SDK

Software Development Kit

Windows 95/98/NT/2000
Version 1.07

pixelfly

pixelfly

pixelfly

pco.
imaging

Contents

Software Development Kit for PixelFly under Windows 95/98/NT/2000

Basics

Windows

Camera and PCI Interface Board control is managed on two levels, represented by the 32 Bit DLL **pccam.dll** and the driver **pccamz.vxd** (95/98) or **pccam.sys** (NT/2000).

This Description includes in detail all functions of the upper level of the SDK (pccam.dll) and some notes how to use it.

Hardware

The PixelFly PCI-Controller-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 buffers are allocated in kernel memory.
In the SDK you will find **Memory Control Functions** to allocate and free buffers, mark one or more buffers for grabing pictures (set the buffers in a queue), give the buffers an Event-handle, which is set, when the transaction is done, and at least map the buffer to the normal user space.
Because each buffer requires some memory overhead it is recommended, that you don't use too much of the buffers. For large sequences allocate your own memory and use a copy function to transfer the data from one of the buffers to your memory, while transfering data from the camera to another 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, binnig, …. 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-Controller-Board.
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 an 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.

After installation you will have the following new directories and files:

... Programs / Digital Camera ToolBox / PixelFly SDK

pccam.h	SDK-fiunction definitions
pccamdef.h	SDK defines and makros
pccam.lib	Library File for pccam.dll
pccam.dll	DLL with SDK calls for PixelFly
pcocnv.dll	DLL with 12bit Convert functions
pcocnv.h	Convert functions definitions
cnv_s.h	Convert defines and structures
Readme.txt	readme file with additional information

Demo Programs

On the CDROM you will find a C++ demo program ,with source files and a project file for Microsoft VisualC++ 6.0 Compiler.

To run the programs, please copy the complete directory on your hard disk.

General Control Functions

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

This command initializes the PCI-Controller-Boards 0…3, the board functions and the hardware are tested automatically when opening the driver for the first time. The board parameters are set to the default values. When calling this function the first time the HANDLE *hdriver must be set to NULL. The function then returns in *hdriver the HANDLE for the selected board. If reinitialisation is needed during the process flow call this function with the HANDLE according to the board number. Board numbers start from 0. If only one board is installed board number must be 0.

IN

board = 0...3	number of the PCI-Controller-Board
hdriver	pointer to HANDLE
*hdriver = NULL	the driver is loaded and the board initialized
*hdriver = !NULL	the board is initialized

OUT

*hdriver returns the filehandle of the driver for this board

int **CLOSEBOARD**(HANDLE *hdriver)

This command resets the PCI-Controller-Board and closes the driver

IN

*hdriver	pointer to HANDLE

*hdriver must be set to the HANDLE returned from the INITBOARD function.

OUT

*hdriver is set to NULL, if the function does not fail

Camera Control Functions

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

This command returns len status **bytes** from the BOARDVAL structure of the board into the buffer ptr. In the headerfile pccamdef.h there are macro definitions to extract certain information out of this structure. Ensure that the buffer is large enough and that value of len is great enough to transfer all values, you need in the makros.
(The structure BOARDVAL is only defined in the driver.)

IN

hdriver = HANDLE returned from INITBOARD
ptr = address of the buffer
len = bytes to read

OUT

*ptr = values of structure BOARDVAL

int **SETMODE**(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 parameter for the next exposures.
It cannot be called if the camera is running. All parameters are checked.

IN

hdriver HANDLE returned from INITBOARD

mode set mode of the camera

mode = 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 % 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 VGA camera

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 (D06..D0), digital gain x32

int **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	= HANDLE returned from INITBOARD	
time	=	= 10…65535µs

int **TRIGGER_CAMERA**(HANDLE hdriver)

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

IN

hdriver	= HANDLE returned from INITBOARD

int **START_CAMERA**(HANDLE hdriver)
int **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 START_CAMERA and a hardware or software trigger.

IN

hdriver	= HANDLE returned from INITBOARD

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

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

IN

hdriver	= HANDLE returned from INITBOARD
cmnd	= command to send (only the low byte is valid)

OUT

*data	= the data sent back, by the ORION-controller

implemented comands in ORION 2.01:

10h	rd_portA
11h	rd_portB
13h	rd_portD
20h	wr_portC

int **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 of one picture in pixel.

IN

hdriver	= HANDLE returned from INITBOARD

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

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

IN

hdriver	= filehandle returned from INITBOARD

OUT

*ccd	= temperature in °C

The temperature range is from -55°C to +125°C.

Memory Control Functions

int **GETBUFFER_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 pccam-def.h there are macro definitions to extract certain information out of this structure. Ensure that the buffer is large enough and that value of len is great enough to transfer all values, you need in the makros.
(The structure DEVBUF is only defined in the driver.)

IN

hdriver	= HANDLE returned from INITBOARD
bufnr	= valid buffer number from ALLOCATE_BUFFER
mode	= 0
len	= bytes to read

OUT

*ptr	= values of structure DEVBUF

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

This commad 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 blocksize. 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	= HANDLE returned from INITBOARD
*size	= size of buffer in byte
*bufnr	= -1 for allocating a new buffer or buffer number of allocated buffer to reallocate with other size

OUT

*size	= allocated size, which might be greater
*bufnr	= number of buffer

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

Free allocated buffer.
If the buffer was set into the buffer queue and no transfer was done to this buffer call REMOVE_BUFFER_FROM_LIST first.
If an event was created for this buffer it will be closed and must not be used any longer.

IN

hdriver	= HANDLE returned from INITBOARD
bufnr	= valid buffer number from ALLOCATE_BUFFER

int **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 transfer is done 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 create a buffer event with SETBUFFER_EVENT, reset the event to nonsignaled state before calling ADD_BUFFER_TO LIST and then wait with any Windows Wait Functions i.e. WaitForSingleObject (…).

IN

hdriver	= HANDLE returned from INITBOARD
bufnr	= valid buffer number from ALLOCATE_BUFFER
size	= number of bytes to transfer
offset	= offset of bytes in the 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 occure in the status byte of the buffer. Size must always be a value greater than 4096.
If transfer size is lower than camera size, the transfer is done with the specified transfer size and no error should occure.
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.
Offset must be a multiple of 4096.

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

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

IN

hdriver	= HANDLE returned from INITBOARD
bufnr	= valid buffer number from ALLOCATE_BUFFER

int **SETBUFFER_EVENT**(HANDLE hdriver, int bufnr, HANDLE *hPicEvent)

This command creates an event handle for this buffer.
The event is created as a manual reset event in nonsignaled state with default security descriptor and without a name.
The event is set to signaled state when the DMA-transfer into the buffer is finished or if a error occurred in transfer. Use i.e WaitForSingleObject(*hPicEvent,TimeOut); to wait until a transfer is done. The event state remains signaled until a new transfer is started to this buffer or it is reseted from the application.

IN

hdriver	= HANDLE returned from INITBOARD
bufnr	= valid buffer number from ALLOCATE_BUFFER

OUT

*hPicEvent	= handle of event

int **MAP_BUFFER**(HANDLE hdriver, int bufnr, int size, int offset, DWORD *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.

IN

hdriver	= HANDLE returned from INITBOARD
bufnr	= valid buffer number from ALLOCATE_BUFFER
size	= number of bytes to map
offset	= 0

OUT

*linadr	= address of buffer

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

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

IN

hdriver	= HANDLE returned from INITBOARD
bufnr	= valid buffer number from ALLOCATE_BUFFER

Commands for the ORION processor

The ORION processor is called automatically by the driver, shortly after a DMA transfer is done.
With the following functions you can set the commands and data byte, which belongs to every comand.
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 read out has been.
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 **SETORIONINT**(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	= HANDLE returned from INITBOARD
bufnr	= valid buffer number from ALLOCATE_BUFFER
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 **GETORIONINT**(HANDLE hdriver, int bufnr, int mode, unsigned char *data, int len)

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

IN

hdriver	= filehandle returned from INITBOARD
bufnr	= number of buffer
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

int **READEEPROM**(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 returned from INITBOARD
mode	= 0 HEAD-EEPROM
	= 1 CARD-EEPROM
adr	= address of byte to read (0…255)

OUT

*data	= byte to read

int **WRITEEEPROM**(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 returned from INITBOARD
mode	= 0 HEAD-EEPROM
	= 1 CARD-EEPROM
adr	= address of byte to write (0…127)

OUT

data	= byte to write

int **SETTIMEOUTS**(HANDLE hdriver, DWORD dma, DWORD proc, DWORD head)

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

IN

hdriver	= filehandle returned from INITBOARD
dma	= timeout in milliseconds for dma
proc	= timeout in milliseconds for card-io
head	= timeout in milliseconds for headpoll

int **SETDRIVER_EVENT**(HANDLE hdriver, int mode, HANDLE *hHeadEvent)

This command creates an event handle for driver events.
The only defined event now is the head event.
The event is created as a manual reset event in nonsignaled state with default security descriptor and without a name.
The head event is set to signaled state when the driver detects that the camera head is connected or disconnected.
Use i.e WaitForSingleObject(*hHeadEvent,TimeOut); to wait and react to these events.

IN

hdriver	= filehandle returned from INITBOARD	
mode	= low word	0x0000 = head event
	= high word	0x0000 = open and enable event
	= high word	0x8000 = disable event
	= hifh word	0xC000 = disable and close event

OUT

*hHeadEvent	= handle of event

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	
-101	timeout in any driver function
-102	board is hold by an other process
-103	wrong boardtype
-104	cannot match processhandle to a board
-105	failed to init PCI
-106	no board found
-107	read configuratuion registers failed
-108	board has wrong configuration
-110	camera is busy
-111	board is not idle
-112	wrong parameter sent
-113	head is disconnected
-116	board initialisation FPGA failed
-117	board initialisation NVRAM failed
-121	not enough IO-buffer space for return values

-130	picture buffer not prepared for transfer
-131	picture buffer in use
-132	picture buffer hold by another process
-133	picture buffer not found
-134	picture buffer cannot be freed
-135	cannot allocate more picture buffer
-136	no memory left for picture buffer
-137	memory reserve failed
-138	memory commit failed
-139	allocate internal memory LUT failed
-140	allocate internal memory PAGETAB failed
-148	event not available
-149	delete event failed
-156	enable interrupts failed
-157	disable interrupts failed
-158	no interrupt connected to the board
-164	timeout in DMA
-165	no dma buffer found
-166	locking of pages failed
-167	unlocking of pages failed
-168	DMA buffersize to small
-169	PCI-Bus error in DMA
-170	DMA is runnig, command not allowed
-228	get processor failed
-229	reserved
-230	wrong processor found
-231	wrong processor size
-232	wrong processor device
-233	read flash failed

EEPROM Tables

There are two 256 byte EEPROM Ram's avaiable. 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