The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC500-SW-82

Linux Device Driver

Optical Isolated 32 Channel 12 Bit ADC

Version 2.0.x

User Manual

Issue 2.0.1

May 2019

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

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
2.0.0	General revision. API documentation added.	July 21, 2014
2.0.1	General revision, COPYING-File added to file-list	May 23,2019

Table of Contents

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 software 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

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC500-SW-82':

TPMC500-SW-82-2.0.1.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
COPYING	Copy of the GNU Public License (GPL)
api/tpmc500api.h	API include file
api/tpmc500api.c	API source file
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 and api/tpmc500api.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 -a**

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 API Documentation

3.1 General Functions

3.1.1 tpmc500Open

NAME

tpmc500Open – Opens a Device

SYNOPSIS

```
TPMC500_HANDLE tpmc500Open
(
      char   *DeviceName
)
```

DESCRIPTION

Before I/O can be performed to a device, a file descriptor must be opened by a call to this function.

PARAMETERS

DeviceName

This parameter points to a null-terminated string that specifies the name of the device. The first TPMC500 device is named “/dev/tpmc500_0” the second device is named “/dev/tpmc500_1” and so on.

EXAMPLE

```
#include “tpmc500api.h”

TPMC500_HANDLE hdl;

/*
** open file descriptor to device
*/
hdl = tpmc500Open(“/dev/tpmc500_0”);
if (hdl == NULL)
{
   /* handle open error */
}
```

RETURNS

A device handle, or NULL if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno.*

The error code is a standard error code set by the I/O system.

3.1.2 tpmc500Close

NAME

tpmc500Close – Closes a Device

SYNOPSIS

```
TPMC500_STATUS tpmc500Close
(
      TPMC500_HANDLE              hdl
)
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tpmc500api.h”

TPMC500_HANDLE hdl;
TPMC500_STATUS result;

/*
** close file descriptor to device
*/
result = tpmc500Close( hdl );

if (result != TPMC500_OK)
{
  /* handle close error */
}
```

RETURNS

On success, TPMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC500_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.3 tpmc500SetModelType

NAME

tpmc500SetModelType – Set the module type of the TPMC500

SYNOPSIS

```
TPMC500_STATUS tpmc500SetModelType
(
    TPMC500_HANDLE      hdl,
    int                 ModuleType
)
```

DESCRIPTION

This function configures the model type of the TPMC500.

This function must be called before the first AD conversion can be started.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

ModuleType

This argument specifies the model type of the TPMC500. The following model types are supported.

Value	Description
TPMC500_TYPE_10	TPMC500-10 (Gain 1/2/5/10, +/-10V, Front I/O)
TPMC500_TYPE_11	TPMC500-11 (Gain 1/2/4/8, +/-10V, Front I/O)
TPMC500_TYPE_12	TPMC500-12 (Gain 1/2/5/10, 0-10V, Front I/O)
TPMC500_TYPE_13	TPMC500-13 (Gain 1/2/4/8, 0-10V, Front I/O)
TPMC500_TYPE_20	TPMC500-20 (Gain 1/2/5/10, +/-10V, Back I/O)
TPMC500_TYPE_21	TPMC500-21 (Gain 1/2/4/8, +/-10V, Back I/O)
TPMC500_TYPE_22	TPMC500-22 (Gain 1/2/5/10, 0-10V, Back I/O)
TPMC500_TYPE_23	TPMC500-23 (Gain 1/2/4/8, 0-10V, Back I/O)

EXAMPLE

```
#include “tpmc500api.h”

TPMC500_HANDLEhdl;
TPMC500_STATUSresult;

result = tpmc500SetModelType(hdl, TPMC500_TYPE_11);

if (result != TPMC500_OK)
{
  /* handle error */
}
```

RETURNS

On success, TPMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC500_ERR_INVALID_HANDLE	The specified TPMC500_HANDLE is invalid.
TPMC500_ERR_RANGE	Invalid channel number.

3.1.4 tpmc500GetModuleInfo

NAME

tpmc500GetModuleInfo – Get module information data

SYNOPSIS

```
TPMC500_STATUS tpmc500GetModuleInfo
(
    TPMC500_HANDLE          hdl,
    TPMC500_INFO_BUFFER     *pModuleInfo
)
```

DESCRIPTION

This function reads module information data such as configured module type, location on the PCI bus and factory programmed correction data.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pModuleInfo

This argument specifies a pointer to the module information buffer.

```
typedef struct
{
   unsigned int     Variant;
   unsigned int     PciBusNo;
   unsigned int     PciDevNo;
   unsigned int     ADCOffsetCal[4];
   unsigned int     ADCGainCal[4];
} TPMC500_INFO_BUFFER;
```

Variant

This parameter returns the configured module variant (e.g. 10 for a TPMC500-10).

PciBusNo, PciDevNo

These parameters specify the PCI location of this module

ADCOffsetCal[4]

This array returns the factory programmed offset correction values for the different gain settings. Array index 0 contains the value for gain 1, index 1 contains the value for gain 2 and so forth.

ADCGainCal[4]

This array returns the factory programmed gain correction for the different gain settings. Array index 0 contains the value for gain 1, index 1 contains the value for gain 2 and so forth.

EXAMPLE

```
#include “tpmc500api.h”

TPMC500_HANDLE       hdl;
TPMC500_STATUS       result;
TPMC500_INFO_BUFFER  ModuleInfo

result = tpmc500GetModuleInfo(hdl, &ModuleInfo);

if (result != TPMC500_OK)
{
   /* handle error */
}
```

RETURNS

On success, TPMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC500_ERR_INVALID_HANDLE	The specified TPMC500_HANDLE is invalid.

3.2 Device Access Functions

3.2.1 tpmc500Read

NAME

tpmc500Read – Read converted AD value

SYNOPSIS

```
TPMC500_STATUS tpmc500Read
(
    TPMC500_HANDLE      hdl,
    int                 channel,
    int                 gain,
    int                 flags,
    int                 *pAdcVal
)
```

DESCRIPTION

This function starts an AD conversion on the specified channel and returns the converted value.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

channel

This argument specifies the input channel number. Valid channels for single-ended mode are 1…32, for differential mode 1...16.

gain

This argument specifies the gain for this channel. Valid gains are 1, 2, 5, 10 for *TPMC500-10/-12/-20/-22* and 1, 2, 4, 8 for *TPMC500-11/-13/-21/-23.*

flags

Set of bit flags that control the AD conversion. The following flags could be OR'ed:

Flag	Meaning
TPMC500_DIFF	If this flag is set the ADC input works in differential mode otherwise in single-ended (default).
TPMC500_CORR	Perform an offset and gain correction with factory calibration data stored in the TPMC500 EEPROM.
TPMC500_FAST	If this flag is set the fast (polled) mode will be used. The driver will not use interrupts, instead it will wait in a busy loop until the settling time (if necessary) and the conversion is finished. Conversions using this mode will be handled faster, but the processor executes a busy loop and other tasks will not be handled during the loops.

pAdcVal

This argument points to an integer variable where the AD value will be returned. 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 "tpmc500api.h"

TPMC500_HANDLE  hdl;
TPMC500_STATUS  result;
int             AdcData;
int             channel, gain, flags;

channel  = 32;
gain     = 2;
flags    = TPMC500_CORR | TPMC500_FAST;

result = tpmc500Read(hdl, channel, gain, flags, &AdcData);

if (result != TPMC500_OK)
{
  /* handle error */
}
```

RETURNS

On success, TPMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC500_ERR_ACCESS	The module type has not been configured.
TPMC500_ERR_INVALID_HANDLE	The specified TPMC500_HANDLE is invalid.
TPMC500_ERR_INVAL	At least one of the parameters is invalid.
TPMC500_ERR_TIMEOUT	ADC conversion timed out.
TPMC500_ERR_RANGE	Invalid channel number.
TPMC500_ERR_BUSY	This error occurs if the sequencer is still running. Please stop the sequencer before executing this function.

3.2.2 tpmc500StartSequencer

NAME

tpmc500StartSequencer – Start sequencer operation

SYNOPSIS

```
TPMC500_STATUS tpmc500StartSequencer
(
    TPMC500_HANDLE      hdl,
    unsigned int        CycleTime,
    unsigned int        NumOfBufferPages,
    unsigned int        NumOfChannels,
    TPMC500_CHAN_CONF   *ChanConf
)
```

DESCRIPTION

This function sets up and starts the sequencer. The setup specifies the channels to be used in sequencer mode and how they will be setup, defining gain, correction and input interface. Additional the sequencer cycle time is defined and depth of the drivers sequencer FIFO will be configured.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

CycleTime

This argument specifies the repeat frequency of the sequencer in 100 µs steps. Each time the sequencer timer reaches the programmed cycle time a new AD conversion of all active channels is started. Valid values are in the range from 100 microseconds to 6.5535 seconds.

NumOfBufferPages

This argument specifies the number of sample blocks in the ring buffer. A sample block contains the samples of all channels (NumOfChannels) per sequencer cycle.

NumOfChannels

This argument specifies the number of active channels for this job. The maximum number is 32.

ChanConf

This array of channel configuration structures specifies the configuration of the active channels. The channel configuration defines the channel number, the gain and some flags. The ordering of channels in a ring buffer page is the same as defined in this array.

```
typedef struct
{
        unsigned int        ChanToUse;
        unsigned int        gain;
        unsigned int        flags;
} TPMC500_CHAN_CONF;
```

ChanToUse

This parameter specifies the input channel number. Valid channels for single-ended mode are 1…32, for differential mode 1...16.

gain

This Parameter specifies the gain for this channel. Valid gains are 1, 2, 5, 10 for *TPMC500-10/-12/-20/-22* and 1, 2, 4, 8 for *TPMC500-11/-13/-21/-23.*

flags

Set of bit flags that control the AD conversion. The following flags could be OR'ed:

Flag	Meaning
TPMC500_DIFF	If this flag is set the ADC input works in differential mode otherwise in single-ended (default).
TPMC500_CORR	Perform an offset and gain correction with factory calibration data stored in the TPMC500 EEPROM.

EXAMPLE

```
#include "tpmc500api.h"

TPMC500_HANDLE     hdl;
TPMC500_STATUS     result;
unsigned int       CycleTime;
unsigned int       NumOfBufferPages;
unsigned int       NumOfChannels;
TPMC500_CHAN_CONF  ChanConf[TPMC500_MAX_CHAN];

CycleTime          = 5000;
NumOfBufferPages   = 100;
NumOfChannels      = 2;

ChanConf[0].ChanToUse= 1;
ChanConf[0].gain     = 1;
ChanConf[0].flags    = TPMC500_CORR;

ChanConf[1].ChanToUse= 20;
ChanConf[1].gain     = 5;
ChanConf[1].flags    = TPMC500_CORR;
…
```

```
// start the sequencer
result = tpmc500StartSequencer(hdl, CycleTime, NumOfBufferPages,
                 NumOfChannels, ChanConf);

if (result != TPMC500_OK)
{
  /* handle error */
}
```

RETURNS

On success, TPMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC500_ERR_ACCESS	The module type has not been configured.
TPMC500_ERR_INVALID_HANDLE	The specified TPMC500_HANDLE is invalid.
TPMC500_ERR_INVAL	At least one of the parameters is invalid.
TPMC500_ERR_RANGE	Invalid channel number.
TPMC500_ERR_BUSY	This error occurs if the sequencer is still running. Please stop the sequencer before executing this function.

3.2.3 tpmc500ReadSequencer

NAME

tpmc500ReadSequencer – Read next data block of sequencer samples

SYNOPSIS

```
TPMC500_STATUS tpmc500ReadSequencer
(
    TPMC500_HANDLE      hdl,
    int                 *pData,
    unsigned int        *pStatus
)
```

DESCRIPTION

This function returns the next available sequencer data block within the ring buffer. If no data block is available the function returns TPMC500_ERR_NODATA. In this case it must be called again until new data is available.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pData

This argument points to an array of integer items where the converted data of all configured channels of a sequencer cycle is returned. The number of channels and the channel configuration was setup using the tpmc500StartSequencer function. The first array item [0] belongs to the channel configured by ChanConfig[0], the second array item [1] belongs to the channel configured by ChanConfig[1] and so forth. Please refer to the example application for details.

pStatus

This argument is a pointer to a variable which returns the actual sequencer status. Keep in mind to check this status before each reading. If status is 0 no error is pending. A set of bits specifies the error condition.

Value	Description
TPMC500_BUF_OVERRUN	This bit indicates a ring buffer overrun. The error occurred if there is no space in ring buffer to write the new AD data. In this case the new AD values are discarded. The sequencer was not stopped.
TPMC500_DATA_OVERFLOW	This indicates an overrun in the sequencer data RAM. The error occurred if the driver is too slow to read the data in time. The sequencer was stopped after this error occurred.
TPMC500_TIMER_ERR	Sequencer timer error (see also TPMC500 hardware manual). The sequencer was stopped after this error occurred.
TPMC500_INST_RAM_ERR	Sequencer instruction RAM error (see also TPMC500 hardware manual). The sequencer was stopped after this error occurred.
TPMC500_SEQ_STOPPED	The Sequencer is not running.

EXAMPLE

```
#include “tpmc500api.h”

TPMC500_HANDLE   hdl;
TPMC500_STATUS   result;
unsigned int     seqStatus;
int              Data[32];

result = tpmc500ReadSequencer(hdl, Data, &seqStatus);

if (result != TPMC500_OK)
{
  if (result == TPMC500_ERR_NODATA)
  {
    /* try again reading data */
  }
  else
  {
    /* handle error */
  }
}
```

RETURNS

On success, TPMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC500_ERR_INVALID_HANDLE	The specified TPMC500_HANDLE is invalid.
TPMC500_ERR_NODATA	No new data available in the ring buffer

3.2.4 tpmc500ReadSequencerTimeout

NAME

tpmc500ReadSequencerTimeout – Wait for and read next data block of sequencer samples

SYNOPSIS

```
TPMC500_STATUS tpmc500ReadSequencerTimeout
(
    TPMC500_HANDLE      hdl,
    int                 *pData,
    unsigned int        *pStatus,
    int                 Timeout_ms
)
```

DESCRIPTION

This function returns the next available sequencer data block within the ring buffer. If no data block is available, the function waits for the next sequencer cycle, or until the specified timeout occurs.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pData

This argument points to an array of integer items where the converted data of all configured channels of a sequencer cycle is returned. The number of channels and the channel configuration was setup using the tpmc500StartSequencer function. The first array item [0] belongs to the channel configured by ChanConfig[0], the second array item [1] belongs to the channel configured by ChanConfig[1] and so forth. Please refer to the example application for details.

pStatus

This argument is a pointer to a variable which returns the actual sequencer status. Keep in mind to check this status before each reading. If status is 0 no error is pending. A set of bits specifies the error condition.

Value	Description
TPMC500_BUF_OVERRUN	This bit indicates a ring buffer overrun. The error occurred if there is no space in ring buffer to write the new AD data. In this case the new AD values are discarded. The sequencer was not stopped.
TPMC500_DATA_OVERFLOW	This indicates an overrun in the sequencer data RAM. The error occurred if the driver is too slow to read the data in time. The sequencer was stopped after this error occurred.
TPMC500_TIMER_ERR	Sequencer timer error (see also TPMC500 hardware manual). The sequencer was stopped after this error occurred.
TPMC500_INST_RAM_ERR	Sequencer instruction RAM error (see also TPMC500 hardware manual). The sequencer was stopped after this error occurred.
TPMC500_SEQ_STOPPED	The Sequencer is not running.

Timeout_ms

This argument specifies the timeout in milliseconds. The resulting timeout granularity depends on the system.

EXAMPLE

```
#include “tpmc500api.h”

TPMC500_HANDLE   hdl;
TPMC500_STATUS   result;
unsigned int     seqStatus;
int              Data[32];

/* wait up to 1 second for sequencer data */
result = tpmc500ReadSequencerTimeout(hdl, Data, &seqStatus, 1000);

if (result != TPMC500_OK)
{
   /* handle error */
}
```

RETURNS

On success, TPMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC500_ERR_INVALID_HANDLE	The specified TPMC500_HANDLE is invalid.
TPMC500_ERR_NODATA	No new data available in the ring buffer

3.2.5 tpmc500StopSequencer

NAME

tpmc500StopSequencer – Stop the sequencer

SYNOPSIS

```
TPMC500_STATUS tpmc500StopSequencer
(
      TPMC500_HANDLE          hdl
);
```

DESCRIPTION

This function stops execution of the sequencer mode on the specified device.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tpmc500api.h”

TPMC500_HANDLE   hdl;
TPMC500_STATUS   result;

result = tpmc500StopSequencer(hdl);

if (result != TPMC500_OK)
{
   /* handle error */
}
```

RETURNS

On success, TPMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC500_ERR_INVALID_HANDLE	The specified TPMC500_HANDLE is invalid.

4 Diagnostic

If the TPMC500 does not work properly it is helpful to get some status information from the driver respective kernel. To get debug output from the driver enable the following symbols in '*tpmc500.c*' by replacing "#undef" with "#define":

```
#define DEBUG_TPMC500
```

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 TPMC500 driver (see also the proc man pages).

```
# tail –f /var/log/messages /* before modprobing the TPMC500 driver */
Jul 21 12:14:59 linux kernel: TEWS TECHNOLOGIES - TPMC500 32 Channel 12 Bit ADC
version 2.0.x (<Release Date>)

Jul 21 12:14:59 linux kernel: TPMC500:  Installing device (vendor=0x10B5,
device=0x9050) at 4:2.0

…

# lspci -v
…
04:02.0 Signal processing controller: PLX Technology, Inc. PCI <-> IOBus Bridge
(rev 01)
    Subsystem: TEWS Technologies GmbH Device 01f4
    Flags: medium devsel, IRQ 17
    Memory at feb9fc00 (32-bit, non-prefetchable) [size=128]
    I/O ports at e000 [size=256]
    I/O ports at e400 [size=256]
    Memory at feb9f000 (32-bit, non-prefetchable) [size=2K]
    Kernel driver in use: TEWS TECHNOLOGIES - TPMC500 32 Channel 12 Bit ADC
…

# cat /proc/devices
Character devices:
  1 mem
…
226 drm
253 tpmc500drv

Block devices:
  1 ramdisk
…
```

```
# cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  0:         42          0          0          0   IO-APIC-edge      timer
  1:          4          3          1          2   IO-APIC-edge      i8042
  6:          1          1          1          0   IO-APIC-edge      floppy
  7:          1          0          0          0   IO-APIC-edge      parport0
  8:          0          0          1          0   IO-APIC-edge      rtc0
  9:          0          0          0          0   IO-APIC-fasteoi   acpi
 12:         39         42         44         38   IO-APIC-edge      i8042
 14:       1890         31         33       2735   IO-APIC-edge      ata_piix
 15:          0          0          0          0   IO-APIC-edge      ata_piix
 16:          0          0          0          0   IO-APIC-fasteoi   uhci_hcd:usb5
 17:         13          9          0         97   IO-APIC-fasteoi   TPMC500
 18:          0          0          0          0   IO-APIC-fasteoi   uhci_hcd:usb4
…

# cat /proc/ioports
…
  e000-efff : PCI Bus 0000:04
    e000-e0ff : 0000:04:02.0
    e400-e4ff : 0000:04:02.0
      e400-e4ff : TPMC500
    ec00-ec3f : 0000:04:00.0
      ec00-ec3f : e1000
  ffa0-ffaf : 0000:00:1f.1
    ffa0-ffaf : ata_piix
…
```