SDK

Software Development Kit

Linux 2.2 , Linux 2.4

Version 1.15.14

pixelfly

pixelfly

pixelfly

pco.
imaging

Contents

Driver Functions

Software Development Kit for PixelFly under Linux

Basics

Camera and PCI-Board control is managed on two levels, represented by the 32 Bit library **libpfcam** and the driver **pccam.o**.

This description includes in detail all functions of the upper level of the SDK (libpfcam) and some notes how to use it. The important parts of the driver (pccam.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 ability to lwrite error messages to any output stream or with the syslog() call.

Hardware and Library
The PixelFly PCI-Board does not have a large memory, to grab one or more pictures. The pictures read out from CCD will be sent directly by a Master-DMA transfer (without interaction of the PC-CPU) to the main memory of the PC.
This requires a special memory management for the picture buffers. Therefore these picture buffers are allocated in kernel memory. In the SDK you will find **Memory Control Functions** to allocate and free picture buffers, mark one or more buffers for grabbing pictures (set the buffers in a queue) and at least map the picture buffer to the normal user space.
Because each picture 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 picture buffers to your memory, while transferring data from the camera to another picture buffer.
For each board a maximum of 31 special picture buffers are reserved, which are accessible with normal file IO-functions directly. With the **Buffer Control Calls** you can manage these picture buffers.
Most of **Memory Control Functions** can also be called with the handle of one of the special picture buffer.

On the board there are three processors, one (PLUTO) for communication with the PC-CPU via PCI-Bus, one (CIRCE) for handling camera timing and one (ORION) for communication with the external world via the highside-drivers and optocouplers. You can write your own programs for the last one, to manage your special tasks.
In the SDK you will find **Camera Control Functions** to set camera parameters like exposure time, binning, … . You can start and stop the camera readout at any time, give trigger commands, write data to and read data from the ORION processor and get status information from the PCI-Board and the camera head.

Because interaction with the ORION processor could be time critical, a table of ORION commands for each picture buffer can be set which is executed when the readout to this buffer is done. I.e. this option can be used to get timestamps for each picture.

The **General Control Functions** are used to open, close and reset the driver. The driver can manage up to four boards. So if more than one PixelFly 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 to connect to a given board if the board was opened before from another process with different access rights.

The driver also includes a proc interface, which gives detailed description on the current state of the PCI-board and buffers.

General Control Functions

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

This command initializes the PCI-Controller-Boards 0…3, the board functions and the hardware is tested automatically when opening the driver for the first time. The board parameters are set to the default values.
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 to the driver you can also call the system-function open(…) with the accurate device name. INITBOARD() must then be called to initialize the board.
The filehandle of the driver is needed in any library function to be in connection with the wanted 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.
After initializing the board a short test routine is executed. This routine read head temperature, set the camera mode, start camera, grab one picture, stop camera. If any of this functions fail, initialisation is done again and an new test loop is done. If not successful after 5 loops the function returns with error

IN

board		number of the PCI-Controller-Board
	=	0…3
hdriver		pointer to filehandle
*hdriver	=	NULL
		the driver is opened and the board initialized
*hdriver	=	filehandle of opened driver
		the board is initialized

OUT

*hdriver	=	filehandle of the opened driver

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

This command initializes the PCI-Controller-Boards 0…3. Unlike the INITBOARD() function the test routine after initialisation is not executed. Otherwise there is no difference between the both function. Parameters see INITBOARD().

int **pcc_closeboard** (HANDLE *hdriver)

This command resets 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 **pcc_freeboard** (HANDLE hdriver)

This command resets the PCI-Controller-Board. It does not close the driver. To close the driver you must call system-function close(…).

IN

hdriver	=	filehandle of opened driver

void **pcc_set_syslog_facility** (int m)

Set 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

m	=	syslog facility

void **pcc_get_errortext** (int err, char *text,int length)

This command returns an textstring for each error-code returned from one of the library functions.

IN

err	=	errorcode from any function
text	=	pointer to buffer which will receive the textstring
length	=	size of buffer

OUT

*text	=	textstring for errorcode

Camera Control Functions

int **pcc_get_boardpar** (HANDLE hdriver, void *ptr, int len)

This command returns len status **bytes** from the BOARDVAL structure of the board. In the headerfile pccamdef.h there are macro definitions to extract certain information out of this structure.
(The structure BOARDVAL is defined in the driver.)

IN

hdriver	=	filehandle of opened driver
ptr		pointer to memory address
	=	address of allocated memory
len		number of bytes to read
	=	4…size of allocated memory

OUT

*ptr	=	values of structure BOARDVAL

int **pcc_trigger_camera** (HANDLE hdriver)

This command releases a single exposure in the software trigger mode.

IN

hdriver	=	filehandle of opened driver

int **pcc_start_camera** (HANDLE hdriver)
int **pcc_stop_camera** (HANDLE hdriver)

These commands start and stop the camera. Before setting any of the camera parameters, like binning or gain etc. the camera has to be stopped with STOP_CAMERA. When this command returns without error then the CCD is cleared and ready for setting new parameters or starting new exposures.
A new exposure can be released with **pcc_start_camera** and a hardware or software trigger.

IN

hdriver	=	filehandle of opened driver

int **pcc_set_mode** (HANDLE hdriver, int mode, int explevel, int exptime, int hbin, int vbin, int gain, int offset, int bit_pix, int shift)

This command sets the parameters of the next exposures.
It cannot be called if the camera is running. All parameters are validated and error WRONGVAL is returned if one of the parameters has a invalid value.

IN

hdriver	=	filehandle of opened driver

mode mode of camera

=	0x10	single asnyc shutter hardware trigger
=	0x11	single asnyc shutter software trigger
=	0x20	double shutter hardware trigger
=	0x21	double shutter software trigger
=	0x30	video mode hardware trigger
=	0x31	video mode software trigger
=	0x40	single auto exposure hardware trigger *
=	0x41	single auto exposure software trigger *

In video mode a sequence of exposures is started with the next trigger, in all other modes only one exposure is released by a hardware or a software trigger.
Timing: making an exposure and then readout the CCD. After this a new trigger is accepted.
For double shutter and auto exposure modes (0x20, 0x21, 0x40*, 0x41*) special camera versions are necessary!

*** Special command! Not available in standard SDK!**

explevel * set level in % at which time to stop the auto exposure mode. Only valid if mode is set to auto exposure (0x40, 0x41).

explevel *	= 0…255
	step width = 0.5%
	200 = 100% = 4095 counts

*** Special command! Not available in standard SDK!**

exptime set exposure time of the camera

single async mode (0x10, 0x11)

exptime	= 10…10000µs

*up to 65535µs with latest Board-SW-revisions
video mode (0x30, 0x31)

exptime	= 1…10000ms

hbin set horizontal binning and region of the camera

hbin	= 0x00000	horizontal x1 normal readout
	= 0x00001	horizontal x2 normal readout
	= 0x10000	horizontal x1 wide readout
	= 0x10001	horizontal x2 wide readout

wide readout include 8 dark pixel at the beginnig of each line.

vbin set vertical binning of the camera

vbin	= 0	vertical x1
	= 1	vertical x2
	= 2	vertical x4*

* only avaiable for VGA-cameras

gain set gain value of camera

gain	= 0	low gain
	= 1	high gain

offset not used

bit_pix set how many bits per pixel are transferred

bit_pix	= 12
	= 8

bit_pix = 12:
12 bits per pixel, no shift possible. Two bytes with the upper four bits set to zero are sent. Therefore two pixel values are moved with one PCI (32 bit) transfer.

bit_pix = 8:
8 bits per pixel, shift possible. 8 bit values are generated with a programmable barrel shifter from the 12 bit A/D values. Therefore four pixel are moved with one PCI transfer. This half's the pixel data per image and frees the PCI bus

shift set the digital gain value
Only valid in the 8 bit per pixel mode!

shift	= 0	8 bit (D11...D4), digital gain x1
	= 1	8 bit (D10...D3), digital gain x2
	= 2	8 bit (D09…D2), digital gain x4
	= 3	8 bit (D08…D1), digital gain x8
	= 4	8 bit (D07...D0), digital gain x16
	= 5	8 bit (D07...D0), digital gain x16

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

This command is only avaiable with latest Board SW-revisions. and in mode single async shutter (0x010 or 0x011). It can be called while the camera is running. The exposuretime is changed at the next possible frame.

IN

hdriver	=	filehandle of opened driver
time	=	exposuretime 10…65535µs

int **pcc_wrrd_orion** (HANDLE hdriver, int cmnd, int *data)

This command writes a command to the ORION-controller and reads back the data value sent.

IN

hdriver	=	filehandle of opened driver	
cmnd		command to send (only the low byte is valid)	
implemented commands in ORION 1.14:			
	=	0x10	rd_portA
	=	0x11	rd_portB
	=	0x13	rd_portD
	=	0x20	rd_portC

OUT

*data	data sent back, by the ORION-controller

int **pcc_wrrd_orion_block** (HANDLE hdriver, unsigned char *cmnd, unsigned char *datain, unsigned char *dataout, int len);

This command writes a block of commands to the ORION-controller and reads back the data value sent, for each command. The block is executed as a contiguous sequence. If the Block is interrupted from an Orion-Command at end of DMA-Transfer, it starts again with the first command.

IN

hdriver	=	filehandle of opened driver
cmnd	=	address of buffer of commands to execute
datain	=	address of buffer of data send to the ORION-controller with each command
len	=	number of commands to sent

OUT

*dataout	=	data sent back by the ORION-controller with each command

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

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

IN

hdriver	=	filehandle of opened driver

OUT

*ccdxsize	= x-resolution of CCD
*ccdysize	= y-resolution of CCD
*actualxsize	= x-resolution of picture
*actualysize	= y-resolution of picture
*bit_pix	= bits per pixel in picture (12 or 8 bit)

int **pcc_read_temperature** (HANDLE hdriver, int *ccd)

This command returns the actual CCD-temperature.
The temperature range is from -55°C to +125°C.

IN

hdriver	=	filehandle of opened driver

OUT

*ccd	=	temperature in °C

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

This command returns the exposure time of the last exposed picture. For video mode in ms all others in µs.

IN

hdriver	=	filehandle of opened driver

OUT

*time	=	time of last exposure

int **pcc_read_version** (HANDLE hdriver, int typ, char *vers, int len)

This command returns len characters of the version string from the specified typ. There are version strings for each processor, board hardware, head hardware and programmable CPLD's on board. All version strings consist of ASCII-characters.

IN

hdriver	=	filehandle of opened driver	
typ	=	selection	
	=	1	PLUTO
	=	2	CIRCE
	=	3	ORION
	=	4	Hardware
	=	5	Head
	=	6	CPLD
vers		pointer to memory address	
	=	address of allocated memory	
len		bytes to read	
	=	size of allocated memory	

OUT

*vers	=	version string of selected typ

int **pcc_read_eeprom** (HANDLE hdriver, int mode, int adr, char *data)

This command reads one byte from the EEPROM at the address adr.

Do not call this command while the camera is running!

IN

hdriver	=	filehandle of opened driver	
mode	=	0	HEAD-EEPROM
	=	1	CARD-EEPROM
adr	=	address of byte to read (0…255)	

OUT

*data	=	byte to read

int **pcc_write_eeprom** (HANDLE hdriver, int mode, int adr, char data)

This command writes one byte to the EEPROM at the address adr.

Do not call this command while the camera is running!

IN

hdriver	=	filehandle of opened driver	
mode	=	0	HEAD-EEPROM
	=	1	CARD-EEPROM
adr	=	address of byte to write (0…127)	
data	=	byte to write	

int **pcc_set_timeouts** (HANDLE hdriver, DWORD dma, DWORD proc)

This command can set timout values for card-io and dma.

IN

hdriver	=	filehandle of opened driver
dma	=	timeout in milliseconds for dma
proc	=	timeout in milliseconds for card-io

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

This command enables or disables the select function on the driver IO-interface for different events. Because the driver allocated buffers also invoke this event on picture done, one has to prove the reason of the event
The only defined driver event is the head event.

IN

hdriver	=	filehandle of opened driver
mode	= low word	0x0000 = head event
	= high word	0x0000 = open and enable event
	= high word	0x8000 = disable event
	= high word	0xC000 = disable and close event

int **pcc_set_drv_param** (HANDLE hdriver, DWORD *param, int count)

Set the driver parameters, which control kernel debug messages, open access requirements and demo mode. Normally this parameters are given at loading time of the driver (see Driver Basics).

IN

hdriver	=	filehandle of opened driver	
*params	=	table of parameters	
		1. = pcc_message	= 0 - 5
		2. = pcc_process	= 0 - 3
		3. = pcc_demo	= 0 - 1
count	=	number of bytes in param	

int **pcc_get_drv_param** (HANDLE hdriver, DWORD *param, int count)

Get the actual driver parameters values.

IN

hdriver	=	filehandle of opened driver
count	=	number of bytes to read

OUT

*params	=	parameters read back
		1. = pcc_message
		2. = pcc_process
		3. = pcc_demo

Memory Control Functions

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

This command returns len status bytes from the buffer structure DEVBUF of the specified buffer bufnr. In the headerfile pccamdef.h there are macro definitions to extract certain information out of this structure. (The structure DEVBUF is defined in the driver.)

IN

hdriver	=	filehandle of opened driver
bufnr	=	picture-buffer number returned from pcc_allocate_buffer()
	=	0x80000000+num num is the number of the buffer device from 1 to 32
or		
hdriver	=	filehandle of special picture 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

void **pcc_get_bufferstatustext** (int status, char *text,int length)

This command returns an textstring for the status of the buffer. Status is the first ‘integer’ in structure DEVBUF returned from pcc_get_buffer_status().

IN

status	=	status
text	=	pointer to buffer which will receive the textstring
length	=	size of buffer

OUT

*text	=	textstring for status

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

This command allocates a buffer for the camera in the PC main memory.

The value of size has to be set to the number of **bytes, which** should be allocated. The return value of size might be greater because the buffer is allocated with a certain block size. To allocate a new buffer, the value of bufnr must be set to –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
	=	picture-buffer number returned from previous pcc_allocate_buffer(), to reallocate with different size, or
	=	0x80000000+num num is the number of the buffer device from 1 to 32
or		
hdriver	=	filehandle of special picture buffer
bufnr	=	not used
*size		size of picture-buffer in byte

OUT

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

int **pcc_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 pcc_remove_buffer_from_list first.

IN

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

int **pcc_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	=	picture-buffer number returned from pcc_allocate_buffer() or
	=	0x80000000+num num is the number of the buffer device from 1 to 32
or		
hdriver	=	filehandle of special picture buffer
bufnr	=	not used

int **pcc_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 **pcc_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 picture released from the camera. If a picture 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.

IN

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

recommended size for 12 bit data
actualxsize*actualysize*2
recommended size for 8 bit data
actualxsize*actualysize

Get actualxsize and actualysize with function GET_SIZES(…).

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.
If transfer size is greater than camera size the transfer will produce a timeout and generate an error.
With offset set to other values then 0, you can have more small camera pictures in one large buffer.
Size must always be a value greater than 4096.
Offset must be a multiple of 4096.

int **pcc_set_buffer_event** (HANDLE hdriver, int bufnr,int mode);

This command enables or disables the select function on the driver IO-interface for driver allocated buffers. This event is invoked if a picture is done on one of the buffers or one of the buffers is removed from the list. After returning from the select, one has to prove the reason of the event Also a head event can occur.

IN

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

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

This command controls the access from user-space to the driver allocated buffers, which are allocated in kernel space. Before mapping and unmapping (mmap() and munmap()) and also before any read or write operation to the buffer data is done, this function has to be called.

IN

hdriver	=	filehandle of opened driver
bufnr	=	picture-buffer number returned from pcc_allocate_buffer()

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

This command maps a buffer to user space address “linadr”. Mapping is done with mmap() command permissions set to PROT_READ. A buffer can not be mapped a second time with other input parameters. Buffer device buffers can not be mapped with this function instead use mmap() on the buffer device.

IN

hdriver	=	filehandle of opened driver
bufnr	=	picture-buffer number returned from pcc_allocate_buffer()
size	=	number of bytes to map
offset	=	offset in bytes from start of buffer

OUT

*linadr	=	userspace address of mapped buffer

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

This command unmaps a buffer, which was previously mapped with pcc_map_buffer().

IN

hdriver	=	filehandle of opened driver
bufnr	=	picture-buffer number returned from pcc_allocate_buffer()

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

This command returns the mapping parameters of a buffer, which was previously mapped with pcc_map_buffer().

IN

hdriver	=	filehandle of opened driver
bufnr	=	picture-buffer number returned from pcc_allocate_buffer()

OUT

*size	=	size in bytes of mapping
*offset	=	offset in bytes of mapping
*linadr	=	userspace address of mapped buffer

Commands for the ORION processor

The driver calls the ORION processor automatically, shortly after a DMA transfer is done.
With the following functions you can set the commands and data byte, which belongs to every command.
You can send up to 16 commands. If the driver finds a command in the command table, it will catch the data_in byte from the same table position and send it to the ORION processor.
After the ORION has finished the command and has written back its data byte, this byte will be stored in the data_back table at the same table position, from where the command is read out.
If the command has the value 0x00 or position 16 is reached, the driver will stop sending commands.
Each buffer has its own table, so you can define different commands for each buffer.
When allocating a buffer all tables are set to 0x00, so no commands are sent to the ORION processor.

int **pcc_set_orion_int** (HANDLE hdriver, int bufnr, int mode, unsigned char *cmnd, int len)

This command writes len bytes to the command or data table for the driver internal ORION call.

IN

hdriver	=	filehandle of opened driver
bufnr	=	picture-buffer number returned from ALLOCATE_BUFFER() or
	=	0x80000000+num num is the number of the buffer device from 1 to 32
or		
hdriver	=	filehandle of special picture buffer
bufnr	=	not used
mode	=	1 orion data_back
	=	2 orion data_in
	=	3 orion command
cmnd	=	address of buffer of commands or data to set, maximal 16 bytes
len	=	length of the buffer

int **pcc_get_orion_int** (HANDLE hdriver, int bufnr, int mode, unsigned char *cmnd, int len)

This command reads len bytes from the command or data tables for the driver internal ORION call.

IN

hdriver	=	filehandle of opened driver
bufnr	=	picture-buffer number returned from ALLOCATE_BUFFER() or
	=	0x80000000+num num is the number of the buffer device from 1 to 32
or		
hdriver	=	filehandle of special picture buffer
bufnr	=	not used
mode	=	1 orion data_back
	=	2 orion data_in
	=	3 orion command
cmnd	=	address of buffer
len	=	length of the buffer

OUT

*cmnd	=	commands or data read

Driver Basics

The driver is a loadable LINUX module. It includes the IO-interface to the PixelFly PCI-Controller-Board and preparation and maintenance of picture buffers for each board in the PC main memory. The driver can handle up to four boards and up to 31 picture buffers for each board. The driver communicates with the PLUTO-processor on the PCI-board through an interrupt controlled IO-interface.

Major and Minor numbers

The Major number of the module is currently defined to 0 in the header-file pccam.h, which means that the Majornumber is dynamical assigned to the driver. The Major number should only be changed to a fixed number (i.e.100dec) when the driver is 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 picture 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 pccamx and pccbufx_yy, where x is the board number and yy is the picture buffer number. I.e. build the nodes for two boards each with two picture buffers. (MJ = Major number, after loading the driver you can get its MJ from /proc/devices, look for pccam_dev in the output from *cat /proc/devices*).

mknod /dev/pccam0 c MJ 0 (IO-Device board 0)
mknod /dev/pccbuf0_01 c MJ 1 (first picture buffer board 0)
mknod /dev/pccbuf0_02 c MJ 2 (second picture buffer board 0)

mknod /dev/pccam1 c MJ 64 (IO-Device board 1)
mknod /dev/pccbuf1_01 c MJ 65 (first picture buffer board 1)
mknod /dev/pccbuf1_02 c MJ 66 (second picture buffer board 1)

To increase functionality of the driver an additional feature was included in driver-versions above 1.15. Each device gets now an separate major number. To run the driver as it was before add the switch pcc_multi=x , when starting the driver.

When loading the module (i.e. *insmod /PATH/pccam*) 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 picture buffer.

When removing the device all internal buffers and eventually allocated pictures buffer are freed.

Startparameters

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 IO. So it runs without having a PCI Interface board installed.	
	x= 0	demo mode disabled (default)
	X= 1	demo mode enabled
pcc_message=x	set message level of driver The driver outputs debug messages. Greater values send more messages.	
	x= 0	no messages (default)
	x= 1..5	messages send
pcc_process=x	set open access requirements	
	x= 0	one user, open multi (default)
	x= 1	one user, one open
	x= 2	one process, open multi
	x= 3	any user, open multi
pcc_initlev=x	set the different levels for initialisation of the board. The heigher the number is, the less is the number of tests made.	
	x= 4	no test for head
	x= 3	no test of headparameters
	x= 2	no setting of camera default values
	x= 1	no timer for head disconnection
	x= 0	(default)
pcc_multi=x	number of devices which get the same Major number	
	x= 0	different Major number for each device. (default)
	x= 1..4	number of devices which get same Major number.
pcc_majortab=a,b,c,d		
	table of Major numbers if pcc_multi is 0 If explicit values for each device are set the table must include values for all installe devices.	
	a=-1	use predefined Major number and increase this number for each device found(default)
	a=0..256	number for first device
	b=0..256	number for second device
	c=0..256	number for third device
	d=0..256	number for fourth device

Driver DMA Transfer

If the driver invokes a DMA-Transfer, some data out of the driver structures VXDBOARDVAL and DEVBUF is used. For better understanding, what is done 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 picture-buffer. The whole picture-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 address 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 IO-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 picture-buffer to which the next DMA-transfer should appear or is still running. In the list BUFLIST the picture-buffers for the following transfers are stored, with its actual parameters size and offset. The function call pcc_add_buffer_to_list or “pcc:c4” append the picture buffer at the end of the list. If actdma is currently empty, the picture buffer is directly assigned to actdma with function set_new_actdma(). This function proofs, if a picture buffer is in the list. If this is true it compares the actual settings of this picture-buffer with the parameters for the given transfer and calls function build_transfer() if parameters have changed. At least the picture-buffer is removed from the list.

Using Scatter/Gather techniques on the PCI-Board the main PC-Processor is free for other task during the whole 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 PLUTO-Processor. The PLUTO-Processor then does all necessary actions for this transfer when the camera starts to transmit data and sends a return code when the transfer is finished. This creates an interrupt on the host side, 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 picture-buffer.

To start a DMA-Transfer function startdma() is called. This function checks, whether the camera is already started, whether actdma references a picture buffer, whether this picture-buffer has a valid transfer table and there is no DMA-Transfer running to this buffer. If all is true, the transfer table of the selected picture-buffer is sent to the PLUTO-Processor. Function startdma() is called from the driver if either START_CAMERA is called from a process or a picture-buffer is set to actdma.

Driver Buffer-Device

The driver Buffer-Device maintains up to 31 picture buffers. It supports status reads, mapping of the buffer to users space, invoking 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 at one time. But only one program should use the WRITE and SELECT interface. If the Buffer-device is opened the first time, it allocates 2*size of the CCD-chip 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 transferable of the buffer it is initialized with picture size 2*ccdsize and offset 0.
When closing the Buffer-device, the allocated memory is freed.

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. If a buffer was mapped, please call *munmap(…)*, before closing the buffer device.

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 one can wait for multiple buffers. The select call returns if the DMA-Transfer to 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 conform to 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.
pcc:c3	Read command 3 is set. The next read on the Buffer-device will return the ORION-tables.

Setting for DMA and Flags reset

	All following commands are setting the READ-command to 0.
pcc:c4	Set a picture 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: actualxsize*actualysize
offset	offset in picture buffer (start address). Default: 0
data	extra data Default: 0
pcc:c5	Removes a picture 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: if camera is initialized
ccdxsize*ccdysize*2
else
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.

pcc:c9 Clear the write-done Flag in the Buffer-status.

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

pcc:c10 Set all ORION-tables. Additional Byte of command and data values must be added to this command. All data-bytes in the tables are set to 0x00, before writing the new values

data Bytes of data, 16Byte ORION data_back, 16Byte ORION data_in, 16Byte ORION command.

pcc:c11 Set ORION-command table. Additional Bytes of command values must be added to this command. All data-bytes in the table are set to 0x00, before writing the new values

data max. 16Bytes ORION command

pcc:c12 Set ORION-data_in table. Additional Bytes of data values must be added to this command. All data-bytes in the tables are set to 0x00, before writing the new values

data max. 16Bytes ORION data_in

pcc:c13	Set ORION-data_back table. Additional Bytes of data values must be added to this command. All data-bytes in the tables are set to 0x00, before writing the new values
data	max. 16Bytes ORION data_back

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 invoke DMA-Transfers to picture buffers as well as programming the processors and FPGA's on 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 running 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 (pccamio.c).
This function uses the system call ioctl(). All usable commands are defined in the header file pccamc.h, all supported commands are described below with valid input and output parameters.
All commands have at least one unsigned int (=DWORD) Output parameter which contains the internal error codes defined in the Header file pccame.h.

Common functions after loading driver

GET_TIMEMS

Reads System time in ms.

Input:	
Output:	DWORD error code
	DWORD time

GET_TIMEUS

Reads System time in µs.

Input:	
Output:	DWORD error code
	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, connection to the head is tested and valid head parameters are readout. Version numbers of the on board processors and FPGA's are readout and checked for conformity.

Input:	DWORD boardnr
Output:	DWORD error code
	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 picture buffers for this board. Reset all Hardware on the board. INIT_BOARD must be called before working with the board again.

Input:	
Output:	DWORD error code

SAVE_DEVICE

Saves the actual state of the board and driver. The outbuf buffer must be large enough to hold the error code and all parameters of the struct VXDBOARDVAL.

Input:	
Output:	DWORD error code struct VXDBOARDVAL

RESTORE_DEVICE

Restores the board to the state, which is given as INPUT-Parameter.

Input:	struct VXDBOARDVAL
Output:	DWORD error code

GET_BOARD_PAR

Get Parameters of board. If single mode is used one can get single DWORDS out of VXDBOARDVAL, else the size of the Output-buffer defines how many DWORDS from VXDBOARDVAL are copied.

Input:	DWORD mode DWORD offset
Output:	DWORD error code struct VXDBOARDVAL as whole or in parts
Single mode:	
Output:	DWORD error code DWORD value
mode=1:	Single mode
offset	Offset in DWORDS in struct VXDBOARDVAL

GET_PROZVERS

Get the version strings (16 bytes) of the processors, the head and the FPGA's. The versions are readout in the INIT_BOARD call. OUTBUF must be large enough to hold 20 bytes.

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

boardnr:	Number of board starting with 1.
devnr:	device number
	1 = PLUTO processor
	2 = CIRCE processor
	3 = ORION processor
	4 = Board Hardware
	5 = Head 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 runs 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:	DWORD error code

NVRAM_RB

Read a byte from the NVRAM-interface of the PCI Controller. This interface is used to upload the actual software of the processors and FPGA's.

Input:	DWORD adr
Output:	DWORD error code
	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. This interface is used to download new versions of the software of the processors and FPGA's.

Input: DWORD adr
DWORD value
Output: DWORD error code

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

SET_DRIVER_EVENT

Enables or disables the driver events. Actual only the head connect/disconnect event is defined in the driver.

Input: DWORD event_com
Output: DWORD error code

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: DWORD error code

GET_DRV_VAR

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

Output: DWORD error code
DWORD pcc_message
DWORD pcc_process
DWORD pcc_demo

Buffer functions

The buffer functions are handling picture buffers with the IOCTL-Interface of the driver. These buffers can be used in parallel to the picture buffers, which are allocated directly with file-operations. Some functions can work with both kinds of buffers. The number of picture buffers one can allocate is limited.
All functions which can access picture 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 error code
	DWORD bufnr
bufnr:	buffer number (0…BUFCOUNT)

GET_STATUS_DEVBUF

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

Input:	DWORD mode
	DWORD bufnr
Output:	DWORD error code
	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

SET_ORION_LIST

Set the command or data tables for the ORION processor.

Input:	DWORD bufnr DWORD mode 16 Byte of Data
Output:	DWORD error code
bufnr:	number of buffer picbuf
mode:	1= copy Data to ORION data_out table 2= copy Data to ORION data_in table 3= copy Data to ORION command table

GET_ORION_LIST

Get the command or data tables for the ORION processor.

Input:	DWORD bufnr DWORD mode
Output:	DWORD error code 16 Byte of Data
bufnr:	number of buffer picbuf
mode:	1= copy ORION data_out table to Data 2= copy ORION data_in table to Data 3= copy ORION command table to Data

ALLOCATE_DEVICE_BUFFER

Allocates a picture buffer of given size.

Input:	DWORD bufnr DWORD size_in
Output:	DWORD error code 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 picture buffer from the DMA-queue.

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

SET_WORKING_BUFFER

Set a picture 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:	DWORD error code

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

SET_MAP_BUFFER

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

Input:	DWORD bufnr
Output:	DWORD error code

bufnr:	number of buffer

SET_BUFFER_EVENT

Enable/disable event on this buffer.

Input:	DWORD bufnr
	DWORD mode
Output:	DWORD error code

bufnr:	number of buffer
mode:	0 disable
	1 enable

SET_TRANSFERTAB_BUFFER

Set the transfer table of a picture 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:	DWORD error code

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:	DWORD error code

bufnr:	number of buffer picbuf

Camera functions

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

SET_MODE

Set all camera parameters.

Input:	DWORD mode DWORD exposure level DWORD exposure time DWORD binning DWORD gain DWORD offset DWORD conversion
Output:	DWORD error code

bufnr:	number of buffer
mode:	camera modes (see library)
exposure level:	auto gain level (see library)
exposure time:	time to expose (see library)
binning:	0x00 h1,v1 0x01 h1,v2 0x02 h1,v4 0x80 h2,v1 0x81 h2,v2 0x82 h1,v4
gain:	analog gain (see library)
offset:	0
conversion:	digital gain (see library)

SET_EXP

Set camera exposuretime

Input:	DWORD exposure time
Output:	DWORD error code

READ_ATMEL

Read from PLUTO processor. Only temperature read can be done if camera is busy.

Input:	DWORD cmd
	DWORD data_in2
	DWORD data_in3
Output:	DWORD error code
	DWORD data_out2
	DWORD data_out3

cmd:	PLUTO command
data_in2:	data for Mailbox 2, if necessary
data_in3:	data for Mailbox 3, if necessary
data_out2:	data from Mailbox 2
data_out3:	data from Mailbox 3

WRITE_ATMEL

Write to PLUTO processor. No write can be done if camera is busy.

Input:	DWORD cmd
	DWORD data_in2
	DWORD data_in3
Output:	DWORD error code

cmd:	PLUTO command
data_in2:	data for Mailbox 2, if necessary
data_in3:	data for Mailbox 3, if necessary

WR_RD_ORION

Write to and read from the ORION processor. This can be done at nearly any time after INIT_BOARD. If a ORION command from the Orion list is running error code –110 is returned

Input:	DWORD cmd
	DWORD data_in
Output:	DWORD error code
	DWORD data_out

cmd:	lowest byte ORION command
data_in:	lowest byte ORION data send
data_out:	lowest byte ORION data back

TRIGGER

Send a trigger command to the camera.

Input:	
Output:	DWORD error code

START

Start the camera with the actual mode settings, if it is not busy yet. If the camera is started successfully and a picture buffer is waiting for data, this starts also the DMA transfer to this picture buffer.

Input:
Output: DWORD error code

STOP

Stop the camera, if it is busy. If the camera is sending data already, the stop is done after all data is sent.

Input:
Output: DWORD error code

Installation Notes

Copy the SW_PFSDKLNX_0107.tar.gz to a distinct directory (e.g. PCOSDK).
Use "tar -xpvzf SW_PFSDKLNX_0107.tar.gz" to get the files:
pfsdk1_15_13.tar.gz and this manual.
To get the driver and library
Use "tar -xpvzf pfsdk1_15_13.tar.gz".
The following new directories are created.
Each directory contains a makefile and the source files for different parts of the driver or library. The makefile builds a debug and a release version of each part, in the directories *debug* respectivly *rel*. The debug version writes a lot of Kernel messages, so it should not be used for normal working.
Errors during build create error files in directory *make_err,* for each source file.

./pfsdk1_15_14

Driver compilation and driver installation should be done as root.
After uncompressing change to directory .../pfsdk1_15_14 and do
make driver
make install
reboot
Make install calls script file pccam_load.
The script pccam_load must be called only once after building the driver. It removes older versions of driver module pccam.o and copies the new built to lib/modules/'uname -r'/pco.
Then an entry for the driver in modules.conf is created.
Th eskript waits for user input to get the number of devices and buffers and creates the devnodes in the /dev directory according to the user input.
Startparameters of the pccam module (pcc_major and pcc_message) can be given as command line parameters.
The script pccam_unload undos all the actions done from pccam_load.

./pccdrv
Driver C–source-files and header files. Makefile to build the driver.
./pccdrivh
Header Files for the driver.
./pcclib
Library C-source-files and header-files for the main functions of the SDK. Makefile to build the library.
./pcctest
C-source-files for small ‘textmode’ demonstration programs. Makefile to build the the demonstration programs.
Pccmain: use of driver functions
Pccbuf: use of buffer functions
Pccgrab: grabs series of pictures

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	reserved
-9	invalid camera mode
-10	reserved
-11	device is hold by another process
-12	error in reading or writing data to board
-13	wrong driver function
-14	reserved
...	

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
-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	Picture-Buffer is not prepared for DMA-Transfer
-131	A DMA-Transfer is started on this Picture-Buffer
-132	Another process has exclusive access to this Picture-Buffer
-133	Picture-Buffer cannot be found
-134	Deallocating of the Picture-Buffer failed
-135	No more Picture-Buffers can be allocated, Maxcount reached
-136	No more Picture-Buffers can be allocated, Maxalloc reached
-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 Picture-Buffer is defined for this DMA-Transfer
-168	Size of Picture-Buffer is to small for the DMA-Transfer
-169	An Error occurred during DMA-Transfer
-170	DMA-Transfer is running, function not allowed

EEPROM Tables

There are two 256 byte EEPROM Ram's available. One is located in the Camera head (CCD); the other is on the PCI board. Both have a 128 byte user area (addr.: 00h - 7Fh), which can be read and written by user. The upper 128-byte block is reserved for status and system information and is read only.

EEPROM table CCD (camera head):

addr:	**data:**		**comment:**
00h - 7Fh	-		user area
80h - 8Fh	text	-ascii-	header text
90h - 92h	vers	-ascii-	3 byte version number (format: 1.23)
93h - 98h	date	-ascii-	6 byte version date (format: dd.mm.yy)
99h - 9Dh	ser.num	-ascii-	5 byte serial number (alphanumeric)
9Eh - 9Fh			reserved
A0h	ccd	-8-	CCD type: 00h CCD VGA (640x480) black/white 01h CCD VGA (640x480) color 10h CCD SVGA2/3“ (1280x1024) black/white 11h CCD SVGA2/3“ (1280x1024) color 20h CCD HVGA1/2“ (1360x1024) black/white 21h CCD HVGA1/2“ (1360x1024) color
A1h - A2h	sub	-ascii-	2 byte substrate voltage code for CCD
A3h	hw_double_sh	-8-	double shutter: 00h no 01h double shutter standard
A4h *****	hw_prisma	-8-	prisma: 00h no 01h slit 02h pin
A5h	hw_special	-8-	special hw design: 00h no 01h special
A6h	hw_special_id	-8-	special hw identification code
A7h - AFh	-		reserved
B0h - B1h	offset	-16-	12 bit D/A offset value for binning code 00,01 (Hx1), Gain normal
B2h - B3h	offset	-16-	12 bit D/A offset value for binning code 00,01 (Hx1), Gain high
B4h - B5h	offset	-16-	12 bit D/A offset value for binning code 80,81 (Hx2), Gain normal
B6h - B7h	offset	-16-	12 bit D/A offset value for binning code 80,81 (Hx2), Gain high
B8h - BFh			reserved
C0h - C1h *****	l_meter	-16-	light meter full scale, gain normal
C2h - C3h *****	l_meter	-16-	light meter full scale, gain high
C4h - C5h *****	offset	-16-	offset correction for light meter
C6h - FFh	-		reserved

-8- = 8 bit value -16- = 16 bit value (high byte / low byte) -ascii- = ASCII value

*** Special command! Not available in standard SDK!**

EEPROM table PCI board:

addr:	data:	comment:
00h - 7Fh	-	user area
80h - 8Fh	text -ascii-	header text
90h - 92h	vers -ascii-	3 byte version number (format: 1.23)
93h - 98h	date -ascii-	6 byte version date (format: dd.mm.yy)
99h - 9Dh	ser.num -ascii-	5 byte serial number (alphanumeric)
9Eh - 9Fh		reserved
A0h	pci -8-	PCI type: 00h standard PCI board 01h Compact PCI board
A1h	hw_special -8-	special hw design: 00h no 01h special
A2h	hw_special_id -8-	special hw identification code
A3h - A7h	-	reserved
A8h - A9h	hour -16-	hour meter; every hour this value is incremented by 1
AAh - ABh	on -16-	on meter; if switched on, this value is incremented by 1
ACh - FFh	-	reserved

-8- = 8 bit value -16- = 16 bit value (high byte / low byte) -ascii- = ASCII value

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