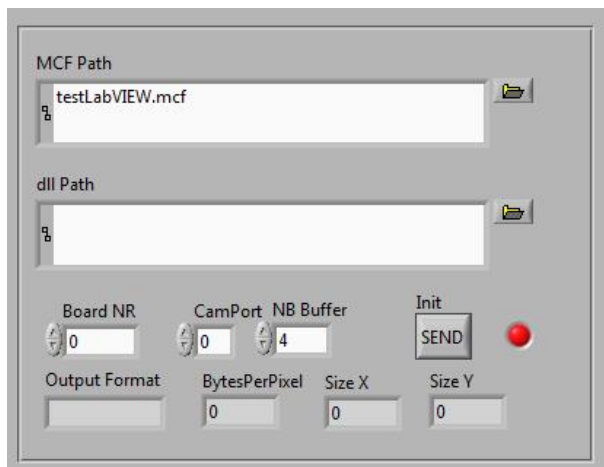


Figure 3: GUI elements in 1st panel (right upper corner)

Element Name	Element Type	Description
MCF Path	Input field	Path to/file name of configuration file (which contains the name of applet to be used and the according parameter values, see section 3.1.2)
DLL Path	Input field	Path to/file name of interface DLL file <i>SiSo_LabviewInterface.dll</i> , the actual implementation of the LabView interface; for more details, see section 3.1.3
Board NR	Input field	Board index of target frame grabber
CamPort	Input field	Number of the frame grabber port the camera is connected to
NB Buffer	Input field	Number of image buffers
Init [SEND]	Button	Starting initialization of frame grabber and applet load

Element Name	Element Type	Description
Indicator	Indicator	Status of initialization (red: error, green: successful)
Output format	Static input field	Output format (output color depth, i.e., bits per pixel) of applet
Bytes per Pixel	Static input field	Number of bytes per pixel
Size X	Static input field	Dimension of image in pixel
Size Y	Static input field	Dimension of image in pixel

Table 2: Functional description of GUI elements in 1st panel (right upper corner)

Acquisition control (2nd panel right upper corner):

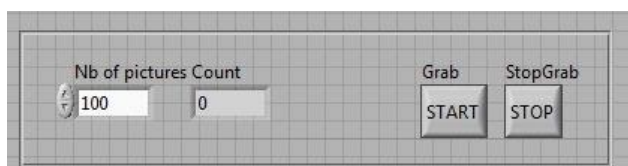


Figure 4: GUI elements in 2nd panel (right upper corner)

Element Name	Element Type	Description
Nb of pictures	Input field	Number of pictures to be acquired
Count	Static input field	Number of frames already transferred
START [Grab]	Button	Starts image acquisition
STOP [Stop Grab]	Button	Stops image acquisition – Count is set back to 0

Table 3: Functional description of GUI elements in 2nd panel (right upper corner)

Shut Down (3rd panel right upper corner):



Figure 5: GUI elements of 3rd panel (right upper corner)

Element Name	Element Type	Description
Stop Acquire	Indicator (red: error, green: successful)	Status display for action „Stop Acquire“
FreeMem		Status display for action „Free up memory“
FreeGrabber		Status display for action „Free up grabber“
Stop [SEND]	Button	Shut down of the whole system. After clicking this button, all resources that are used for acquiring image data are freed up. If you want to start acquisition after clicking this button, the grabber first needs to be initialized again (via the SEND [Init] button in the 1 st panel).

Table 4: Functional description of GUI elements in 3rd panel (right upper corner)

Image Area Display (display of transferred data in raw format, bottom panel):



Figure 6: GUI elements of bottom panel

Element Name	Element Type	Description	
[array_4]	Input field	Positioning, X value (definition of start pixel of display)	For Example, values 0 and 0 define the top left pixel of the processed image as starting point for the display.
[array_4]	Input field	Positioning, Y value (definition of start pixel of display)	
Display field	Display field	Image array display: Hexadecimal values of all bytes of the image array	
Display Array	Button	Switching image array display on or off	
Exit [OFF]	Button	Application is being stopped	

Table 5: Functional description of GUI elements in 4th panel (bottom)



3.1.2 How to Use this Sample Program for Image Acquisition

Before going into detail as to how the program has been designed in LabVIEW, we will give you a short overview on how the program is to be used by a user.

This is the instructions a user should follow when starting the actual image acquisition:

How to use the Program

1. Make sure you have a configuration file (.mcf) ready.

Prerequisite : Configuration File

The configuration file contains the following information:

- Name of applet that is to be used with the frame grabber
- Parameters and their values that have to be set when using the specified applet

For using the sample application, you can use the configuration files provided in the installation package. Two files are available for this application:

- ExampeConfig_Gray_8bit.mcf (for 8-bit color depth)
- ExampeConfig_Gray_16bit.mcf (for 16-bit color depth)

Alternatively, or when using other applets, the configuration file should be created as follows:

- a. Start the program microDisplay.
- b. Load the applet you want to use.
- c. Set the parameters as you need them.
- d. Save this configuration to file (.mcf) using the menu options *File* → *Save this Configuration*.

2. Start the sample program SiSoLabviewInterface_Example_Gray.vi (which has been designed in LabVIEW):

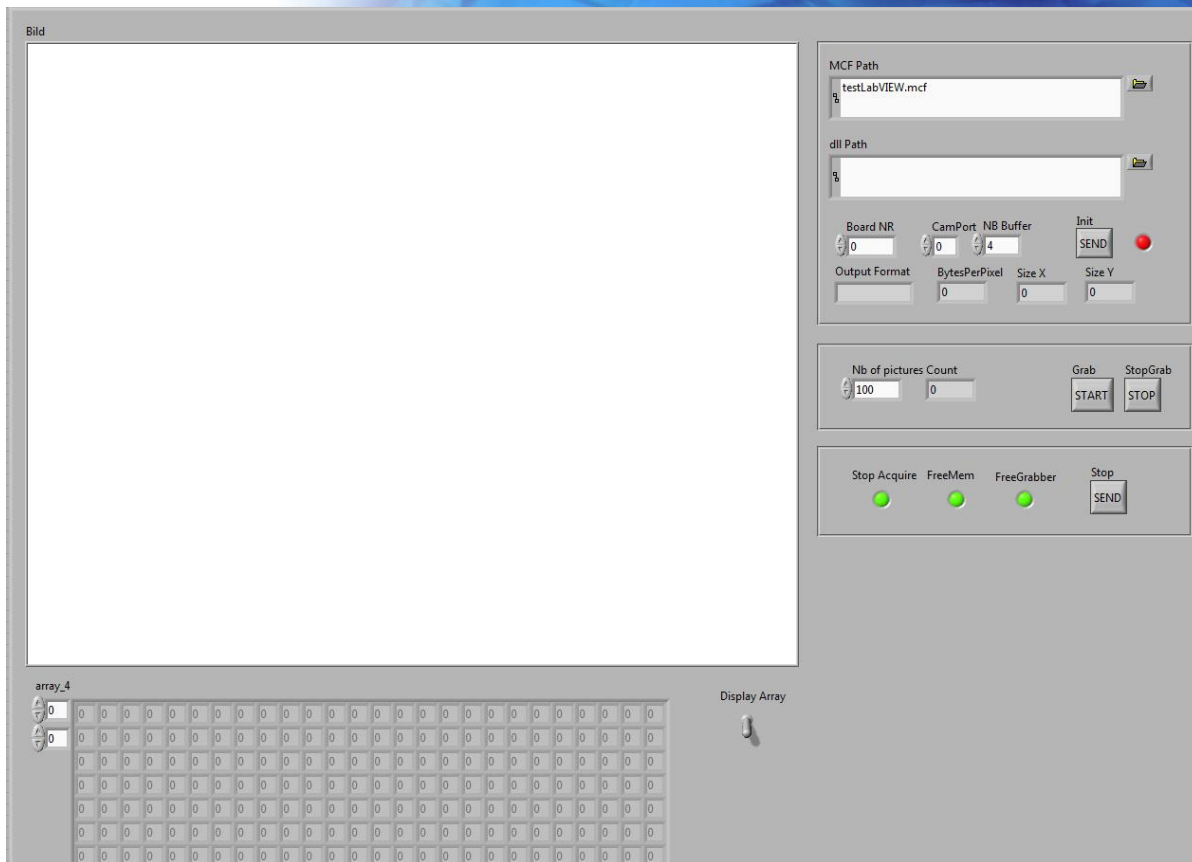


Figure 7: Graphical User Interface

3. Specify path and/or name of configuration file in the input field **MCF Path**. You can do so by using the explorer.
4. The LabView Interface is implemented as DLL file (*SiSo_LabviewInterface.dll*). Specify path and/or name of the DLL file in the input field **dll Path** (if not displayed automatically). You can do so by using the explorer.
5. Enter the board number (index number) of the frame grabber you want to use under **Board NR**.
6. Enter the CamPort number (number of frame grabber port the camera is connected to) under **CamPort**.
7. Enter the total of image buffers you want to use under **NB Buffer**.
8. Click on the **SEND** [Init] button to start frame grabber initialization.
If initialization has been successful, the indicator to the right of the button turns green.

9. Enter the number of pictures you want to acquire under ***Nb of pictures***.

10. Click on the **START** [Grab] button.

Now, the acquired images are displayed in the image display panel of the program window.

If no images are displayed, please check

- if the camera is connected correctly (in case of questions, refer to the camera documentation of the camera you are using), and/or
- if the parameters are set correctly (in case of questions, refer to the runtime documentation of the Silicon Software Runtime).


**Tip**

When the *Array Display* is switched to **on**, all byte values of the image array are displayed in hexadecimal format during acquisition.

11. If you want to stop image acquisition, click on the **STOP** [Grab] button.

12. If you are done with image acquisition, click on the **SEND** [Stop] button to free up all resources.

3.1.3 Designing the Sample Program in LabVIEW (Block Diagrams)

	<p>Reference and Further Reading</p> <p>The functions available over the <i>Silicon Software Interface Library for NI LabVIEW</i> are the same as the functions provided by the <i>Silicon Software Runtime SDK Library</i>. Therefore, in case of questions, you can always refer to the Silicon Software SDK documentation provided on the Silicon Software Homepage.</p>
---	--

Just open the block diagrams in LabVIEW to follow our explanations. The modules of the sample program will be described in the order as they are called.

The Silicon Software Labview Interface DLL File

The LabView interface is implemented as DLL file. The name of this DLL file is **SiSo_LabviewInterface.dll**. All interface modules are designed to show the DLL path in the diagram:

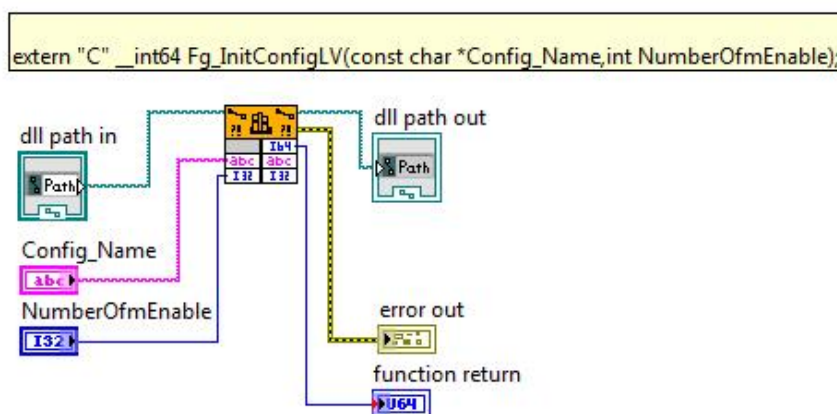


Figure 1: Block diagram of Modul Fg_InitConfigLV.vi

Location of Interface DLL



The interface DLL file is part of the Silicon Software SDK library. It is installed in directory (%SISODIR5%)\bin. (SISODIR5 is an environment variable pointing to the installation directory of the Silicon Software runtime library.)

The location of the interface DLL file has always to be stated since LabVIEW not always finds this file automatically (and warnings and error messages may be displayed in such cases).

Therefore, when using the interface, in the **Call Library Function** dialog:

1. Specify the path to the **SiSo_LabviewInterface.dll** file in each module.
2. Activate the option **Specify path on diagram**.

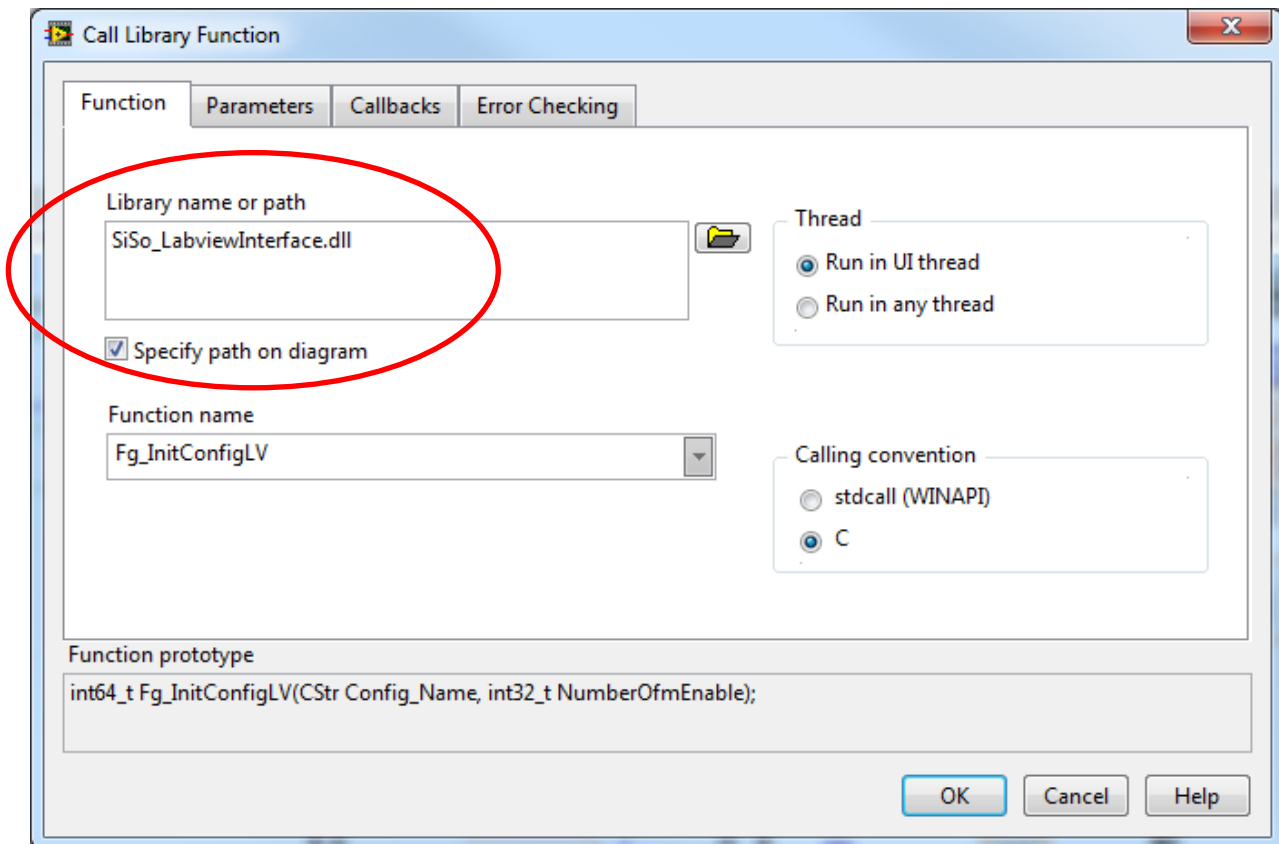


Figure 2: Dialog for module Fg_InitConfigLV.vi

Automatic readout of installation path via module FindWrapperDll.vi

The path to the interface DLL file is ascertained by the module **FindWrapperDll.vi**. The module reads the value of environment variable SISODIR5 and outputs the value as a path object in LabVIEW (see [Figure 3](#), field *DLL Path*). If the DLL file has been found, the full path to the file is displayed in the according field. Otherwise, the error message *Not found SiSo_LabviewInterface.dll* is displayed. In this case, the DLL file can be specified manually by the user.

Initialization of the frame grabber

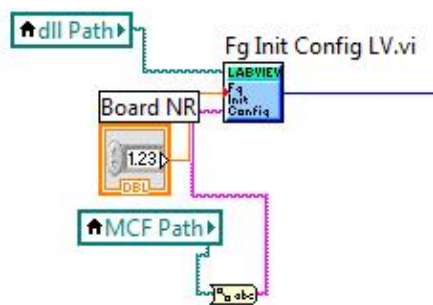


Figure 3: Part of Block Diagram: Initialization

There are two ways to initialize the frame grabber:

- A) Initialization on the basis of a configuration file (as implemented in the example program).
- B) Initialization without configuration file.

A) Initialization on the basis of a configuration file:

The configuration file specifies the applet that is to be used and contains pre-set parameter values. (For information on how to create a configuration file, see section [3.1.2](#)).

For initialization on the basis of a configuration file, module *Fg_InitConfigLV.vi* is available. When using *Fg_InitConfigLV.vi* (together with the configuration file), the applet specified in the configuration file is loaded and all parameters relevant for image acquisition are set automatically during initialization.

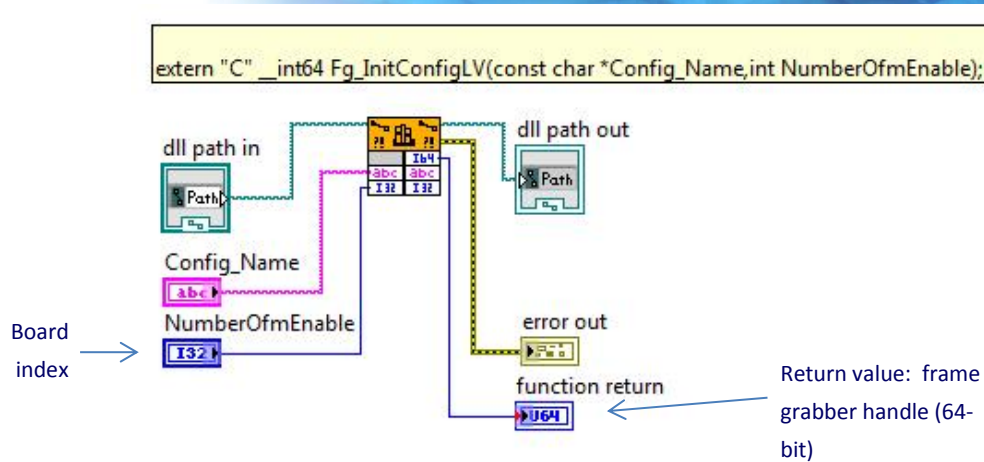


Figure 4: Definition diagram of Fg_InitConfigLV.vi

For testing, you find two example configuration files already prepared for you in the installation package:

- ExampleConfig_Gray_8bit.mcf (for 8-bit image format)
- ExampleConfig_Gray_16bit.mcf (for 16-bit image format)

B) Initialization without configuration file:

For initializing without configuration file, module *Fg_InitLV.vi* is available. When using *Fg_InitLV.vi*, the name of the applet has to be entered (e.g., *DualAreaGray16.dll*). All parameter values have to be set explicitly using module *Fg_setParameterLV.vi* (see below, section [Parameterization](#)). Otherwise, all parameters will use default values.

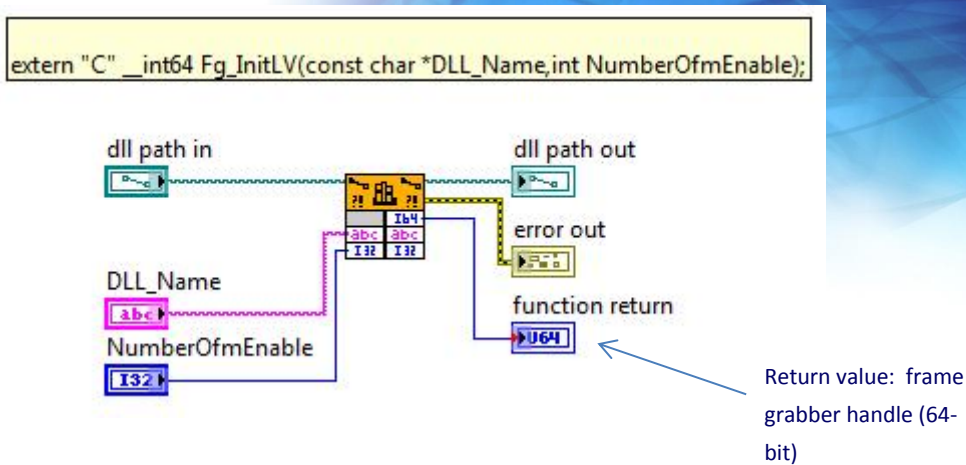


Figure 5: Definition diagram of Fg_InitLV.vi

In both cases (A and B), the return value is the frame grabber handle (pointer to the frame grabber instance). This 64-bit integer is saved to the local variable *BoardHandleAddr*.

In many cases (as you will see below), the *BoardHandleAddr* has to be entered as a parameter value.

The advantage of this model is that multiple frame grabbers can be managed.

Parameterization

For parameterization, the module *Fg_setParameterLV.vi* is available.



Note

In the sample program, parameterization is not necessary as the frame grabber is initialized via *Fg_InitConfigLV.vi* (using a configuration file). Therefore, all parameters are set automatically during initialization.

When calling *Fg_setParameterLV.vi*, you have to enter the values for two variables:

- *BoardHandleAddr* (return value of initialization, see above)
- *Parameter*

Parameter is a parameter ID pre-defined in the *Silicon Software Runtime SDK Library*. The value can be directly entered as an integer constant.

Parameters of a Specific Applet

Information on

- which parameters can be set for a certain applet,
- the names of the individual parameters

you will find in the applet documentation for the applet you are using. All applet documentation you find here:

http://www.siliconsoftware.de/download/live_docu/RT5/en/ind_acquisition.html

Looking up Parameter IDs

For looking up individual parameter IDs and to have an overview over the complete parameter ID definition, refer to the *Silicon Software Runtime SDK Library documentation*:

http://www.siliconsoftware.de/download/live_docu/RT5/en/documents/SDK/functions/fgrab_define_8h.html



Retrieving parameter ID by parameter name

As an alternative to looking up the parameter ID in the definition file (see above), you can also get the parameter ID by parameter name. To do so, call module *Fg_getParameterIdByNameLV.vi*. Here, the parameter name has to be entered:

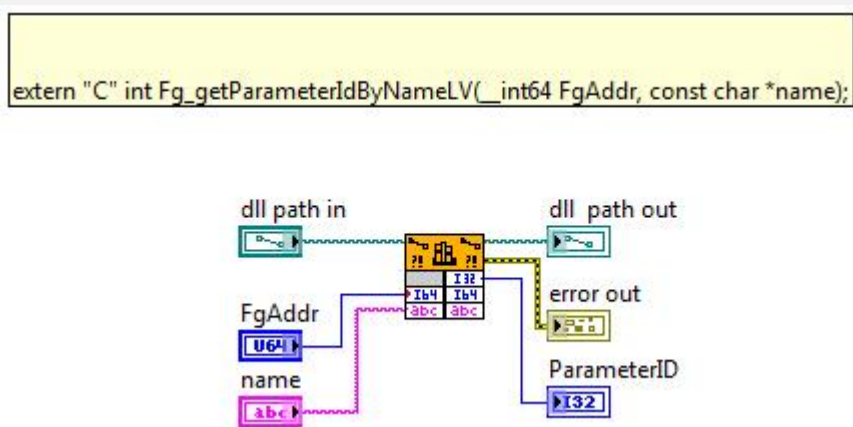


Figure 6: Definition of Module *Fg_getParameterIdByNameLV.vi*

To get information on the value currently set for a certain parameter, you can use the module *Fg_getParameterLV.vi*. When calling *Fg_getParameterLV.vi*, you have to enter *BoardHandleAddr* and *Parameter* (as you have to when calling *Fg_setParameterLV.vi*).

In the following example, module *Fg_getParameterLV.vi* is called three times to retrieve the values for the parameters width, height, and image output format.

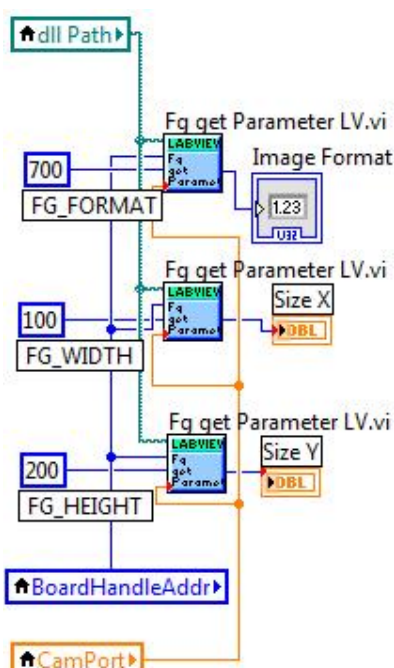


Figure 7: Using Module *Fg_getParameter LV.vi*

Further Information on Parameterization

For further information on parameterization, refer to the *Silicon Software Runtime SDK Library documentation*, *Fg_setParameter ()*:

http://www.siliconsoftware.de/download/live_docu/RT5/en/documents/SDK/function_s/fgrab_prototyp_8h.html#a8f448043d18dbdbcd9189a156f8f16b1

```
extern "C" int Fg_getParameterLV(_int64 FgAddr, int Parameter,void *Value,int CamPort);
```

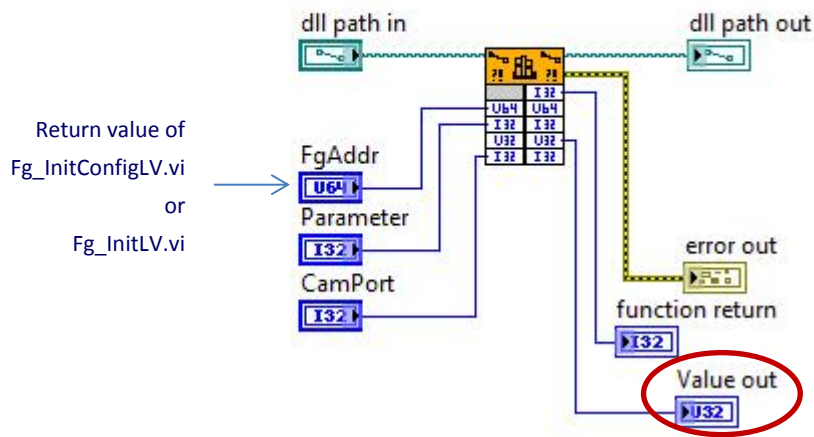


Figure 8: Definition of `Fg_getParameterLV.vi` (value output)

```
extern "C" int Fg_setParameterLV(_int64 FgAddr, int Parameter,void *Value,int CamPort);
```

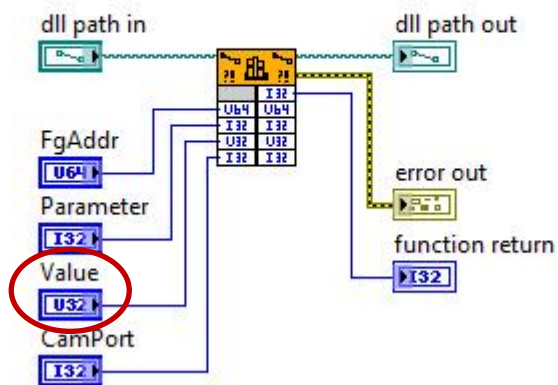


Figure 9: Definition of `Fg_setParameterLV.vi` (value input)

Memory Allocation

For allocating memory on the PC, the module *Fg_AllocMemLVEx.vi* is available.

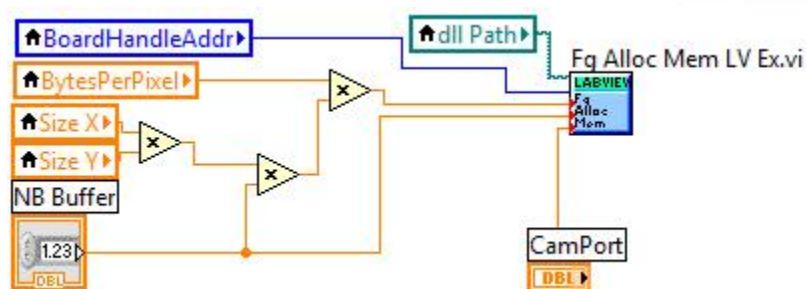


Figure 10: Block Diagram, Section “Memory Allocation”

Return value is the address of the allocated DMA memory block on the PC. The return value is stored in the local variable *DmaHandleAddr*.

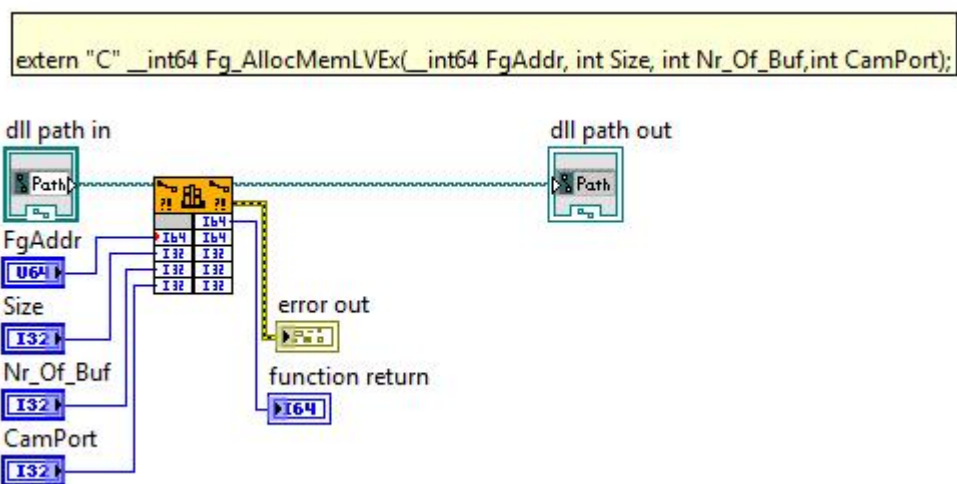


Figure 11: Definition of Module *Fg_AllocMemLVEx.vi*

As a rule of thumb, the parameter *Size* is calculated by the formula

$$\text{Height} * \text{Width} * \text{Number of Subbuffers} * \text{BytesPerPixel}.$$

The value of *BytesPerPixel* can be retrieved by calling module *Fg_getBytesPerPixelLV.vi*.

Prerequisite: Parameter *FG_FORMAT* has to be set correctly before (see section *Parameterization*)

Image Acquisition

On calling *Fg_AcquireLVEx.vi*, image acquisition is started.

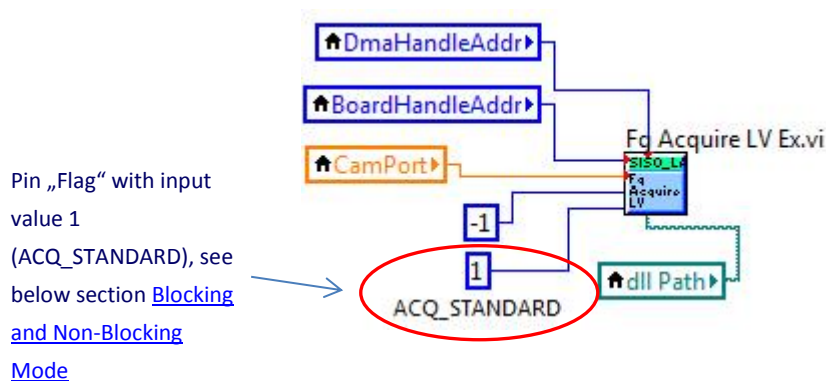


Figure 12: Block Diagram, Section "Image Acquisition"

On call of *Fg_AcquireLVEx.vi*, continuous image acquisition is started on the camera port specified in variable *CamPort*. Data is transferred to the DMA channel specified in variable *DmaHandleAddr*. Return value is error code describing the current status.



Starting Infinite Acquisition

When *PicCount* is set to *PicCount* = -1, images are acquired infinitely until acquisition is stopped by calling *Fg_stopAcquireLV.vi*.

Blocking and Non-Blocking Mode

The Silicon Software Runtime offers two completely different ways to arrange the memory:

- The **Standard Mode** or non-blocking mode (ACQ_STANDARD)
- The **Blocking Mode** (ACQ_BLOCK).

Which of the two modes is used for image acquisition you define via input pin “flag” in module *Fg_AcquireLVEx.vi*:

flag = 1 : The standard mode ACQ_STANDARD is used.

flag = 2 : The blocking mode ACQ_BLOCK is used.

The Standard Mode ACQ_STANDARD

The standard mode (ACQ_STANDARD) arranges the memory buffers using a roundrobin model. This model is also called ring buffer and is the most common method:

The first frame is transferred to buffer number one. The second frame is transferred to buffer number two, etc. When the last buffer has been filled, buffer number one is used again. There is a direct map between image and memory buffer (image buffer = image number modulo number of buffers).

The Blocking Mode ACQ_BLOCK

If the memory is arranged in blocking mode, each buffer is filled on a FIFO (first in first out) basis. At the beginning, all buffers of the memory are available and may be filled with frame data. As soon as a buffer is filled, it is deleted from the list of available buffers and hence, cannot be overwritten. As soon as the image data in a buffer is post-processed and hence not needed anymore, the buffer is added to the FIFO list of available buffers and may be reused again.

The frame buffers can be locked and freed in random order. Declaring a frame buffer as free and thus adding it back to the list of free buffers is entirely controlled by the user application.

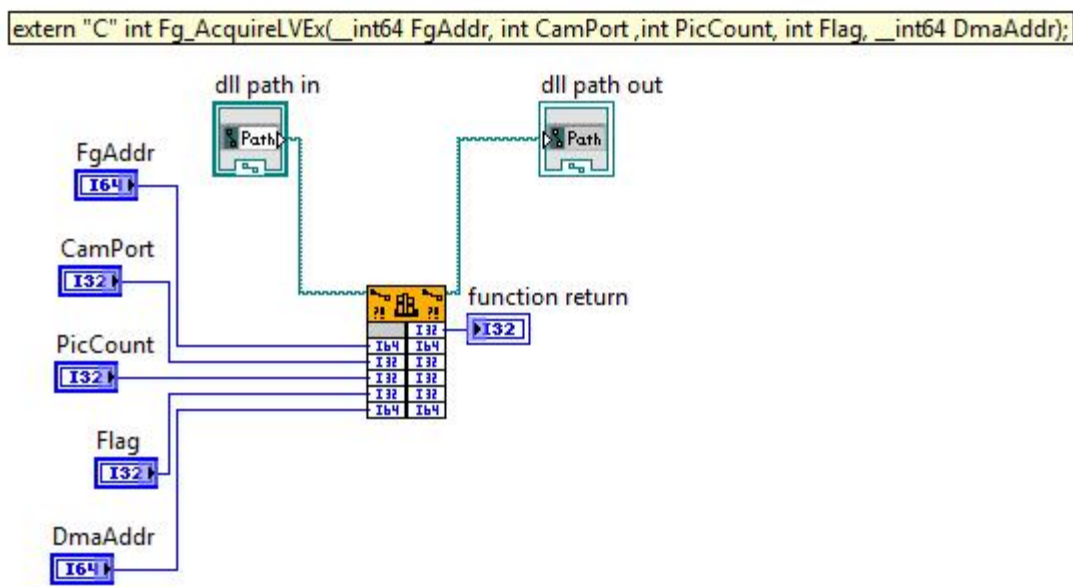


Figure 13: Definition of Module Fg_AcquireLVEx.vi

Fetching Image Data out of the LabView Interface into the LabView Application

The LabView Interface offers two ways to retrieve image data:

- A) Image Copying: This approach is optimized for direct use of image information within the LabView components. The LabView Programmer can fetch images in a very easy manner: The LabView Interface provides specific functions (not contained in the native C interface) which copy the image data into arrays that can be displayed by LabView. Amongst other things, they convert 16-bit image data into 8-bit image data.

The following modules are available:

- Fg_getLastImageArray8LVEx.vi,
- Fg_getLastImageArray16LVEx.vi,
- Fg_getLastImageArray24LVEx.vi
- Fg_getLastImageArray8BlockingLVEx.vi
- Fg_getLastImageArray16BlockingLVEx.vi,
- Fg_getLastImageArray24BlockingLVEx.vi

The modules are available for use in Blocking Mode and Non-Blocking Mode. (For more information about the blocking and the non-blocking mode, see section [Blocking and Non-Blocking Mode](#) in this document, and the [Silicon Software SDK documentation](#).)

The advantage of this functions is that the data can be used directly within LabVIEW. The downside of this functions is a longer processing time caused by the conversion processes (copy functions and, if necessary, pixel-by-pixel conversion functions).

You find examples for fetching image data using this approach in the following example diagrams:

- SiSoLabviewInterface_Example_Gray_Blocking.vi
- SiSoLabviewInterface_Example_Gray_Non_Blocking.vi
- SiSoLabviewInterface_Example_Gray16_Blocking.vi
- SiSoLabviewInterface_Example_RGB24_Non_Blocking.vi
- SiSoLabviewInterface_Example_Gray_GigE_Non_Blocking.vi

B) This approach allows to access the frame numbers (image numbers) and image pointers of the image buffer management. This approach is similar to the approach described in the Silicon Software SDK standard documentation and offers extended controlling options. For details, refer to the document [\doc\en\app_notes\san014_hiperf_image_aq.pdf](#) of the runtime documentation.

The following modules are available for Blocking and Non-Blocking Mode:

Acquisition Modus	Fetching Image Buffer Index	Fetching Number of Frame (Image)	Reading Image Data
Non-Blocking Mode	Fg_getImageLVEx.vi	Fg get Last Pic Number Blocking LV Ex.vi	Fg get Image Data Array LV Ex.vi
		Fg get Last Pic Number LV Ex.vi	Fg get Image Data Array24 LV Ex.vi
Blocking Mode	Fg_getImageLVEx.vi	No possibility	Fg get Image Data Array LV Ex.vi
			Fg get Image Data Array24 LV Ex.vi

You find examples for fetching image data using this approach in the following example diagrams:

- SiSoLabviewInterface_Example_Gray_Extension_Blocking.vi
- SiSoLabviewInterface_Example_Gray_Extension_GigE.vi
- SiSoLabviewInterface_Example_Gray_Extension_Non_Blocking.vi
- SiSoLabviewInterface_Example_RGB_Extension_Non_Blocking.vi

Image Data Array

With module *Fg_getLastImageArray8BlockingLVEx.vi* and *Fg_getLastImageArray8LVEx.vi*, image data is transferred from the specified DMA channel to RAM. The module outputs either 8-bit or 16-bit image data, depending on the image output format defined in parameter *FG_FORMAT*. Parameter *FG_FORMAT* has default value *FG_GRAY* (8-bit). When set to *FG_GRAY16*, module *Fg_getLastImageArray8BlockingLVEx.vi* outputs 16-bit image data.



Setting the Output Format

Information on parameterization issues you find in section [Parameterization](#) and in the [Silicon Software SDK documentation](#).

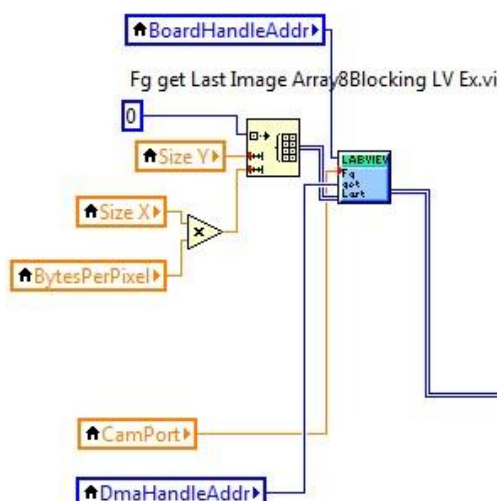


Figure 14: Block Diagram, Section “Image Data”

2D arrays of one-byte elements (I8, U8) are handed over as input to module *Fg_getLastImageArray8-BlockingLVEx.vi*. The size (length) of the array is calculated by the formula

Height * Width * BytesPerPixel.

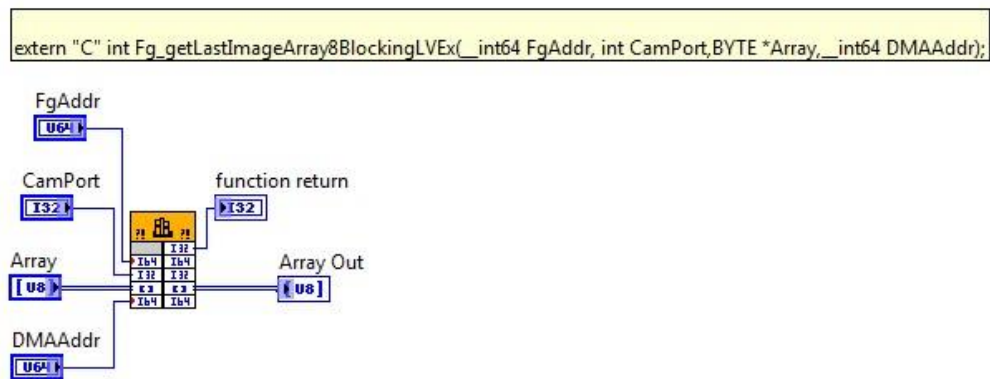


Figure 15: Definition of Module **Fg_getLastImageArray8BlockingLVEx.vi**

For processing 16-bit image data exclusively, the modules **Fg get Last Image Array16LV Ex.vi** and **Fg get Last Image Array16Blocking LV Ex.vi** are available.

Both modules can be used in the same way as **Fg_getLastImageArray8BlockingLVEx.vi**. Actually, both 16-bit modules are implemented the same way as the 8-bit modules, only the output image data array is converted into 16-bit format.

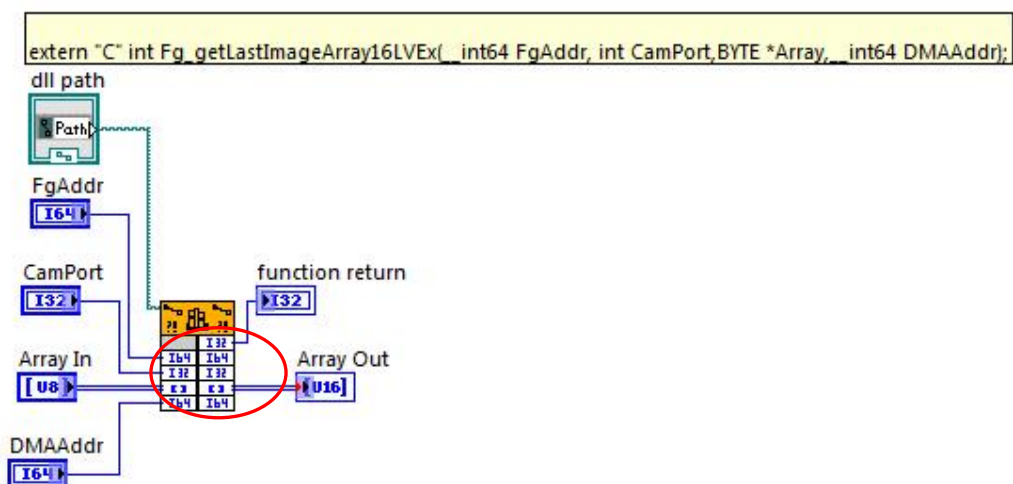


Figure 16: Definition of Module **Fg_getLastImageArray16LVEx.vi**

When working with 16-bit image acquisition:

Since module *Flatten Pixmap.vi* does not offer any 16-bit conversion functions, you need to convert your image data before using it. For converting 16-bit image data to an 8-bit array, the module *Fg_convertImageArray16to8LV.vi* is available.

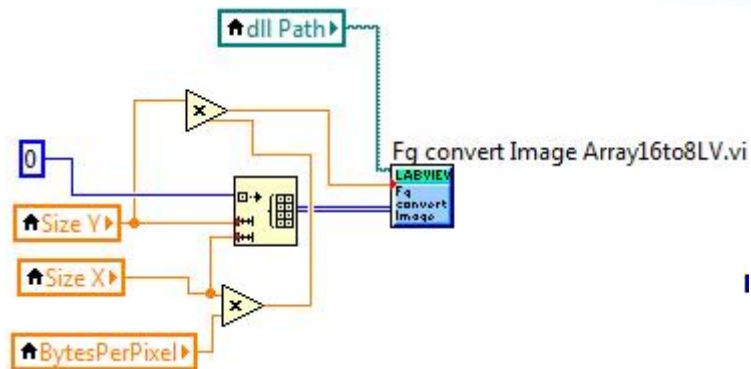


Figure 17: Block Diagram, Section *Fg_convertImageArray16to8LV.vi*

Input for *Fg_convertImageArray16to8LV.vi* are:

- The 16-bit data array that is to be converted
- The initialized target 8-bit array
- Size of original 16-bit array

```
extern "C" int Fg_convertImageArray16to8LV(int byteSize, BYTE* InputArray16, BYTE* OutputArray8);
```

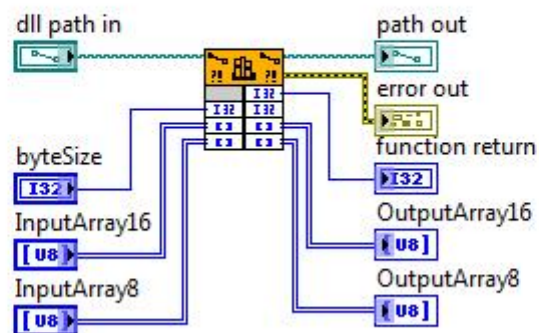


Figure 18: Definition of Module *Fg_convertImageArray16to8LV.vi*

Freeing up Resources

When the image acquisition is completed, all resources are freed up by calling the following modules in the given order:

- Fg_stopAcquireLV.vi
- Fg_FreeMemLVEx.vi
- Fg_FreeGrabberLV.vi

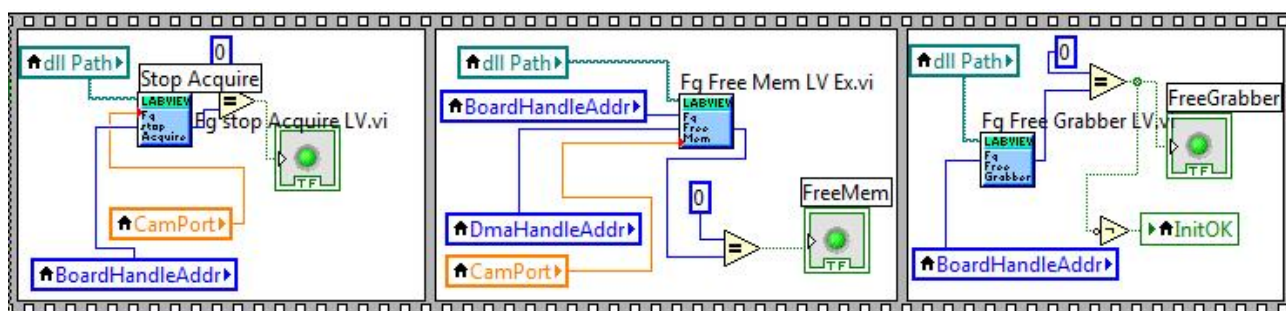


Figure 19: Block Diagram, Section "Freeing Up Resources"

```
extern "C" int Fg_stopAcquireLV(_int64 FgAddr, int CamPort);
```

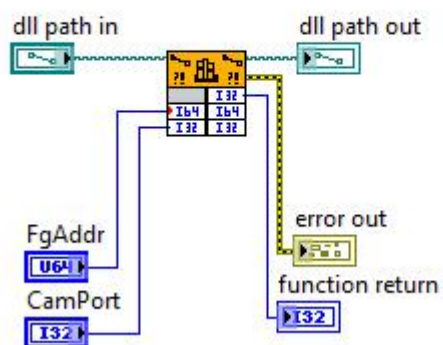


Figure 20: Definition of Module Fg_stopAcquireLV.vi

```
extern "C" int Fg_FreeMemLVEx(_int64 FgAddr, int CamPort, _int64 DMAAddr);
```

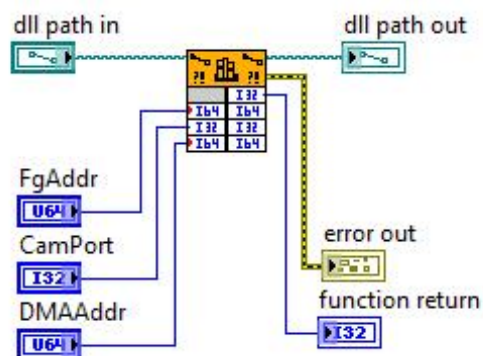


Figure 21: Definition of Module Fg_FreeMemLVEx.vi

```
extern "C" int Fg_FreeGrabberLV(_int64 FgAddr);
```

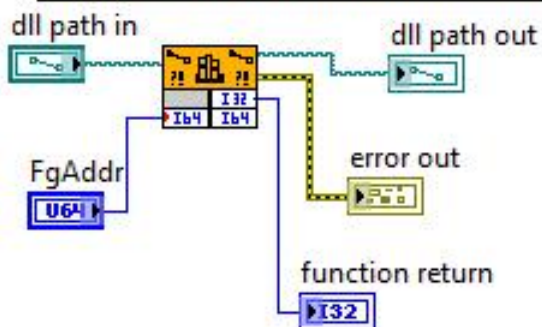



Figure 22: Definition of Module Fg_FreeGrabberLV.vi

3.2 Sample Program: SiSoLabviewInterface_Example_Color24.vi for Acquisition of Camera Link RGB Color Images

	Important
	<p>In this section, we describe the differences in programming the CL color example compared to the CL gray scale example above (section 0). Thus, we assume you had a look at the CL gray scale example (section 0) first.</p> <p>For using this sample application, you can use the configuration file provided in the installation package:</p> <ul style="list-style-type: none"> ▪ ExampeConfig_Color_24bit.mcf (for 24-bit color depth)

The logic behind the acquisition of CL 24-bit RGB color images is quite similar to the one used for image acquisition of CL gray scale images (see section 0). The program structure differs only in a few details from its gray scale pendant.

The most important difference is that instead of image transfer module *Fg_getLastImageArray8BlockingLVEx.vi* (see section Image Data Array), module **Fg_getLastImageArray24-BlockingLVEx.vi** is used to transfer the image data with the correct RGB format.

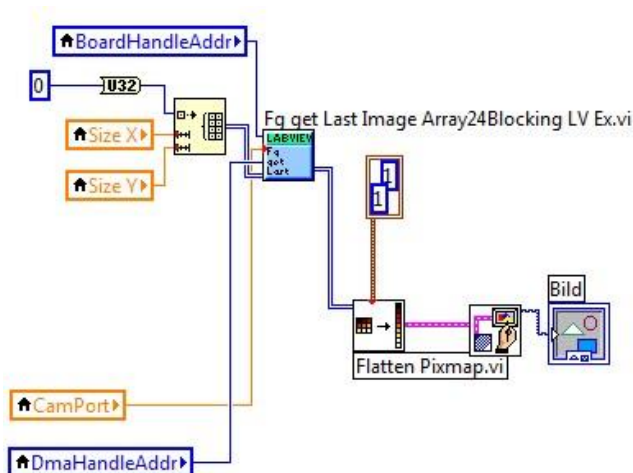


Figure 23: Block Diagram, Section “Image Display”

Here, the image data array is initialized as a 4-Bytes component.



Important

The output array out of module *Fg_getLastImageArray24BlockingLVEx.vi* has to be connected to the input port „24-bit pixmap“ of module *Flatten Pixmap.vi* (and not, as in the gray scale example above, to the „8-bit pixmap“ input port).

```
extern "C" __int64 Fg_getLastImageArray24BlockingLVEx(__int64 FgAddr, int CamPort, BYTE *Array, __int64 DMAAddr);
```

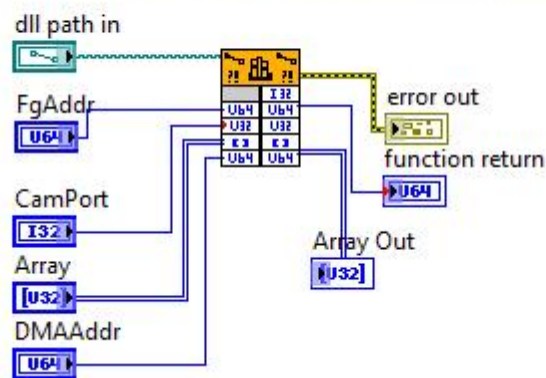


Figure 24: Definition of Module *Fg_getLastImageArray24BlockingLVEx.vi*

Fetching Image Data out of the LabView Interface into the LabView Application

The LabView Interface offers two ways to retrieve image data:

- A) Image Copying: This approach is optimized for direct use of image information within the LabView components. The LabView Programmer can fetch images in a very easy manner: The LabView Interface provides specific functions (not contained in the native C interface) which copy the image data into arrays that can be displayed by LabView. Amongst other things, they convert 16-bit image data into 8-bit image data.

The following modules are available:

- *Fg_getLastImageArray8LVEx.vi*
- *Fg_getLastImageArray16LVEx.vi*
- *Fg_getLastImageArray24LVEx.vi*

- Fg_getLastImageArray8BlockingLVEx.vi
- Fg_getLastImageArray16BlockingLVEx.vi
- Fg_getLastImageArray24BlockingLVEx.vi

The modules are available for use in Blocking Mode and Non-Blocking Mode. (For more information about the blocking and the non-blocking mode, see section [Blocking and Non-Blocking Mode](#) in this document, and the [Silicon Software SDK documentation](#).)

The advantage of this functions is that the data can be used directly within LabView. The downside of this functions is a longer processing time caused by the conversion processes (copy functions and, if necessary, pixel-by-pixel conversion functions).

You find examples for fetching image data using this approach in the following example diagrams:

- SiSoLabviewInterface_Example_Gray_Blocking.vi
- SiSoLabviewInterface_Example_Gray_Non_Blocking.vi
- SiSoLabviewInterface_Example_Gray16_Blocking.vi
- SiSoLabviewInterface_Example_RGB24_Non_Blocking.vi
- SiSoLabviewInterface_Example_Gray_GigE_Non_Blocking.vi

- B) Pointers and frame numbers: This approach allows to access the frame numbers (image numbers) and image pointers of the image buffer management. This approach is similar to the approach described in the Silicon Software SDK standard documentation and offers extended controlling options. For details, refer to the document [\doc\en\app_notes\san014_hiperf_image_aq.pdf](#) of the runtime documentation.

The following modules are available for Blocking and Non-Blocking Mode:

Acquisition Modus	Fetching Image Buffer Index	Fetching Number of Frame (Image)	Reading Image Data
Non-Blocking Mode	Fg_getImageLVEx.vi	Fg get Last Pic Number Blocking LV Ex.vi	Fg get Image Data Array LV Ex.vi
		Fg get Last Pic Number LV Ex.vi	Fg get Image Data Array24 LV Ex.vi


Acquisition Modus	Fetching Image Buffer Index	Fetching Number of Frame (Image)	Reading Image Data
Blocking Mode	Fg_getImageLVEx.vi	No possibility	Fg get Image Data Array LV Ex.vi
			Fg get Image Data Array24 LV Ex.vi

You find examples for fetching image data using this approach in the following example diagrams:

- SiSoLabviewInterface_Example_Gray_Extension_Blocking.vi
- SiSoLabviewInterface_Example_Gray_Extension_GigE.vi
- SiSoLabviewInterface_Example_Gray_Extension_Non_Blocking.vi
- SiSoLabviewInterface_Example_RGB_Extension_Non_Blocking.vi

3.3 Sample Program: *SiSoLabviewInterface_Example_Gray_GigE.vi* for image acquisition with a GigE Vision Frame Grabber

The sample program described in this section is designed for image acquisition with a GigE Vision frame grabber. It is able to output 8-bit as well as 16-bit images.

	<p>Important</p> <p>In this section, we describe the differences in programming the GigE example compared to the CL gray scale example above (section 3.1). Thus, we assume you had a look at the CL gray scale example (section 3.1) before.</p> <p>For using the sample application, you can use the configuration files provided in the installation package. Two files are available for this application:</p> <ul style="list-style-type: none"> • ExampeConfig_ Gray_GigE_8bit.mcf (for 8-bit color depth) • ExampeConfig_ Gray_GigE_16bit.mcf (for 16-bit color depth)
---	--

3.3.1 Graphical User Interface

The Graphical user interface and the way of usage are nearly the same as in the Camera Link example above (see section [3.1](#)). The only difference on the GUI is the static input field *Camera Count* which displays the number of cameras connected to the selected port.

The graphical user interface of this sample program looks as follows:

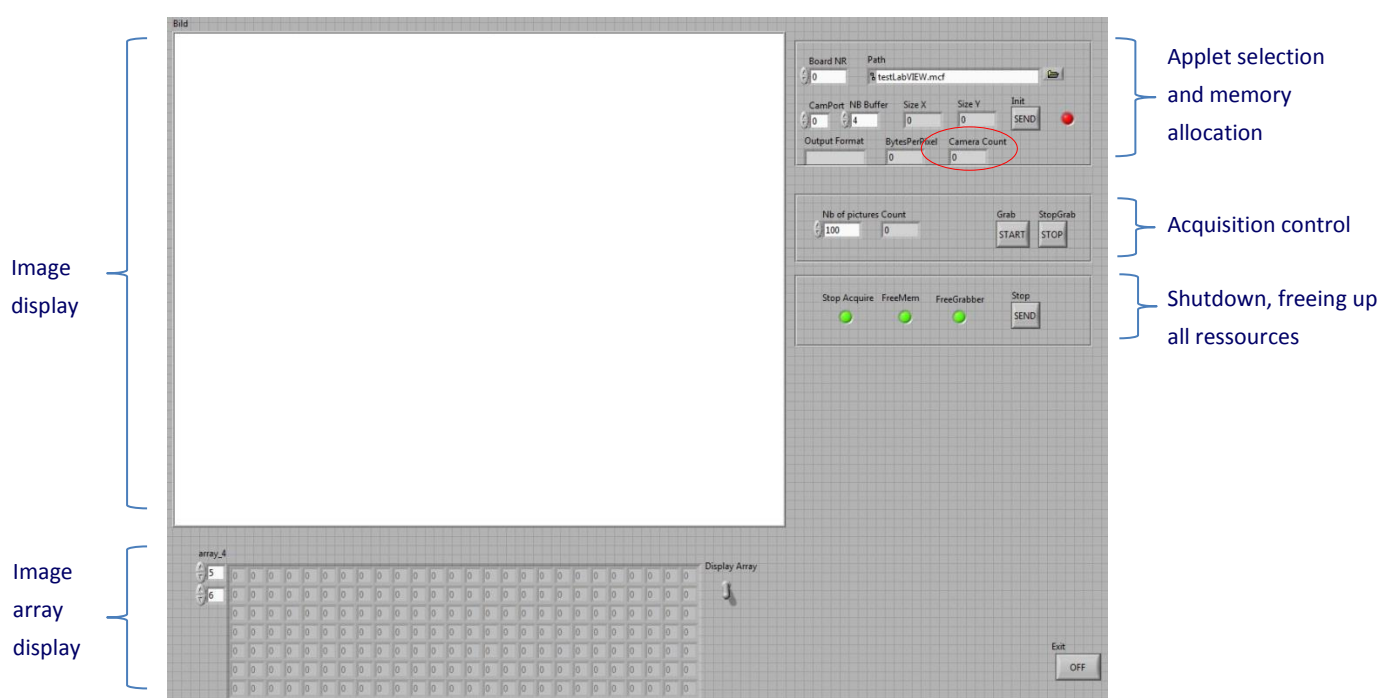


Figure 25: Graphical User Interface

A description of all GUI elements you find in section [3.1.1](#). End user instructions on how to use the sample program you find in section [3.1.2](#) (as there is no difference between CL and GigE sample programs).

3.3.2 Programming for GigE Vision Frame Grabbers

GigE Vision frame grabbers and cameras support the *GenICam* Standard. The *GenICam* protocol is defined by the „Gbe_“ modules of the *Silicon Software Interface Library for NI LabVIEW*.

GigE Vision modules can be used the same way as all the other modules.

Frame grabber initialization and memory allocation are carried out using the same scheme as the CL sample program described above (see section 3.1.3). The instance of the frame grabber and the pointer to the memory block are stored in the same local variables *BoardHandleAddr* and *DmaHandleAddr*.

However, the program varies as there are two additional variables defined:

- *GbeBoardHandle* (instance of *GenICam* protocol)
- *CameraHandle* (instance of camera)

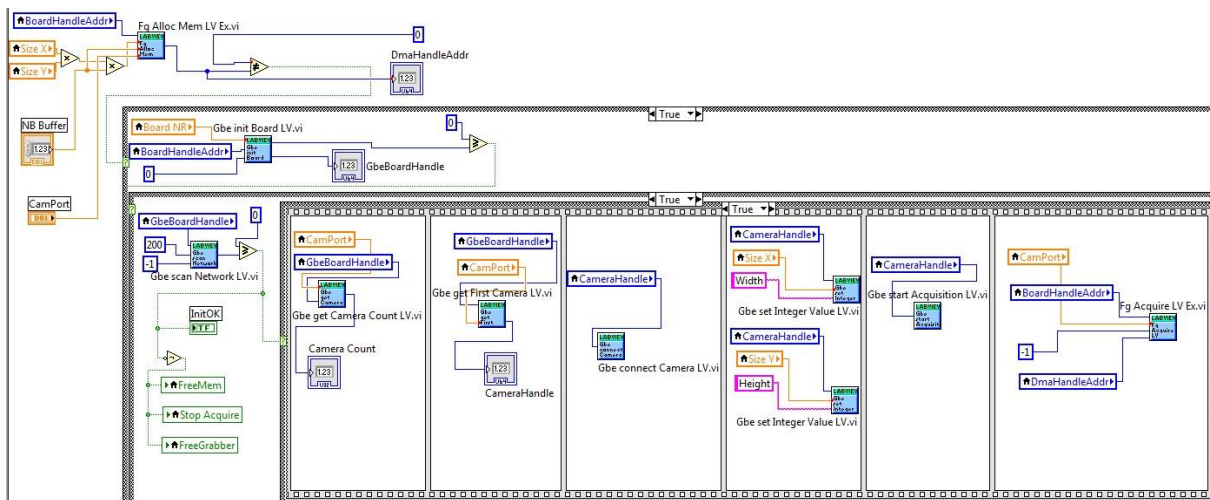


Figure 26: Section of Block Diagram

After memory allocation (via call of module *Fg_AllocMemLVEx.vi*), module *Gbe_initBoardLV.vi* has to be called to initialize the GenICam protocol.

Modules *Gbe_scanNetworkLV.vi* and *Gbe_getCameraCountLV.vi* are used to get the number of cameras on the target frame grabber. Then, via calling module *Gbe_getFirstCameraLV.vi*, the camera instance is retrieved.

Module *Gbe_connectCameraLV.vi* is used to connect the camera.

Since you are using the GenICam protocol, you should set parameters explicitly. In this sample program, module `Gbe_setIntegerValueLV.vi` has been used to set *Width* and *Height*. For parameterization of other data types, according modules are available (`Gbe_set/getFloatValueLV.vi`, `Gbe_set/getStringValueLV.vi`, etc.)



Important

These “Gbe_ ...” parameterization modules differ from the usual parameterization modules in so far as the value of the variable *Parameter* is a string (e.g., „Width“ and „Height“) and not the parameter ID.

To start the acquisition, both modules

- `Gbe_startAcquisitionLV.vi`, and
- `Fg_AcquireLVEx.vi`

need to be called.



Call `Gbe_startAcquisitionLV.vi` first

It is important that you call module `Gbe_startAcquisitionLV.vi` first and module `Fg_AcquireLVEx.vi` only in a second step. Other ways, a loss of image data might occur.

3.4 Sample Program:

SiSoLabviewInterface_Example_Gray_Ext_NonBlo_UserMem_TrueP ointer.vi

The Silicon Software runtime offers two ways to manage the memory, one called “Grabber-Allocated Memory”(standard memory management), and another called “User-Allocated Memory”. This example demonstrates how to use the “User-Allocated Memory”.

For more information about the memory management in the Silicon Software Runtime, refer to our runtime documentation: Software Development Kit -> SDK Manual, section [3 Memory Management](#).

As you already know, standard memory management can be implemented via the modules *Fg_AllocMemLVEx.vi* and *Fg_FreeMemLVEx.vi*.

A new feature of the LabVIEW Interface version 2.1 is the additional option to implement a user-controlled memory management. As always, the memory management consists of two parts: Allocating the memory, and freeing the memory.

Memory Allocation

For user-controlled memory allocation on the PC, two modules need to be used:

- **Fg Alloc Mem Head LV.vi** (for defining the starting address of the memory block)
- **Fg Add Mem LV.vi** (for defining the individual frame buffers)

Start with **Fg Alloc Mem Head LV.vi** to define the address of the new memory block.

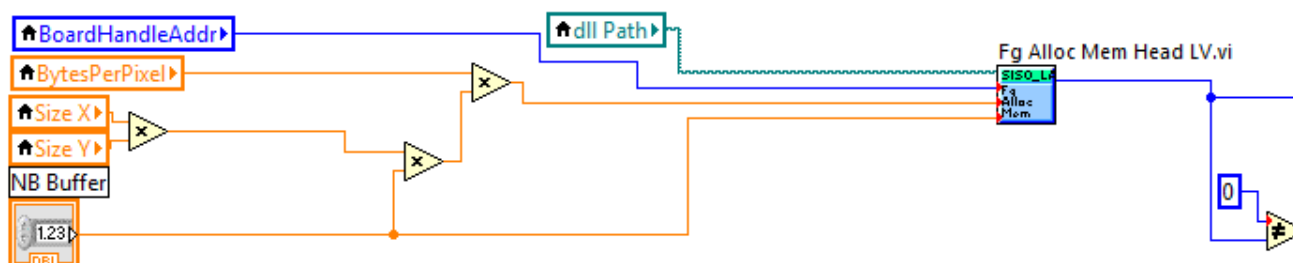


Figure 27: Block Diagram, Section “User-Controlled Memory Allocation, calling Fg Alloc Mem Head LV.vi”

```
extern "C" _int64 Fg_AllocMemHeadLV(_int64 FgAddr, int totalBufSize, int nr_of_buffer);
```

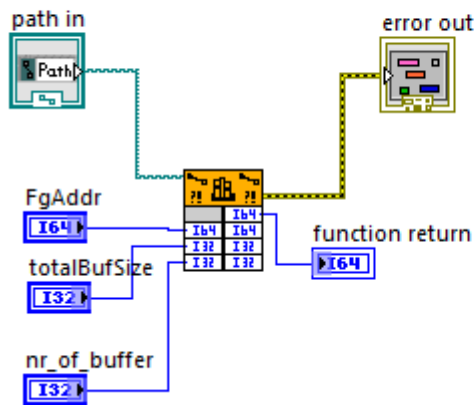


Figure 28: Definition of Module Fg Alloc Mem Head LV.vi

Return value is the address of the allocated DMA memory block on the PC.

After the starting address of the memory block is defined via module *Fg Alloc Mem Head LV.vi*, the individual frame buffers have to be allocated. To do so, the module *Fg Add Mem LV.vi* is available.

The call of module *Fg Add Mem LV.vi* has to be repeated in a loop until the required number of frame buffers is allocated.

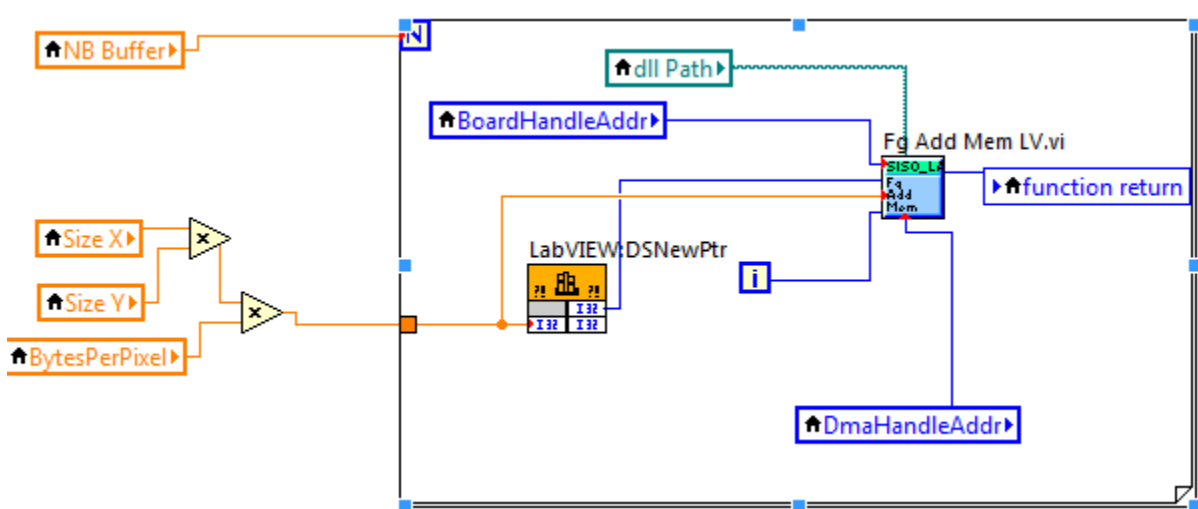


Figure 29: Block Diagram, Section "User-Controlled Frame Buffer Allocation"

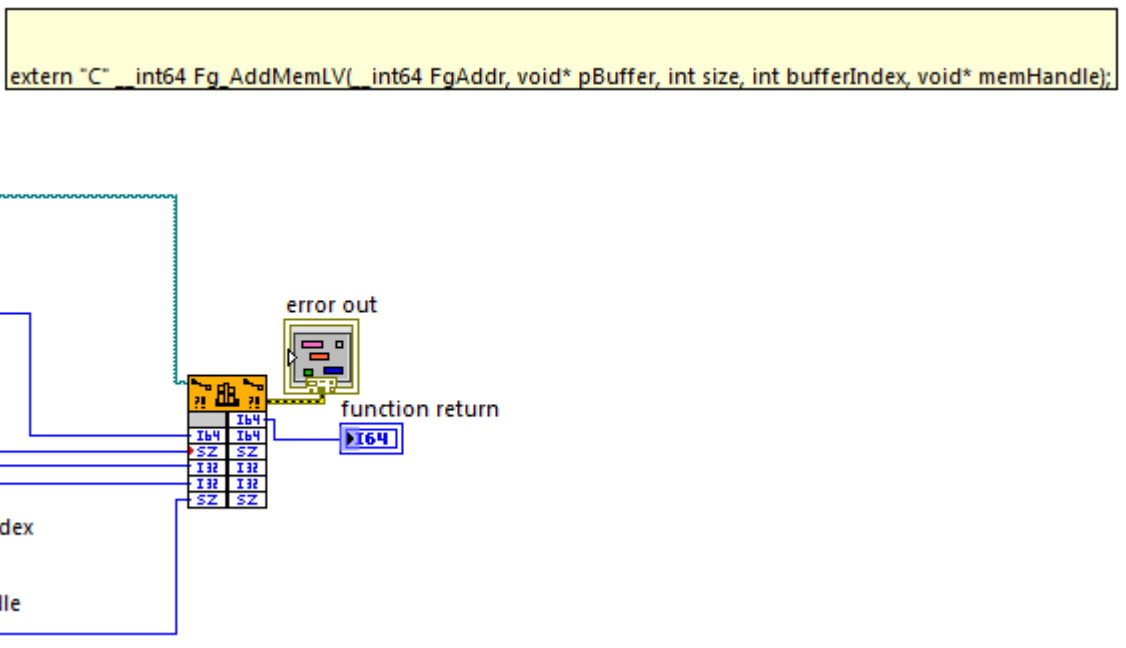



Figure 30: Definition of Module Fg Add Mem LV.vi

	<p>Parameter bufferSize</p> <p>Parameter bufferIndex is zero-based.</p>
---	---

The Labview Code Interface Node(CIN)

Module **LabVIEWWL:DSNewPtr** is the Labview Code Interface Node (CIN) function. The CIN function is not part of the Silicon Software LabView Interface, but a standard library of LabView. The module allows memory allocation for frame buffers, like, e.g., **Malloc** in C/C++.

Module LabVIEWWL:DSNewPtr can be imported via module Call Library Function Node.

These are the required settings for **LabVIEWL:DSNewPtr**:

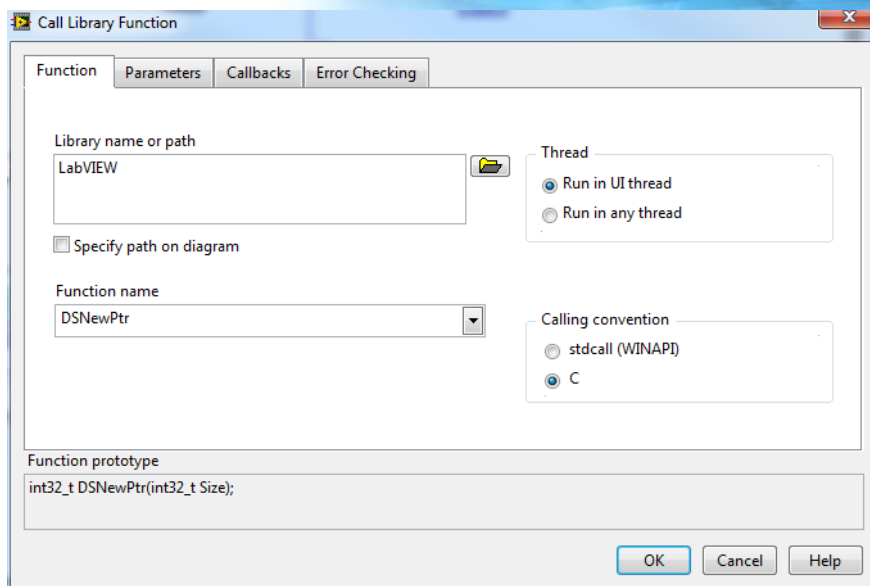


Figure 31: Settings for LabVIEWL:DSNewPtr

Parameters of LabVIEWL:DSNewPtr:

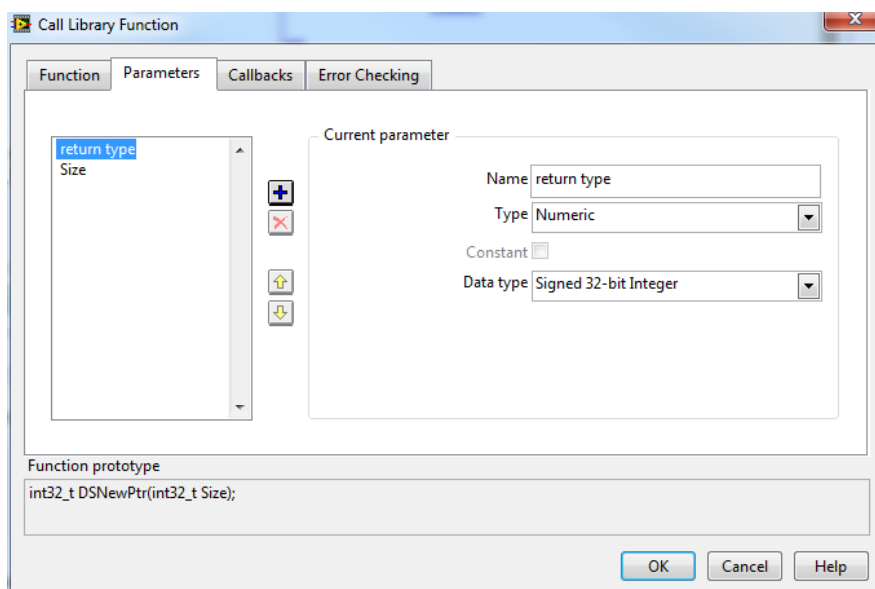


Image Acquisition with User-Controlled Memory Management

When working with user-controlled memory management, module **Fg get Image Ptr LV Ex.vi** has to be called for image acquisition.

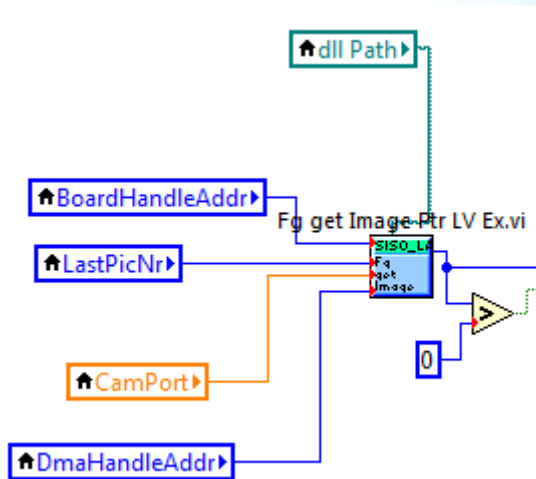


Figure 32: Block Diagram, Section “Image Acquisition with User-Controlled Memory Management”

This module outputs the address or a pointer that points to the image data.

Therefore, the pointer has to be de-referenced.

For de-referencing, the module *LabVIEW:MoveBlock* can be used. *LabVIEW:MoveBlock* is also part of the Labview Code Interface Node (CIN).



Alternative Ways to De-Reference the Pointer

If you wish to de-reference the pointer within LabView in another way, you can do so. It is not obligatory to use exclusively module *LabVIEW:MoveBlock* for de-referencing.

These are the required settings for LabVIEW:MoveBlock:

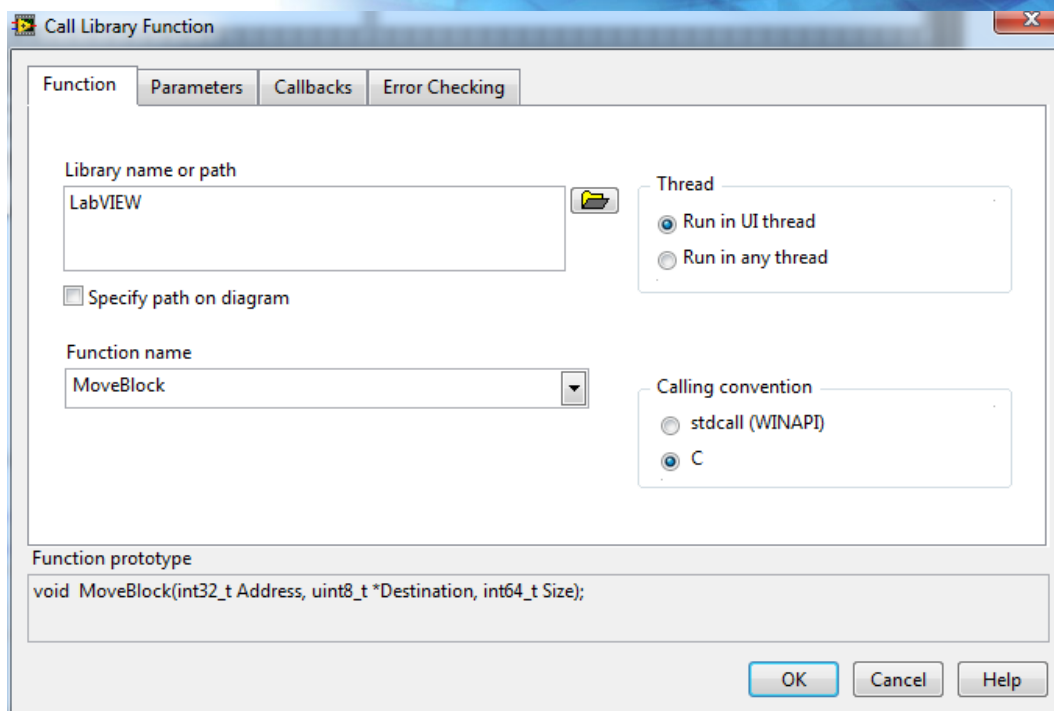


Figure 33: Settings for Module LabVIEW:MoveBlock

Parameters of LabVIEW:MoveBlock:

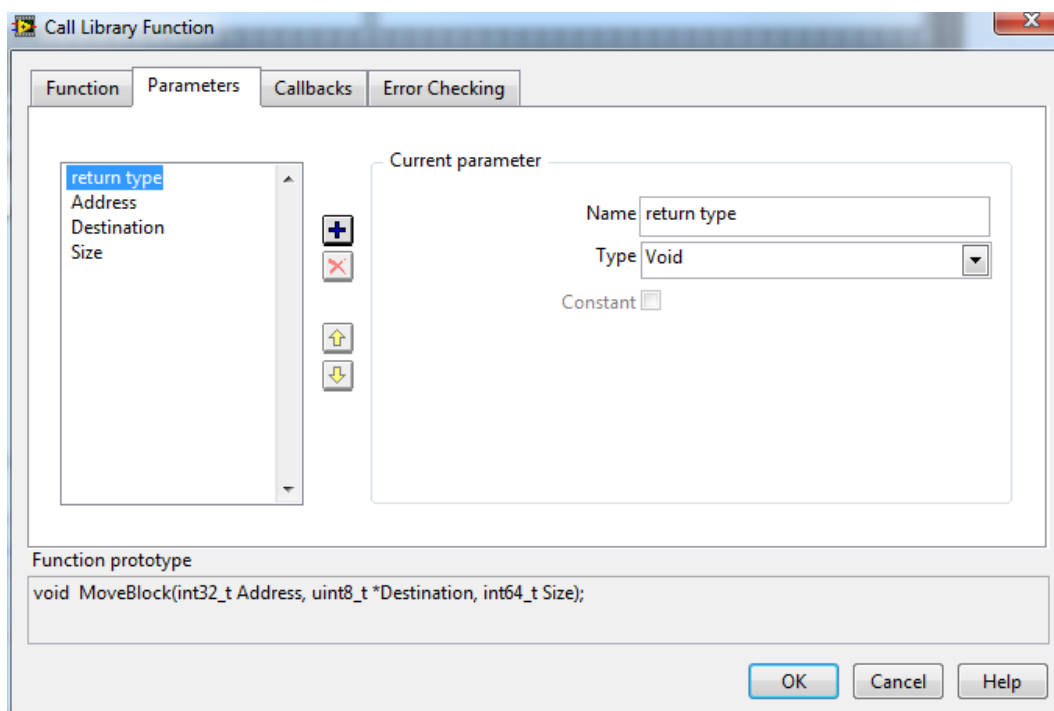


Figure 34: Parameters of module LabVIEW:MoveBlock

Freeing the Memory with User-Controlled Memory Management

The first step is to free the memory space allocated for the individual frame buffers. To free the frame buffer memory space, the module **Fg Del Mem LV.vi** has to be called.



Delete all Frame Buffer Memories first

It is of utmost importance to free all frame buffer memories before you proceed.

Call **Fg Del Mem LV.vi** in a loop (like **Fg Add Mem LV Ex.vi**) until all allocated frame buffers of the memory block are freed.

```
extern "C" __int64 Fg_DelMemLV( __int64 FgAddr, void* memHandle, int bufferSize);
```

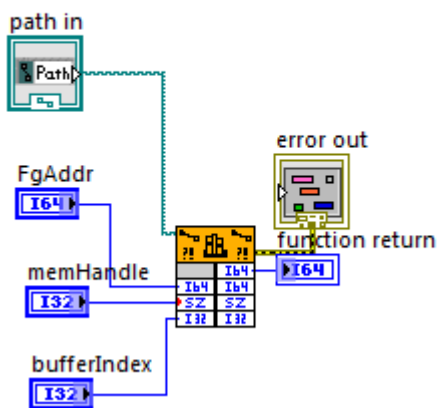


Figure 35: Block Diagram, Section "Calling Fg Free Mem LV.vi"

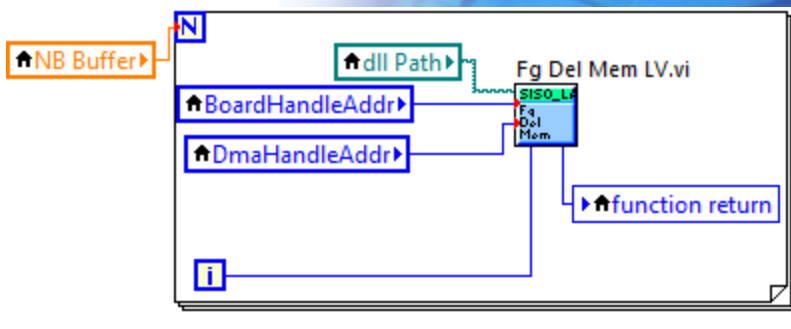


Figure 36: Definition of Module Fg Del Mem LV.vi

After all frame buffers of the memory block are deleted, you can proceed and free the address of the memory block.



Important

Make sure all frame buffers of the memory block are deleted before freeing the address of the memory block. Otherwise, the frame buffer space can not be freed, since it can not be found without the address of the memory block.

To free the memory/memory address, the module **Fg Free Mem Head LV.vi** is available.

```
extern "C" int Fg_FreeMemHeadLV(_int64 FgAddr, void* pMemHead);
```

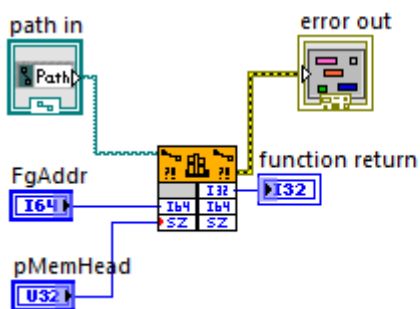


Figure 37: Block Diagram, Section "Freeing Memory Block Address using module Fg Free Mem Head LV.vi "

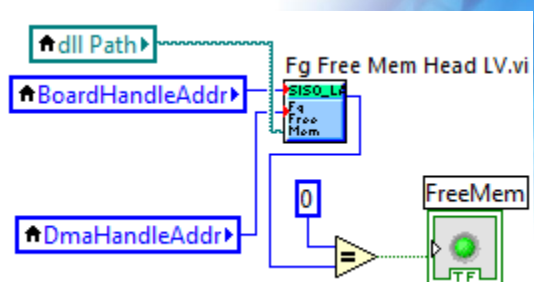


Figure 38: Definition of Module Fg Free Mem Head LV.vi

Contact Details

SILICONSOFTWARE GmbH

Steubenstrasse 46

D - 68163 Mannheim, Germany

Phone: +49(0)621.789 507 39

Fax: +49(0)621.789 507 10

Email: vertrieb@silicon-software.de

Web: www.silicon-software.info

SILICONSOFTWARE Inc.

1 Tara Boulevard, Suite 200

Nashua, NH 03062, USA

Phone: +1 603 324 7172

Fax: +1 603 324 7101

Email: info@silicon-software.com

Web: www.silicon-software.info

Disclaimer

While every precaution has been taken in the preparation of this manual, Silicon Software GmbH assumes no responsibility for errors or omissions. Silicon Software GmbH reserves the right to change the specification of the product described within this manual and the manual itself at any time without notice and without obligation of Silicon Software GmbH to notify any person of such revisions or changes.

Trademarks

All trademarks and registered trademarks are the property of their respective owners.

Copyright Note

© Copyright 2000–2014 Silicon Software GmbH. All rights reserved. This document may not in whole or in part, be reproduced, transmitted, transcribed, stored in any electronic medium or machine readable form, or translated into any language or computer language without the prior written consent of Silicon Software GmbH.