Pxdrv 0.20

A driver for the PX500, PX510 and PX610 frame grabbers.
February 2001

by Alessandro Rubini (`rubini@linux.it`)

This file documents version 0.20 of the Pxdrv device driver and its support programs (February 2001).

1 Introduction

This text introduces to the features of the PX500/510/610 device driver for Linux. Not every feature of the device is supported: feel free to ask for features if you need them: I'll feel more inclined to write code if someone is going to use it.

In this document, the term PX500 is used to refer to all devices in the family (unless otherwise specified). The terms PX510 and PX610 are used to refer to features that are not available in earlier models. Refer to the ImageNation manual to get information about hardware features.

The driver currently works with version 2.0 and 2.2 of the Linux kernel. Note that in order for DMA to work, you need to change your *Lilo* configuration, as described in Chapter 4 [The DMA Buffer], page 2.

As of version 0.20 of the driver, progressive scan cameras are supported, as well as using WEN as an input to synchronize the frame grabber (The WEN signal is used by some resettable cameras to signal the beginning of a new video field. See Chapter 11 [Special Cameras], page 10.

2 General Information abut the driver

As usual in the Unix environment, you need device special files in order to access a device driver of any kind. This section enumerates the entry points used to communicate with the driver.

The following devices are used for the first grabber (device number 0). They are automatically created by `px_load`. See Chapter 3 [Compiling and Loading], page 2.

`/dev/px0` A binary device, returning a continuous stream of 320x240 (NTSC) or 384x288 (PAL) raw images. On open it allocates a DMA buffer to fit the image.

`/dev/px0pgm`
The pmg device associated to the previous one. It returns a single PGM image of size 320x240 or 384x288.

`/dev/px0ctl`
A control device. Opening it doesn't allocate the DMA buffer. Reading is not allowed, but you can still use ioctl() on this device.

`/dev/px0H`
The hi-res binary node: it returns a continuous stream of 640x480 (NTSC) or 768x576 (PAL) images in raw format.

`/dev/px0Hpgm`
The pgm node. Returns a single 640x480 or 768x576 pgm image.

Whenever the device is active in low-resolution, opening an high-resolution entry point return `-EBUSY`. The same happens if the device is working in high resolution and you open a low-resolution entry point. The control node can always be opened because it isn't involved with data acquisition.

Note that if you open a device and then change its resolution via `ioctl`, further *open* calls on the device won't be prohibited; the device will instead remain configured as selected via *ioctl*. See Chapter 6 [Ioctl], page 3.

3 Compiling, Loading, Unloading

To compile the driver run `"make"`. If the source of the kernel you are using is not in /usr/src/linux, you can specify its location as KERNELDIR, either in the environment or at the command line of make. For example I currently use `"make KERNELDIR=/usr/src/linux-2.2"`.

Alternatively, you can specify INCLUDEDIR in the same way. I used INCLUDEDIR in older releases of this drivers as well as other drivers. Now I prefer to use KERNELDIR instead. INCLUDEDIR, if not specified by the user, defaults to `$(KERNELDIR)/include` .

To load the driver, run `./px_load`, Edit the script to configure permission bits and owners of the devices being created. Any argument that you pass at the `px_load` command line are passed to *insmod*. This allows load-time configuration of internal variables.

To unload the driver, run `./px_unload`. It also removes the '`/dev`' entry points.

4 Allocating a DMA Buffer

One of the most compelling problems with any DMA-capable device is the allocation of a suitable memory buffer. In *pxdrv* I chose to use a module of mine, called "allocator". The allocator is able to use high memory (above the one used in normal operation) for DMA allocation. In order to use it, however, there must be some real RAM memory at the end of the memory used by the Linux kernel.

To prevent the kernel for using high memory, so that it remains available for DMA, you should pass a command line argument to the kernel. Command line arguments can be passed to Lilo, to Grub or to whichever loader you are using (unless it's very poor in design). For Lilo, either use `"append="` in '`/etc/lilo.conf`' or add command line arguments to the interactive prompt. For example, I have a 64MB box and reserve two megs for DMA by telling the kernel `mem=62M`. Actually, the px500 uses half a meg at most, but I have another driver that acquires RGB images and needs 1.5 megs.

In lilo.conf:

```
image = /zImage
label = linux
append = "mem=62M"
```

Or, interactively:

```
LILO: linux mem=62M
```

Please note that the `mem=` argument must be lowercase or the kernel won't use it.

5 Reading and Writing

The devices return image data when read. For example, you can use "`cat /dev/px0H | split -b <size>`" to create a burst of raw graphic files in your disk (the *size* value depends on whether you use NTSC (size == 307200) or PAL (size == 442368). To see the current image, you can issue "`xv - < /dev/px0pgm`" or any equivalent command.

There is currently no way to control grabbing in time, like grabbing one frame every 200ms or so. This will be implemented in a future release of the driver.

Devices can't be written to, currently. Maybe I'll arrange for PGM devices to receive text strings to configure their behavior (regions of interest and so on), and the same will apply to the control node. If I won't conceive more interesting write semantics for binary nodes, they will behave in the same way.

6 Ioctl

The `ioctl()` function is supported by the device, to see and/or change what is happening in the internals. A few of the ioctl commands are very low level, and I'd suggest using them only if you know how the device works internally.

6.1 Low-Level Commands

Some of these commands can only be issued by the super-user, independent of the file permission of the '`/dev`' entry point accessed. This is a security measure necessary to protect hardware from potentially bad actions. The super-user can damage anything anyways, so allowing these commands does not harm.

If you are just a device user, feel free to skip to the next section.

`PX_IOCRESET`
: This command resets the device to the default state. Note that if the "persist" flag is set, the reset is not complete. See Chapter 10 [Device Flags], page 9. The command is restricted to the super-user.

`PX_IOCHARDRESET` *(no third argument needed)*
: This command is mainly a debugging aid: it resets the device and drops the module's usage count to one (therefore, on the next close it will drop to 0 and you'll be able to unload the module even if an Oops happened. The command is restricted to the super-user.

`PX_IOCGDEV` *(third argument needed)*
: This command returns the whole device structure to user space, using the third argument as a `Px_Dev *` item. The command is mainly useful for debugging and is not restricted to the super-user.

`PX_IOCSREG` *(third argument needed)*
`PX_IOCGREG` *(third argument needed)*
: These commands are used to read or write a 32-bit PCI register. The third argument to `ioctl` is a pointer to a `Px_Register` structure. The *reg* field

identifies the register to be acted upon, the *val* field is the value being read or written. If the register number is out of the allowed range, `-EINVAL` is returned. Only the superuser can write device registers.

`PX_IOCCMD` *(third argument needed)*
: This command issues a command to the grabber's microcontroller. The commands are defined in '`px500.h`' and are often used internally by the driver. The argument of this `ioctl` command is a pointer to a `Px_Register` structure and is used like in `PX_IOCSREG` and `PX_IOCGREG` above. For output commands, the structure defines the command number and its argument; for input commands the structure defines the command number on entry and the result is filled by the driver. Some of the commands are restricted to the super user and some are not. Refer to `px_command_list` in the driver for details.

`PX_IOCSPORT` *(third argument needed)*
`PX_IOCGPORT` *(third argument needed)*
: These commands read or write a "port" in the driver. The allowed "port" numbers are in the range from `PX_PORT_MIN` to `PX_PORT_MAX` inclusive (otherwise, `-EINVAL` is returned). The third `ioctl` argument is a pointer to a `Px_Register` structure and is used just like in `PX_IOCSREG` and `PX_IOCGETREG` adove. Only the superuser can write ports.

6.2 Higher-Level Commands

Use of these commands doesn't require intimate knowledge of the frame grabber device, and the commands are meant for use by actual application code.

`PX_IOCSLUT` *(third argument needed)*
`PX_IOCGLUT` *(third argument needed)*
: These commands are used to assign and retrieve the gray-level look-up table of the device. The third argument to `ioctl` is a pointer to `struct PX_Lut`, defined in '`px500.h`'. A sample applications using these commands is '`pxlut`'. See Section 13.6 [pxlut], page 13.

`PX_IOCSGAIN` *(third argument needed)*
`PX_IOCGGAIN` *(third argument needed)*
: These commands are used to assign and retrieve the image gain. The gain is made up of two numbers: a "gross gain" (ranging from 0 to 3) and a "fine gain" ranging from 0 to 255. The third argument of `ioctl` is a pointer to `struct Px_Gain`, defined in '`px500.h`'. Use of this command is exemplified in '`pxgain`' and '`pxcontrol`'.

`PX_IOCAUTOGAIN` *(third argument needed)*
: This command enables or disables the auto-gain feature of the device. The third argument of `ioctl` is a pointer to an integer item. If the item is positive, auto-gain is enabled; if the item is zero or negative auto-gain is disabled. When auto-gain is active, the user-specified gain won't be effective and the driver will adapt the image gain on the fly to achieve maximum image range. Sample applications using this command are '`pxgain`' and '`pxcontrol`'.

`PX_IOCSMUX` *(third argument needed)*
`PX_IOCGMUX` *(third argument needed)*

These commands set and retrieve the current value of the input multiplexer. By default the device grabs from input 0; `IOCSMUX` can be used to switch the multiplexer to either of the four input channels (numbered from 0 to 3). The third argument to `ioctl` is a pointer to `unsigned long`. `IOCSMUX` returns `-EINVAL` if the value specified is out of range; `IOCGMUX` always succeeds. When switching video input, the behavior of the grabber is controlled by `PX_FLAG_NONSYNC`. See Chapter 10 [Device Flags], page 9.

`PX_IOCSOFF` *(third argument needed)*
`PX_IOCGOFF` *(third argument needed)*

These commands are used to get and set the offset value for video input. Offset adjustment is usually paired with gain adjustment. The third argument of `ioctl` is a pointer to `int`. See the ImageNation manual for details about theoffset.

`PX_IOCSFLAGS` *(third argument needed)* `(unsigned long * argument)`
`PX_IOCGFLAGS` *(third argument needed)* `(unsigned long * argument)`

These commands are used to retrieve and modify the flags associated to the PX500 device. The third argument of `ioctl` is a pointer to `unsigned long`. While `IOCGFLAGS` returns all of the flags to user space, `IOCSFLAGS` modifies only the bits in `PX_FLAG_USERMASK` and doesn't touch other bits. The flags are described in Chapter 10 [Device Flags], page 9.

`PX_IOCSCOMPARE`
`PX_IOCGCOMPARE`

Support for user control of the PX500 compare value is not implemented (even though the feature is used to implement auto-gain).

`PX_IOCSSIZE` *(third argument needed)*
`PX_IOCGSIZE` *(third argument needed)*

These commands are used to retrieve and change the current image size and other acquisition parameters. The third argument to `ioctl` is a pointer to `Px_Parameters`. The role of this structure is detailed in Chapter 11 [Special Cameras], page 10. Since `IOCSSIZE` is potentially dangerous, its use is restricted to the super-user.

`PX_IOCSTRIGGER` *(third argument needed)*
`PX_IOCGTRIGGER` *(third argument needed)*

These commands are used to configure trigger operation and retrieve the current trigger setup. The third argument of `ioctl` is a pointer to `unsigned long`. The argument being exchanged with kernel space is a bit field and is described in Chapter 8 [Triggers], page 7.

`PX_IOCWTRIGGER` *(no third argument needed)*

Wait for a triggered acquisition. When a process issues this command it will be stopped until a frame or field (according to current configuration) is acquired to the DMA buffer. Note that read() won't block waiting for a triggered acquisition (it will just return the last acquired image), if you want to block you must use

this `ioctl` command. The command returns 0 for success and -1 (with `errno` set to `EINTR`) if the command is interrupted by a signal.

`PX_IOCSTROBEENA` *(third argument needed)*
: Enable or disable the strobe outputs (which, by default, are configured as inputs). The third argument to `ioctl` is a pointer to `int`. If the value pointed to is positive, then strobes are enabled, if it is zero or negative strobes are disabled. Strobe management is implemented by macros that use `PX_IOCCMD` and is documented in Chapter 9 [Strobes], page 8.

  PX_IOCTRIGSTROBE *(third argument needed)*

  The command tells the driver whether the trigger input should fire the strobe sequence or not. The third argument to `ioctl` is a pointer to `int`. If that integer value is positive, the feature is enabled; if it is zero or negative the feature is disabled.

`PX_IOCDRIVEENA` *(third argument needed)*
: Enable or disable the output synchronization signals. The third argument to `ioctl` is a pointer to `unsigned long`. The pointed-to value is a bit mask, made up of the following bits. `PX_BIT_DRIVE_ENA` enable the output pins (1) or disable them (0). `PX_BIT_DRIVE_HNEG` if set makes the output H-sync a normally-high signal with negative-going pulses; if set the signal is normally-low with high pulses. `PX_BIT_DRIVE_VNEG` is the same for the V-sync signal. The command can be tested using the `pxcontrol` sample tool but the bitmask must be specified in numeric form. See Section 13.7 [pxcontrol], page 13. The command fails with `-EINVAL` if any invalid bit is set in the argument value.

`PX_IOCPALQZ` *(third argument needed)*
: Enable or disable use of the PAL quarts. The PAL family of European video formats requires sampling pixels at an higher frequency in order to get square-pixel images. While US standard require 640 pixels per line, European standards requi 768 pixels per line. If you use a normal (interlaced) camera you won't need to run this command (the right quarts is automatically selected). If you run US progressive-scan cameras you may need to disable the higher-frequency quartz. The third argument to `ioctl` is a pointer to `int`. If the pointed-to value is positive than the PAL quartz is selected; if it is zero or negative the standard (slower) quartz is selected.

7 Mmap

Memory mapping is implemented on any device node. If you mmap() the control entry point, you will map the current DMA memory, either low resolution, high resolution or no DMA at all (if no DMA is active, mapping the control device will return the last-acquired image). Memory-mapping other devices assures that DMA is being run at low resolution ('`/dev/px0`' or '`/dev/px0pgm`') or high resolution ('`/dev/px0H`' or '`/dev/px0Hpgm`').

You can test a simple `mmap()` client by running the `mapper` program distributed with the driver. See Section 13.8 [mapper], page 14.

8 Triggers

The PX500 device is able to perform triggered acquisition. The feature is controlled by means of the `PX_IOCSTRIGGER` `ioctl` command (and the current setup can be retrieved by issuing `PX_IOCGTRIGGER`. See Chapter 6 [Ioctl], page 3.

The argument passed to (and received from) the `ioctl` commands is a bit mask made up of the following bits:

`PX_BIT_TRIGGER_ENABLE`
: If the bit is set, trigger-driven grabbing is active. If clear the trigger input is disabled.

`PX_BIT_TRIGGER_POL`
: If set, use negative-level (or falling edge) triggers. If unset, positive level (or raising edge).

`PX_BIT_TRIGGER_EDGE`
: If the bit is set, the trigger is edge-sensitive, if the bit is clear the trigger is level-sensitive.

`PX_BIT_TRIGGER_DEBN`
: If the bit is set, debouncing is enabled on the trigger input. Refer to the ImageNation manuals for details.

`PX_BIT_TRIGGER_OCCURRED`
: This bit is only valid in IOCGTRIGGER, and tells the application if the frame grabber received a trigger after the last `IOCSTRIGGER` command. The bit is cleared every time `IOCSTRIGGER` is issued.

When the `TRIGGER_ENABLE` bit gets set, data transfer is stopped until a trigger arrives (if you read the device, you'll find the last grabbed image). When a trigger is received, the driver will acquire a single image (one frame or one field, according to the current setup) and will wait for one further trigger. Clearing `TRIGGER_ENABLE` turns back the system to continuous grabbing.

To wait for a triggered acquisition use `PX_IOCWTRIGGER` described in Chapter 6 [Ioctl], page 3.

If you want to check whether a trigger has already been managed, you can check the TRIGGER_OCCURRED bit. The bit is set after a triggered acquisition is over (i.e., when the bit is set you are assured that the in-memory image is current). The bit is automatically cleared if you choose to wait for the trigger (using `IOCWTRIGGER`) instead of polling (using `IOCGTRIGGER`). As outlined above, the bit is cleared when read, so that waiting for another trigger doesn't require any further action.

9 Strobes

The PX510/610 devices have two digital I/O lines, called strobes. The lines can be acted upon using microcontroller commands, so no special `ioctl` command is implemented for them. However, in order to save the programmer from boring hacking with hex values, the '`px500.h`' header defines the following functions to play with the strobes. Refer to the ImageNation manuals for details on the provided mechanisms.

The return value is always zero or a positive number for success, and -1 in case of error (with `errno` set to a proper value). This comes straight from the underlying ioctl system call.

`Px_EnableStrobes(int fd, int output);`
: The `output` argument must be 1 to enable the strobe lines and 0 to disable them. The macro uses `IOCSTROBEENA`.

`Px_InitStrobes(int fd);`
: Restore strobes to their default power-on values.

`Px_Fire_Strobe(int fd, int which);`
: The macro fires or stops the strobe sequence, according to the `which` argument. The argument must be 0 to stop a strobe sequence, and to 1, 2, or 3 to fire the first strobe, the gap, or the second strobe. For clarity, the following symbolic names can be used as well: `PX_FIRE_STROBE_STOP`, `PX_FIRE_STROBE_1`, `PX_FIRE_STROBE_GAP`, `PX_FIRE_STROBE_2`. The result of the macro is undefined if invalid values of `which` are passed.

`Px_TimeStrobe(int fd, int which, int val);`
: The macro sets the duration of strobe 1, gap or strobe2, according to whether `which` is 1, 2, 3. The duration is expressed in multiples of the duration of one horizontal line. Like above, the following Once again, the following symbolic names can be used instead: `PX_FIRE_STROBE_1`, `PX_FIRE_STROBE_GAP`, `PX_FIRE_STROBE_2`. The result of the macro is undefined for invalid values of `which`. The `val` argument (the duration) must fit 16 bits so it must be in the range 0–65535.

`Px_TimeStrobes(int fd, int t1, int t2, int t3);`
: The macro sets all three durations.

`Px_StrobePolarity(int fd, int which, int negative);`
: The macro sets the polarity of one of the strobes: if `negative` is non-zero, the polarity will be inverted from the default. The `which` argument should be 1 (strobe 1) or greater (strobe 2). The values `PX_FIRE_STROBE_1` and `PX_FIRE_STROBE_2` can be used. This macro individually controls the polarity of the two output lines.

`int Px_GetStrobes(int fd);`
: The macro returns the contents of the *status* port into *resultptr. The status ports includes the bits `PX_BIT_STROBE_1`, `PX_BIT_STROBE_2` and `PX_BIT_STROBING`. The first two bits report the current voltage level at the strobe pins

(which, thus can be used as input pins as well, provided strobing output is disabled). The `PX_BIT_STROBING` value tells whether or not a strobing sequence is active; since both bits are inactive during the gap period, the bit is needed to tell the gap period from the no-strobe-active case.

10 Device Flags

The device structure includes a 32-bit field of flags, which can be retrieved from user space either as an unsigned long value or within the whole device" structure (using `IOCGDEV`). Only flags intended for user consumption are described here. Some of them are read-only for user-space programs.

Informative flags:

`PX_FLAG_DEVICENAME`
: This is is an AND mask to extract the device number from the flags field. In other words, "`flags & PX_FLAG_DEVICENAME`" is either 500, 510 or 610 according to the device type.

`PX_FLAG_DEVICE510`
: This bit is set in the flags field if the current device is either a PX510 or a PX610 device (and is not net if this is an older PX500 device). Use it to check whether some capabilities are available.

`PX_FLAG_DEVICE610`
: The bit is only set if this is a PX610 (i.e., it supports progressive scan cameras).

`PX_FLAG_DEVICE104`
: The bit is set if this device is a PC104+ device. It is clear if this is a PCI device.

`PX_FLAG_HASCACHE`
: The bit, if setm reports that this device is equipped with on-board cache RAM.

Read/write flags, that can be modified via `IOCSFLAGS`:

`PX_FLAG_AUTOGAIN`
: The bit can be set to enable automatic gain control. It is clear by default.

`PX_FLAG_PERSIST`
: The bit, if set, tells that the device must not be reconfigured with default values on the first open, so that configuration persists from a close to the next open. Setting this bit might be useful during development of new features but you should think twice if you think this is needed in your application. Remember that despite persistent configuration, the DMA buffer is released anyway on the last close. If an application program needs to preserve configuration across close it should better open an idle file descriptor on the device (see `pxlive` for an example).

`PX_FLAG_NONSYNC`
: The bit specifies that cameras are not synchronized. It is only used when you switch the input multiplexer. If the bit is set, any `read()` system call will sleep

for the duration of two image frames after the digitazer is switched to another camera. This delay allows the grabber to re-sync with the new video stream. The bit is off by default thus assuming that cameras are synchronized (in which case, no extra delay is added by the driver after the input MUX is switched).

11 Progressive-Scan and Special Cameras

Pxdrv supports progressive-scan cameras and special cameras by means of its `struct Px_Parameters` structure and `PX_IOCSSIZE ioctl` command.

The structure is defined in '`px500.h`':

```
typedef struct Px_Parameters {
__u16 xsize, ysize;
__u16 firstline, lastline;
__u16 firstcol, lastcol;
__u16 dmahole, decimate;
__u16 fieldlen, blanklen;
__u16 flags;
} Px_Parameters;
```

The role of the various fields is as follows:

`__u16 xsize, ysize;`
: The size of the image. This is the information used to create the PGM header for the PGM special files. Please note that when the parameters are changed via `IOCSSIZE` the dimension of the image is changed from the default.

`__u16 firstline, lastline;`
: The first and last line of digitized video. The first line is normally 0 and the last line equals full-frame height (i.e., 480 or 576). Setting the values differently allows cropping the image in the Y direction. Please note that the `ysize` parameter above must be consistent with `firstline` and `lastline`.

`__u16 firstcol, lastcol;`
: The first and last column of digitized video. The first column is normally 0 and the last column equals full-frame width (i.e., 640 or 768). Setting the values differently allows cropping the image in the X direction. Please note that the `xsize` parameter above must be consistent with `firstcol` and `lastcol`.

`__u16 dmahole;`
: `dmahole` is the amount (in bytes) of space to leave blank between successive video lines in the same field. When acquiring single fields it is usually 0, when acquiring whole frames it is the same as the line length. Note that when acquiring whole frames with progressive-scan cameras the `dmahole` must be set to zero.

`__u16 decimate;`
: Horizontal and vertical decimation. Use 0 for whole-frame and 1 for single-field acquisition. Note that while `decimation` can be set to values greater than 1, the feature is untested.

`__u16 fieldlen, blanklen;`
: Length, in multiples of the video line, of the field and of the leading blank lines after the vsync signal. Either value can be set to zero in order to use default values.

`__u16 flags;`
: Flags that modify acquisition parameters.

Currently, the following flags are supported:

`PX_PARAMETER_FLAG_PROGRESSIVE`
: The flag enable s progressive-scan support if the current grabber is a PX610 (other models can't deal with progressive scan). With progressive-scan cameras you'll need to specify the field length (by setting `fieldlen` in the structure.

`PX_PARAMETER_FLAG_WEN`
: The flag enables use of the trigger input as a substitute for vertical sync. If the Wen feature is enabled, you'll also need to configure the trigger for proper operation.

Note that the driver automatically configures the frame grabber for PAL operation whenever the field length is greater than the default NTSC length. Therefore, when using a progressive-scan camera you may need to explicitly choose the pixel clock to use. To this aim, use the `PX_IOCPALQZ` `ioctl` command. See Chapter 6 [Ioctl], page 3.

11.1 An Example Using WEN

As an example, this is how I acquire images using a Jai CV-M10 progressive-scan camera:

- Open the device (I use '`/dev/px0Hpgm`' but it doesn't matter).
- Use `IOCSSIZE` to set image parameters and flags. This enables WEN and progressive-scan; I use `xsize`=640, `ysize`=480, `dmahole`=0, `decimate`=0, `blanklen`=20, `fieldllen`=640. First and last lines and columns are unchanged, while `flags` are set to `PX_PARAMETER_FLAG_PROGRESSIVE|PX_PARAMETER_FLAG_WEN`.
- Use `IOCSTRIGGER` to choose trigger parameters. I select a positive-edge trigger: `PX_BIT_TRIGGER_ENABLE | PX_BIT_TRIGGER_EDGE`.
- Use `IOCPALQZ` with an argument of 0 in order to disable the higher-frequency PAL quartz and digitize 640 pixels per line.

If you use '`pxlive`' and '`pxcontrol`' to change parameters, you can try this sequence:

```
$ ./pxlive /dev/px0Hpgm &
$ ./pxcontrol setparams 0 640 480 3 25 480
$ ./pxcontrol  settrigger 10 palqz 0
```

To get details on `setparams`, Section 13.7 [pxcontrol], page 13.

12 Bugs and Missing Features

The driver still misses any time-related operation (like single-frame acquisition).

The driver didn't undergo serious auditing and testing.

13 Sample Programs

All the sample application parse the command line in the same way. The device name can be either the first or last command line argument. The device name is identified by being a full pathname (i.e., it has a leading "/"). If none is specified on the commandline, the `PXDEVICE` environment variable is used to retrieve a default device entry point; if the variable is unset, the default is using device 0 (the actual default entry point is either the control node or the low-resolution PGM node. See Chapter 2 [General Information], page 1.

13.1 pxtest

The `pxtest` program prints information about the frame grabber associated to the device special file it opens. It uses the control entry point by default in order not to allocate a DMA buffer.

This is an example of a '`pxtest`' run on the host called `borea`, as of version 0.20 of the device driver:

```
borea% ./pxtest
./pxtest: using device /dev/px0ctl
Device name: PX610, interface bus: PC104+
Hardware information:
hardware revision: 0x34
protection key: 0x55
has cache ram: no
PIC revision date: 4/21/97
Software information:
dma buffer: 0x03e00000 (size 0x0006c000), remapped at 0xc403c000
registers at 0xefffe000, remapped at 0xc403a000
image size (640x500): 320000
usage 3, isup 5
irq 11, currently 11
Status information:
status: 0x37
status2: 0x09
port X: 0x00
port Y: 0x4c
port Z: 0x01
has video: y
video type: pal
```

13.2 pxshow

The program displays the flow of images from the device by printing it to the text screen. I use mostly text mode, so I like this tool. It reads /dev/px0 by default and it assumes the device is 320x240 pixels (i.e., an NTSC camera). It adapts the output to the actual screen size (whether it is an `xterm` or a text console).

13.3 pxlive

The program makes live video on an X window. This tool is quite slow as it doesn't directly access X data structures; it's not like a real live video, just a simple demonstration.

If the `pacco` tool is available, pxlive uses it to gain better speed and better resolution. If `pacco` is not available, the tool runs using the photo widget of Tk, which is much slower. Pacco is available from `ftp://ftp.prosa.it/pub/software` and `ftp://ftp.systemy.it/pub/develop`. Like other sample programs, this accepts a device name on the commandline or as an environment variable. The program must access a pgm entry point (either low or high resolution).

The `More...` button of the program opens an additional window that lets you play with some of the configurable parameters of the grabber. Operation with the `More...` window is especially slow since it executes an external command (`pxcontrol`) for each configuration change. Again, my aim is not speed but something easy to show things out.

13.4 pxstereo

The program is a slightly modified pxlive that uses two cameras (/dev/px0 and /dev/px1). Thanks to Rob Steele for contributing this: although he didn't consider it ready for prime time (as it lacks some details as well as Pacco support), I think it is interesting. I planned to reintegrate it back to the normal pxlive but maybe it will never happen for real.

13.5 pxgain

The program shows or sets the gain for the frame grabber. Remember that the gain is reset to "1 0" on first open, so you need to keep the device open while changing the gain. You can run `pxgain` while running `pxshow` or `pxlive` to see real-time changes (but note that `pxlive` in its configuration window allows playing with the gina.

The `auto` option to `pxgain` enables auto-gain. An optional argument in addition to `auto` tells the maximum amount of gain change from one frame to the next. Default is 5; 0 disables autogain.

`pxgain auto 10`	*auto-gain, maxstep is 10*
`pxgain auto`	*default step*
`pxgain auto 0`	*disables auto-gain*

13.6 pxlut

The `pxlut` program allows reading and writing the gray level look-up table of the device. Call it without arguments to get help.

13.7 pxcontrol

The program is a general-purpose wrapper for `ioctl()`. Call it with no arguments to have a list of allowed command line options. You can specify several commands at the same time on the command line, one after another.

All commands accepted by the program get numeric arguments. The meaning of the arguments should be clear by itself or needs checking the '`px500.h`' header (for example, to pass a bitmask to the program you'll need to look up the value of individual bits; this applies for instance to the `setrigger` command.

The only command that needs a special description is `setparams`. The command is meant as an helper to invoke the `IOCSSIZE ioctl` command. The data structure used in `ioctl` is described in Chapter 11 [Special Cameras], page 10, and the `setparams` command takes 6 numeric arguments:

- Low resolution (1) or high resolution (0). If the argument is zero, decimation is set to zero and `dmahole` is set to be equal to the horizontal size (unless progressive-scan input is enabled). If the argument is not zero, decimation is set to 1 and `dmahole` is cleared. The value must be specified as non-zero to acquire single fields (the default if you open the low-resolution devices).
- Image width. The value is used to set `xsize`.
- Image height. The value is used to set `ysize`.
- Image flags, and or-combination of `PX_PARAMETER_FLAG_PROGRESSIVE` (1) and `PX_PARAMETER_FLAG_WEN` (2).
- Length of vertical blank. Set to zero to use the default value that is appropriate for the current video input (NTSC or PAL).
- Length of image field. Set to zero to use the default value that is appropriate for the current video input.

Use of `setparams` is exempified in Section 11.1 [An Example Using WEN], page 11.

13.8 mapper

The `mapper` is general-purpose memory-map frontend. It receives arguments on the command line: the device to map, the initial offset and the size of the memory map. Mapped data is printed to the standard output. The following example shows how to acquire a image from an NTSC camera via `mmap()`:

```
./mapper /dev/px0  0 `exptr 320 \* 240` | rawtopgm 320 240 > img.pgm
```

14 Library Functions

To use the library you should include `px500.h` and link with `-lpxdrv` (which is installed in '`/usr/local/lib`'). The library is made up of functions, preprocessor macros and external commodity variables (the variables are instrumental to the workings of some of the macros. They are not documented here as such.

Please note that direct use of `ioctl` is still the preferred way to interact with the device; some of the library functions I offer haven't been tested.

The `fd` argument of all the functions is a file descriptor that must refer to a frame grabber device. If you use thee standard I/O library (`fopen()` and file pointers), you can extract the descriptor from a FILE pointere by calling `fileno()`. The return argument of the library functions, unless otherwise described, is 0 or positive in case of success and a negative value in case of failure, with `errno` set to describe the type of error.

The following functions are implemented, most as simple wrappers around ioctl():

`int Px_Command(int fd, int command, int value);`
: The function invokes a low-level command in the frame grabber. If the command is read-only, "value" is unused and the read value is returned as a non-negativa value. If the command is write-only, the return value is 0 for success. Note that some of the commands can only be invoked by the superuser. See Chapter 6 [Ioctl], page 3.

`int Px_SetReg(int fd, unsigned long register, unsigned long val);`
`int Px_GetReg(int fd, unsigned long register);`
: Write and read a 32-bit register. The "set" command can only be issued by the superuser. The functions return -1 in case of error and 0 or the value of the register in case of success. Note that register values with the high bit set will be returned as negative numbers. Use `ioctl(PX_IOCGETREG)` to avoid ambiguity. As a matter of facts, reading and writing registers should not be needed during grabber operation.

`int Px_Setport(int fd, unsigned long port, unsigned long val);`
`int Px_GetPort(int fd, unsigned long port);`
: These are teh eequivalent of set/get register, but they act on ports instead.

`int Px_GetDev(int fd, Px_Dev *ptr);`
: The function copies to user space the device data structure.

`int Px_Reset(int fd);`
: Resets the device.

`int Px_ProtKey(int fd);`
: Returns the protection key associated to the device. Refer to ImageNation manuals for details.

`int Px_Revision(int fd);`
`int Px_ReleaseYear(int fd);`
`int Px_ReleaseMonth(int fd);`
`int Px_ReleaseDay(int fd);`
: These functions return information about the physical device. The first returns the hardware revision number, the other return the PIC revision date (only for new devices, the PX500 return 0xff for the date queries). The information, which might reveal useful when reporting bugs, is displayed by the program `pxtest` using these macros.

`int Px_Status(int fd);`
`int Px_Status2(int fd);`
`int Px_PortX(int fd);`
`int Px_PortY(int fd);`
`int Px_PortZ(int fd);`
: The functions return information about the software status of the device. They are not expected to be useful but in debugging. The `pxtest` sample program uses them to print the current value of the status ports.

`int Px_HasRam(int fd);`

Returns 0 or 1 (or -1 in case of error), according to whether the device has cache RAM memory installed.

`int Px_HaveVideo(int fd);`

The function returns 0 or 1 (modulo errors) according to whether a video signal is present on the grabber's input.

`int Px_VideoType(int fd);`
`int Px_PalVideo(int fd);`
`int Px_NtscVideo(int fd);`

Check the current video mode. The first function returns either `PX_NTSC_VIDEO` or `PX_PAL_VIDEO` (or -1 if an error occcurs), while the other two are booleean (an error is reported as 0). Note that the video mode is not a meaningful information unless `Px_HaveVideo()` returns true.

`int Px_SetGain(int fd, Px_Gain *ptr);`
`int Px_GetGain(int fd, Px_Gain *ptr);`

The functions transfer to/from kernel space a description of the internal amplifier of the grabber. The gain is described by a "gross" field (0-3) and a "fine" field (0-255). Both are expressed as fields of `struct Px_Gain`.

`int Px_AutoGain(int fd, int value);`

Enables/disables the autogain feature. If the value is 0 autogain is disabled. If it is non-zero, it represents the speed of gain change; suggested values are in the range 1-10, as too big values may result in oscillation.

`int Px_SetLut(int fd, Px_Lut *ptr);`
`int Px_GetLut(int fd, Px_Lut *ptr);`

The functions transfer to/from kernel space a copy of the look-up table used by the grabber's digitizer. The LUT is described by a `struct Px_Lut` data structure, and use of the two functions is exemplified by the `pxlut` sample program.

`int Px_GetParameters(int fd, Px_Parameters *ptr);`
`int Px_SetParamaters(int fd, Px_Parameters *ptr);`

The functions read/write the current acquisition parameters. Since the structure includes a few low-level parameters, the suggested way to set them is a read/modify/write sequence. The actual paramters are described in Chapter 11 [Special Cameras], page 10.

Concept Index

P

Q

R

S

T

V

W

X

Y

Table of Contents