The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC501-SW-82

Linux Device Driver

32 Channel 16 Bit ADC

Version 1.3.x

User Manual

Issue 1.3.3

June 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

TPMC501-SW-82

Linux Device Driver

32 Channel 16 Bit ADC

Supported Modules:
TPMC501

Issue	Description	Date
1.0	First Issue	October 24, 2001
1.1	New ioctl() function TP501_IOCSMODTYPE	May 14, 2002
1.2	Distribution format has changed	December 17, 2003
1.3.0	Kernel 2.6.x Support	March 10, 2005
1.3.1	Filelist modified, New Address TEWS LLC, general revision	November 08, 2006
1.3.2	Installation description modified, read() parameter corrected	August 25, 2008
1.3.3	Address TEWS LLC removed	June 15, 2010

Table of Contents

1 Introduction

The TPMC501-SW-82 Linux device driver allows the operation of a TPMC501 ADC PMC on Linux operating systems.

The TPMC501 device driver includes the following features:

- read value from a selected ADC channel
- use sequencer mode for continuously read from selected channels
- correction of input values with the factory programmed correction data
- select hardware supported gains

In case of difficulties during driver installation please contact TEWS TECHNOLOGIES.

The TPMC501-SW-82 device driver supports the modules listed below:

TPMC501	Optically Isolated 32 Channel 16 Bit ADC	PMC

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

TPMC501 Hardware User manual

TPMC501 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path '.\TPMC501-SW-82\':

TPMC501-SW-82-1.3.3.pdf	This manual in PDF format
TPMC501-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

The GZIP compressed archive TPMC501-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tpmc501/':

tpmc501.c	Driver source code
tpmc501def.h	Driver include file
tpmc501.h	Driver include file for application program
Makefile	Device driver make file
makenode	Script for device node creation
include/config.h	Driver independent library header file
include/tpxxxhwdep.c	Low level hardware access functions source file
include/tpxxxhwdep.h	Access functions header file
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
example/tpmc501exa.c	Example application
example/Makefile	Example application makefile

In order to perform an installation, extract all files of the archive TPMC501-SW-82.tar.gz to the desired target directory. The command 'tar -xzvf TPMC501-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tpmc501.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

- 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 automatically load dependent kernel modules.

  # **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 tpmc501drv

- 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 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 TPMC501 found. The first TPMC501 module can be accessed with device node /dev/tpmc501_0, the second module with device node /dev/tpmc501_1, the third TPMC501 module with device node /dev/tpmc501_2 and so on.

The assignment of device nodes to physical TPMC501 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 tpmc501drv

If your kernel has enabled devfs or sysfs (udev), all /dev/tpmc501_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 "*tpmc501drv: 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

The TPMC501 driver uses dynamic allocation of major device numbers by default. If this isn't suitable for the application it is possible to define a major number for the driver. If the kernel has enabled devfs the driver will not use the symbol TPMC501_MAJOR.

To change the major number edit the file *tpmc501def.h*, change the following symbol to appropriate value and enter **make install** to create a new driver.

TPMC501_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TPMC501_MAJOR   122
```

Be sure that the desired major number isn't used by other drivers. Please check /proc/devices to see which numbers are free.

3 Device Input/Output 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. 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/tpmc501_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 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() closes 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 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() reads from a device.

SYNOPSIS

#include <unistd.h>

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

DESCRIPTION

The **read** function reads an ADC value from the specified channel.

A pointer to the callers read buffer *TP501_READBUF* and the size of this structure are passed by the parameters *buffer* and *size* to the device.

The *TP501_READBUF* structure has the following layout:

```
typedef struct
{
        unsigned short          channel;
        unsigned short          gain;
        unsigned short          flags;
        long                    value;
} TP501_READBUF, *PTP501_READBUF;
```

channel

This value specifies the ADC channel that will be used. Allowed values are 1 to 32 for single-ended input and 1 to 16 for differential input.

gain

Specifies the input gain that will be used. The following table shows the allowed values. These values are predefined in 'tpmc501.h'.

Name	TPMC501-10/-12/-20/-22	TPMC501-11/-13/-21/-23
TP501_GAIN1	gain = 1	gain = 1
TP501_GAIN2	gain = 2	gain = 2
TP501_GAIN4	not supported	gain = 4
TP501_GAIN5	gain = 5	not supported
TP501_GAIN8	not supported	gain = 8
TP501_GAIN10	gain = 10	not supported

flags

This value is an ORed value of the flags shown in the following table.

Name	Meaning
TP501_FL_DIFF	If this flag is set, the driver will use differential input signal. If the flag is not set, the driver will use single-ended input signal.
TP501_FL_CORR	If this flag is set, the driver will correct the ADC input value with the factory programmed correction data. If this flag is not set, the driver will return the ADC input.
TP501_FL_FAST	If this flag is set, the driver will start a conversion on the last programmed channel, with the last selected gain and the last selected input mode. The parameters *gain*, *channel* and the flag *TP501_FL_DIFF* will be ignored if this flag is set. If this flag is used, the hardware coded settling time is not needed and not used, this makes the access faster. If the flag is not set, the driver will work in the normal mode.

value

This parameter returns the converted ADC value.

EXAMPLE

```
#include <tpmc501.h>

int           hCurrent;
ssize_t       NumBytes;
TP501_READBUF ADCBuf;
…
/*****************************************
 Read channel 5 with differential input
 use gain 2
 correct the input data
*****************************************/
ADCBuf.channel      = 5;
ADCBuf.gain         = TP501_GAIN2;
ADCBuf.flags        = TP501_FL_DIFF | TP501_FL_CORR;

NumBytes = read(hCurrent, &ADCBuf, sizeof(ADCBuf));
if (NumBytes >= 0)
   printf( "\nADC Value = %ld\n", ADCBuf.value);
else
   printf("\nRead failed --> Error = %d\n", errno );
```

RETURNS

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

RETURNS

On success **read** returns the size of the structure *TP501_READBUF*. In 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 buffer is too small.
EFAULT	Invalid pointer to the read buffer
EBUSY	The sequencer mode is active on the specified device.
ETIME	The settling or conversion exceeds the supposed range.
ENOTYPEINIT	Driver specific error (150) – The module type has not been set. (Use ioctl-function TP501_IOCSMODTYPE)

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.4 ioctl()

NAME

ioctl() – device control functions

SYNOPSIS

#include <sys/ioctl.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 *tpmc501.h*:

Value	Meaning
TP501_IOCGREADPARAM	Get module parameters including module type and the factory programmed correction values.
TP501_IOCSEQSTOP	Stop the sequencer
TP501_IOCSSEQSETUP	Setup and start the sequencer
TP501_IOCGSEQREAD	Read ADC data from sequencer data RAM, synchronized on the sequencer cycle
TP501_IOCGSEQIMMREAD	Read ADC data from sequencer data RAM, make an immediate read, use the latest values. This read is asynchronous to the sequencer clock cycle.
TP501_IOCSMODTYPE	Setup model type.

See below for more detailed information on each control code.

Note: To use these TPMC501 specific control codes the header file tpmc501.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 also 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 TPMC501 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.4.1 TP501_IOCGREADPARAM

NAME

TP501_IOCGREADPARAM - Get the module parameters

DESCRIPTION

This ioctl function returns modules parameters. This includes the module type and the factory programmed correction data.

A pointer to the callers parameter buffer (TP501_PARABUF) is passed by the parameter argp to the driver.

The TP501_PARABUF structure has the following layout:

```
typedef struct
{
        int             ModuleType;
        signed short    OffsCorr[4];
        signed short    GainCorr[4];
} TP501_PARABUF, *PTP501_PARABUF;
```

ModuleType

This parameter returns the module type. ‘10’ will be returned for TPMC501-10, ‘11’ will be returned for TPMC501-11 and so on.

OffsCorr

This array returns the factory programmed offset correction data, which is used if the *TP501_FL_CORR* flag is set. The index of the array specifies the gain.

Value	Gain
0	1
1	2
2	4/5
3	8/10

GainCorr

This array returns the factory programmed gain correction data, which is used if the *TP501_FL_CORR* flag is set. The index of the array specifies the gain.

Value	Gain
0	1
1	2
2	4/5
3	8/10

EXAMPLE

```
#include <tpmc501.h>

int           hCurrent;
int           result;
TP501_PARABUF ParamBuf;

result = ioctl(hCurrent, TP501_IOCGREADPARAM, &ParamBuf);
if (result >= 0)
{
  printf("\nModule type = TPMC501-%02d\n",
    ParamBuf.ModuleType);
  printf("Offset Error = %d, %d, %d, %d\n",
    ParamBuf.OffsCorr[0],
    ParamBuf.OffsCorr[1],
    ParamBuf.OffsCorr[2],
    ParamBuf.OffsCorr[3]);
  printf("Gain Error   = %d, %d, %d, %d\n",
    ParamBuf.GainCorr[0],
    ParamBuf.GainCorr[1],
    ParamBuf.GainCorr[2],
    ParamBuf.GainCorr[3]);
}
else
{
  printf("\nRead module parameter failed -->  Error = %d\n", errno);
}
```

ERRORS

EINVAL	Invalid pointer to the parameter buffer. Please check the argument *argp*.

SEE ALSO

ioctl man pages

3.4.2 TP501_IOCSEQSTOP

NAME

TP501_IOCSEQSTOP – Stop Sequencer Mode

DESCRIPTION

This ioctl function stops the sequencer mode.

EXAMPLE

```
#include <tpmc501.h>

int  hCurrent;
int  result;


result = ioctl(hCurrent, TP501_IOCSEQSTOP);
if (result >= 0)
{
  printf("\nStopping sequencer successful\n");
}
else
{
  printf("\nStopping sequencer failed -->  Error = %d\n",
    errno);
}
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages

3.4.3 TP501_IOCSSEQSETUP

NAME

TP501_IOCSSEQSETUP - Setup and start the sequencer, enter sequencer mode

DESCRIPTION

This ioctl function sets up the TPMC501 to work in sequencer mode. The cycle time and the channel configuration are set up.

A pointer to the callers parameter buffer (*TP501_SEQSETBUF*) is passed by the parameter *argp* to the driver.

The *TP501_SEQSETBUF* structure has the following layout:

```
typedef struct
{
        unsigned short          cycleTime;
        struct
        {
                unsigned short  flags;
                unsigned short  gain;
        }       channel[TP501_SNGLCHANS];
} TP501_SEQSETBUF, *PTP501_SEQSETBUF;
```

cycleTime

This parameter specifies the cycle time that will be used. The value will be copied into the sequencer timer register. The value has a resolution of 100µs steps. If this value is set to zero, the sequencer will work in continuous mode.

structure channel

This array structure holds information for the channels. The index of the channel structure specifies the channel. Index 0 is advised to channel 1, index 1 is advised to channel 2 and so on. The array has 32 elements.

flags

This parameter is an ORed value of the following described flags.

Name	Meaning
TP501_FL_DIFF	If this flag is set, the driver will use differential input signal. If the flag is not set, the driver will use single-ended input signal.
TP501_FL_CORR	If this flag is set, the driver will correct the ADC input value with the factory programmed correction data. If this flag is not set, the driver will return the ADC input.

TP501_FL_ENABLE	If this flag is set the channel will be used in sequencer mode. If this flag is not set, the channel will be ignored in sequencer mode.

gain

This parameter specifies the gain.

Name	TPMC501-10/-12/-20/-22	TPMC501-11/-13/-21/-23
TP501_GAIN1	gain = 1	gain = 1
TP501_GAIN2	gain = 2	gain = 2
TP501_GAIN4	not supported	gain = 4
TP501_GAIN5	gain = 5	not supported
TP501_GAIN8	not supported	gain = 8
TP501_GAIN10	gain = 10	not supported

EXAMPLE

```
#include <tpmc501.h>

int             hCurrent;
int             result;
TP501_SEQSETBUF  SeqSetBuf;


/*********************************************************
 Start sequencer with a cycle time of 1 sec
 Enable following channels:
    Channel 1: Gain=1, Correction enabled, single-ended
    Channel 6: Gain=2, Correction disabled, differential
*********************************************************/
SeqSetBuf.cycleTime = 10000; /* 10000 * 100µs */

for (i = 0; i < TP501_SNGLCHANS; i++)
{
   SeqSetBuf.channel[i].flags = 0;    /* disable channel */
}

SeqSetBuf.channel[0].flags = TP501_FL_ENABLE | TP501_FL_CORR;
SeqSetBuf.channel[5].flags = TP501_FL_ENABLE | TP501_FL_DIFF;

SeqSetBuf.channel[0].gain = TP501_GAIN1;
SeqSetBuf.channel[5].gain = TP501_GAIN2;
…
```

…

```
result = ioctl(hCurrent, TP501_IOCSSEQSETUP, &SeqSetBuf);
if (result >= 0)
{
   printf("\nStarting sequencer successful\n");
}
else
{
   printf("\nStarting sequencer failed -->  Error = %d\n", errno);
}
```

ERRORS

EFAULT	Invalid pointer to the parameter buffer. Please check the argument argp.
ENOTYPEINIT	Driver specific error (150) – The module type has not been set.

SEE ALSO

ioctl man pages

3.4.4 TP501_IOCGSEQREAD

NAME

TP501_IOCGSEQREAD – Read a set of data value synchronized with cycle time

DESCRIPTION

This ioctl function returns a set of ADC data. The function returns ADC values for the channels, which had been enabled with the *TP501_IOCSSEQSETUP* function. The specified modes of the *TP501_IOCSSEQSETUP* function are used.

A pointer to the callers parameter buffer (*TP501_SEQREADBUF*) is passed by the parameter *argp* to the driver.

The *TP501_SEQREADBUF* structure has the following layout:

```
typedef struct
{
    int     overrunCount;
    int     error;
    long    values[TP501_SNGLCHANS];
} TP501_SEQREADBUF, *PTP501_SEQREADBUF;
```

overrunCount

This parameter returns the number of lost sequencer cycles. A value of '-1' means there has not been a valid cycle (only in error case), a value of '0' means no data has been lost. If the value is greater '0', than value set(s) had been lost.

error

This value returns an ORed value of the following error flags. This value should be checked for every call of the function.

Name	Meaning
TP501_FL_HWOVERRUN	The hardware has detected an overflow; the data sequencer has not been serviced in one cycle time.
TP501_FL_TIMERERR	The hardware has signaled that the specified cycle time is too short to make the specified conversions.
TP501_FL_INSTRAMERR	The hardware has detected an error in the instruction RAM. (No channel selected)
TP501_FL_SWOVERRUN	The driver can not make the data corrections in one cycle time.

values

This array returns a full set of ADC values. Only the values of the channels selected in *TP501_IOCSSEQSETUP* will be valid. The index specifies the channel. Index 0 is advised to channel 1, index 1 is advised to channel 2 and so on. The array has 32 elements.

EXAMPLE

```
#include <tpmc501.h>

int                hCurrent;
int                result;
TP501_SEQREADBUF   SeqReadBuf;

/*****************************************
 read values of the enabled channel 1 and 6
*****************************************/
result = ioctl(hCurrent, TP501_IOCGSEQREAD, &SeqReadBuf);
if (result >= 0)
{
  printf("Error %04Xh - Overruns %d\n",
    SeqReadBuf.error,
    SeqReadBuf.overrunCount);
  printf("Channel 1: %ld\n", SeqReadBuf.values[0]);
  printf("Channel 6: %ld\n", SeqReadBuf.values[5]);
}
else
{
  printf("\nReading values failed --> Error = %d\n", errno);
}
```

ERRORS

EFAULT	Invalid pointer to the parameter buffer. Please check the argument argp.

SEE ALSO

ioctl man pages

3.4.5 TP501_IOCGSEQIMMREAD

NAME

TP501_IOCGSEQIMMREAD – Read a set of data value unsynchronized with cycle time

DESCRIPTION

This ioctl function returns a set of ADC data. The function returns ADC values for the channels, which had been enabled with the *TP501_IOCSSEQSETUP* function. The specified modes of the *TP501_IOCSSEQSETUP* function are used.

A pointer to the callers parameter buffer (*TP501_SEQREADBUF*) is passed by the parameter *argp* to the driver.

The *TP501_SEQREADBUF* structure has the following layout:

```
typedef struct
{
    int     overrunCount;
    int     error;
    long    values[TP501_SNGLCHANS];
} TP501_SEQREADBUF, *PTP501_SEQREADBUF;
```

overrunCount

This parameter returns the number of lost sequencer cycles. A value of '-1' means there has not been a valid cycle since the last read, a value of '0' means no data has been lost. If the value is greater '0', than value set(s) had been lost.

error

This value returns an ORed value of the following error flags. This value should be checked for every call of the function.

Name	Meaning
TP501_FL_HWOVERRUN	The hardware has detected an overflow; the data sequencer has not been serviced in one cycle time.
TP501_FL_TIMERERR	The hardware has signaled that the specified cycle time is too short to make the specified conversions.
TP501_FL_INSTRAMERR	The hardware has detected an error in the instruction RAM. (No channel selected)
TP501_FL_SWOVERRUN	The driver can not make the data corrections in one cycle time.

values

This array returns a full set of ADC values. Only the values of the channels selected in *TP501_IOCSSEQSETUP* will be valid. The index specifies the channel. Index 0 is advised to channel 1, index 1 is advised to channel 2 and so on. The array has 32 elements.

EXAMPLE

```
#include <tpmc501.h>

int                hCurrent;
int                result;
TP501_SEQREADBUF   SeqReadBuf;


/*****************************************
 read values of the enabled channel 1 and 6
*****************************************/
result = ioctl(hCurrent, TP501_IOCGSEQIMMREAD, &SeqReadBuf);
if (result >= 0)
{
  printf("Error %04Xh - Overruns %d\n",
    SeqReadBuf.error,
    SeqReadBuf.overrunCount);
  printf("Channel 1: %ld\n", SeqReadBuf.values[0]);
  printf("Channel 6: %ld\n", SeqReadBuf.values[5]);
}
else
{
  printf("\nReading values failed --> Error = %d\n", errno);
}
```

ERRORS

EFAULT	Invalid pointer to the parameter buffer. Please check the argument argp.

SEE ALSO

ioctl man pages

3.4.6 TP501_IOCSMODTYPE

NAME

TP501_IOCSMODTYPE - Setup model type

DESCRIPTION

This ioctl function sets up the TPMC501 model type. The driver will store the model type and will return and correct ADC values depending from this value.

This function must be called before any read or sequencer access is done.

A pointer to the callers parameter buffer (*TP501_TYPEBUF*) is passed by the parameter *argp* to the driver.

The *TP501_TYPEBUF* structure has the following layout:

```
typedef struct
{
        int     ModuleType;         /* TPMC501 variant type */
} TP501_TYPEBUF, *PTP501_TYPEBUF;
```

ModuleType

This parameter specifies the model type. The following table shows the allowed values and the corresponding module types.

value	module type
10	TPMC501-10
11	TPMC501-11
12	TPMC501-12
13	TPMC501-13
20	TPMC501-20
21	TPMC501-21
22	TPMC501-22
23	TPMC501-23

EXAMPLE

```
#include <tpmc501.h>

int           hCurrent;
int           result;
TP501_TYPEBUF TypeBuf;


/*************************************************************
 Setup model type as TPMC501-10
 *************************************************************/
TypeBuf.ModuleType = 10;     /* TPMC501-10 */

result = ioctl(hCurrent, TP501_IOCSMODTYPE, &TypeBuf);
if (result >= 0)
{
   printf("\nSetting module type successful\n");
}
else
{
   printf("\nSetting module type failed -->  Error = %d\n",
     errno);
}
```

ERRORS

EFAULT	Invalid pointer to the parameter buffer. Please check the argument argp.

SEE ALSO

ioctl man pages

4 Diagnostic

If the TPMC501 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 TPMC501 driver (see also the proc man pages).

```
# cat /proc/pci
…
  Bus  0, device   9, function  0:
    Class 1180: PCI device 10b5:9050 (rev 1).
      IRQ 12.
      Non-prefetchable 32 bit memory at 0xe7000000 [0xe700007f].
      I/O at 0xe000 [0xe07f].
      I/O at 0xd800 [0xd8ff].
      Non-prefetchable 32 bit memory at 0xe6800000 [0xe68007ff].

# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 ttyS
  5 cua
  7 vcs
 10 misc
 29 fb
128 ptm
136 pts
162 raw
254 tpmc501drv

# cat /proc/interrupts
           CPU0
  0:     969329          XT-PIC  timer
  1:       4439          XT-PIC  keyboard
  2:          0          XT-PIC  cascade
  4:       2537          XT-PIC  serial
  8:          2          XT-PIC  rtc
 10:         14          XT-PIC  ncr53c8xx
 11:       5457          XT-PIC  eth0
 12:     924341          XT-PIC  TPMC501
 14:      58562          XT-PIC  ide0
 15:          5          XT-PIC  ide1
```

```
NMI:          0
ERR:          0
MIS:          0

# cat /proc/ioports
…
03f6-03f6 : ide0
03f8-03ff : serial(auto)
0cf8-0cff : PCI conf1
d000-d07f : PCI device 1011:0014
  d000-d07f : tulip
d400-d4ff : PCI device 1000:0001
  d400-d47f : ncr53c8xx
d800-d8ff : PCI device 10b5:9050
  d800-d8ff : TPMC501 (PCI)
e000-e07f : PCI device 10b5:9050
e800-e80f : PCI device 8086:7010
  e800-e807 : ide0
  e808-e80f : ide1

#cat /proc/iomem
00000000-0009ffff : System RAM
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000f0000-000fffff : System ROM
00100000-03ffffff : System RAM
  00100000-002327d1 : Kernel code
  002327d2-0031bdcb : Kernel data
e4000000-e4ffffff : PCI device 1002:4758
e5800000-e580007f : PCI device 1011:0014
  e5800000-e580007f : tulip
e6000000-e60000ff : PCI device 1000:0001
e6800000-e68007ff : PCI device 10b5:9050
e7000000-e700007f : PCI device 10b5:9050
ffff0000-ffffffff : reserved

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)
8000-807f : eth0
8100-811f : TPMC501(PCI)
```