sensicam

dicam pro

SDK-Description

This document describes the API interface to the sensicam and dicam pro camera series.

Table of Contents

1 General

Important:
The PCI520 Board is not supported from this SDK.

Camera and PCI-Board control are managed on two levels, represented by the library **sen_cam.dll** and the driver **pco525.sys**.

This description includes in detail all functions of the upper level of the SDK (sen_cam.dll) and some hints how to use it.

Most of the SDK functions are declared as *int name()*. The returned integer value is one of the SDK errorcode values. A list of all errorcodes can be found in the header file pco_err.h a short description of each errorcode in the header file pco_errt.h. The OUT part in the description of each function does not include this return value.

Hardware and Library
The PCI525-Board has two memory banks on board to hold maximum two images of the camera. The image data read out from CCD will be sent to one of these memory banks. The transfer of the data to the main memory of the PC is done by Master-DMA transfer (without interaction of the PC-CPU) from one of the banks. Simultaneously a readout from the camera to the other bank can occur.
In the SDK you will find **Memory Control Functions** to allocate and free image buffers, select one or more buffers for grabbing images (set the buffers in a queue). These Buffers can be controlled by using the returned buffer number. Additional there exist functions to directly write in an application allocated and controlled memory area.

The **General Control Functions** are used to open, close and setup the driver and the board. The driver can manage up to eight boards. So if more than one PCI525 Board is installed in the PC, the driver creates a unique handle for the selected board, if opened the first time. This unique handle must be used for all subsequent operations with this board.
The driver refuses the connection to a given board if the board was opened before from another process.

The **Camera Control Functions** are used to control the connected camera and to get status information from the board and the camera. The timing and readout operations of the camera are controlled from a COC (CameraOperationCode), which is loaded into a small fifo-memory onboard. The COC is created and downloaded from Function SET_COC*()*. To start the camera RUN_COC*()* must be called, to stop it STOP_*COC()*. After RUN_COC is called the camera starts the current COC and the image data is transmitted to one of the memory banks on the board. A call to STOP_COC*()* cancels the current COC and erases the buffers on board. The camera must be stopped before a new COC can be set from Function SET_COC*()*.
Status information from the PCI-Board and the camera can be readout at any time. For fast access to different parameters there are three functions, which will fill different stuctures defined in the header file “sencam_def.h”. In this file you will also find further useful definitions.

The SDK is ***not thread save*** in relation to simultaneous operations such as setting the camera and grabbing images on each camera. Two or more cameras can work in independent threads. Thus it is not possible to setup two or more threads getting images with different settings and sizes. However threading is possible in case the developer takes care for correct thread synchronization, e.g. one thread changes the settings and a second one grabs the images. In this case the second thread has to stop grabbing till the first one has changed the settings. In principle the order of commands shown in the typical implementation must be met.

Migrating from an older SDK:
First step will be to replace the INIT-functions, delete all SET_BOARD and GET_BOARD functions and include the Handle Parameter in all other function and compile with the new library.
Second step might be to change the image grabbing to the new buffer handling functions.

2 Library Functions

2.1 General Control Functions

2.1.1 Initializing and close

INITBOARD

int **INITBOARD** (int board, HANDLE *hdriver)

This command initializes the PCI-Controller-Boards 0…7. The communication with the board through the driver is tested and the necessary ressources are claimed. This function does not fail, if no camera is connected or the camera is not switched on.
When calling this function with parameter *hdriver set to NULL, the function opens the driver and returns in *hdriver the file handle of the driver for the selected board.
The file handle of the driver is needed in any library function to communicate with a specific board.
If reinitialization is needed during the process flow, call this function with the filehandle according to the board number. Board numbers start from zero. If only one board is installed board number must be 0.

IN

board		number of the PCI-Controller-Board
	=	0…7
hdriver		pointer to filehandle
*hdriver	=	NULL the driver will be opened and the board initialized
*hdriver	=	filehandle of opened driver the board will be initialized

OUT

*hdriver	=	filehandle of the opened driver

CLOSEBOARD

int **CLOSEBOARD** (HANDLE *hdriver)

This command generates a reset of the PCI-Controller-Board and closes the driver. If the function returns without error, *hdriver is also set to NULL.

IN

hdriver		pointer to filehandle
*hdriver	=	filehandle of opened driver

OUT

*hdriver	=	NULL

SETUP_CAMERA

int **SETUP_CAMERA** (HANDLE hdriver)

The board and camera parameters are set to the default values. Then the connection to the camera is tested and the parameters are set according to the found camera. If no camera is found the function returns an errorvalue. This function must be called if a new camera is connected to the board and should be called again after the camera is switched off and on.

IN

hdriver	=	filehandle of opened driver

2.2 Camera Control Functions

2.2.1 Start and stop camera COC

RUN_COC

int **RUN_COC** (HANDLE hdriver, int mode)

Processing of the COC is started during which the COC program describes the read out procedure for the CCD as well as the delay and exposure times for capturing an image.
In continuous mode the COC program is started repeatedly until the STOP_COC command is given.
This call also enables the driver to transfer data from the memory on board to the PC-main memory

IN

hdriver	=	filehandle of opened driver
mode	=	0 continuous trigger
	=	4 single trigger

When choosing 'continuous trigger' mode immediately after the first call and then after each exposure, a new exposure is automatically started (restart of the COC program) as long as one or both buffers of the PCI interface board are empty. The sequence speed depends on the selected delay and exposure times and on the CCD read out time.

When choosing 'single trigger mode' one single exposure is started (restart of the COC program).
Make sure that the STOP_COC command terminates an eventually running COC process before the camera has to record a new image.
The STOP_COC command also clears the PCI Interface Board buffer. Consequently, when calling the RUN_COC command afterwards, an image can be written into the now empty buffer space.
If the buffers should already be occupied by other images, RUN_COC does not write a new image into the buffer. Only after any image read is done or CLEAR_BOARD_BUFFER or STOP_COC command is processed another exposure can be started.
Single trigger mode should not be used while running simultaneous mode, otherwise this causes the camera and library to make some additional processing which will decrease performance.

Example Recording an image with a non-empty buffer

```
STOP_COC                cleans the memory
RUN_COC                 starts an exposure
ADD_BUFFER_TO_LIST      loads an image from the PCI buffer into the
                        PC memory
repeat
  GET_BUFFER_STATUS
until                   buffer is done

STOP_COC                terminates the exposure
```

STOP_COC

int **STOP_COC** (HANDLE hdriver,int mode)

This function interrupts an active exposure (execution of the COC program). It can also be used as a break option, e.g. in the case of very long delay and exposure times.
Additionally, the PCI Interface Board Buffers are erased and the ability to transfer data is disabled.
If the camera is running when this function is called, a waiting loop is started after stop, to clear the CCD-Chip register contents.

IN

hdriver	=	filehandle of opened driver
Mode	=	0

2.2.2 Build COC for sensicam

The following sensicam camera types are supported with this SDK:
'sensicam long exposure', 'sensicam qe',
'sensicam qe standard', 'sensicam qe double shutter',
'sensicam fast shutter', 'sensicam double shutter',
'sensicam em' and 'sensicam uv'.

SET_COC

int **SET_COC**	(HANDLE hdriver int mode, int trig, int roixmin, int roixmax, int roiymin, int roiymax, int hbin, int vbin, char *table)

This function generates a COC (**C**amera **O**peration **C**ode) which is loaded into the program memory of the camera. All parameters are checked to ensure that a valid set is generated. If any of the parameters is wrong the function returns error `PCO_ERROR_WRONGVALUE`. To get a valid set of parameters TEST_COC can be used.
Camera type specific settings can only be made as described below.

IN

For exact description of all parameters see notes below

hdriver	=	filehandle of opened driver
mode		operation mode
trig		trigger and start mode (auto, hw, …)
roixmin		start of horizontal ROI (Region of Interest)
roixmax		end of horizontal ROI
roiymin		start of vertical ROI
roiymax		end of vertical ROI
hbin		horizontal binning
vbin		vertical binning
table		pointer to zero terminated ASCII string with values for delay and exposure times

mode

Set the camera type, operation mode and analog gain. It is a combination of the following parameters
(**type**+(**gain***256)+(**submode***65536)) respectively
((**type**&0xFF) | ((**gain**&0xFF)*<<8) | ((**submode**&0xFF)<<16))
See also the defines in the cam_types.h file

type Long Exposure:
The type 'long exposure' is for the use with the 'sensicam long exposure', all 'sensicam qe', the 'sensicam em' and 'sensicam uv' versions of the sensicam camera series.

type	0	long exposure (M_LONG)	
gain	0	normal analog gain	
	1	extended analog gain	
	3	Low Light Mode[1]	
submode	0	sequential, busy out	(NORMALLONG)
	1	simultaneous, busy out	(VIDEO)
	2	sequential, expos out[2]	(MECHSHUT)
	3	simultaneous, expos out*	(MECHSHUTV)
	8	fast qe, busy out[3]	(QE_FAST)
	9	double qe, busy out[4]	(QE_DOUBLE)
	20	fast em, busy out[5]	(EM_FAST)
	21	simult. fast em, busy out[5]	(EM_FAST_V)

1) only for cameras sensicam qe, sensicam qe standard and sensicam qe double shutter
2) the TRIG IN BNC plug at the rear of the PCI-Board is used as an output. The Signal on this output follows the exposure time in default mode. Setting additional values in the exposure time string alters the output signal. For exact description contact PCO support.
3) only for camera sensicam qe standard and sensicam qe double shutter
4) only for camera sensicam qe double shutter
5) only for camera sensicam em , sensicam uv

submode NORMALLONG:
In the 'Sequential' mode delay, exposure and CCD readout are done sequentially in chronological order. All possible trigger combinations are allowed.

VIDEO:
The mode 'Simultaneous' does not allow a delay setting. Exposure and CCD readout are done simultaneously. The longer duration of either exposure time or readout time determines the maximum achievable repetition rate. For exposure times, which are longer as twice readout time using the mode NORMALLONG is recommended. The only allowed trigger combinations are auto start and sequence start (trig = 0x000, trig = 0x100, trig = 0x200).

MECHSHUT:
BNC-Plug at the PCI-Board is used as an output to monitor exposure time. No trigger settings are possible. Delay, exposure and CCD readout are done sequentially

MECHSHUTV:
BNC-Plug at the PCI-Board is used as an output to monitor exposure time. No trigger settings are possible. Exposure and CCD readout are done simultaneously

QE_FAST:
Sequential mode with possibility to set short exposure times. All trigger combinations are allowed.

QE_DOUBLE:
Two images are taken in a sequence which is started by the external trigger input 'TRIG' on the PCI Interface Board. This sequence can be started by a rising edge (trig = 0x001) or by a falling edge (trig = 0x002). The two exposed images are linked together to one data set (one image with double height is transferred). The inter-framing time between the two images has to be at least 500ns.

EM_FAST:
Sequential mode with possibility to set short exposure times. All trigger combinations are allowed.

EM_FAST_V:
Simultaneous mode with possibility to set short exposure times. All trigger combinations are allowed.

type Fast Shutter:
The type Fast Shutter is for the use with the 'sensicam fast shutter' version only.

type	1	fast shutter (M_FAST)	
gain	0	normal analog gain	
	1	extended analog gain	
submode	0	standard	(NORMALFAST)
	5	cycle	(CYCLE)

submode NORMALFAST:
Single and multiple exposures with delay and exposure times between 100ns and 1ms can be done. On a single trigger the complete time table is started. All possible trigger combinations are allowed.

CYCLE:
In this mode, every exposure is synchronized with an external trigger event. Only external trigger modes are allowed. Every delay-, exposure time pair can be repeated up to 1000 times. Each event must be released by its own trigger pulse. This means if 20 exposure times are set, 20 trigger events are needed to record **one** image.

type

Double Shutter:
The type Double Shutter is for the use with the 'sensicam double shutter' version only.

type	1	Double Shutter (M_FAST)	
gain	0	normal analog gain	
	1	extended analog gain	
submode	0	standard	(NORMALFAST)
	1	double 200ns	(DOUBLE)
	2	double 1µs	(DOUBLEL)
	5	cycle	(CYCLE)

To be compatible with older versions,
'Double Shutter 200ns' is also type = 2 and
'Double Shutter 1µs' is also type = 3.

submode

NORMALFAST:
See Fast Shutter mode.

DOUBLE:
Two images are taken in a sequence which is started by the external trigger input 'TRIG' on the PCI Interface Board. This sequence can be started by an rising edge (trig = 0x001) or by a falling edge (trig = 0x002). The two exposed images are linked together to one data set (one image with double height is transferred). The inter-framing time between the two images has to be at least 200ns. This short interval time between the two images happens at the expense of reduced anti-blooming.

DOUBLEL:
Two images are taken in a sequence which is started by the external trigger input 'TRIG' on the PCI Interface Board. This sequence can be started by an rising edge (trig = 0x001) or by a falling edge (trig = 0x002). The two exposed images are linked together to one data set (one image with double height is transferred). The inter-framing time between the two images has to be at least 1000ns. This submode offers a widely increased anti-blooming compared to submode DOUBLE.

CYCLE
See Fast Shutter mode.

trig

Set the camera trigger mode. When "trig" is set to any external trigger mode, delay and exposure times are started with a TTL- trigger signal applied at the external trigger input "TRIG" of the PCI Interface Board.

For 'sensicam long exposure', all 'sensicam qe' , 'sensicam em' and 'sensicam uv' version

trig	0x000	auto start, auto frame
	0x001	auto start, frame with external rising edge
	0x002	auto start, frame with external falling edge
	0x100	sequence start with external rising edge
	0x200	sequence start with external falling edge
	0x101	sequence + frame start with external rising edge [1]
	0x202	sequence + frame start with external falling edge [1]

All other combinations are forbidden

There are three different modes to trigger the camera:

- **auto start (trig = 0x000, 0x001, 0x002)**
  Each frame will be triggered, either by an internal software trigger or by an external trigger signal.

- **sequence start (trig = 0x100, 0x200)**
  An external trigger signal starts a complete sequence.

- **sequence + frame start (trig = 0x101, 0x202H)**
  The first external trigger starts a sequence, the second trigger starts the first exposure (frame). The following exposures must be triggered, too.

For 'sensicam fast shutter' and 'sensicam double shutter' version

trig	0x000	no external synchronization
	0x001	external falling edge
	0x002	external rising edge

All other combinations are forbidden.

roixmin, roixmax

Set the start and end values for the horizontal region of interest (ROI). One unit is 32 pixels. This setting affects the readout of the CCD-Chip. Less data is transferred, but readout time is not affected.

roixmin	start value of horizontal ROI
roixmax	end value of horizontal ROI
	range 1 ... 20 for CCD chip type 640 x 480
	range 1 ... 40 for CCD chip type 1280 x 1024
	range 1 ... 43 for CCD chip type 1376 x 1040
	range 1 ... 32 for CCD chip type 1024 x 1002

roiymin, roiymax

Set the start and end value for the vertical region of interest (ROI). One unit is 32 pixels. The last segment may be smaller, if CCD width is not a multiple of 32. This setting affects the readout of the CCD-Chip. Less data is transferred and the readout time is decreased.

roiy1	start value of vertical ROI
roiy2	end value of vertical ROI
	range 1 ... 15 for CCD chip type 640 x 480
	range 1 ... 32 f or CCD chip type 1280 x 1024
	range 1 ... 33 for CCD chip type 1376 x 1040
	range 1 ... 32 for CCD chip type 1024 x 1002

Thus the smallest ROI is 32 pixels in square.
To get for example the upper right corner 32*32 pixel the ROI settings should be roixmin=1, roixmax=1, roiymin=1, roiymax=1.
In the case of any ‘Double Shutter’ mode, the ROI is set for the two half images which are then transferred as one data set of double height.

hbin

Set the horizontal binning. This setting affects the readout of the CCD-Chip. Less data is transferred, but readout time is not affected.

hbin	horizontal binning
	1 no binning
	selectable values
	1, 2, 4, 8

vbin

Set the vertical binning. The maximal vertical binning setting depends on the selected vertical ROI. This setting affects the readout of the CCD-Chip. Less data is transferred and the readout time is decreased.

vbin	vertical binning 1 no binning selectable values SVGA 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 selectable values VGA 1, 2, 4, 8, 15, 16, 30, 32, 60, 64, 120, 128, 240, 256, 480 selectable values 'sensicam qe' 1, 2, 4, 8, 16 selectable values 'sensicam em', 'sensicam uv' 1, 2, 4, 8, 16 , 32, 64, 128

table

Set the delay and the exposure times. The parameter is a pointer to a zero terminated ASCII string. The time-values are separated by comma (,) or space ( ). The array is concluded by the sequence "-1,-1", so that variable key lengths can be handed over. The characters 'CR' (13D) and 'LF' (10D) may be used to structure the input string.

table	pointer to string array

a) Long Exposure (NORMALLONG)

string array

DELAY , EXPOS_WIDTH,
-1 , -1

The delay and exposure time is in ms with a range from 0 to 1,000,000 for DELAY and from 1 to 1,000,000 for EXPOS_WIDTH, in steps of 1 ms. Exactly one pair of values is allowed.

b) Long Exposure (VIDEO)

string array

0 , EXPOS_WIDTH,
-1 , -1

The exposure time is in ms with a range from 1 to 1,000,000 for EXPOS_WIDTH, in steps of 1 ms. Exactly one pair of values is allowed.

c) Long Exposure (MECHSHUT)

string array

DELAY ,	EXPOS_WIDTH,
AV ,	AV,
START ,	STOP,
-1 ,	-1

The delay and exposure time is in ms with a range from 0 to 1,000,000 for DELAY and from 1 to 1,000,000 for EXPOS_WIDTH, in steps of 1 ms.
If both AV values are '-1', the default values are used for start and stop time of the output Signal
If both AV values are '0' the start and stop time of the output signal is calculated according to the START and STOP values given.
Range of START is DELAY*(-1) to EXPOS+STOP-1.
Negative values are setting the start time of output signal before exposure time start, positive values after exposure time start.
Range of STOP is (DELAY*(-1))+1 to 1,000,000
Negative values are setting the stop time of output signal before exposure time ends, positive values after exposure time end.

d) Long Exposure (MECHSHUTV)

string array

0 ,	EXPOS_WIDTH,
AV ,	AV,
START ,	STOP,
-1 ,	-1

The exposure time is in ms with a range from 0 to 1,000,000 for EXPOS_WIDTH, in steps of 1 ms.
If both AV values are '-1', the default values are used for start and stop time of the output Signal
If both AV values are '0' the start and stop time of the output signal is calculated according to the START and STOP values given.
Range of START is 0 to EXPOS+STOP-1.
No negative values are allowed, positive values set the start time of output signal after exposure time start.
Range of STOP is EXPOS*(-1) to 1,000,000
Negative values set the stop time of output signal before exposure time ends, positive values after exposure time end.

e) Long Exposure (QE_FAST)

string array

DELAY	,	EXPOS_WIDTH,
-1	,	-1

The delay and exposure time is in ns with a range from 0 to 50,000,000 for DELAY and from 500 to 10,000,000 for EXPOS_WIDTH, in steps of 100 ns (nearest value is selected, get exact times with GET_CAM_VALUES. Exactly one pair of values is expected.

f) Long Exposure (QE_DOUBLE)

string array

-1	,	-1

The exposure times for the two half images are determined by the sequence of the TRIG input signal on the PCI Interface Board. No timevalues are given in this mode

g) Long Exposure (EM_FAST)

string array

DELAY	,	EXPOS_WIDTH,
-1	,	-1

The delay and exposure time is in µs with a range from 0 to 75,000 for DELAY and from 75 to 15,000 for EXPOS_WIDTH, in steps of 75 µs.
Exactly one pair of values is expected.

h) Long Exposure (EM_FAST_V)

string array

0	,	EXPOS_WIDTH,
-1	,	-1

The exposure time is in µs with a range from 75 to 15,000 for EXPOS_WIDTH , in steps of 75 µs. Exactly one pair of values is expected.

h) sensicam em
Two additional strings can be added to the exposure string, to set the electron multiplying gain and the setpoint for the ccd-temperature.

string gain			
“mg1”	electron multiplying gain	=	2
“mg2”		=	5
“mg3”		=	10
“mg4”		=	20
“mg5”		=	50
“mg6”		=	100
“mg7”		=	200
“mg8”		=	500
“mg9”		=	1000

string setpoint	
“ptVAL”	ccd_temperature

VAL is in the range -20 to actual electronic temperature. Default value is –12.

i) sensicam uv
Two additional strings can be added to the exposure string, to set the electron multiplying gain and the setpoint for the ccd-temperature.

string gain			
“mg1”	electron multiplying gain	=	1
“mg2”		=	2
“mg3”		=	5
“mg4”		=	10
“mg5”		=	20
“mg6”		=	50
“mg7”		=	100
“mg8”		=	200
“mg9”		=	500

string setpoint	
“ptVAL”	ccd_temperature

VAL is in the range -20 to actual electronic temperature. Default value is –12.

aa) Fast Shutter (NORMALFAST)

string array

delay1	, expos_width1,
delay2	, expos_width2,
	...
delay100	, expos_width100,
-1	, -1

All delay and exposure times are set in 'ns' with a range from 0 to 1,000,000, in steps of 100 ns. The values are given in sequential order each time is following the preceding time. All values added result in the complete imaging sequence time.
Up to 100 pairs of values are possible!

bb) Fast Shutter (CYCLE)'

string array

1	, cycle1,
delay1	, expos_width1,
1	, cycle2,
delay2	, expos_width2,
	...
1	, cycle50,
delay50	, expos_width50,
-1	, -1

All delay and exposure times are set in 'ns' with a range from 0 to 1,000,000, in steps of 200 ns.
The cycle value must be in the range of 1 ... 1000.
Up to 50 pairs of values are possible.
Every delay, expos_width must be triggered externally.
Parameter **trig**: trig = 0x001 or trig = 0x002
Delay + expos_width must be ≥ 1µs.

cc) Double Shutter

string array

-1	,	-1

The exposure times for the two half images are determined by the sequence of the TRIG input signal on the PCI Interface Board. No timevalues are given in this mode

2.2.3 Examples for sensicam

Example 1

An ROI of a 640 x 480 sensor with 32 pixels horizontal and 64 pixels vertical in the top right corner has the following settings:

```
int roixmin, roixmax = 20, 20;
int roiymin, roiymax = 1, 2;
```

Example 2

If in addition to the situation in example 1 a horizontal binning of 2 pixels (hbin = 2) and a vertical binning of 16 lines (vbin = 16) is set, the image size is reduced to 16 x 4 pixels.

2.2.4 Build COC for dicam pro

All 'dicam pro' camera types are supported with this SDK.

SET_COC

int **SET_COC**	(HANDLE hdriver int mode, int trig, int roixmin, int roixmax, int roiymin, int roiymax, int hbin, int vbin, char *table)

This function generates a COC (**C**amera **O**peration **C**ode) which is loaded into the program memory of the camera. All parameters are checked to ensure that a valid set is generated. If any of the parameters is not valid the function returns `PCO_ERROR_WRONGVALUE`. To get a valid set of parameters TEST_COC can be used.

IN

For exact description of all parameters see notes below

hdriver	=	filehandle of opened driver
mode		operation mode
trig		trigger and start mode (auto, hw, …)
roixmin		start of horizontal ROI (Region of Interest)
roixmax		end of horizontal ROI
roiymin		start of vertical ROI
roiymax		end of vertical ROI
hbin		horizontal binning
vbin		vertical binning
table		pointer to zero terminated ASCII string with values for delay and exposure times

mode

Set the camera type, operation mode and analog gain. It is a combination of the following parameters
(**type**+(**gain***256)+(**submode***65536)) respectively
((**type**&0xFF) | ((**gain**&0xFF)*<<8) | ((**submode**&0xFF)<<16))
See also the defines in the cam_types.h file

typ	5	dicam pro (M_DICAM)
gain	0	normal analog gain
	1	extended analog gain
submode	0	single trigger mode (DPSINGLE)
	1	multi trigger mode (DPMULTI)
	2	double trigger mode (DPDOUBLE)

submode

DPSINGLE:
A single exposure is started with one trigger event and stored into one frame.

DPMULTI:
Multiple exposures are started with one trigger event and stored into one frame.

DPDOUBLE:
A double exposure with short interframing time is started with one trigger event and stored into two frames.

trig

Trigger setting of 'dicam pro' camera is done in the string-table, so trig must always be set to zero.

trig	0:	auto start, auto frame

All other values are not allowed!

roixmin, roixmax

Set the start and end value for the horizontal Region of Interest. One unit is 32 pixels. This setting affects the readout of the CCD-Chip. Less data is transferred, but readout time is not affected.

roixmin	start value of horizontal ROI	
roixmax	end value of horizontal ROI	
	range 1 ... 20	for CCD chip type 640 x 480
	range 1 ... 40	for CCD chip type 1280 x 1024

roiymin, roiymax

Set the start and end value for the vertical Region of Interest. One unit is 32 pixels. This setting affects the readout of the CCD-Chip. Less data is transferred and the readout time is decreased.

roiymin	start value of vertical ROI	
roiymax	end value of vertical ROI	
	range 1 ... 15	for CCD chip type 640 x 480
	range 1 ... 32	for CCD chip type 1280 x 1024

Thus the smallest ROI is 32 pixels in square.
In the case of the double trigger mode, the ROI is set for the two half images which are then transferred as one data set of double height.

hbin

This variable sets the horizontal binning. This setting affects the readout of the CCD-Chip. Less data is transferred but the readout time is not affected.

hbin	horizontal binning 1 no binning selectable values 1, 2, 4, 8

vbin

This variable sets the vertical binning. The maximal vertical binning setting depends on the selected vertical ROI . This setting affects the readout of the CCD-Chip. Less data is transferred and the readout time is decreased.

vbin	vertical binning 1 no binning selectable values SVGA 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 selectable values VGA 1, 2, 4, 8, 15, 16, 30, 32, 60, 64, 120, 128, 240, 256, 480

table

Set the special 'dicam pro' values and delay and exposure times. The parameter is a pointer to a zero terminated ASCII string. All values are separated by comma (,) or space ( ). The array is concluded by the sequence "-1,-1", so that variable key lengths can be handed over. The characters 'CR' (13D) and 'LF' (10D) may be used to structure the input string.

table	pointer to string array

string array

phosphordecay, mcpgain, trigger, loops,
delayhigh1, delaylow1, timehigh1, timelow1,
delayhigh2, delaylow2, timehigh2, timelow2,
delayhigh3, delaylow3, timehigh3, timelow3,
-1, -1

phosphordecay	0 … 100 in [ms]
mcpgain	0 ... 999
trigger	0 no trigger 1 external rising edge
loops	1 ... 256
mintime	minimum pulse time, depending on the pulser
mindeltime	minimum time between two pulses

If loop is set greater than 1, first delay value must also be greater than mindeltime.

Three modes are defined for 'dicam pro' camera:

a) dicam pro single trigger mode
mintime and mindeltime depends on the HVP pulser type, see table below

time and delay setting steps as follows (in ns):
3, 5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100, 120, 140, ... , 1000 in 20ns steps

pulser type	mintime in [ns]	mindeltime in [ns]
HVP3X-3	3 - 10 - 20 - 25 - 30	500
HVP3X-5	5 - 10 - 20 - 25 - 30	500
HVP3X-20	20 - 25 - 30	500
HVP3N	3 - 10 - 15 - 20 - 25	300000
HVP5N	5 - 10 - 15 - 20 – 25	300000
HVP2N	100	500

delayhigh1 0 ... 999999ms
delaylow1 0 ... 999999ns
timehigh1 0 ... 999999ms
timelow1 mintime ... 999999ns

b) dicam pro multi trigger mode
mintime = 20
mindeltime = 500ns or 300µs (depends on the pulser type)
time and delay settings in 20ns steps.

delayhigh1 0 ... 999ms
delaylow1 0 ... 999980ns
timehigh1 0 ... 999ms
timelow1 mintime ... 999980ns

delayhigh2 0 ... 999ms
delaylow2 mindeltime ... 999980ns
timehigh2 0 ... 999ms
timelow2 mintime ... 999980ns

delayhigh3 0 ... 999ms
delaylow3 mindeltime ... 999980ns
timehigh33 0 ... 999ms
timelow3 mintime ... 999980ns

c) dicam pro double trigger mode
mintime = 20
mindelpulser = 500ns or 300µs (depends on the pulser type)
time and delay settings in 20ns steps.

delayhigh1	0 ... 10ms
delaylow1	0 ... 999980ns
timehigh1	0 ... 999ms
timelow1	mintime ... 999980ns

delayhigh2	0 ... 10ms
delaylow2	mindeltime ... 999980ns
timehigh2	0 ... 999ms
timelow2	mintime ... 999980ns

Note: **The sum of delay1 + delay2 + time1 must be greater than 1000ns.**

2.2.5 Get COC and size information

TEST_COC

int **TEST_COC** (HANDLE hdriver, int *mode, int *trig, int *roix1, int *roix2, int *roiy1, int *roiy2, int *hbin, int *vbin, char *table, int *tablength)

Test all parameters. If the parameters have a valid value, they will be accepted, otherwise the valid value, next closed to the given value, will be used.

IN

hdriver	=	filehandle of opened driver
*mode	=	value to test
*trig	=	value to test
*roix1	=	value to test
*roix2	=	value to test
*roiy1	=	value to test
*roiy2	=	value to test
*hbin	=	value to test
*vbin	=	value to test
*table	=	string to test
*tablength	=	length of the allocated buffer for table

OUT

*mode	=	corrected value
*trig	=	corrected value
*roix1	=	corrected value
*roix2	=	corrected value
*roiy1	=	corrected value
*roiy2	=	corrected value
*hbin	=	corrected value
*vbin	=	corrected value
*table	=	corrected string, if string-buffer is great enough
*tablength	=	length of new builded table

Return Codes:

`PCO_NOERROR`
no error, function call successful

`PCO_WARNING_SDKDLL_COC_VALCHANGE`
one or more values changed

`PCO_WARNING_SDKDLL_COC_STR_SHORT`
buffer for builded string too short

GET_COC_SETTING

int **GET_COC_SETTING** (HANDLE hdriver, int *mode, int *trig, int *roix1, int *roix2,
int *roiy1, int *roiy2, int *hbin, int *vbin,
char *table, int len)

Get actual COC parameters.

IN

hdriver	=	filehandle of opened driver
len	=	length of the allocated buffer for table

OUT

*mode	=	actual mode value
*trig	=	actual trigger value
*roix1	=	actual roixmin value
*roix2	=	actual roixmax value
*roiy1	=	actual roiymin value
*roiy2	=	actual value
*hbin	=	actual value
*vbin	=	actual value
*table	=	actual timetable string, if string-buffer is great enough

GETSIZES

int **GETSIZES**(HANDLE hdriver, int *ccdxsize, int *ccdysize, int *actualxsize,
int *actualysize, int *bit_pix)

This command returns the size of the CCD, the actual size in pixel and the dynamics.

IN

hdriver	=	filehandle of opened driver

OUT

*ccdxsize	= x-resolution of CCD
*ccdysize	= y-resolution of CCD
*actualxsize	= x-resolution of image
*actualysize	= y-resolution of image
*bit_pix	= bits per pixel in image (i.e. 12 bit)

2.2.6 Get common camera information

GET_CAM_PARAM

int **GET_CAM_PARAM** (HANDLE hdriver, struct cam_param *param)

This command fills the structure cam_param with the actual parameters of the connected camera. The structure cam_param is defined in the header file “sencam_def.h”.

IN

hdriver = filehandle of opened driver

OUT

*param = new parameters

GET_CAM_VALUES

int **GET_CAM_VALUES** (HANDLE hdriver, struct cam_values *val)

This command fills the structure cam_values with the actual values of the board, the actual COC and the connected camera.
The structure cam_values is defined in the header file “sencam_def.h”.

IN

hdriver = filehandle of opened driver

OUT

*val = new values

GET_CAM_SETTINGS

int **GET_CAM_SETTINGS** (HANDLE hdriver, struct cam_settings *set)

This command fills the structure cam_settings with the actual settings of the COC from the last accepted SET_COC() call.
The structure cam_settings is defined in the file “sencam_def.h”.

IN

hdriver = filehandle of opened driver

OUT

*set = new settings

GET_DICAM_SETTINGS

int **GET_DICAM_SETTINGS** (HANDLE hdriver, struct dicam_set *set)

This command fills the structure dicam_set with the actual settings of the COC from the last accepted SET_COC() call.
The structure dicam_set is defined in the header file “sencam_def.h”.

IN

hdriver = filehandle of opened driver

OUT

*set = new settings

GET_CAMERA_DESC

int **GET_CAMERA_DESC** (HANDLE hdriver, SC_Camera_Description *strDescription)

This command fills the structure SC_Camera_Description for the actual connected camera The structure SC_Camera_Description is defined in the header file “SC_SDKStructures.h”.

IN

hdriver = filehandle of opened driver

OUT

*strDescription = actual description

GET_DICAMPRO_DESC

int **GET_DICAMPRO_DESC** (HANDLE hdriver, DP_Control_Description *strDescription)

This command fills the structure DP_Control_Description for the actual connected dicam pro camera The structure DP_Control_Description is defined in the header file “SC_SDKStructures.h”.

IN

hdriver = filehandle of opened driver

OUT

*strDescription = actual description

GET_CAMERA_LINETIME_DESC

int **GET_CAMERA_LINETIME_DESC**(HANDLE hdriver, SC_Linetimes_Description *strDescription)

This command fills the structure SC_Linetimes_Description for the actual connected camera. The structure SC_Linetimes_Description is defined in the header file “SC_SDKStructures.h”.

IN

hdriver = filehandle of opened driver

OUT

*strDescription = actual description

GET_STATUS

int **GET_STATUS** (HANDLE hdriver, int camtype, int *eletemp, int *ccdtemp)

Get status information from the camera and the PCI Interface board and read the temperature of the camera circuits and of the CCD sensor. For values out of range an error is returned.
This function is mainly for better portability of applications, which have used one of the older sensicam SDK. It should not be used for new applications.
New CCD-Types cannot be recognized with this function, use GET_CAMERA_PARAM instead.

IN

hdriver = filehandle of opened driver

OUT

*camtype	status information, see below
*eletemp	temperature of camera electronic valid range -30 ... +65 °C (-22...+149 °F)
*ccdtemp	temperature of CCD-Chip valid range -30 ... +65 °C (-22...+149 °F)

cam_type:

D17,D16	gain 00 = normal gain 01 = extended gain 02 = extended gain invers
D15,D14	CCD-type 00 = 640 x 480 01 = 640 x 480 10 = 1280 x 1024 or extended CCD
D13,D12	Camera-type 00 = LongExposure 01 = FastShutter / DoubleShutter 10 = special version 11 = dicam pro
D11,D10,D9	sensicam: version of the camera 000 = vers 1.0 001 = vers 1.5 010 = vers 2.0 … dicam pro: type of installed pulser 000 = HVP2N 001 = HVP5N 011 = HVP5NE 100 = HVP3X, 5ns 101 = HVP3X, 3ns 111 = HVP3X, 20ns
D8	CCD-color 0 = black/white CCD 1 = RGB CCD
D7,D6	FastShutter/Doubleshutter type 01 = Fastshutter 10 = Doubleshutter
D8	temperature regulation 0 = regulating 1 = locked (-13°C = 10.6 °F)
D4…D0	reserved

2.2.7 Miscellaneous functions

SET_POWERDOWN

int **SET_POWERDOWN** (HANDLE hdriver, int pdtime, int pdwakeup)

This command does set the time values for the powerdown function of the CCD. Both time values are in ms.
pdtime is the time after which powerdown is invoked in the COC
pdwakeup is the time the chip needs to come back from powerdown

IN

hdriver	=	filehandle of opened driver
pdtime	=	timevalue
pdwakeup	=	timevalue

SET_DICAM_WAIT

int **SET_DICAM_WAIT** (HANDLE hdriver, int time)

This command does set the waiting time for 'dicam pro' camera. The camera must wait a specific time until HV-Modul and Pulser are ready for the next exposure. Waiting is done internal after programming the COC. The time value is in ms. The value cannot be smaller than 200ms.

IN

hdriver	=	filehandle of opened driver
time	=	timevalue

CLEAR_BAORD_BUFFER

int **CLEAR_BOARD_BUFFER** (HANDLE hdriver)

A fast clear of the on board buffers is done. Only one of the buffers is cleared with each call to this function. If none of the buffers have valid data nothing is done.

IN

hdriver	=	filehandle of opened driver

Return Codes:
`PCO_NOERROR`
if a buffer was cleared

`PCO_WARNING_SDKDLL_NO_IMAGE_BOARD`
if no buffer has valid data

2.3 Image and Buffer Control Functions

2.3.1 Wait and readout Image

WAIT_FOR_IMAGE

int **WAIT_FOR_IMAGE** (HANDLE hdriver, int timeout)

This function does wait up to 'timeout' ms (milliseconds) until the next image is transferred from the camera to one of the onboard buffers.
If there is already an image in the onboard buffer the function returns immediately.
If no image is transferred within the waiting time the function returns with errorcode:
`PCO_ERROR_TIMEOUT`

IN

hdriver	=	filehandle of opened driver
timeout	=	maximum time to wait in ms

READ_IMAGE_12BIT

int **READ_IMAGE_12BIT** (HANDLE hdriver, int mode, int width, int height, unsigned short *b12)

This function reads an image from the onboard Board buffer with the selected 'width' and'height' (in pixels) and writes it into the memory area specified by the pointer. In 'Double Shutter' mode (cf. SET_COC) the two half images are read as one data set of double height. The number of bytes which are read is 'width' * 'height' *2.
Direct DMA-transfer used to write the data from the onboard buffer to the memory region.
If the function was successful, the onboard buffer containing the image is released and the image cannot be read again.

IN

hdriver	=	filehandle of opened driver
mode	=	convert mode, a combination of the following flags
		NORMAL 0x0000
		FLIP 0x0001 (change lines)
		MIRROR 0x0001 (change rows)
witdh	=	image width
height		image height
b12		pointer to memory address

OUT

*ptr	=	image data

If FLIP flag or MIRROR flag is set, the total image is flipped (mirrored horizontal) or mirrored (mirrored vertical) resp. Combination of both flags (mode = FLIP + MIRROR) is possible which results in a 180° rotated image. FLIP and/or MIRROR require additional processing time compared to 'normal'

If no image is in the onboard buffer when the function is called, it enters a sleep state for actual readout time, if readout time is below 1000ms.
Then another image available check is done. If this reports that no image is onboard the function returns with warning
`PCO_WARNING_SDKDLL_NO_IMAGE_BOARD`

It is recommended that WAIT_FOR_IMAGE is called before calling this function, to ensure that an image is available.

2.3.2 Image buffer status information

GET_BUFFER_STATUS

int **GET_BUFFER_STATUS** (HANDLE hdriver, int bufnr, int mode, int *ptr, int len)

This command returns a given number of status bytes from the buffer structure DEVBUF of the specified buffer. In the header-file "senbuf_def.h" there are macro definitions to extract certain information of this structure. (The structure DEVBUF is defined in the driver.)

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER
mode	=	0
ptr		pointer to memory address
	=	address of allocated memory
len		bytes to read
	=	4…size of allocated memory

OUT

*ptr	= values of structure DEVBUF

2.3.3 Allocating and free image buffers

ALLOCATE_BUFFER

int **ALLOCATE_BUFFER** (HANDLE hdriver, int *bufnr, int *size);

This command allocates a buffer for the camera in memory.
The value of size has to be set to the number of **bytes, which** should be allocated. The returned value of size might be greater because the buffer is allocated with a certain block size and therefore should not be used in a following ADD_BUFFER_TO_LIST call.
To allocate a new buffer, the value of bufnr must be set to –1 (*bufnr=-1).
The return value of bufnr must be used in the calls to the other Memory Control Functions. If a buffer should be reallocated *bufnr must be set to its buffer number and *size to the new size.
If the function fails the return values of size and bufnr are not valid and must not be used.

IN

hdriver	=	filehandle of opened driver
*bufnr	=	-1 for allocating a new buffer
*bufnr	=	image-buffer number returned from previous ALLOCATE_BUFFER, to reallocate with different size
*size		size of image-buffer in byte

OUT

*bufnr	=	number of image-buffer
*size	=	allocated size, which might be greater as the size wanted

ALLOCATE_BUFFER_EX

int **ALLOCATE_BUFFER_EX** (HANDLE hdriver, int *bufnr, int size,HANDLE *hPicEvent,void** adr);

If value of *adr is NULL this command allocates a buffer for the camera in memory and does return the allocated address

The user can apply a self allocated buffer to this function call. If *adr is not zero and is a valid data block, the SDK-Dll will commit this externally allocated memory block and attaches it to a new buffer number.

Additionally it is possible to attach an event to the newly created buffer. The user can create this event within his own code and set it to hEvent. If *hEvent is NULL an event will be created inside the SDK-Dll

The value of size has to be set to the number of **bytes, which** should be allocated.

To allocate a new buffer, the value of bufnr must be set to –1 (*bufnr=-1).
The return value of bufnr must be used in the calls to the other Memory Control Functions. If a buffer should be reallocated *bufnr must be set to its buffer number and *size to the new size.
If the function fails the return values of size and bufnr are not valid and must not be used.

IN

hdriver	=	filehandle of opened driver
*bufnr	=	-1 for allocating a new buffer
*bufnr	=	image-buffer number returned from previous ALLOCATE_BUFFER, to reallocate with different size
size		size of image-buffer in byte
*hPicEvent		= NULL create event internally
*hPicEvent		= HANDLE of an event created outside SDK-Dll
*adr		= NULL memory will be allocated internally
*adr		= valid address of memory allocated outside SDK-Dll

OUT

*bufnr	=	number of image-buffer
*hPicEvent		= HANDLE of created event
*adr		= address of allocated memory

FREE_BUFFER

int **FREE_BUFFER** (HANDLE hdriver, int bufnr);

Free allocated buffer. If the buffer was set into the buffer queue and no transfer was done to this buffer call REMOVE_BUFFER_FROM_LIST first.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER

2.3.4 Image buffer queue handling

ADD_BUFFER_TO_LIST

int **ADD_BUFFER_TO_LIST** (HANDLE hdriver, int bufnr, int size, int offset, int data)

Set a buffer into the buffer queue. The SDK-DLL can manage a queue of 64 buffers. A buffer cannot be set to the queue a second time.
If other buffers are already in the list the buffer is set at the end of the queue. If no other buffers are set in the queue the buffer is immediately prepared to read in the data of the next image of one of the board buffers.
If an image transfer is finished the driver changes the buffer status word and searches for the next buffer in the queue. If a buffer is found, it is removed from the queue and prepared for the next transfer.
To wait until a transfer to one of the buffers is finished, poll the buffer status word or wait until the Event of the buffer is fired.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()
size	=	number of bytes to transfer
offset	=	0 (not implemented yet)
data	=	0 (not implemented yet)

recommended size for 12 bit data
actualxsize*actualysize*sizeof(unsigned short)

Get actualxsize and actualysize with function GETSIZES()

If the number of bytes of the transfer does not match the number of bytes which the camera sends to the PCI-board errors may occur in the status of the buffer.

REMOVE_BUFFER_FROM_LIST

int **REMOVE_BUFFER_FROM_LIST** (HANDLE hdriver, int bufnr);

This command removes the buffer from the buffer queue.
If a transfer is actual in progress to this buffer, an error is returned.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()

ADD_BUFFER

int **ADD_BUFFER** (HANDLE hdriver, int size, void *adr, HANDLE hEvent, DWORD* Status)

Set an external allocated memory buffer into the buffer queue. The SDK-DLL can manage a queue of 64 buffers. A buffer cannot be set to the queue a second time.
The event is set when the DMA-Transfer into the buffer is finished or if a error occurred during the transfer.
Use i.e WaitForSingleObject(*hPicEvent,TimeOut); to wait until a transfer is done. Status must then be checked for errors.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()
size	=	number of bytes to transfer
adr	=	valid address of memory allocated outside SDK-Dll
hEvent	=	HANDLE of an event
Status	=	address of a DWORD, which will be filled success status of the buffer

recommended size for 12 bit data
actualxsize*actualysize*sizeof(unsigned short)

Get actualxsize and actualysize with function GETSIZES()

If the number of bytes of the transfer does not match the number of bytes which the camera sends to the PCI-board errors may occur in the status byte of the buffer.

REMOVE_BUFFER

int **REMOVE_BUFFER** (HANDLE hdriver, void * adr);

This command removes the buffer from the buffer queue.
If a transfer is actual in progress to this buffer, an error is returned.

IN

hdriver	=	filehandle of opened driver
*adr	=	valid address of memory which was set into the buffer queue with an ADD_BUFFER() call.

REMOVE_ALL_BUFFERS_FROM_LIST

int **REMOVE_ALL_BUFFERS_FROM_LIST** (HANDLE hdriver);

This command removes all buffers from the buffer queue.
If a transfer is actual in progress to one of the buffers, an error is returned.

IN

hdriver	=	filehandle of opened driver

SET_BUFFER_EVENT

int **SET_BUFFER_EVENT** (HANDLE hdriver, int bufnr, HANDLE *hEvent)

Create or attach an eventhandle for this buffer. The event is set when the DMA-Transfer into the buffer is finished or if a error occurred during the transfer Use i.e WaitForSingleObject(*hPicEvent,TimeOut); to wait until a transfer is done

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()
*hEvent	=	NULL create event internally
*hEvent	=	HANDLE of an event created outside SDK-Dll

OUT

* hEvent	=	HANDLE of created event

CLEARBUFFER_EVENT

int **CLEARBUFFER_EVENT** (HANDLE hdriver, int bufnr, HANDLE *hEvent)

Detach the eventhandle from this buffer. If the event was internally created close the event handle

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()
*hEvent	=	attached HANDLE

OUT

* hEvent	=	NULL if handle is closed

2.3.5 Image buffer mapping

MAP_BUFFER

int **MAP_BUFFER** (HANDLE hdriver, int bufnr, int size, int offset, void **linadr)

This command maps the buffer and returns the address.
If size is greater than allocated buffer size an error is returned.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()
size	=	number of bytes to map
offset	=	0

OUT

*linadr	= address of buffer

UNMAP_BUFFER

int **UNMAP_BUFER**(HANDLE hdriver, int bufnr)

This command does unmap the buffer.
Please unmap all mapped buffers before closing the driver.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from ALLOCATE_BUFFER()

3 Typical Implementation

This typical step by step implementation shows the basic handling.
Use camera model used in the example is a sensicam vga long exposure camera.
The camera is set to full resolution, no binning and should grab images with an exposure time of 10ms.
Error handling must be added.

1. Open the camera

```
int board=0;
HANDLE hdriver=NULL;

err=INITBOARD(board,&hdriver);
```

2. Setup the camera and fill the structures:

```
err=SETUP_CAMERA(hdriver);

SC_Camera_Description strCamDesc;
memset(&strCamDesc,0,sizeof(SC_Camera_Description));
strCamDesc.wSize=sizeof(SC_Camera_Description);
err=GET_CAMERA_DESC(hdriver,&strCamDesc);
```

3. Set camera settings (exposure, modes, etc.) and sizes (binning, ROI, etc.).

```
char times[200];
int mode=M_LONG;
int submode=NORMALLONG;
int trig=0
int roixmin=1;
int roiymin=1;
int roixmax=20;
int roiymax=15;
int hbin=1;
int vbin=1;
sprintf(times,"0,10,-1,-1");

int mode_full=mode+(submode<<16);

err=SET_COC(hdriver,mode_full,trig,roixmin,roixmax,
           roiymin,roiymax,hbin,vbin,times);
```

4. Get the sizes and allocate buffer(s)

```
int width,height,ccdxsize,ccdysize,bitpix;
err=GETSIZES(hdriver,&ccdxsize,&ccdysize,
             &width,&height,&bitpix);

int size=width*height*((bitpix+7)/8);
int bufnr=-1;
err=ALLOCATE_BUFFER(hdriver,&bufnr,&size);

void *adr=NULL;
err=MAP_BUFFER(hdriver,bufnr,size,0,&adr);

HANDLE buffer_event=NULL;
err=SETBUFFER_EVENT(hdriver,bufnr,&buffer_event);
```

5. Set camera state to 'RUN'

```
err=RUN_COC(hdriver,0);
```

6. Readout directly or add your buffer(s)

```
err=READ_IMAGE_12BIT(hdriver,0,width,height,adr);
```

or

```
err=ADD_BUFFER_TO_LIST(hdriver,bufnr[x],size,0,0);

int bufstat;
DWORD status=WaitForSingleObject(buffer_event,0);
if(status==WAIT_OBJECT_0)
 GETBUFFER_STATUS(hdriver,bufnr,0,
                  &bufstat,sizeof(bufstat));
```

7. Do a convert and show the image.

Use your own convert routines or one out of convert library from PCO to create a displayable (Bitmap) data-array
Display or store the image.

8. Stop the camera.

```
err=STOP_COC(hdriver,0);
```

9. Free all buffers and close the camera.

```
REMOVE_ALL_BUFFERS_FROM_LIST(hdriver);
err=FREE_BUFFER(hdriver,bufnr);
err=CLOSEBOARD(&hdriver);
```

4 Return Codes

The error codes are standardized as far as possible. The error codes contain the information of the error layer, the source (microcontrollers, CPLDs, FPGAs) and an error code (error cause). All values are combined by a logical OR operation. Error codes and warnings are always negative values, if read as signed integers, or if read as unsigned word, the MSB is set. Errors have the general format 0x80######, warnings have the format 0xC0######.The error numbers are not unique. Each layer and the common errors have their own error codes. You have to analyze the error in order to get error source. This can easily be done with a call to PCO_GetErrorText.

// e.g.: 0xC0000080 indicates a warning,
// 0x800A3001 is an error inside the SC2-SDK-dll.
// MSB LSB
// XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
// |||| |||| |||| |||| |||| |||| |||| ||||
// |||| |||| |||| |||| |||| ---------- Error or warning code
// |||| |||| |||| |||| ||||
// |||| |||| |||| |||| ---------- Layer code
// |||| |||| |||| ||||
// |||| |||| ---------- Device code
// |||| ||||
// |||---------- reserved for future use
// |||
// ||---------- Common error code flag
// ||
// |---------- Warning indication bit
// |
// ---------- Error indication bit

Error layer:
0x00001000 PCO_ERROR_FIRMWARE // error inside the firmware
0x00002000 PCO_ERROR_DRIVER // error inside the driver
0x00003000 PCO_ERROR_SDKDLL // error inside the SDK-dll
0x00004000 PCO_ERROR_APPLICATION // error inside the application

Error / Warning source:
0x00010000 SC2_ERROR_PCOCAM_POWER_CPLD // error at CPLD in pco.power unit
0x00020000 SC2_ERROR_PCOCAM_HEAD_UP // error at uP of head board in pco.camera
0x00030000 SC2_ERROR_PCOCAM_MAIN_UP // error at uP of main board in pco.camera
0x00040000 SC2_ERROR_PCOCAM_FWIRE_UP // error at uP of firewire board in pco.camera
0x00050000 SC2_ERROR_PCOCAM_MAIN_FPGA // error at FPGA of main board in pco.camera
0x00060000 SC2_ERROR_PCOCAM_HEAD_FPGA // error at FGPA of head board in pco.camera
0x00070000 SC2_ERROR_PCOCAM_MAIN_BOARD // error at main board in pco.camera
0x00080000 SC2_ERROR_PCOCAM_HEAD_CPLD // error at CPLD of head board in pco.camera
0x00090000 SC2_ERROR_SENSOR // error at image sensor (CCD or CMOS)
0x000A0000 SC2_ERROR_SDKDLL // error inside the SDKDLL
0x000B0000 SC2_ERROR_DRIVER // error inside the driver
0x000D0000 SC2_ERROR_POWER // error within power unit
0x00100000 PCO_ERROR_CAMWARE // error in CamWare (also some kind of "device")
0x00110000 PCO_ERROR_CONVERTDLL // error inside the convert dll

Error codes:
Please take a look at the file pco_err.h.
Warnings:
Please take a look at the file pco_err.h.
In case of successful operation the standard Response Message is returned.
To get detailed error information you can call the function PCO_GetErrorText, which is defined inside the PCO_errt.h header file.

4.1.1 PCO_GetErrorText

Gets a detailed description for an error.
This function is part of the header file pco_errt.h. If you want to use this function include the pco_errt.h header file and define PCO_ERRT_H_CREATE_OBJECT in **one** of your modules.

a.) Prototype:
void PCO_GetErrorText(DWORD dwerr, char* pbuf, DWORD dwlen)

b.) Input parameter:
- DWORD dwerr: DWORD which holds the error number.
- char* pbuf: Address of the first char of an char array.
- DWORD dwlen: DWORD which holds the length of the char array in byte.

pco.
imaging

PCO AG
Donaupark 11
D-93309 Kelheim
fon +49 (0)9441 2005 0
fax +49 (0)9441 2005 20
eMail: info@pco.de
www.pco.de

The Cooke Corporation
6930 Metroplex Drive
Romulus, MI 48174
USA
www.cookecorp.com