

TPMC500-SW-82

Linux Device Driver

32 / 16 Channel 12 bit ADC

Version 1.3.x

User Manual

Issue 1.3.0

March 2005

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7
Phone: +49-(0)4101-4058-0
e-mail: info@tews.com

25469 Halstenbek / Germany
Fax: +49-(0)4101-4058-19
www.tews.com

TEWS TECHNOLOGIES LLC

1 E. Liberty Street, Sixth Floor
Phone: +1 (775) 686 6077
e-mail: usasales@tews.com

Reno, Nevada 89504 / USA
Fax: +1 (775) 686 6024
www.tews.com

TPMC500-SW-82

32 / 16 Channel 12 bit ADC

Linux Device Driver

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2001-2005 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	September 2001
1.1		May 2002
1.2	General Revision	February 2004
1.3.0	Kernel 2.6.x Support	March 14, 2005

Table of Contents

1	INTRODUCTION.....	4
2	INSTALLATION.....	5
	2.1 Build and install the device driver.....	5
	2.2 Uninstall the device driver	5
	2.3 Install device driver into the running kernel	5
	2.4 Remove device driver from the running kernel	6
	2.5 Change Major Device Number	7
	2.6 Setting module type.....	7
3	DEVICE INPUT/OUTPUT FUNCTIONS	8
	3.1 open()	8
	3.2 close().....	10
	3.3 read()	11
	3.4 ioctl()	14
	3.4.1 TP500_IOCGRADPARAM	16
	3.4.2 TP500_IOCSEQSTOP.....	18
	3.4.3 TP500_IOCSEQSETUP	19
	3.4.4 TP500_IOCSEQREAD.....	22
	3.4.5 TP500_IOCSEQIMMREAD.....	24
	3.4.6 TP500_IOCSEMODTYPE	26
4	DIAGNOSTIC.....	28

1 Introduction

The TPMC500-SW-82 Linux device driver allows the operation of a TPMC500 ADC PMC on Linux operating systems.

The TPMC500 device driver includes the following features:

- read value from a selected ADC channel
- use sequencer mode for continuously read from selected channel
- 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.

2 Installation

The directory TPMC500-SW-82 on the distribution media contains the following files:

TPMC500-SW-82.pdf	This manual in PDF format
TPMC500-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information

The GZIP compressed archive TPMC500-SW-82-SRC.tar.gz contains the following files and directories:

tpmc500.c	Driver source code
tpmc500def.h	Driver private include file
tpmc500.h	Driver public include file for application program
Makefile	Device driver make file
Makefile.elinos	Device driver make file for ELinOS
makenode	Script to create device nodes on the file system
tpxxhwdep.c	Low level WINNT style hardware access functions source file
tpxxhwdep.h	Access functions header file
tpmodule.c	Driver independent library
tpmodule.h	Driver independent library header file
Example/tpmc500example.c	Example application
Example/Makefile	Example application make file

In order to perform an installation, extract all files of the archive TPMC500-SW-82-SRC.tar.gz to the desired target directory.

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
```

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 device driver into the running kernel

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

```
# modprobe tpmc500drv
```

- 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 TPMC500 module found. The first TPMC500 module can be accessed with device node `/dev/tpmc500_0`, the second with device node `/dev/tpmc500_1` and so on.

The assignment of device nodes to physical TPMC500 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 tpmc500drv

If your kernel has enabled devfs, all `/dev/tpmc500_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 "*tpmc500drv: 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 TPM500 driver use 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 `tpmc500def.h`, change the following symbol to appropriate value and enter `make install` to create a new driver.

`TPMC500_MAJOR` Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TPMC500_MAJOR 122
```

2.6 Setting module type

Before any read or sequencer accesses are done, the module type must be set. This must be done with the `ioctl()` function `TP500_IOCSMODTYPE`. A description of this function can be found below in chapter "`TP500_IOCSMODTYPE`".

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 has 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 hCurrent;  
  
. . .  
  
hCurrent = open("/dev/tpmc500_0, O_RDWR);  
  
. . .
```

RETURNS

The usual 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 hCurrent;  
  
. . .  
  
if (close(hCurrent) != 0)  
{  
    * handle close error conditions */  
}
```

RETURNS

The usual 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 *TP500_READBUF* and the size of this structure are passed by the parameters *buffer* and *size* to the device.

The *TP500_READBUF* structure has the following layout:

```
typedef struct
{
    unsigned short    channel;           /* channel number */
    unsigned short    gain;             /* selected gain */
    unsigned short    flags;
    short             value;            /* ADC input value */
} TP500_READBUF, *PTP500_READBUF;
```

channel

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

It specifies the input gain that will be used, following table shows the allowed values. These values are predefined in *tpmc500.h*.

Name	TPMC500-10/-12/-20/-22	TPMC500-11/-13/-21/-23
<i>TP500_GAIN1</i>	gain = 1	gain = 1
<i>TP500_GAIN2</i>	gain = 2	gain = 2
<i>TP500_GAIN4</i>	not supported	gain = 4
<i>TP500_GAIN5</i>	gain = 5	not supported
<i>TP500_GAIN8</i>	not supported	gain = 8
<i>TP500_GAIN10</i>	gain = 10	not supported

flags

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

Name	Meaning
<i>TP500_FL_DIFF</i>	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.
<i>TP500_FL_CORR</i>	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.
<i>TP500_FL_FAST</i>	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 <i>gain</i> , <i>channel</i> and the flag <i>TP500_FL_DIFF</i> 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 speed up the access. If the flag is not set, the driver will work in the normal mode.

value

Parameter returns the converted ADC value.

EXAMPLE

```

int                hCurrent;
ssize_t           NumBytes;
TP500_READBUF     ADCBuf;

. . .

/*****
Read channel 5 with differential input
use gain 2
correct the input data
*****/
ADCBuf.channel     = 5;
ADCBuf.gain        = TP500_GAIN2;
ADCBuf.flags       = TP500_FL_DIFF | TP500_FL_CORR;

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

. . .

```

RETURNS

On success **read** returns a positive value. 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 also 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.

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 fildes, int request [, void *argp])
```

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *fildes*, 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 *tpmc500.h*:

Value	Meaning
<i>TP500_IOCGREADPARAM</i>	Get module parameters including module type and the factory programmed correction values
<i>TP500_IOCSEQSTOP</i>	Stop the sequencer
<i>TP500_IOCSEQSETUP</i>	Setup and start the sequencer
<i>TP500_IOCSEQREAD</i>	Read ADC data from sequencer data RAM, synchronized on the sequencer cycle
<i>TP500_IOCSEQIMMREAD</i>	Read ADC data from sequencer data RAM, make an immediate read, use the latest values. This read is asynchronous to the sequencer clock cycle.

See below for more detailed information on each control code.

To use these TPMC500 specific control codes the header file *tpmc500.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

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 TPMC500 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.4.1 TP500_IOCGBREADPARAM

NAME

TP500_IOCGBREADPARAM gets 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 *TP500_PARABUF* is passed by the parameter *argp* to the driver.

The *TP500_PARABUF* structure has the following layout:

```
typedef struct
{
    int             ModuleType;           /* TPMC500 variant type */
    signed char     OffsCorr[4];         /* Offset correction data */
    signed char     GainCorr[4];        /* Gain correction data */
} TP500_PARABUF, *PTP500_PARABUF;
```

ModuleType

Parameter returns the module type. '10' will be returned for TPMC500-10, '11' will be returned for TPMC500-11 and so on.

OffsCorr[]

Array returns the factory programmed offset correction data that is used if the TP500_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[]

Array returns the factory programmed gain correction data that is used if the TP500_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

```
int          hCurrent;
int          result;
TP500_PARABUF ParamBuf;

. . .

result = ioctl(hCurrent, TP500_IOCTL_READPARAM, &ParamBuf);
if (result >= 0)
{
    printf("\nModule type = TPMC500-%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

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

SEE ALSO

ioctl man pages

3.4.2 TP500_IOCSEQSTOP

NAME

TP500_IOCSEQSTOP stops sequencer mode.

DESCRIPTION

This ioctl function stops the sequencer mode.

EXAMPLE

```
int hCurrent;
int result;

. . .

result = ioctl(hCurrent, TP500_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 TP500_IOCSEQSETUP

NAME

TP500_IOCSEQSETUP setups and starts the sequencer, enter sequencer mode.

DESCRIPTION

This ioctl function sets up the TPM550 to work in sequencer mode. The cycle time and the channel configuration are set up.

A pointer to the callers parameter buffer (*TP500_SEQSETBUF*) is passed by the parameter *argp* to the driver.

The *TP500_SEQSETBUF* structure has the following layout:

```
typedef struct
{
    unsigned short    cycleTime;        /* value of cycletime register */
    struct
    {
        unsigned short    flags;
        unsigned short    gain;        /* selected gain */
    } channel[TP500_SINGLCHANS];      /* channel configuration */
} TP500_SEQSETBUF, *PTP500_SEQSETBUF;
```

cycleTime

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[]

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

Parameter is an ORed value of the following described flags.

Name	Meaning
<i>TP500_FL_DIFF</i>	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.
<i>TP500_FL_CORR</i>	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.
<i>TP500_FL_ENABLE</i>	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

Parameter specifies the gain.

Name	TPMC500-10/-12/-20/-22	TPMC500-11/-13/-21/-23
<i>TP500_GAIN1</i>	gain = 1	gain = 1
<i>TP500_GAIN2</i>	gain = 2	gain = 2
<i>TP500_GAIN4</i>	not supported	gain = 4
<i>TP500_GAIN5</i>	gain = 5	not supported
<i>TP500_GAIN8</i>	not supported	gain = 8
<i>TP500_GAIN10</i>	gain = 10	not supported

EXAMPLE

```

int          hCurrent;
int          result;
TP500_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 < TP500_SINGLCHANS; i++)
{
    SeqSetBuf.channel[i].flags = 0; /* disable channel */
}
SeqSetBuf.channel[0].flags = TP500_FL_ENABLE | TP500_FL_CORR;
SeqSetBuf.channel[5].flags = TP500_FL_ENABLE | TP500_FL_DIFF;

SeqSetBuf.channel[0].gain = TP500_GAIN1;
SeqSetBuf.channel[5].gain = TP500_GAIN2;

result = ioctl(hCurrent, TP500_IOCSEQSETUP, &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 <i>argp</i> .
ENOTYPEINIT	Driver specific error (150) – The module type has not been set.

SEE ALSO

ioctl man pages

3.4.4 TP500_IOCSEQREAD

NAME

TP500_IOCSEQREAD reads a set of data values synchronized with cycle time

DESCRIPTION

This ioctl function returns a set of ADC data. The function returns ADC values for the channels that have been enabled with the *TP500_IOCSEQSETUP* function. The specified modes of the *TP500_IOCSEQSETUP* function are used.

A pointer to the callers parameter buffer (*TP500_SEQREADBUF*) is passed by the parameter *argp* to the driver.

The *TP500_SEQREADBUF* structure has the following layout:

```
typedef struct
{
    int      overrunCount;          /* number of lost cycles */
    int      error;                /* error flags */
    short    values[TP500_SINGLCHANS]; /* ADC input value */
} TP500_SEQREADBUF, *PTP500_SEQREADBUF;
```

overrunCount

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

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

Name	Meaning
<i>TP500_FL_HWOVERRUN</i>	The hardware has detected an overflow. The data sequencer has not been serviced in one cycle time.
<i>TP500_FL_TIMERERR</i>	The hardware has signaled that the specified cycle time is too short to make the specified conversions.
<i>TP500_FL_INSTRAMERR</i>	The hardware has detected an error in the instruction RAM (no channel selected).
<i>TP500_FL_SWOVERRUN</i>	The driver can not make the data corrections in one cycle time.

values[]

Array returns a full set of ADC values. Only the values of the channels selected in *TP500_IOCSEQSETUP* 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

```
int          hCurrent;
int          result;
TP500_SEQREADBUF  SeqReadBuf;

. . .

/*****
Read values of the enabled channel 1 and 6
*****/
result = ioctl(hCurrent, TP500_IOCTLGSEQREAD, &SeqReadBuf);
if (result >= 0)
{
    printf("Error %04Xh - Overruns %d\n",
           SeqReadBuf.error,
           SeqReadBuf.overrunCount);
    printf("Channel 1: %d\n", SeqReadBuf.values[0]);
    printf("Channel 1: %d\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 TP500_IOCSEQIMMREAD

NAME

TP500_IOCSEQIMMREAD reads 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 that have been enabled with the *TP500_IOCSEQSETUP* function. The specified modes of the *TP500_IOCSEQSETUP* function are used.

A pointer to the callers parameter buffer (*TP500_SEQREADBUF*) is passed by the parameter *argp* to the driver.

The *TP500_SEQREADBUF* structure has the following layout:

```
typedef struct
{
    int      overrunCount;          /* number of lost cycles */
    int      error;                /* error flags */
    short    values[TP500_SINGLCHANS]; /* ADC input value */
} TP500_SEQREADBUF, *PTP500_SEQREADBUF;
```

overrunCount

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

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

Name	Meaning
<i>TP500_FL_HWOVERRUN</i>	The hardware has detected an overflow. The data sequencer has not been serviced in one cycle time.
<i>TP500_FL_TIMERERR</i>	The hardware has signaled that the specified cycle time is too short to make the specified conversions.
<i>TP500_FL_INSTRAMERR</i>	The hardware has detected an error in the instruction RAM (no channel selected).
<i>TP500_FL_SWOVERRUN</i>	The driver can not make the data corrections in one cycle time.

values[]

Array returns a full set of ADC values. Only the values of the channels selected in *TP500_IOCSEQSETUP* 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

```
int          hCurrent;
int          result;
TP500_SEQREADBUF  SeqReadBuf;

. . .

/*****
Read values of the enabled channel 1 and 6
*****/
result = ioctl(hCurrent, TP500_IOCTL_SEQIMMREAD, &SeqReadBuf);
if (result >= 0)
{
    printf("Error %04Xh - Overruns %d\n",
        SeqReadBuf.error,
        SeqReadBuf.overrunCount);
    printf("Channel 1: %d\n", SeqReadBuf.values[0]);
    printf("Channel 1: %d\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 TP500_IOCSMODTYPE

NAME

TP500_IOCSMODTYPE sets up model type.

DESCRIPTION

This ioctl function sets up the TPMC500 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 (*TP500_TYPEBUF*) is passed by the parameter *argp* to the driver.

The *TP500_TYPEBUF* structure has the following layout:

```
typedef struct
{
    int      ModuleType;    /* TPMC500 variant type */
} TP500_TYPEBUF, *PTP500_TYPEBUF;
```

ModuleType

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

Value	Module Type
10	TPMC500-10
11	TPMC500-11
12	TPMC500-12
13	TPMC500-13
20	TPMC500-20
21	TPMC500-21
22	TPMC500-22
23	TPMC500-23

EXAMPLE

```
int          hCurrent;
int          result;
TP500_TYPEBUF TypeBuf;

. . .

/*****
Setup module type as TPMC500-10
*****/
TypeBuf.ModuleType = 10;    /* TPMC500-10 */

result = ioctl(hCurrent, TP500_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 TPMC500 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 TPMC500 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 [0xe680007f].
```

```
# cat /proc/devices
Character devices:
 1 mem
 2 pty
 3 tty
 4 ttyS
 5 cua
 7 vcs
10 misc
29 fb
128 ptm
136 pts
162 raw
254 tpmc500drv
```

```
# cat /proc/interrupts
          CPU0
 0:         47346          XT-PIC  timer
 1:           413          XT-PIC  keyboard
 2:             0          XT-PIC  cascade
 4:           341          XT-PIC  serial
 8:             2          XT-PIC  rtc
10:            14          XT-PIC  ncr53c8xx
11:           262          XT-PIC  eth0
12:             0          XT-PIC  TPMC500
14:          13582          XT-PIC  ide0
15:             5          XT-PIC  ide1
NMI:             0
ERR:             0
MIS:             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
0170-0177 : ide1
01f0-01f7 : ide0
02f8-02ff : serial(auto)
0376-0376 : ide1
03c0-03df : vga+
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 : TPMC500 (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
00100000-03fffffff : System RAM
    00100000-002327d1 : Kernel code
    002327d2-0031bdcb : Kernel data
e4000000-e4fffffff : 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
```