The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC680-SW-82

Linux Device Driver

64 Digital Input/Output

Version 1.1.x

User Manual

Issue 1.1.3

April 2010

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7 25469 Halstenbek, Germany

Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19

e-mail: info@tews.com www.tews.com

TPMC680-SW-82

Linux Device Driver

64 Digital Input/Output

Supported Modules:
TPMC680-10

Issue	Description	Date
1.0	First Issue	April 9, 2003
1.1.0	Kernel 2.6.x Revision	April 8, 2005
1.1.1	Description of installation revised	January 29, 2006
1.1.2	Filelist modified, new address TEWS LLC, general revision	November 9, 2006
1.1.3	address TEWS LLC removed	April 1, 2010

Table of Contents

1 Introduction

The TPMC680-SW-82 Linux device driver allows the operation of a TPMC680 digital I/O PMC on Linux operating systems.

Supported features:

- read digital input value (8 bit / 64 bit ports)
- write digital output value(8 bit / 64 bit ports)
- receive and transmit parallel data (16 bit / 32 bit handshake ports)
- configure port size, direction and handshake mode
- wait for a specified input event (8 bit / 64 bit ports)

The TPMC680-SW-82 device driver supports the modules listed below:

TPMC680-10	8 x 8 Bit Digital Inputs/Outputs (5V TTL)	(PMC)

To get more information about the features and usage of TPMC680 devices it is recommended to read the manuals listed below.

TPMC680 User Manual

TPMC680 Engineering Manual

2 Installation

The directory TPMC680-SW-82 on the distribution media contains the following files:

TPMC680-SW-82-1.1.3.pdf	This manual in PDF format
TPMC680-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

The GZIP compressed archive TPMC680-SW-82-SRC.tar.gz contains the following files and directories:

tpmc680/tpmc680.c	Driver source code
tpmc680/tpmc680def.h	Driver private include file
tpmc680/tpmc680.h	Driver public include file for application program
tpmc680/Makefile	Device driver make file
tpmc680/makenode	Script to create device nodes on the file system
tpmc680/include/tpxxxhwdep.c	Low level hardware access functions source file
tpmc680/include/tpxxxhwdep.h	Access functions header file
tpmc680/include/tpmodule.c	Driver independent library
tpmc680/include/tpmodule.h	Driver independent library header file
tpmc680/example/tpmc680exa.c	Example application
tpmc680/example/Makefile	Example application make file

In order to perform an installation, extract all files of the archive TPMC680-SW-82-SRC.tar.gz to the desired target directory and copy tpmc680.h into the '/usr/include' path.

2.1 Build and install the device driver

- Login as *root*
- Change to the target directory
- To create and install the driver in the module directory */lib/modules/<version>/misc* enter:

  # make install

- Only after the first build we have to execute *depmod* to create a new dependency description for loadable kernel modules. This dependency file is later used by modprobe to load the driver module.

  # depmod -aq

2.2 Uninstall the device driver

- Login as *root*
- Change to the target directory
- To remove the driver from the module directory */lib/modules/<version>/misc* enter:

  # make uninstall

- Update kernel module dependency description file

  # depmod -aq

2.3 Install device driver into the running kernel

- To load the device driver into the running kernel, login as root and execute the following commands:

  # modprobe tpmc680drv

- After the first build or if you are using dynamic major device allocation it's necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled the device file system (devfs) then you have to skip running the makenode script. Instead of creating device nodes from the script the driver itself takes creating and destroying of device nodes in its responsibility.

  # sh makenode

On success the device driver will create a minor device for each TPMC680 module found. The first TPMC680 module can be accessed with device node /dev/tpmc680_0, the second with device node /dev/tpmc680_1 and so on.

The assignment of device nodes to physical TPMC680 modules depends on the search order of the PCI bus driver.

2.4 Remove device driver from the running kernel

- To remove the device driver from the running kernel login as root and execute the following command:

  # modprobe –r tpmc680drv

If your kernel has enabled DEVFS, all /dev/tpmc680_x nodes will be automatically removed from your file system after this.

Be sure that the driver isn't opened by any application program. If opened you will get the response "*tpmc680drv: Device or resource busy*" and the driver will still remain in the system until you close all opened files and execute *modprobe –r* again.

2.5 Change Major Device Number

This paragraph is only for Linux kernels without DEVFS installed. The TPMC680 driver uses dynamic allocation of major device numbers per default. If this isn't suitable for the application it's possible to define a major number for the driver.

To change the major number edit the file tpmc680def.h, change the following symbol to appropriate value and enter `make install` to create a new driver.

TPMC680_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TPMC680_MAJOR       122
```

3 Device Input/Output functions

This chapter describes the interface to the device driver I/O system.

3.1 open()

NAME

open() - open a file descriptor

SYNOPSIS

#include <fcntl.h>

int open (*const char *filename, int flags*)

DESCRIPTION

The open function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file is to be opened. This is a bit mask; you create the value by the bitwise OR of the appropriate parameters (using the | operator in C).

See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int fd;

...

fd = open("/dev/tpmc680_0", O_RDWR);
if (fd < 0)
{
     /* handle open error conditions */
}
```

RETURNS

The normal return value from open is a non-negative integer file descriptor. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

E_NODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during open. For more information about open error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.2 close()

NAME

close() – close a file descriptor

SYNOPSIS

#include <unistd.h>

int **close** *(int filedes)*

DESCRIPTION

The close function closes the file descriptor *filedes*.

EXAMPLE

```
int fd;

...

if (close(fd) != 0)
{
     /* handle close error conditions */
}
```

RETURNS

The normal return value from close is 0. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

E_NODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during close. For more information about close error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 read()

NAME

read() – read from a device

SYNOPSIS

#include <unistd.h>
#include <tpmc680.h>

ssize_t read(*int filedes, void *buffer, size_t size*)

DESCRIPTION

The read function reads data from the TPMC680 device. How the data is read, differs dependent on the selected port size.

A pointer to the callers read buffer (*TP680_IOBUF*) and the size of this structure are passed by the parameters *buffer* and *size* to the device.

The *TP680_IOBUF* structure has the following layout:

```
typedef struct
{
        int             port;       /* 0.. 7                          */
        int             dataWidth;  /* size of data value             */
        int             dataSize;   /* size of data buffer in bytes   */
        void*           buf;
} TP680_IOBUF, *PTP680_IOBUF;
```

port

Specifies the port number, allowed values are between 0 and 7. Dependent on the selected configuration, some port numbers will not be allowed or some bits will not be available for data. The table below gives an overview.

Port Mode Configuration [1]		Connected to Port							
Port 0	Port 2	7	6	5	4	3	2	1	0
8 bit	8 bit	7	6	5	4	3	2	1	0
16 bit	8 bit	7	6	5 [2]	4 [2]	3	2	0	0
8 bit	16 bit	7	6	5 [2]	4 [2]	2	2	1	0
16 bit	16 bit	7	6	5 [2]	4 [2]	2	2	0	0
32 bit	---	7	6	5 [2]	4 [2]	0	0	0	0
64 bit	---	0	0	0	0	0	0	0	0

[1] The port mode configurations are assigned to the following driver configurations values:

Port Mode Configuration	**Driver Configuration Value** (tp680def.h)
8 bit	TP680_MODE_SIZE_8BIT
16 bit	TP680_MODE_SIZE_16BIT
32 bit	TP680_MODE_SIZE_32BIT
64 bit	TP680_MODE_SIZE_64BIT

[2] Bits 0/1 may be used for HS and are unreadable than.

portWidth

Specifies the size of the port, the allowed port size depends on the module configuration.

dataSize

Specifies the size of the data buffer. The *buf* parameter must point to a buffer of the specified size. The *dataSize* must be specified in bytes.

Port Mode Configuration	Allowed Parameter Values		
size	port	portWidth	dataSize
8 bit	0, 1, 2, 3, 4, 5, 6, 7	sizeof(unsigned char)	1
16 bit	0, 2	sizeof(unsigned short)	(1, 2, ..., *TP680_MAXTRANSFERLEN)* * sizeof(unsigned short)
32 bit	0	sizeof(unsigned long)	(1, 2, ..., *TP680_MAXTRANSFERLEN)* * sizeof(unsigned long)
64 bit	0	2 * sizeof(unsigned long)	2 * sizeof(unsigned long)

buf

Points to a data buffer the driver will copy the input value(s) to. The size of the buffer is specified with the parameter *dataSize*. The size of the buffer is limited by the value of the macro *TP680_MAXTRANSFERLEN* which specifies the maximum number of data elements; this means the maximum size of the buffer is dependent from the selected portWidth. The application will find the read data in the buffer.

EXAMPLE

```
#include <tpmc680.h>

int                hCurrent = 0;
TP680_IOBUF        rdBuf;
unsigned char      ucVal;
unsigned short     usBuf[5];
int                NumBytes;

hCurrent = open(...);

...

/*
** Read a value from port 7 using the 8-bit mode
*/
rdBuf.port = 7;
rdBuf.dataWidth = sizeof(unsigned char);
rdBuf.dataSize = sizeof(unsigned char);
rdBuf.buf = &ucVal;

NumBytes = read(hCurrent, &rdBuf, sizeof(rdBuf));
if (NumBytes > 0)
{
     /* Input data in ucVal */
}
else
{
     /* read error */
}

...
```

```
...

/*
** Read a maximum of 5 values from port 2 using the 16-bit mode
*/
rdBuf.port = 2;
rdBuf.dataWidth = sizeof(unsigned short);
rdBuf.dataSize = 5 * sizeof(unsigned short);
rdBuf.buf = &usBus[0];

NumBytes = read(hCurrent, &rdBuf, sizeof(rdBuf));
if (NumBytes > 0)
{
     /* Input data in array usBuf[] */
}
else
{
     /* read error */
}
```

RETURNS

On success read returns the size of written data. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL	Invalid argument. This error code is returned if the size of the read or data buffer is too small.
EFAULT	Invalid pointer to the read or data buffer.
EACCES	Access is not allowed, port has a false configuration.
ECHRNG	Invalid port number specified

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.4 write()

NAME

write() – read to a device

SYNOPSIS

#include <unistd.h>
#include <tpmc680.h>

ssize_t write(*int filedes, void *buffer, size_t size*)

DESCRIPTION

The write function writes data to the TPMC680 device. How the data is written, differs dependent on the selected port size.

A pointer to the callers write buffer (*TP680_IOBUF*) and the size of this structure are passed by the parameters *buffer* and *size* to the device.

The *TP680_IOBUF* structure has the following layout:

```
typedef struct
{
        int             port;       /* 0.. 7                          */
        int             dataWidth;  /* size of data value             */
        int             dataSize;   /* size of data buffer in bytes   */
        void*           buf;
} TP680_IOBUF, *PTP680_IOBUF;
```

port

Specifies the port number, allowed values are between 0 and 7. Dependent on the selected configuration, some port numbers will not be allowed or some bits will not be available for data. The table below gives an overview.

Port Mode Configuration [1]		Connected to Port							
Port 0	Port 2	7	6	5	4	3	2	1	0
8 bit	8 bit	7	6	5	4	3	2	1	0
16 bit	8 bit	7	6	5 [2]	4 [2]	3	2	0	0
8 bit	16 bit	7	6	5 [2]	4 [2]	2	2	1	0
16 bit	16 bit	7	6	5 [2]	4 [2]	2	2	0	0
32 bit	---	7	6	5 [2]	4 [2]	0	0	0	0
64 bit	---	0	0	0	0	0	0	0	0

[1] The port mode configurations are assigned to the following driver configurations values:

Port Mode Configuration	**Driver Configuration Value** (tp680def.h)
8 bit	TP680_MODE_SIZE_8BIT
16 bit	TP680_MODE_SIZE_16BIT
32 bit	TP680_MODE_SIZE_32BIT
64 bit	TP680_MODE_SIZE_64BIT

[2] Bits 0/1 may be used for HS and are unreadable than.

portWidth

Specifies the size of the port, the allowed port size depends on the module configuration.

dataSize

Specifies the size of the data buffer. The *buf* parameter must point to a buffer of the specified size. The *dataSize* must be specified in bytes.

Port Mode Configuration	Allowed Parameter Values		
size	port	portWidth	dataSize
8 bit	0, 1, 2, 3, 4, 5, 6, 7	sizeof(unsigned char)	1
16 bit	0, 2	sizeof(unsigned short)	(1, 2, ..., *TP680_MAXTRANSFERLEN)* * sizeof(unsigned short)
32 bit	0	sizeof(unsigned long)	(1, 2, ..., *TP680_MAXTRANSFERLEN)* * sizeof(unsigned long)
64 bit	0	2 * sizeof(unsigned long)	2 * sizeof(unsigned long)

buf

Points to a data buffer the driver will copy the output value(s) from. The size of the buffer is specified with the parameter *dataSize*. The size of the buffer is limited by the value of the macro *TP680_MAXTRANSFERLEN* which specifies the maximum number of data elements; this means the maximum size of the buffer is dependent from the selected portWidth. The application must fill in the output values in this buffer.

EXAMPLE

```
#include <tpmc680.h>

int                hCurrent = 0;
TP680_IOBUF        wrBuf;
unsigned char      ucVal;
unsigned short     usBuf[5];
int                NumBytes;

hCurrent = open(...);

...

/*
** Write a value (0x55) to port 7 using the 8-bit mode
*/
wrBuf.port = 7;
wrBuf.dataWidth = sizeof(unsigned char);
wrBuf.dataSize = sizeof(unsigned char);
wrBuf.buf = &ucVal;

ucVal = 0x55;

NumBytes = read(hCurrent, &wrBuf, sizeof(wrBuf));
if (NumBytes > 0)
{
     /* 8-bit value written */
}
else
{
     /* write error */
}

...
```

```
...

/*
** Write 5 values to port 2 using the 16-bit mode
*/
wrBuf.port = 2;
wrBuf.dataWidth = sizeof(unsigned short);
wrBuf.dataSize = 5 * sizeof(unsigned short);
wrBuf.buf = &usBus[0];

usBus[0] = 0x1111;
usBus[1] = 0x1112;
usBus[2] = 0x1122;
usBus[3] = 0x1222;
usBus[4] = 0x2222;

NumBytes = write(hCurrent, &wrBuf, sizeof(wrBuf));
if (NumBytes > 0)
{
     /* data written to 16-bit port */
}
else
{
     /* write error */
}
```

RETURNS

On success read returns the size of written data. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL	Invalid argument. This error code is returned if the size of the write or data buffer is too small.
EFAULT	Invalid pointer to the write or data buffer.
EACCES	Access is not allowed, port has a false configuration.
ECHRNG	Invalid port number specified

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.5 ioctl()

NAME

ioctl() – device control functions

SYNOPSIS

#include <sys/ioctl.h>
#include <tpmc680.h>

int ioctl(int filedes, int request [, void *argp])

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *filedes*, causing the corresponding device to perform the requested operation.

The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *tpmc680.h*:

Value	Meaning
TP680_IOC_SETMODE	Configure ports
TP680_IOC_EVENTWAIT	Wait for a specified event

See behind for more detailed information on each control code.

To use these TPMC680 specific control codes the header file tpmc680.h must be included in the application

RETURNS

On success, zero is returned. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL	Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument *request*.

Other function dependant error codes will be described for each ioctl code separately. Note, the TPMC680 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.5.1 TP680_IOC_SETMODE

NAME

TP680_IOC_SETMODE – Configure a port

DESCRIPTION

This ioctl function configures the specified port of the TPMC680.

A pointer to the callers configuration buffer (*TP680_SETMODEBUF*) is passed by the parameter *argp* to the driver. The optional argument can be omitted for this ioctl function.

The *TP680_SETMODEBUF* structure has the following layout:

```
typedef struct
{
        unsigned long       port;       /* Port number to handle */
        unsigned long       Size;       /* Port size */
        unsigned long       Direction;  /* Port direction */
        unsigned long       HSMode;     /* Handshake Output Mode */
        unsigned long       HSFifo;     /* Handshake Output Fifo Mode */
} TP680_SETMODEBUF, *PTP680_SETMODEBUF;
```

port

This member specifies the port that shall be configured. Valid values are between 0 and 7.

Size

This argument specifies the port size. The following table describes the allowed port sizes and for which ports they are allowed.

Value	Ports	Description
TP680_MODE_SIZE_8BIT	0, 1, 2, 3, 4, 5, 6, 7	The port has a width of 8 bit. Each port can be accessed separately.
TP680_MODE_SIZE_16BIT	0,2	The port has a width of 16 bit and the output is controlled by the handshake signals. Two ports are used together. If port 0 is selected port 1 is used also. If port 2 is selected also port 3 will be used. The configuration of the connected ports is always adapted. **If this mode is selected for any port the handshake port 4 will be configured as an 8-bit input port.**
TP680_MODE_SIZE_32BIT	0	The port has a width of 32 bit and the output is controlled by the handshake signals. The ports 0, 1, 2 and 3 will be used together. The configuration of the connected ports is always set together. **If this mode is selected the handshake port 4 will be configured as an 8-bit input port.**
TP680_MODE_SIZE_64BIT	0	All ports are connected and can be used as simple 64 bit input or output port. All ports get the same configuration.

Direction

This member specifies direction of the port. All connected ports will get the same direction. Allowed values are:

TP680_MODE_DIR_INPUT	The port will be used as an input port.
TP680_MODE_DIR_OUTPUT	The port will be used as an output port.

HSMode

This value specifies the handshake mode and is only valid if the port shall be configured in 16 or 32 bit handshake mode (*TP680_MODE_SIZE_16BIT, TP680_MODE_SIZE_32BIT*). Using an output handshake, will change the direction of port 5 to output. The allowed values are:

TP680_MODE_HSFLAG_NO	No output handshake will be used.
TP680_MODE_HSFLAG_INTERLOCKED	The interlocked output handshake mode will be used.
TP680_MODE_HSFLAG_PULSED	The pulsed output handshake mode will be used.

HSFifo

This value specifies the handshake event depending on the handshake FIFO fill level. This value is only used if an output handshake is configured. The values are:

TP680_MODE_HSFIFOEV_NOTFULL	The event announces FIFO is not full.
TP680_MODE_HSFIFOEV_EMPTY	The event announces FIFO is empty.

When setting up ports other that depends on the selected, may change direction or mode. (Please refer to the TPMC680 User Manual.

Changing a port size from big to small will also change the mode of the previously connected ports. The ports will be set into 8 bit mode and they will keep their direction.

EXAMPLE

```
#include <tpmc680.h>

int                hCurrent = 0;
int                result;
TP680_SETMODEBUF   modeBuf;

hCurrent = open(...);

...

/* Configure port (2) for 16-bit output handshake mode */

modeBuf.port       = 2;
modeBuf.Size       = TP680_MODE_SIZE_16BIT;
modeBuf.Direction  = TP680_MODE_DIR_OUTPUT;
modeBuf.HSMode     = TP680_MODE_HSFLAG_INTERLOCKED;
                           /* interlocked output HS mode */
modeBuf.HSFifo     = TP680_MODE_HSFIFOEV_EMPTY;
                           /* ouput event on FIFO empty */

result = ioctl(hCurrent, TP680_IOC_SETMODE, &modeBuf);
if(result >= 0)
{
     /* Setting port mode successful */
}
else
{
     /* Setting portmode failed */
}
```

ERRORS

EFAULT	Invalid pointer to the configuration buffer.
ECHRNG	Invalid port number specified
EACCES	Access is not allowed, port has a false configuration

SEE ALSO

ioctl man pages

3.5.2 TP680_IOC_EVENTWAIT

NAME

TP680_IOC_EVENTWAIT – Wait for a specified input event

DESCRIPTION

This ioctl function waits for a specified event on a specified input line of the TPMC680.

A pointer to the callers event buffer (*TP680_EVENTWAITBUF*) is passed by the parameter *argp* to the driver.The optional argument can be omitted for this ioctl function.

The *TP680_EVENTWAITBUF* structure has the following layout:

```
typedef struct
{
        unsigned long       port;       /* Port number to handle */
        unsigned long       lineNo;     /* Input Line, event shall occur on */
        unsigned long       transition; /* Specify transition */
        unsigned long       timeout;    /* Timeout in seconds */
} TP680_EVENTWAITBUF, *PTP680_EVENTWAITBUF;
```

port

This member specifies the port to wait for. Valid values are between 0 and 7.

lineNo

This member specified the line to wait for. Valid values are between 0 and 7.

transition

This member specifies the event to wait for. The following events are supported:

TP680_IO_EDGE_HI	The event will occur if the specified input line changes from Low to High.
TP680_IO_EDGE_LO	The event will occur if the specified input line changes from High to Low.
TP680_IO_EDGE_ANY	The event will occur if the specified input line changes its value.

timeout

This argument specifies the timeout in ticks. If the specified event does not occur in the specified time, the function will return with an error code.

This function is only supported for 8 bit and 64 bit ports. Other configurations will return an error code.

EXAMPLE

```
#include <tpmc680.h>

int                hCurrent = 0;
int                result;
TP680_EVENTWAITBUF eventBuf;

hCurrent = open(...);

...

/*
** Wait for a high to low transition on line 3 of port 3, timeout after
** 10000 ticks.
*/
eventBuf.port           = 3;
eventBuf.lineNo         = 3;
eventBuf.transition     = TP680_IO_EDGE_LO;
eventBuf.timeout        = 10000;

result = ioctl(hCurrent, TP680_IOC_EVENTWAIT, &eventBuf);
if(result >= 0)
{
     /* Event occurred */
}
else
{
     /* Event did not occur or access failed */
}
```

ERRORS

EFAULT	Invalid pointer to the configuration buffer.
ECHRNG	Invalid port number specified
EACCES	Access is not allowed, port has a false configuration
EBUSY	The input line is already connected to a waiting event

SEE ALSO

ioctl man pages

4 Diagnostic

If the TPMC680 does not work properly it is helpful to get some status information from the driver respective kernel.

The Linux */proc* file system provides information about kernel, resources, driver, devices and so on. The following screen dumps displays information of a correct running TPMC680 driver (see also the proc man pages).

```
# cat /proc/pci
...
  Bus  0, device  17, function  0:
    Signal processing controller: PCI device 1498:02a8 (TEWS Datentechnik
GmBH) (rev 0).
      IRQ 11.
      Non-prefetchable 32 bit memory at 0xcfffff00 [0xcfffff7f].
      I/O at 0xcc00 [0xcc7f].
      Non-prefetchable 32 bit memory at 0xcffffd00 [0xcffffdff].


# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 ttyS
  5 cua
  7 vcs
 10 misc
 13 input
 14 sound
 29 fb
 36 netlink
162 raw
180 usb
226 drm
254 tpmc680drv
```

```
# cat /proc/interrupts
           CPU0
  0:     308553          XT-PIC  timer
  1:        958          XT-PIC  keyboard
  2:          0          XT-PIC  cascade
  5:       1309          XT-PIC  usb-ohci, eth0
  8:          1          XT-PIC  rtc
 11:      22796          XT-PIC  usb-ohci, SiS 7012, TPMC680
 12:        373          XT-PIC  PS/2 Mouse
 14:      10642          XT-PIC  ide0
 15:          1          XT-PIC  ide1
NMI:          0
ERR:          0

# cat /proc/ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
...
03f8-03ff : serial(auto)
0cf8-0cff : PCI conf1
a000-afff : PCI Bus #01
dc00-dc7f : PCI device 1498:02a8 (TEWS Datentechnik GmBH)
ff00-ff0f : Silicon Integrated Systems [SiS] 5513 [IDE]
  ff00-ff07 : ide0
  ff08-ff0f : ide1
```

```
# cat /proc/iomem
00000000-0009fbff : System RAM
0009fc00-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000f0000-000fffff : System ROM
00100000-0ffeffff : System RAM
  00100000-00247f2e : Kernel code
  00247f2f-0033ed03 : Kernel data
0fff0000-0fff7fff : ACPI Tables
0fff8000-0fffffff : ACPI Non-volatile Storage
c7c00000-cfcfffff : PCI Bus #01
  c8000000-cbffffff : ATI Technologies Inc Rage 128 Pro Ultra TF
cfe00000-cfefffff : PCI Bus #01
  cfefc000-cfefffff : ATI Technologies Inc Rage 128 Pro Ultra TF
cfffc000-cfffcfff : Standard Microsystems Corp [SMC] 83C170QF
  cfffc000-cfffcfff : epic100
cfffd000-cfffdfff : Silicon Integrated Systems [SiS] 7001
  cfffd000-cfffdfff : usb-ohci
cfffe000-cfffefff : Silicon Integrated Systems [SiS] 7001 (#2)
  cfffe000-cfffefff : usb-ohci
cffffe00-cffffeff : PCI device 1498:02a8 (TEWS Datentechnik GmBH)
  cffffe00-cffffeff : TPMC680
cfffff80-ffffffff : PCI device 1498:02a8 (TEWS Datentechnik GmBH)
d0000000-d3ffffff : Silicon Integrated Systems [SiS] 735 Host
fec00000-fec00fff : reserved
fee00000-fee00fff : reserved
ffee0000-ffefffff : reserved
fffc0000-ffffffff : reserved
```