The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV003-SW-82

Linux Device Driver

16 (8) Bit Digital I/O

Version 2.1.x

User Manual

Issue 2.1.3

November 2017

powerBridge Computer

Ehlbeek 15a
30938 Burgwedel
fon 05139-9980-0
fax 05139-9980-49

www.powerbridge.de
info@powerbridge.de

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7 25469 Halstenbek, Germany
) 4101 4058 0 Fax: +49 (0) 4101 4058 19
nfo@tews.com www.tews.com

TDRV003-SW-82

Linux Device Driver

16 (8) Bit Digital I/O

Supported Modules:
TPMC670
TPMC671

Issue	Description	Date
1.0	First Issue	August 21, 2000
1.1	Kernel 2.4.4 Support	August 23, 2001
1.2	General Revision	February 27, 2004
1.3.0	Kernel 2.6 Support	March 10, 2005
2.0.0	TPMC680-SW-82 changed to TDRV003-SW-82 TPMC671 Support added File list changed	August 8, 2006
2.0.1	New address TEWS LLC	August 29, 2006
2.0.2	New file list (config.h added), description of archive extraction	September 26, 2007
2.1.0	Description of new output functions added	June 12, 2008
2.1.1	Address TEWS LLC removed	July 20, 2010
2.1.2	Layout specific modifications	February 11, 2011
2.1.3	File-List modified	November 24, 2017

Table of Contents

1 Introduction

The TDRV003-SW-82 Linux device driver allows the operation of the TPMC670 family digital I/O PMC conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close(), read()*, *write()* and *ioctl()* functions.

Special I/O operation that do not fit to the standard I/O calls will be performed by calling the *ioctl()* function with a specific function code and an optional function dependent argument.

The TDRV003-SW-82 device driver supports the following features:

- Reading digital input value immediately or after a selected event occurs
- Writing digital output value
- Set and clear single output lines
- Watchdog operation
- Input hardware debouncing

The TDRV003-SW-82 device driver supports the modules listed below:

TPMC670	16 (8) Channel Digital I/O	(PMC)
TPMC671	16 Channel Digital I/O	(PMC)

In this document all supported modules and devices will be called TDRV003. Specials for certain devices will be advised.

To get more information about the features and usage of TDRV003 devices it is recommended to read the manuals listed below.

TPMC670/TPMC671 User Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV003-SW-82':

TDRV003-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
TDRV003-SW-82-2.1.3.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

The GZIP compressed archive TDRV003-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tdrv003/':

tdrv003.c	TDRV003 device driver source
tdrv003def.h	TDRV003 driver include file
tdrv003.h	TDRV003 include file for driver and application
Makefile	Device Driver Makefile
makenode	Script for Device Node Creation in File System
include/tpxxxhwdep.c	Hardware dependent library
include/tpxxxhwdep.h	Hardware dependent library header file
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
include/config.h	Driver independent library header file
example/tdrv003exa.c	Example Application
example/Makefile	Makefile for Example Application
COPYING	Copy of the GNU Public License (GPL)

In order to perform an installation, extract all files of the archive TDRV003-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TDRV003-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tdrv003.h to */usr/include*

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

- To update the device driver's module dependencies, enter:

  # **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

2.3 Install the Device Driver into a running Kernel

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

  # modprobe tdrv003drv

- After the first build or if you are using dynamic major device allocation it is necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled a device file system (devfs or sysfs with udev) 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 TDRV003 module found. The first TDRV003 module can be accessed with device node /dev/tdrv003_0, the second module with device node /dev/tdrv003_1, and so on.

The assignment of device nodes to physical TDRV003 modules depends on the search order of the PCI bus driver.

2.4 Remove the Device Driver from a running Kernel

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

  # modprobe -r tdrv003drv

If your kernel has enabled devfs or sysfs (udev), all /dev/tdrv003_x nodes will be automatically removed from your file system after this.

Make sure that the driver isn't opened by any application program. If opened you will get the response "*tdrv003drv: 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 TDRV003 driver use dynamic allocation of major device numbers per default. If this isn't suitable for the application it is possible to define a major number for the driver.

To change the major number edit the file tdrv003def.h, change the following symbol to appropriate value and enter *make install* to create a new driver.

TDRV003_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TDRV003_MAJOR        122
```

Be sure that the desired major number is not used by other drivers. Please check */proc/devices* to see which numbers are free.

Keep in mind that it is necessary to create new device nodes if the major number for the TDRV003 driver has changed and the makenode script is not used.

3 I/O Functions

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

3.1 open

NAME

open() opens 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

```
#include <fcntl.h>

int  fd;

…

fd = open("/dev/tdrv003_0", O_RDWR);
if (fd == -1)
{
     /* handle error condition */
}
```

RETURNS

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

ERRORS

Error Code	Description
ENODEV	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() closes a file descriptor.

SYNOPSIS

#include <unistd.h>

int close
(
  int filedes
)

DESCRIPTION

The close function closes the file descriptor *filedes*.

EXAMPLE

```
#include <unistd.h>

int  fd;

…

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

RETURNS

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

ERRORS

Error Code	Description
ENODEV	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() reads from a device.

SYNOPSIS

```
#include    <sys/ioctl.h>
#include    <tdrv003.h>

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

DESCRIPTION

The read function reads the contents of the input port either immediately or after a specified event has occurred. Possible events are rising or falling edge, or any of it at a specified input bit, or a pattern match of masked input bits.

A pointer to the callers read buffer *TDRV003_READBUF* and the size of this structure are passed by the parameters buffer and size to the device.

```
typedef struct
{
        unsigned short      value;
        unsigned short      mode;
        unsigned short      mask;
        unsigned short      match;
        unsigned long       timeout;
} TDRV003_READBUF, *PTDRV003_READBUF;
```

value

This parameter receives the contents of the input port. For TPMC670-11/-21 modules only bits 0..7 are relevant. Bit 0 corresponds to the first input line, bit 1 corresponds to the second input line and so on.

mode

This parameter specifies the “event” mode for this read request.

Event	Description
TDRV003_NOW	The driver reads the input port and returns immediately to the caller. The parameters *mask*, *match* and *timeout* are not relevant in this mode.
TDRV003_MATCH	The driver reads the input port if the masked input bits match to the specified pattern. The input mask must be specified in the parameter *mask*. A 1 value in *mask* means that the input bit value “must-match” identically to the corresponding bit in the *match* parameter.
TDRV003_HIGH_TR	The driver reads the input port if a high-transition at the specified bit position occurs. A 1 value in *mask* specifies the bit position of the input port. If more than one bit position is specified the events are ORed. That means the read operation is completed if a high-transition at least at one relevant bit position occurs.
TDRV003_LOW_TR	The driver reads the input port if a low-transition at the specified bit position occurs. A 1 value in *mask* specifies the bit position of the input port. If more than one bit position is specified the events are ORed. That means the read operation is completed if a low-transition at least at one relevant bit position occurs.
TDRV003_ANY_TR	The driver reads the input port if a transition (high or low) at the specified bit position occurs. A 1 value in *mask* specifies the bit position of the input port. If more than one bit position is specified the events are ORed. That means the read operation is completed if a transition at least at one relevant bit position occurs.

mask

This parameter specifies a bit mask. A 1 value marks the corresponding bit position as relevant.

match

This parameter specifies a pattern that must match to the contents of the input port. Only the bit positions specified by *mask* must compare to the input port.

timeout

This parameter specifies the amount of time (in ticks) the caller is willing to wait for the specified event to occur. A value of 0 means wait indefinitely.

Remember interrupt latency:

The returned value for all event reads (*TDRV003_MATCH*, *TDRV003_HIGH_TR*, *TDRV003_LOW_TR*, and *TDRV003_ANY_TR*) may not return the value of the moment the event has occurred, because the input value is read in the ISR.

***TDRV003_MATCH* may miss very short events, because match check is made in the ISR. Therefore it is recommended not to use this event for fast changing inputs**

EXAMPLE

```
#include <sys/ioctl.h>
#include <tdrv003.h>

int                 fd;
ssize_t             NumBytes;
TDRV003_READBUF     ReadBuf;

…

/* Read input port immediately without waiting for any event */
ReadBuf.mode = TDRV003_NOW;

NumBytes = read(fd, &ReadBuf, sizeof(ReadBuf));
if (NumBytes > 0)
{
     printf("Input port = 0x%x\n", ReadBuf.value);
}
else
{
     /* handle read error */
}

…

/* Read the input port after          */
/* bits 0, 6 = 0 and bits 1, 7 = 1    */
ReadBuf.mode        = TDRV003_MATCH;
ReadBuf.mask        = 0x00C3;          /* bit 0,1,6,7 are relevant */
ReadBuf.match       = 0x0082;
ReadBuf.timeout     = 100;             /* ticks */

NumBytes = read(fd, &ReadBuf, sizeof(ReadBuf));
if (NumBytes > 0)
{
     printf("Input port = 0x%x\n", ReadBuf.value);
}
else
{
     /* handle read error */
}

…
```

…

```
/* Read the input port after a high-transition at bit 7 occurred */
ReadBuf.mode       = TDRV003_HIGH_TR;
ReadBuf.mask       = 1 << 7;          /* high-transition at bit 7 */
ReadBuf.timeout    = 100;             /* ticks */

NumBytes = read(fd, (char*)&ReadBuf, sizeof(ReadBuf));
if (NumBytes > 0)
{
     printf("Input port = 0x%x\n", ReadBuf.value);
}
else
{
     /* handle read error */
}
```

…

RETURNS

On success read returns the size of the structure *TDRV003_READBUF*. In case of an error, a value of –1 is returned. The global variable errno contains the detailed error code.

ERRORS

Error Code	Description
EINVAL	Invalid argument. This error code is returned if the size of the read buffer is too small.
EFAULT	Invalid pointer to the read buffer
EBUSY	The maximum number of concurrent read requests has exceeded. Increase the value of *MAX_REQUESTS* in *tdrv003def.h*.
ETIME	The allowed time to finish the read request has elapsed.
EINTR	Interrupted function call; an asynchronous signal occurred and prevented completion of the call. When this happens, try the call again.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.4 write

write() writes to a device.

SYNOPSIS

#include <unistd.h>

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

DESCRIPTION

The write function writes a new output value to the device specified by the descriptor *filedes*. A pointer to an *unsigned short* buffer and the size of an unsigned short variable is passed by the parameters *buffer* and *size* to the device. Bit 0 corresponds to the first output line, bit 1 corresponds to the second output line and so on.

EXAMPLE

```
#include <unistd.h>

int                fd;
ssize_t            NumBytes;
unsigned short     OutData;


/*-------------------------------------------------
  Set output lines 1, 2, 3, and 12 (bits 0, 1, 2, 11)
  All other outputs will be set to 0.
  -------------------------------------------------*/

OutData = (1 << 11) | (1 << 2) | (1 << 1) | (1 << 0);

NumBytes = write(fd, (char*)&OutData, sizeof(unsigned short));
if (NumBytes > 0)
{
       /* Data successful written */
}
```

RETURNS

On success **write** returns the size of written data (2). In case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

Error Code	Description
EINVAL	Invalid argument. This error code is returned if the size of the write buffer is too small.
EFAULT	Invalid pointer to the write buffer.
EACCES	The output register is locked by a watchdog failure. Execute the ioctl function *TDRV003_IOCWDRESET* to reset the watchdog error.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.5 ioctl

NAME

ioctl() device control functions

SYNOPSIS

```
#include    <sys/ioctl.h>
#include    <tdrv003.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 *tdrv003.h*:

Value	Meaning
TDRV003_IOCWDENABLE	Enable output watchdog
TDRV003_IOCWDDISABLE	Disable output watchdog
TDRV003_IOCWDRESET	Reset output watchdog
TDRV003_IOCDEBENABLE	Enable input debouncer function
TDRV003_IOCDEBDISABLE	Disable input debouncer function
TDRV003_IOCGOUTPUTGET	Read back output value
TDRV003_IOCTOUTPUTSETBITS	Set specific bits of output value
TDRV003_IOCTOUTPUTCLEARBITS	Clear specific bits of output value
TDRV003_IOCSWRITEMASK	Write output value in conjunction with bitmask

See below for more detailed information on each control code.

To use these TDRV003 specific control codes the header file *tdrv003.h* must be included in the application.

RETURNS

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

ERRORS

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

Other function dependent error codes will be described for each ioctl code separately. Note, the TDRV003 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.5.1 TDRV003_IOCWDENABLE

NAME

TDRV003_IOCWDENABLE - enables output watchdog

DESCRIPTION

This ioctl function enables the output watchdog function of the TDRV003 device after the next write operation to the device. Please remember if the watchdog is enabled and no write access occurs within 120 ms all outputs go into the OFF state. To unlock the output register and leave the OFF state the ioctl function *TDRV003_IOCWDRESET* must be executed.

The optional argument can be omitted for this ioctl function.

EXAMPLE

```
#include <sys/ioctl.h>
#include <tdrv003.h>

…

int       fd;
int       result;

…

result = ioctl(fd, TDRV003_IOCWDENABLE);
if (result < 0)
{
     /* handle ioctl error */
}

…
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages

3.5.2 TDRV003_IOCWDDISABLE

NAME

TDRV003_IOCWDDISABLE - disables output watchdog

DESCRIPTION

This ioctl function disables the output watchdog function of the TDRV003 device.

The optional argument can be omitted for this ioctl function.

EXAMPLE

```
#include <sys/ioctl.h>
#include <tdrv003.h>

…

int       fd;
int       result;

…

result = ioctl(fd, TDRV003_IOCWDDISABLE);
 if (result < 0)
{
     /* handle ioctl error */
}

…
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages

3.5.3 TDRV003_IOCWDRESET

NAME

TDRV003_IOCWDRESET - resets output watchdog

DESCRIPTION

This ioctl function resets an output watchdog error. If the write function returns the error code *EACCES* this ioctl function must be executed to unlock the output register.

The optional argument can be omitted for this ioctl function.

EXAMPLE

```
#include <sys/ioctl.h>
#include <tdrv003.h>

…

int       fd;
int       result;

…

result = ioctl(fd, TDRV003_IOCWDRESET);
if (result < 0)
{
     /* handle ioctl error */
}

…
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages

3.5.4 TDRV003_IOCDEBENABLE

NAME

TDRV003_IOCDEBENABLE - enables input debouncer function.

DESCRIPTION

This ioctl function enables the input debouncer function. The argument *argp* passes the new debouncer counter value to the driver.

Please refer to the corresponding TDRV003 device hardware user manual for calculation formulas for appropriate counter values.

EXAMPLE

```
#include <sys/ioctl.h>
#include <tdrv003.h>

…

int       fd;
int       result;

…

/* Enable the debouncer with a debounce time of ~1ms */
result = ioctl(fd, TDRV003_IOCDEBENABLE , (unsigned short)147);
if (result < 0)
{
     /* handle ioctl error */
}

…
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages

3.5.5 TDRV003_IOCDEBDISABLE

NAME

TDRV003_IOCDEBDISABLE - disables input debouncer function.

DESCRIPTION

This ioctl function disables the input debouncer function.

The optional argument can be omitted for this ioctl function.

EXAMPLE

```
#include <sys/ioctl.h>
#include <tdrv003.h>

…

int       fd;
int       result;

…

result = ioctl(fd, TDRV003_IOCDEBDISABLE);
if (result < 0)
{
     /* handle ioctl error */
}

…
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages

3.5.6 TDRV003_IOCGOUTPUTGET

NAME

TDRV003_IOCGOUTPUTGET – Read back output value

DESCRIPTION

This ioctl function reads back the current output value. The function specific control parameter **argp** is a pointer to an *unsigned short* value. Bit 0 corresponds to the first output line, bit 1 corresponds to the second output line and so on.

EXAMPLE

```
#include "tdrv003.h"

int                fd;
unsigned short     OutputState;
int                retval;

/*-------------------------------------------------
  Read current state of output lines
  -------------------------------------------------*/

retval = ioctl(fd, TDRV003_IOCGOUTPUTGET, (int)&OutputState);
if (retval >= 0)
{
     printf( "Current output state: %X\n", OutputState );
} else {
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
EFAULT	Parameter data can not be copied to or from the driver's context. Please check the argument *argp*.

3.5.7 TDRV003_IOCTOUTPUTSETBITS

NAME

TDRV003_IOCTOUTPUTSETBITS – Set specific bits of output value

DESCRIPTION

This ioctl function sets some bits of the output value (set to 1). The function specific control parameter **argp** is a pointer to an *unsigned short* value. Bit 0 corresponds to the first output line, bit 1 corresponds to the second output line and so on. Unset bits will be left unchanged for the output.

EXAMPLE

```
#include "tdrv003.h"

int                fd;
unsigned short     OutputBits;
int                retval;

/*-------------------------------------------------
  Set output lines 2, 3, and 16 (bits 1, 2 and 15)
  -------------------------------------------------*/
OutputBits = (1 << 15) | (1 << 2) | (1 << 1);
retval = ioctl(fd, TDRV003_IOCTOUTPUTSETBITS, (int)&OutputBits);
if (retval < 0)
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
EFAULT	Parameter data can not be copied to or from the driver's context. Please check the argument *argp*.
EACCES	The output register is locked by a watchdog failure. Execute the ioctl function *TDRV003_IOCWDRESET* to reset the watchdog error.

3.5.8 TDRV003_IOCTOUTPUTCLEARBITS

NAME

TDRV003_IOCTOUTPUTCLEARBITS – Clear specific bits of output value

DESCRIPTION

This ioctl function clears some bits of the output value (set to 0). The function specific control parameter **argp** is a pointer to an *unsigned short* value. Bit 0 corresponds to the first output line, bit 1 corresponds to the second output line and so on. Unset bits will be left unchanged for the output.

EXAMPLE

```
#include "tdrv003.h"

int                fd;
unsigned short     OutputBits;
int                retval;

/*-------------------------------------------------
  Clear output lines 1 and 16 (bits 0 and 15)
  -------------------------------------------------*/
OutputBits = (1 << 15) | (1 << 0);
retval = ioctl(fd, TDRV003_IOCTOUTPUTCLEARBITS, (int)&OutputBits);
if (retval < 0)
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
EFAULT	Parameter data can not be copied to or from the driver's context. Please check the argument *argp*.
EACCES	The output register is locked by a watchdog failure. Execute the ioctl function *TDRV003_IOCWDRESET* to reset the watchdog error.

3.5.9 TDRV003_IOCSWRITEMASK

NAME

TDRV003_IOCSWRITEMASK – Write output value in conjunction with bitmask

DESCRIPTION

This ioctl function writes the output value in conjunction with a bitmask. Only specified bits will be changed. The function specific control parameter **argp** is a pointer to a *TDRV003_WRITEBUF* structure.

```
typedef struct
{
   unsigned short   value;
   unsigned short   mask;
} TDRV003_WRITEBUF;
```

value

This parameter specifies a 16bit word which reflects the out lines. Bit 0 corresponds to the first output line, bit 1 corresponds to the seconds output line and so on.

mask

This parameter specifies a 16bit mask. ‘1’ means that the corresponding bit in *value* will be updated. ‘0’ bits will be left unchanged. Bit 0 corresponds to the first output line, bit 1 corresponds to the seconds output line and so on.

EXAMPLE

```
#include "tdrv003.h"

int                 fd;
TDRV003_WRITEBUF    WriteBuf;
int                 retval;

/*-------------------------------------------------
  Set output line 1 and 8 (bit 0 and bit 7), and
  clear output line 16 (bit 15)
  -------------------------------------------------*/
WriteBuf.value = (1 <<  7) | (1 << 0);
WriteBuf.mask  = (1 << 15) | (1 << 7) | (1 << 0);

retval = ioctl(fd, TDRV003_IOCSWRITEMASK, (int)&WriteBuf);
if (retval < 0)
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
EFAULT	Parameter data can not be copied to or from the driver's context. Please check the argument *argp*.
EACCES	The output register is locked by a watchdog failure. Execute the ioctl function *TDRV003_IOCWDRESET* to reset the watchdog error.

4 Diagnostic

If the TDRV003 does not work properly it is helpful to get some status information from the driver respective the kernel.

The Linux */proc* file system provides information about kernel, resources, driver, devices, and so on. The following screen dumps display information of a correct running TDRV003 driver (see also the proc man pages). The examples show a system with 1x TPMC670 and 1x TPMC671.

```
# cat /proc/pci
…

  Bus  0, device  10, function  0:
    Signal processing controller: PCI device 1498:029f (TEWS Technologies
GmbH) (rev 0).
      IRQ 5.
      Non-prefetchable 32 bit memory at 0xeb020000 [0xeb02007f].
      I/O at 0xa400 [0xa47f].
      I/O at 0xa800 [0xa80f].
  Bus  0, device  12, function  0:
    Signal processing controller: PLX Technology, Inc. PCI <-> IOBus Bridge
(rev 1).
      IRQ 10.
      Non-prefetchable 32 bit memory at 0xeb021000 [0xeb02107f].
      I/O at 0xac00 [0xac7f].
      I/O at 0xb000 [0xb00f].

# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 ttyS
  5 cua
  6 lp
  7 vcs
 10 misc
 13 input
 29 fb
 36 netlink
129 ptm
…
142 pts
162 raw
254 tdrv003drv
```

```
# cat /proc/interrupts
           CPU0
  0:     282100          XT-PIC  timer
  1:          6          XT-PIC  keyboard
  2:          0          XT-PIC  cascade
  5:          8          XT-PIC  TDRV003
  8:          1          XT-PIC  rtc
 10:          0          XT-PIC  TDRV003
 11:       4648          XT-PIC  eth0
 12:        223          XT-PIC  PS/2 Mouse
 14:      12968          XT-PIC  ide0
 15:          0          XT-PIC  ide1
NMI:          0
ERR:          0

# cat /proc/ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
01f0-01f7 : ide0
02f8-02ff : serial(auto)
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(auto)
0cf8-0cff : PCI conf1
9000-9fff : PCI Bus #01
  9000-90ff : PCI device 1002:5964 (ATI Technologies Inc)
a000-a03f : Intel Corp. 82557/8/9 [Ethernet Pro 100]
  a000-a03f : e100
a400-a47f : PCI device 1498:029f (TEWS Datentechnik GmBH)
a800-a80f : PCI device 1498:029f (TEWS Datentechnik GmBH)
  a800-a80f : TDRV003
ac00-ac7f : PLX Technology, Inc. PCI <-> IOBus Bridge
b000-b00f : PLX Technology, Inc. PCI <-> IOBus Bridge
  b000-b00f : TDRV003
b400-b40f : VIA Technologies, Inc. VT82C586B PIPC Bus Master IDE
b800-b81f : VIA Technologies, Inc. USB
```