The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC500-SW-82

Linux Device Driver

Optical Isolated 32 Channel 12 Bit ADC

Version 1.4.x

User Manual

Issue 1.4.0

September 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

TPMC500-SW-82

Linux Device Driver

Optical Isolated 32 Channel 12 Bit ADC

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.

©2001-2010 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	September 26, 2001
1.1	New ioctl() function TP500_IOCSMODTYPE	May 15, 2002
1.2	General Revision	February 27, 2004
1.3.0	Kernel 2.6.x Support	March 15, 2005
1.3.1	File list modified, New Address TEWS LLC, general revision	September 27, 2007
1.4.0	New Flag TP500_FL_RAPID for read(), Address TEWS LLC removed	September 22, 2010

Table of Contents

1 INTRODUCTION 4
2 INSTALLATION 5
2.1 Build and install the device driver 5
2.2 Uninstall the device driver 6
2.3 Install device driver into the running kernel 6
2.4 Remove device driver from the running kernel 6
2.5 Change Major Device Number 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_IOCGREADPARAM 16
3.4.2 TP500_IOCSEQSTOP 18
3.4.3 TP500_IOCSSEQSETUP 19
3.4.4 TP500_IOCGSEQREAD 22
3.4.5 TP500_IOCGSEQIMMREAD 24
3.4.6 TP500_IOCSMODTYPE 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 reads from selected channels
- correction of input values with the factory programmed correction data
- select hardware supported gains

The TPMC500-SW-82 device driver supports the modules listed below:

TPMC500	Optically Isolated 32 Channel 12 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.

TPMC500 Hardware User manual
TPMC500 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC500-SW-82':

TPMC500-SW-82-1.4.0.pdf	This manual in PDF format
TPMC500-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

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

Directory path './tpmc500/':

tpmc500.c	Driver source code
tpmc500def.h	Driver include file
tpmc500.h	Driver include file for application program
Makefile	Device driver make file
makenode	Script for device node creation
include/config.h	Include of system dependent config.h
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/tpmc500exa.c	Example application
example/Makefile	Example application makefile

In order to perform an installation, extract all files of the archive TPMC500-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TPMC500-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tpmc500.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 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 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 TPMC500 module found. The first module of the first TPMC500 module can be accessed with device node /dev/tpmc500_0, the second module with device node /dev/tpmc500_1, the third TPMC500 module with device node /dev/tpmc500_2 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 or sysfs (udev), 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 dynamic device management. The TPCM500 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
```

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 TPMC500 driver has changed and the makenode script is not used.

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/tpmc500_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 *TP500_READBUF* and the size of this structure are passed by the parameters *buffer* and *size* to the device.

```
typedef struct
{
        unsigned short          channel;
        unsigned short          gain;
        unsigned short          flags;
        short                   value;
} TP500_READBUF, *PTP500_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 'tpmc500.h'.

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

flags

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

Name	Meaning
TP500_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.
TP500_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.
TP500_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 *TP500_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.
TP500_FL_RAPID	The driver will wait for settling time and conversion completion in polled mode, interrupts will not be used. This mode is faster than the 'normal' interrupt driven mode, but the CPU will be busy for this time.

value

This parameter returns the converted ADC value. The 12-bit value is always moved to the least significant bits. The returned value is in the range from 0…4095 for unipolar input and -2048…2047 for bipolar input.

EXAMPLE

```
#include <tpmc500.h>

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
{
   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 *TP500_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 TP500_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 *TPMC500*:

Value	Meaning
TP500_IOCGREADPARAM	Get module parameters including module type and the factory programmed correction values.
TP500_IOCSEQSTOP	Stop the sequencer
TP500_IOCSSEQSETUP	Setup and start the sequencer
TP500_IOCGSEQREAD	Read ADC data from sequencer data RAM, synchronized on the sequencer cycle
TP500_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.
TP500_IOCSMODTYPE	Setup model type.

See below for more detailed information on each control code.

Note: 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 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 TPMC500 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.4.1 TP500_IOCGREADPARAM

NAME

TP500_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 (TP500_PARABUF) is passed by the parameter argp to the driver.

```
typedef struct
{
        int                     ModuleType;
        signed short            OffsCorr[4];
        signed short            GainCorr[4];
} TP500_PARABUF, *PTP500_PARABUF;
```

ModuleType

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

OffsCorr

This array returns the factory programmed offset correction data, which 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

This array returns the factory programmed gain correction data, which 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

```
#include <tpmc500.h>

int           hCurrent;
int           result;
TP500_PARABUF ParamBuf;

result = ioctl(hCurrent, TP500_IOCGREADPARAM, &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

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

SEE ALSO

ioctl man pages

3.4.2 TP500_IOCSEQSTOP

NAME

TP500_IOCSEQSTOP – Stop Sequencer Mode

DESCRIPTION

This ioctl function stops the sequencer mode.

EXAMPLE

```
#include <tpmc500.h>

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_IOCSSEQSETUP

NAME

TP500_IOCSSEQSETUP - Setup and start the sequencer, enter sequencer mode

DESCRIPTION

This ioctl function sets up the TPMC500 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.

```
typedef struct
{
        unsigned short          cycleTime;      /* value of cycletime register */
        struct
        {
                unsigned short  flags;
                unsigned short  gain;           /* selected gain */
        }       channel[TP500_SNGLCHANS];       /* channel configuration */
} TP500_SEQSETBUF, *PTP500_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
TP500_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.
TP500_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.
TP500_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	TPMC500-10/-12/-20/-22	TPMC500-11/-13/-21/-23
TP500_GAIN1	gain = 1	gain = 1
TP500_GAIN2	gain = 2	gain = 2
TP500_GAIN4	not supported	gain = 4
TP500_GAIN5	gain = 5	not supported
TP500_GAIN8	not supported	gain = 8
TP500_GAIN10	gain = 10	not supported

EXAMPLE

```
#include <tpmc500.h>

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_SNGLCHANS; 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_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 TP500_IOCGSEQREAD

NAME

TP500_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 *TP500_IOCSSEQSETUP* function. The specified modes of the *TP500_IOCSSEQSETUP* function are used.

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

```
typedef struct
{
        int             overrunCount;
        int             error;
        short           values[TP500_SNGLCHANS];
} TP500_SEQREADBUF, *PTP500_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
TP500_FL_HWOVERRUN	The hardware has detected an overflow; the data sequencer has not been serviced in one cycle time.
TP500_FL_TIMERERR	The hardware has signaled that the specified cycle time is too short to make the specified conversions.
TP500_FL_INSTRAMERR	The hardware has detected an error in the instruction RAM. (No channel selected)
TP500_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 *TP500_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.

The returned values are in the range from 0…4095 for unipolar input and -2048…2047 for bipolar input.

EXAMPLE

```
#include <tpmc500.h>

int                hCurrent;
int                result;
TP500_SEQREADBUF   SeqReadBuf;

/*****************************************
 read values of the enabled channel 1 and 6
*****************************************/
result = ioctl(hCurrent, TP500_IOCGSEQREAD, &SeqReadBuf);
if (result >= 0)
{
  printf("Error %04Xh - Overruns %d\n",
    SeqReadBuf.error,
    SeqReadBuf.overrunCount);
  printf("Channel 1: %d\n", SeqReadBuf.values[0]);
  printf("Channel 6: %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_IOCGSEQIMMREAD

NAME

TP500_IOCGSEQIMMREAD – Read a set of data values 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 *TP500_IOCSSEQSETUP* function. The specified modes of the *TP500_IOCSSEQSETUP* function are used.

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

```
typedef struct
{
        int             overrunCount;
        int             error;
        short           values[TP500_SNGLCHANS];
} TP500_SEQREADBUF, *PTP500_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
TP500_FL_HWOVERRUN	The hardware has detected an overflow; the data sequencer has not been serviced in one cycle time.
TP500_FL_TIMERERR	The hardware has signaled that the specified cycle time is too short to make the specified conversions.
TP500_FL_INSTRAMERR	The hardware has detected an error in the instruction RAM. (No channel selected)
TP500_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 *TP500_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.

The returned values are in the range from 0…4095 for unipolar input and -2048…2047 for bipolar input.

EXAMPLE

```
#include <tpmc500.h>

int                hCurrent;
int                result;
TP500_SEQREADBUF   SeqReadBuf;

/*****************************************
 read values of the enabled channel 1 and 6
*****************************************/
result = ioctl(hCurrent, TP500_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 TP500_IOCSMODTYPE

NAME

TP500_IOCSMODTYPE - Setup 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.

```
typedef struct
{
    int                 ModuleType;
} TP500_TYPEBUF, *PTP500_TYPEBUF;
```

ModuleType

This 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

```
#include <tpmc500.h>

int             hCurrent;
int             result;
TP500_TYPEBUF   TypeBuf;

…
```

…

```
/*************************************************************
 Setup model 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 displays information of a correct running TPMC500 driver (see also the proc man pages).

```
# lspci -v
…
02:07.0 Signal processing controller: PLX Technology, Inc. PCI <-> IOBus
Bridge (rev 01)
        Subsystem: TEWS Datentechnik GmBH: Unknown device 01f4
        Flags: medium devsel, IRQ 16
        Memory at ff5fec00 (32-bit, non-prefetchable)
        I/O ports at a880 [size=128]
        I/O ports at a400 [size=256]
        Memory at ff5fe000 (32-bit, non-prefetchable) [size=2K]
…

# cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  6 lp
  7 vcs
 10 misc
 13 input
 21 sg
 29 fb
128 ptm
136 pts
180 usb
189 usb_device
226 drm
253 tpmc500drv

Block devices:
  1 ramdisk
…
```

```
# cat /proc/interrupts
           CPU0       CPU1
  0:         85          0   IO-APIC-edge      timer
  1:          0         10   IO-APIC-edge      i8042
  6:          0          5   IO-APIC-edge      floppy
  7:          0          0   IO-APIC-edge      parport0
  8:          0          1   IO-APIC-edge      rtc
  9:          0          0   IO-APIC-fasteoi   acpi
 12:          0        128   IO-APIC-edge      i8042
 14:       7730      70590   IO-APIC-edge      ide0
 16:          0          0   IO-APIC-fasteoi   sata_sil, ehci_hcd:usb1,
ohci_hcd:usb4, ohci_hcd:usb5, TPMC500
 17:     579558     296665   IO-APIC-fasteoi   radeon@PCI:1:0:0
 18:          0          0   IO-APIC-fasteoi   uhci_hcd:usb2
 19:      24613         47   IO-APIC-fasteoi   uhci_hcd:usb3, eth0
NMI:          0          0
LOC:    2571037    2571039
ERR:          0
MIS:          0

# cat /proc/ioports
…
1010-1015 : ACPI CPU throttle
1020-1023 : ACPI GPE0_BLK
10b0-10b7 : ACPI GPE1_BLK
7000-9fff : PCI Bus #01
  9000-90ff : 0000:01:00.0
a000-bfff : PCI Bus #02
  a080-a0bf : 0000:02:03.0
    a080-a0bf : e1000
  a400-a4ff : 0000:02:07.0
    a400-a4ff : TPMC500
  a800-a83f : 0000:02:06.0
    a800-a83f : e100
  a880-a8ff : 0000:02:07.0
  ac00-ac7f : 0000:02:0a.0
  b000-b01f : 0000:02:0b.0
    b000-b01f : uhci_hcd
  b080-b09f : 0000:02:0b.1
…
```

```
#cat /proc/iomem
…
  ff5c0000-ff5dffff : 0000:02:06.0
    ff5c0000-ff5dffff : e100
  ff5e0000-ff5effff : 0000:02:06.0
  ff5fb000-ff5fbfff : 0000:02:00.0
    ff5fb000-ff5fbfff : ohci_hcd
  ff5fc000-ff5fcfff : 0000:02:00.1
    ff5fc000-ff5fcfff : ohci_hcd
  ff5fd000-ff5fdfff : 0000:02:06.0
    ff5fd000-ff5fdfff : e100
  ff5fe000-ff5fe7ff : 0000:02:07.0
  ff5fec00-ff5fec7f : 0000:02:07.0
  ff5ff000-ff5ff7ff : 0000:02:0a.0
  ff5ff800-ff5ff8ff : 0000:02:0b.2
    ff5ff800-ff5ff8ff : ehci_hcd
  ff5ffc00-ff5fffff : 0000:02:05.0
    ff5ffc00-ff5fffff : sata_sil
ff780000-ffffffff : reserved
```