SDK

Software Development Kit

Linux
Version 1.02

sensicam dicam pro sensicam

pco.
imaging

Contents

Driver Functions

Software Development Kit for SensiCam under Linux

Basics

Camera and PCI-Board control are managed on two levels, represented by the library **libsencam** and the driver **pco52x.o**.

This description includes in detail all functions of the upper level of the SDK (libsencam) and some hints how to use it. The important parts of the driver (pco52x.o) are also included. Although the driver is described in this manual it is recommended to use the library functions instead of the driver functions directly. The library has the capability to write error messages to stderr, a logfile or with the *syslog()* call.
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 is at the end of this manual and in the header file "errcodes.h". The OUT part in the description of each function does not include this return value. The returned value is below zero for errors, zero for no error and above zero for notifications. SDK functions which are declared as *void name()*, do not return a value.

Hardware and Library

The PCI520 respectively 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.
The Master DMA requires a special memory management for the image buffers. Therefore these image buffers are allocated in kernel memory area. 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) and at least map the image buffer to the normal user space.
Because each image buffer requires some memory overhead it is recommended, that you don't use too much of these buffers. For large sequences allocate your own memory and use a copy function to transfer the data from one of the image buffers to your memory, while transferring data from the camera to another image buffer.
For each board a maximum of 31 special image buffers are reserved, which are accessible with normal file I/O-functions directly. With the **Buffer Control Calls** you can manage these image buffers.
Most of the **Memory Control Functions** can also be called with the handle of one of the special image buffer.

The **General Control Functions** are used to open, close and setup the driver and the board. The driver can manage up to four boards. So if more than one PCI520/525 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 on the board. The COC is created and downloaded from Function *sen_set_coc()*. To start the camera *sen_run_coc()* must be called, to stop it *sen_stop_coc()*. After *sen_run_coc()* the camera starts the current COC and the image data is transmitted to one of the memory banks on the board. A call to *sen_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 *sen_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 driver also includes a proc interface, which gives a short description on the current state of the PCI-board and buffers.

Generating the COC for all Camera-Types and all possible variations is not included in the free part of this SDK. It is part of the firmware of the camera-model and therefore it is given as an precompiled additional library.

General Control Functions

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

This command initializes the PCI-Controller-Boards 0…3. 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 the *hdriver set to NULL, the function opens the driver with the standard device name and returns in *hdriver the file handle of the driver for the selected board.
To get a valid handle for the driver you can also call the system-function *open()* with the accurate device name. After this *sen_initboard()* must be called to initialize the board.
The filehandle 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 0. If only one board is installed board number must be 0.

IN

board		number of the PCI-Controller-Board
	=	0…3
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

int **sen_closeboard**(HANDLE *hdriver)

This command generates a reset of the PCI-Controller-Board and closes the driver. If the function does not fail, *hdriver is set to NULL.

IN

hdriver		pointer to filehandle
*hdriver	=	filehandle of opened driver

OUT

*hdriver	=	NULL

int **sen_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 fuction 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

void **sen_enable_error_messages**(int msg_lev,char *name)

This command enables or disables Library generated error and info message log. The messages are written in the logfile 'name'. If no name is given the standard name 'sencam.log' is used. Parameter msg_lev controls the kind of generated messages. For defined levels see file "loglevel.h".

IN

msg_lev	=	message level a combination of the defines in "loglevel.h"
*name	=	path and filename of logfile
*name		NULL standard filename "sencam.log" will be used

void **sen_set_syslog_facility**(int msg_lev)

This command sets the facility for the *syslog()* calls in the library. The level for syslog is ored to the given facility. The default setting is LOG_LOCAL2. The library does not call the *openlog()* or *closelog()* functions.

IN

msg_lev	=	message level a combination of the defines in "loglevel.h"

Camera Control Functions

int **sen_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, 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 a READ_IMAGE or a 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
  GETBUFFER_STATUS
until                   buffer is done

STOP_COC                terminates the exposure
```

int **sen_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

COC for SensiCam

int **sen_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 WRONGVAL. To get a valid set of parameters *sen_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

The following SensiCam camera types are available at the moment:
‘Long Exposure’, ‘Long Exposure QE’, ‘QE Standard’, ‘QE Double Shutter’, ‘Fast Shutter’ and ‘Double Shutter’.
Specific settings can only be made for distinct types as described below.

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 'Long Exposure' and all 'QE' versions of the SensiCam.

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)

1) only for cameras SensiCam QE, SensiCam QE Standard and SensiCam QE Double
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 cameras SensiCam QE Standard and SensiCam QE Double
4) only for camera SensiCam QE Double

submode NORMALLONG:
In the 'Sequential' mode delay, exposure and CCD readout are done sequentially, i. e. 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 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 interframing time between the two images has to be at least 500ns.

type Fast Shutter:
The type Fast Shutter is for the use of the Fast Shutter version of the SensiCam.

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 Double Shutter version of the SensiCam.

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 interframing 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 interframing 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 LongExposure and QE

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 [1]
	0x200	sequence start with external falling edge [1]
	0x101	sequence + frame start with external rising edge [1]
	0x202	sequence + frame start with external falling edge [1]

All other combinations are forbidden

[1)] These modes will work only with PCI Interface Boards with revision code 17 and later. Please ask PCO support.

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 FastShutter and DoubleShutter

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

roiymin, roiymax

Set the start and end value for the vertical region of interest (ROI). One unit is 32 pixels. 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 for CCD chip type 1280 x 1024
	range 1 ... 33 for CCD chip type 1376 x 1040

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 the 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

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 set 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 set 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 *sen_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) 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!

d) 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.

d) 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

Examples

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.

COC for DiCAM-PRO

int **sen_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 WRONGVAL. To get a valid set of parameters *sen_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 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 DiCAM-PRO modes are defined:

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 delay1 + delay2 + time1 must be higher than 1000ns.**

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

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

0	no error, function call successful
103	one or more values changed
104	buffer for builded string too short

int **sen_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

int **sen_clear_board_buffer**HANDLE hdriver)

Fast clear of the buffer on the board. 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:

0	if a buffer was cleared
100	if no buffer has valid data

int **sen_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 (12 bit)

int **sen_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 file “sencam_def.h” and listed at the end of this manual.

IN

hdriver	=	filehandle of opened driver

OUT

*param	=	new parameters

int **sen_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 file “sencam_def.h” and listed at the end of this manual.

IN

hdriver	=	filehandle of opened driver

OUT

*val	=	new values

int **sen_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” and listed at the end of this manual.

IN

hdriver	=	filehandle of opened driver

OUT

*set	=	new settings

Memory Control Functions

int **sen_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_d.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 sen_allocate_buffer()
	=	0x80000000+num num is the number of the buffer device from 1 to 32
or		
hdriver	=	filehandle of special image buffer
bufnr	=	not used
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

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

This command allocates a buffer for the camera in the kernel memory area.
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 *sen_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 *sen_allocate_buffer()*, to reallocate with different size
*bufnr	=	0x80000000+num num is the number of the buffer device from 1 to 31
or		
hdriver	=	filehandle of special image buffer
bufnr	=	not used
*size		size of image-buffer in byte

OUT

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

int **sen_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 *sen_remove_buffer_from_list()* first.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from *sen_allocate_buffer()*
bufnr	=	0x80000000+num num is the number of the buffer device from 1 to 32
or		
hdriver	=	filehandle of special image buffer
bufnr	=	not used

int **sen_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 *sen_allocate_buffer()*
bufnr	=	0x80000000+num num is the number of the buffer device from 1 to 32
or		
hdriver	=	filehandle of special image buffer
bufnr	=	not used

int **sen_remove_all_buffer_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

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

Set a buffer into the buffer queue. The driver can manage a queue of 32 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 a 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 use the select system call.

IN

hdriver	=	filehandle of opened driver
bufnr	=	image-buffer number returned from *sen_allocate_buffer()*
bufnr	=	0x80000000+num num is the number of the buffer device from 1 to 32
or		
hdriver	=	filehandle of special image buffer
bufnr	=	not used
size	=	number of bytes to transfer
offset	=	start offset of bytes in the image-buffer
data	=	0 (not implemented yet)

recommended size for 12 bit data
actualxsize*actualysize*2

Get actualxsize and actualysize with function *sen_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.
If transfer size is lower than camera size, the transfer is done with the specified transfer size and no error should occur.
With offset set to other values then 0, you can have more small camera images in one large buffer.
Offset must be a multiple of 4096.

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

This command maps the buffer to an user address. The address can be used for Read-, Write- and other operations on the buffer data from the PC-CPU.
If size is lower than the allocated buffer size not all data in the buffer can be accesssed. If size is greater than allocated buffer size an error is returned.
Restriction: Only one of the buffers can be mapped to a linear address at a time. Before mapping another of the buffers unmap the previous mapped buffer.

IN

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

OUT

*linadr	=	address of buffer

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

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

IN

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

int **sen_setbuffer_event(**HANDLE hdriver, int bufnr, int mode)

This command enables or disables the ability of the buffer, to create an event for the *select()* call if a data transfer is finished.

IN

hdriver	=		filehandle of opened driver
bufnr	=		image-buffer number returned from *sen_allocate_buffer()*
mode	=	0	disable event
	=	1	enable event

Driver Basics

The driver is a loadable LINUX module. It includes the I/O-interface to the SensiCam PCI-Controller-Board (PCI520 or PCI525) and preparation and maintenance of image buffers for each board in the PC main memory. The driver can handle up to four boards and up to 31 image buffers for each board. The driver communicates with the the PCI-board through an interrupt controlled I/O-interface.

Major and Minor numbers
The Major number of the module is currently defined to 0 in the header-file “sencamd.h”, so the Majornumber is dynamical assigned to the driver.
The major number can be changed to a free device number value i.e. 100 decimal in “sencamd.h”. This Major number should then only be used in a standalone system, where no interaction with other modules is possible.
The Major number can also be overwritten with symbol pcc_major when loading the driver.
The Minor number is divided into two partitions. The one byte Minor number looks as follows bbssssss, where bb stands for the different boards and ssssss for the image buffers assigned to the board ranging from 1-31. The number 0 for ssssss represents the IO-Interface to the given board.
The driver expects the following nodes in the /dev subdirectory pco52x_a and pco52x_a_bb, where a is the board number and bb is the image buffer number. With comand mknod one can build the nodes. Example below build the nodes for two boards each with two image buffers. (MJ = Major number, after loading the driver one can get its MJ from /proc/devices, look for sencam_dev in the output from *cat /proc/devices*).

mknod /dev/pco52x_0 c MJ 0 (IO-Device board 0)
mknod /dev/pco52x_0_01 c MJ 1 (first image buffer board 0)
mknod /dev/pco52x_0_02 c MJ 2 (second pic. buffer board 0)

mknod /dev/pco52x_1 c MJ 64 (IO-Device board 1)
mknod /dev/pco52x_1_01 c MJ 65 (first image buffer board 1)
mknod /dev/pco52x_1_02 c MJ 66 (second pic. buffer board 1)

When loading the module (i.e. *insmod /PATH/pco52x.o*) all variables in the driver are initialized. Then the driver searches for installed boards and allocates all internal buffers for each board. It then creates a proc interface, which gives essential information of the state of the driver and the image buffers.
When removing the device all internal buffers and previously allocated image buffers are freed.

The scripts sencam_load and sencam_unload can be used to correctly install respectively uninstall the driver into your system.

The following parameters could be set when loading the driver

pcc_major=x set Major number of the module
x=0…256
x=0 (default)
get a dynamic MAJOR Number

pcc_demo=x enable or disable demo mode
In demo mode the driver does no physical Iinput/output. So it runs without having a PCI Interface board installed. (Not all functions work correct in demo mode)
x= 0 demo mode disabled (default)
x= 1 demo mode enabled

pcc_message=x set message level of driver
The driver outputs debug messages. More bIts set send more messages.Definitions are in sencamls.h
x= 0 no messages (default)
x= 1 error messages send

pcc_process=x set open access requirements
x= 0 one user, open multi
x= 1 one user, one open
x= 2 one process, open multi (default)
x= 3 any user, open multi

Driver DMA Transfer

If the driver starts a DMA-Transfer, some data out of the driver structures VXDBOARDVAL and DEVBUF is used. For better understanding, what is going on in the driver a short explanation of the DMA-Transfer is given here.

In the Driver Buffer-Device a structure (struct DEVBUF) with all significant data is created for each image-buffer. The whole image-buffer is splitted in several blocks of allocated memory (default block size is 64kByte). In the structure DEVBUF two tables can be found: the pagetab for the physical addresses of the allocated blocks and the transfertab consisting of physical transfer addresses and transfer size for each block of memory. This table is built with function build_transfer(), which gets the size and starting offset for the actual transfer.

In the Driver I/O-Device the pointer 'actdma' and the list BUFLIST dmabuffers are used to manage the DMA-transfers to different buffers. The pointer actdma references the struct DEVBUF of the image-buffer to which the next DMA-transfer should appear or is still running. In the list BUFLIST the image-buffers for the following transfers are stored, with its actual parameters size and offset.
The function call *sen_add_buffer_to_list()* or "pcc:c4" append the image buffer at the end of the list. If actdma is currently empty, the image buffer is directly assigned to actdma with function set_new_actdma(). This function proofs, if a image buffer is in the list. If this is true it compares the actual settings of this image-buffer with the parameters for the given transfer and calls function build_transfer() if parameters have changed. At least the image-buffer is removed from the list.

The PC-Processor is free for other tasks during most time of the DMA-Transfer. When a transfer has to be started, needed data for this transfer (physical addresses and transfer size out of the transfertab table) is send to the DMA-Controller on the board. If the transfer is done it generates an interrupt and the values for the next transfer are send to the DMA-Controller. If all data for this image buffer are transferred the status words in the structures VXDBOARDVAL and DEVBUF are changed and the driver informs the calling process, if necessary. Furthermore it calls set_new_actdma(), to continue camera-transfer with the next image-buffer.
To start a DMA-Transfer function startdma() is called. This function checks, whether the camera is already started, whether actdma references a image buffer, whether this image-buffer has a valid transfer table and there is no DMA-Transfer operating to this buffer. If all is true and there are a valid data in one of the board buffers, the first values of the transfer table are send to the DMA-Controller. If no valid data is in the board, the driver enables image_in interrupts on the board, if a image_in interrupt is rised, startdma() is called again and the image_in interrupt function disabled.
Function startdma() is called from the driver if either RUN_COC is called from a process or a image-buffer is set to actdma.

Driver Buffer-Device

The driver Buffer-Device maintains up to 31 image buffers. It supports status reads, mapping of the buffer to users space, starting a DMA-Transfer to the buffer and waiting for the transfer to be done (with the system command *select()*).
For every buffer a set of parameters is available (struct DEVBUF).
The Buffer-Device supports the following file–operations:

READ:	read the buffer status or buffer data
WRITE:	write the buffer commands
SELECT:	wait for DMA finished
MMAP:	map the buffer to user space
OPEN:	opens the buffer device
RELEASE:	close the buffer device

The buffer device can be opened from different programs simultaneously. But only one program should use the WRITE and SELECT interface. If the Buffer-device is opened the first time, it allocates 2*1280*1024 bytes in the PC main memory. This memory is segmented in continuous blocks of 64kBytes. If this is not possible the blocks may be smaller. After allocating the transfertable of the buffer is initialized with a small image size.
When closing the Buffer-device, the allocated memory is released.

The MMAP-interface supports the *mmap()* system call. The first read or write operation to the returned addresses will take a longer time as the following, because the page table of the process has to be rebuilt.

The SELECT-interface supports the *select()* system call. You must use the SEL_EX parameter to connect to the buffer device. Within one select call you can wait for multiple buffers. The select call returns, if the DMA-Transfer to one of the buffer is finished.

Buffer-Device Control Calls

Different commands can be given to the Buffer-device with the WRITE-interface. Each command is represented by an ASCII-string. The syntax must exactly match the strings given in the following description. If an error occurs a value <=0 is returned else the bytes written to the Buffer-device.

Setting of next READ command:

pcc:c0 *R*ead command 0 is set. The next read on the Buffer-device will return the buffer-status= struct DEVBUF.

pcc:c1 Read command 1 is set. The next read on the Buffer-device will return the buffer-data.

pcc:c2 *R*ead command 2 is set. The next read on the Buffer-device will return the device-status= struct VXDBOARDVAL.

Setting for DMA and Flags reset

All following commands are setting the READ-command to 0.

pcc:c4 Set a image buffer at the end of the DMA-queue. If no actual DMA-buffer is set, this buffer is set as the actual DMA-buffer. If the camera is started a DMA-Transfer is started. This command can have additional parameters, each separated with a blank (0x20Hex). If no additional parameter is found the default values are used.

size Bytes to transfer.
Default 640*480*2

offset offset in image buffer (start address).
Default: 0

data extra data
Default: 0

pcc:c5 Removes a image buffer from the DMA-queue.

pcc:c6	Set new size. If no additional parameter is found the default value is used.
size	Block size of buffer in Byte. Default: 640*480*2
pcc:c7	Set new block size. If no additional parameter is found the default value is used.
size	Block size of buffer in Byte. Default: 65536 (64kByte)
pcc:c8	Clear the select, select write-done Flag in the Buffer-status word.
pcc:c9	Clear the write-done Flag in the Buffer-status word.

With the READ-interface data can be read back from the Buffer-device. With some of the above WRITE-commands one can select which data should be read. With the *lseek()* system call one can jump to arbitrary locations in the data-buffers.

Driver IO-Device

The driver IO-Device serves as an interface to all camera functions i.e. setting and reading parameters, start, stop etc. It can also be used to start DMA-Transfers to image buffers as well as programming the PCI-Controller-Board.
For every board a set of parameters is available (struct VXDBOARDVAL).
The IO-Device supports the following file–operations

READ:	read the device status = struct VXDBOARDVAL
IOCTL:	all commands described later on
SELECT:	wait for certain actions
MMAP:	map one allocated buffer at a time
OPEN:	opens the device
RELEASE:	close the device

If the device is opened, the presence of the specified board is checked and the access rights are validated. The open function returns –ENODEV if no board is found or –EBUSY if access is denied. The release function waits until all operating actions are finished and sets the board to the IDLE-state.

All commands to the PCI Interface board are sent through the IOCTL-interface. For this purpose a DeviceIoControl function was built. You can find this function in the library (misc_io.c).
This function uses the system call ioctl(). All usable commands are defined in the header file sencamc.h, all supported commands are described below with valid input and output parameters.
On error all commands return a standard error and set an internal value to one of the internal error codes defined in the Header file sencame.h.

Common functions after loading driver

GET_TIMEMS

Reads System time in ms.

Input:
Output: DWORD time

GET_TIMEUS

Reads System time in µs.

Input:
Output: DWORD timesec
DWORD timeus

timesec: second part of system time
timeus: microsecond part of system time

INIT_BOARD

Initializes the board and camera, this must be called before one can do any other interaction with the board. Camera parameters are set to default values. Version numbers of the board is read-out and checked for conformity.

Input: DWORD boardnr
Output: DWORD hold
DWORD base_address
DWORD pci_int
DWORD hdevice
DWORD process

hold: includes board type and board number
base_address: PCI-Controller base address
pci_int: PCI-Controller Interrupt number
hdevice: open count of driver
process: process uid

Common functions after INIT_BOARD

The following functions can only be called after INIT_BOARD has been done.

FREE_BOARD

Close the board and free all IOCTL-allocated image buffers for this board. Reset all Hardware on the board. INIT_BOARD must be called before working with the board again.

Input:
Output:

GET_PROZVERS

Get the version strings (16 bytes) of the board and the FPGA's. The versions are readout in the INIT_BOARD call. OUTBUF must be large enough to hold 16 bytes.

Input:	DWORD boardnr
	DWORD devnr
Output:	16 bytes version string

boardnr:	Number of board starting with 1.
devnr:	device number
	4 = Board Hardware
	6 = FPGA devices

SET_TIMEOUT

For doing physical IO and DMA-Transfers the driver sets two independent timers, which produce a timeout error if the action does not finish in the regular time. For Head-Status check a third timer operates periodically. With this call the timeout values can be set. All values are in ms. The default values should not be changed.

Input:	DWORD dmatimeout
	DWORD iotimeout
	DWORD headtimeout
Output:	

NVRAM_RB

Read a byte from the NVRAM-interface of the PCI Controller.

Input:	DWORD adr
Output:	DWORD value

adr:	address to read from
value:	readback value from address adr

NVRAM_WB

Writes a byte to the NVRAM-interface of the PCI Controller.

Input: DWORD adr
DWORD value
Output:

adr: address to write
value: value to write only lowest byte

SET_DRIVER_EVENT

Enables or disables the driver events. Actual no event is defined in the driver.

Input: DWORD event_com
Output:

event_com: choose event and command
event_com&0x0000FFFF=0 head event
event_com&0x80000000=0 enable event
event_com&0x80000000!=0 disable event

SET_DRV_VAR

Set the driver parameters pcc_message, pcc_process, pcc_demo. (see also Driver Basics)

Input: DWORD pcc_message
DWORD pcc_process
DWORD pcc_demo
Output:

GET_DRV_VAR

Get actual values of the driver parameters pcc_message, pcc_process, pcc_demo. (see also Driver Basics)

Output: DWORD pcc_message
DWORD pcc_process
DWORD pcc_demo

Buffer functions

The buffer functions are handling image buffers with the IOCTL-Interface of the driver. These buffers can be used in parallel to the image buffers, which are allocated directly with file-operations. Some functions can work with both kinds of buffers. The amount of image buffers one can allocate is limited.
All functions which can access image buffers from the buffer device, have added an item picbuf=0x8000000+num. Where num is the number of the buffer device (1 – 32).
All this functions can also be accessed with the handle of the buffer-device. In this case the given buffer number bufnr is not used.

GET_FREE_DEVBUF

Searches for the next free buffer and returns its number.

Input:
Output: DWORD bufnr

bufnr: buffer number (0…BUFCOUNT)

GET_STATUS_DEVBUF

Returns the status of the image buffer by copying the struct DEVBUF.

Input: DWORD mode
DWORD bufnr
Output: struct DEVBUF

mode: 0= get status of buffer bufnr
1= get status of the actual DMA-buffer
2= get status of the actual Mapped-buffer
bufnr: number of buffer
picbuf

SET_PAGESIZE_DEVBUF

Set the block size of the buffers. Default block size is 64Kbyte. Block size can be set in multiples of 4Kbyte.

Input: DWORD bufnr
DWORD block size
Output: DWORD error code
struct DEVBUF

bufnr: number of buffer
picbuf
block size: size of Blocks in Byte

ALLOCATE_DEVICE_BUFFER

Allocates a image buffer of given size.

Input:	DWORD bufnr
	DWORD size_in
Output:	DWORD size_out
bufnr:	number of buffer
	picbuf
size_in:	buffer size in Byte, which should be allocated
size_out:	buffer size in Byte, which has been allocated

CLEAR_WORKING_BUFFER

Removes a image buffer from the DMA-queue.

Input:	DWORD bufnr
Output:	
bufnr:	number of buffer
	picbuf

SET_WORKING_BUFFER

Set a image buffer at the end of the DMA-queue. If no actual DMA-buffer is set, this buffer is set as the actual DMA-buffer. If the camera is started a DMA-Transfer is started.

Input:	DWORD bufnr
	DWORD tr_size
	DWORD offset
	DWORD data
Output:	
bufnr:	number of buffer
	picbuf
tr_size:	Bytes to transfer
	actualxsize*actualysize if no input
offset:	offset in image buffer (start address)
	0 if no input
data	extra data
	0 if no input

SET_MAP_BUFFER

Set one of the image buffers to the actual map buffers, so it can be mapped into the user space.

Input:	DWORD bufnr
Output:	
bufnr:	number of buffer

SET_TRANSFERTAB_BUFFER

Set the transfer table of a image buffer. The buffer must have been allocated with size 0. One entry in the transfer table consists of the physical memory start address of continuous DMA-block and the size of this block. The total size of all entries must also be given. The last entry in the table must have the values address = -1 and size = –1.

Input:	DWORD bufnr
	DWORD total size
	DWORD tabnum
	tabnum table entries DWORD,DWORD
Output:	
bufnr:	number of buffer
total size	total size in bytes of the transfer
tabnum:	number of entries in the table

FREE_DEVICE_BUFFER

Free the allocated memory of the buffer and allow a new allocation. The bufnr is illegal from now on, until it is given back from a ALLOCATE_DEVICE_BUFFER again.

Input:	DWORD bufnr
Output:	
bufnr:	number of buffer
	picbuf

Camera functions

The following functions can only be called after INIT_BOARD has been done.

COMMAND

Write a command to the PCI-Board.

Input: DWORD command
Output:

command: command to write

PCO_READ

Read one dword from the PCI-Board.

Input: DWORD address
Output: DWORD data

address: adress to read from
data: data read from address

PCO_WRITE

Write one dword to the PCI-Board.

Input: DWORD address
DWORD data
Output:

address: adress to read from
data: data to write

RELOAD_FPGA

Load and reset the FPGA on board.

Input: byte *tab
Output:

tab: array of bytes to load in the FPGA

LOAD_COC

Load the COC memory on the board.

Input: word *tab
Output:

tab: array of valid COC-values

FRAME_IN

Read special status register on board.

Input:
Output: DWORD data

data: status register read in

TEST_BUSY

Test busy state of board and returns busy flag.

Input:
Output: DWORD data

data: busy state of board

START

Enable DMA-Transfer from board memory to PC-memory. Call startdma().

Input:
Output:

STOP

Disable DMA-Transfer from board memory to PC-memory.

Input:
Output:

Installation Instructions

Copy the SW_SCDPSDKLNX_0102.tar.gz to a distinct directory (e.g. PCOSDK).
Use "tar -xpvzf SW_SCDPSDKLNX_0102.tar.gz" to get the above files.

To get the driver and library
Use "tar -xpvzf sensdk1_02_02.tar.gz".

After this you will find the following new directories and files as noted.

Each directory contains a makefile and source files for different parts of the SDK. The makefiles build a debug and release version of all parts in the subdirectory "debug" respectivly "rel". Errors during compilation create error files in the "make_err" directory for each source file.

Driver compilation and driver installation should be done as root.
After uncompressing change to directory .../sen1_02_02 and do:
make driver
make install
reboot

Make install will call script file sencam_load.

The script sencam_load must be called only once after building the driver. It removes older versions of driver module pco52x.o and copies the new built to lib/modules/'uname -r'/pco.
An entry for the driver in modules.conf is created.
File pb52xx.bit is copied to lib/modules/'uname -r'/pco.
The script sencam_load creates file pb525.sh in directory /etc/profile.d, which exports environment variable PCOPB. (see File pb5250xx.bit ...)
The script askes for the number of devices and buffers and creates the devnodes in the /dev directory according to the user input.
Startparameters of the pco52x module (sen_major and sen_message) can be given as command line parameters.

The script sencam_unload undos all the actions done from sencam_load.

File pb5250xx.bit and environment variable PCOPB:
When the library function sen_initboard() is called and a PCI525 board is found, the fpga's on board are loaded with the bitstream from file pb5250xx.bit. The current and one upper directories will be searched for finding the newest version of this file, where xx is the version number.
Too shorten this process or to start programs from any directory environment variable PCOPB should be set to the correct filepath i.e. "/usr/pco/sen1_02_02/pb525002.bit".

After the installation the following new directories will be generated. Each directory contains a makefile and the source files for different parts of the driver or library.

./sen1_02_02
Main makefile scripts and bitstream file.

./sen1_02_02/sendrv
Driver C–source-files and makefile to build the driver

./sen1_02_02/sendrvh
Header Files for the driver.

./sen1_02_02/senlib
Library C-source-files and header-files for the main functions of the SDK. Makefile to build the library.

./sen1_02_02/coc_i386
Library libsencoc.a = functions to built the COC. Is linked together with the senlib files to build the Library libsencam.

. /sen1_02_02/sentest
C-source-files and header files to build terminal applications for testing the board and buffer functions. Makefile to build the test-application.

Return Codes

Function ok	
0	no error, function call successful

Library Errors	
-1	initialization failed; no camera connected
-2	timeout in any function
-3	function call with wrong parameter
-4	cannot locate PCI card or card driver
-5	wrong operating system
-6	no or wrong driver installed
-7	IO function failed
-8	camera not connected or power off
-9	invalid camera mode
-10	reserved
-11	device is hold by another process
-12	error in reading or writing data to board
-13	invalid function call to driver
-14	cannot allocate DMA Buffer
-15	FPGA File not foubnd or load error
-16	DMA Timeout
-17	creating wait queue or event failed
-20	COC running, camera busy
-21	COC does not fit in board memeory
-22	Camera temperature failed
-23	Allocating memory failed
-24	Readout running
-25	Set/reset Buffer Flags failed
-26	Buffer is in use
-27	an error occurs in a system call
-28	DMA is running
-29	Open filehandle failed
-32	Newer version of driver neeeded
-33	one of extended status bits shows an error
-34	board memory has an error
-35	function not allowed with this ccdtyp
-36	error in DMA from board to memory
-37	error while reading from file
-38	error while writing to file
-39	error while clearing board buffer
...	

Driver Errors

Values returned from library. Error code returned directly from driver is (value+100)*-1.

-101	timeout in any driver function
-102	board is used from an other user or process
-103	Function is not allowed with this type of board
-104	Board is not initialized
-105	No PCI-Bios was found
-106	No PCI-Board with correct Vendor_ID and Device_ID was found
-107	Configuration of PCI-Board cannot be read
-108	Function is only allowed for IO_Device
-109	Memory allocation failed
-110	Camera does another job
-111	Camera is running, function not allowed
-112	Wrong parameter in function call
-113	Connection to Camera-Head lost
-114	Verify of Camera-Head data failed
-115	Board cannot manage the connected Camera Head
-116	Initialization of on board FPGA failed
-117	Write to board located NVRAM failed
-120	Function is called with too less parameters
-121	Buffer is too small for all return values
-130	Image-Buffer is not prepared for DMA-Transfer
-131	A DMA-Transfer is started on this Image-Buffer
-132	Another process has exclusive access to this Image-Buffer
-133	Image-Buffer cannot be found
-134	Deallocating of the Image-Buffer failed
-135	No more Image-Buffers can be allocated, Maxcount reached
-136	No more Image-Buffers can be allocated, Maxalloc reached
-137	Allocating Memory for Image-buffer failed
-138	Allocating Memory for Image-buffer failed
-139	Allocating Memory for BWLUT failed
-140	Allocating Memory for PageTable failed
-148	No Event Handler defined for this device
-149	Deleting the Event Handler for this device failed
-156	Start of the Interrupt Handler for this device failed
-157	Stop of the Interrupt Handler for this device failed
-158	No Interrupt Handler is installed for this device
-164	DMA-Transfer has a Timeout
-165	No Image-Buffer is defined for this DMA-Transfer
-168	Size of Image-Buffer is to small for the DMA-Transfer
-169	An Error occurred during DMA-Transfer
-170	DMA-Transfer is running, function not allowed

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