The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC501-SW-72

LynxOS Device Driver

Optically Isolated 32 Channel 16 Bit ADC

Version 1.0.x

User Manual

Issue 1.0.0

October 2009

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-72

LynxOS Device Driver

Optically Isolated 32 Channel 16-Bit ADC

Supported Modules:
TPMC501

Issue	Description	Date
1.0.0	First Issue	October 20, 2009

Table of Contents

1 Introduction

The TPMC501-SW-72 LynxOS device driver allows the operation of the TPMC501 Digital Input PMC on LynxOS platforms with DRM based PCI interface.

The standard file (I/O) functions (open, close, ioctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and configuration operations.

The TPMC501-SW-72 device driver supports the following features:

- reading ADC data from a specified input channel
- selection of input gain
- support of differential and single-ended input interface
- support of pipeline mode
- input data correction with factory calibration data
- configuration, start and stop of the sequencer
- read of sequencer data sets

The TPMC501-SW-72 device driver supports the modules listed below:

TPMC501-10	32 Channel 16-bit ADC (Front I/O) (Input Gain: 1,2,5,10) (±10V for gain = 1)	(PMC)
TPMC501-11	32 Channel 16-bit ADC (Front I/O) (Input Gain: 1,2,4,8) (±10V for gain = 1)	(PMC)
TPMC501-12	32 Channel 16-bit ADC (Front I/O) (Input Gain: 1,2,5,10) (0V to 10V for gain = 1)	(PMC)
TPMC501-13	32 Channel 16-bit ADC (Front I/O) (Input Gain: 1,2,4,8) (0V to 10V for gain = 1)	(PMC)
TPMC501-20	32 Channel 16-bit ADC (Back I/O) (Input Gain: 1,2,5,10) (±10V for gain = 1)	(PMC)
TPMC501-21	32 Channel 16-bit ADC (Back I/O) (Input Gain: 1,2,4,8) (±10V for gain = 1)	(PMC)
TPMC501-22	32 Channel 16-bit ADC (Back I/O) (Input Gain: 1,2,5,10) (0V to 10V for gain = 1)	(PMC)
TPMC501-23	32 Channel 16-bit ADC (Back I/O) (Input Gain: 1,2,4,8) (0V to 10V for gain = 1)	(PMC)

To get more information about the features and use of TPMC501 devices it is recommended to read the manuals listed below.

TPMC501 User Manual

TPMC501 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC501-SW-72':

TPMC501-SW-72-SRC.tar.gz	GZIP compressed archive with driver source code
TPMC501-SW-72-1.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

For installation the files have to be copied to the desired target directory.

The GZIP compressed archive TPMC501-SW-72-SRC.tar.gz contains the following files and directories:

Directory path 'tpmc501':

tpmc501.c	TPMC501 device driver source
tpmc501def.h	TPMC501 driver include file
tpmc501.h	TPMC501 include file for driver and application
tpmc501_info.c	TPMC501 Device information definition
tpmc501_info.h	TPMC501 Device information definition header
tpmc501.cfg	TPMC501 Driver configuration file include
tpmc501.import	Linker import file
Makefile	Device driver make file
example/tpmc501exa.c	Example application
example/Makefile	Example application makefile

In order to perform a driver installation, first extract the TAR file to a temporary directory, than follow the steps below:

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/tpmc501 or /sys/drivers.cpci_x86/tpmc501

2. Copy the following files to this directory:
   - tpmc501.c
   - tpmc501def.h
   - tpmc501.import
   - Makefile

3. Copy tpmc501.h to /usr/include/

4. Copy tpmc501_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tpmc501_info.h to /sys/dheaders/

Copy tpmc501.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform. For example: /sys/cfg.ppc or /sys/cfg.x86 ....

2.1 Device Driver Installation

The two methods of driver installation are as follows:

(1) Static Installation
(2) Dynamic Installation (only native LynxOS 4 systems)

2.1.1 Static Installation

With this method, the driver object code is linked with the kernel routines and is installed during system start-up.

2.1.1.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tpmc501, where xxx represents the BSP that supports the target hardware.
2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

2.1.1.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).
2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = … tpmc501_info.x
   ```

   And at the end of the Makefile

   ```
   tpmc501_info.o:$(DHEADERS)/tpmc501_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

2.1.1.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os/ respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry at the end of this file.

   ```
   I:tpmc501.cfg
   ```

2.1.1.4 Rebuild the Kernel

1. Change to the directory /sys/lynx.os/ (/sys/bsp.xxx)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

   The N flag instructs init to run mknod and create all the nodes mentioned in the new nodetab.

4. After reboot you should find the following new devices (depends on the device configuration): /dev/tpmc501a, /dev/tpmc501b, …

2.1.2 Dynamic Installation

This method allows you to install the driver after the operating system is booted. The driver object code is attached to the end of the kernel image and the operating system dynamically adds this driver to its internal structures. The driver can also be removed dynamically.

2.1.2.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tpmc501, where xxx represents the BSP that supports the target hardware.
2. To make the dynamic link-able driver enter:

   ```
   make dldd
   ```

2.1.2.2 Create Device Information Declaration

1. Change to the directory /sys/drivers.xxx/tpmc501, where xxx represents the BSP that supports the target hardware.
2. To create a device definition file for the major device (this works only on native systems)

   ```
   make t501info
   ```

3. To install the driver enter:

   ```
   drinstall –c tpmc501.obj
   ```

   If successful, drinstall returns a unique <driver-ID>

4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> t501info
   ```

   The <driver-ID> is returned by the drinstall command

5. To create the node for the devices enter:

   ```
   mknod /dev/tpmc501a c <major_no> 0
   ```

   The <major_no> is returned by the devinstall command.

If all steps are successfully completed, the TPMC501 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TPMC501 device enter the following commands:

```
devinstall –u –c <device-ID>
drinstall –u <driver-ID>
```

2.1.3 Device Information Definition File

The device information definition contains information necessary to install the TPMC501 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tpmc501_info.h*.

This structure contains the following parameter:

PCIBusNumber Contains the PCI bus number at which the supported device is connected. Valid bus numbers are in range from 0 to 255.

PCIDeviceNumber Contains the device number (slot) at which the supported device is connected. Valid device numbers are in range from 0 to 31.

If both PCIBusNumber and PCIDeviceNumber are –1 then the driver will auto scan for supported devices. The first device found in the scan order will be allocated by the driver for this major device.

Already allocated devices can't be allocated twice. This is important to know if there are more than one TPMC501 major devices.

boardVersion Specifies the used board type. The value depends on the name of the device. Specify 10 for TPMC501-10(R), 11 for TPMC501-11(R), 20 for TPMC501-20, and so on.

A device information definition is unique for every TPMC501 major device. The file *tpmc501_info.c* on the distribution media contains two device information declarations, **tpmc501A** for the first major device and **tpmc501B** for the second major device.

If the driver should support more than two major devices it is necessary to copy and paste an existing declaration and rename it with a unique name, for example **tpmc501C**, **tpmc501D** and so on.

It is also necessary to modify the device and driver configuration file, respectively the configuration include file *tpmc501.cfg*.

The following device declaration information uses the auto find method to detect a supported device on the PCI bus.

```
TDRV501_INFO tpmc501A = {

     -1,            /*  Auto find the device on any PCI bus    */
     -1,
     10,            /* TPMC501-10 */
};
```

2.1.4 Configuration File: CONFIG.TBL

The device and driver configuration file CONFIG.TBL contains entries for device drivers and its major and minor device declarations. Each time the system is rebuild, the config utility read this file and produces a new set of driver and device configuration tables and a corresponding nodetab.

To install the TPMC501 driver and devices into the LynxOS system, the configuration include file tpmc501.cfg must be included in the CONFIG.TBL (see also chapter 2.1.1.3).

The file tpmc501.cfg on the distribution disk contains the driver entry (*C:tpmc501:\....*) and two major device entries (*D:TPMC501 1:tpmc501A:: and D:TPMC501 2:tpmc501B::*).

If the driver should support more than one major device, the following entries for major devices must be enabled by removing the comment character (#). By copy and paste an existing major and minor entries and renaming the new entries, it is possible to add any number of additional TPMC501 devices.

This example shows a driver entry with two major devices and one minor device:

```
#    Format:
#    C:driver-name:open:close:read:write:select:control:install:uninstall
#    D:device-name:info-block-name:raw-partner-name
#    N:node-name:minor-dev

C:tpmc501:tpmc501open:tpmc501close: \
::: \
::tpmc501ioctl: \
:tpmc501install:tpmc501uninstall
D:TPMC501 1:tpmc501A::
N:tpmc501a:0
D:TPMC501 2:tpmc501B::
N:tpmc501b:0
```

The configuration above creates the following nodes in the /dev directory.

```
/dev/tpmc501a /dev/tpmc501b
```

3 TPMC501 Device Driver Programming

LynxOS system calls are all available directly to any C program. They are implemented as ordinary function calls to "glue" routines in the system library, which trap to the OS code.

Note that many system calls use data structures, which should be obtained in a program from appropriate header files. Necessary header files are listed with the system call synopsis.

3.1 open()

NAME

open() - open a file

SYNOPSIS

#include <sys/file.h>
#include <sys/types.h>
#include <fcntl.h>

int open (char *path, int oflags[, mode_t mode])

DESCRIPTION

Opens a file (TPMC501 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TPMC501 device ***oflags*** must be set to **O_RDWR** to open the file for both reading and writing.

The ***mode*** argument is required only when a file is created. Because a TPMC501 device already exists this argument is ignored.

EXAMPLE

```
int fd

fd = open ("/dev/tpmc501a", O_RDWR);
if (fd == -1)
{
     /* Handle error */
}
```

RETURNS

open returns a file descriptor number if successful, or –1 on error.

SEE ALSO

LynxOS System Call - open()

3.2 close()

NAME

close() – close a file

SYNOPSIS

int close(int fd)

DESCRIPTION

This function closes an opened device.

EXAMPLE

```
int  result;

result = close(fd);
if (result == -1)
{
     /* Handle error */
}
```

RETURNS

close returns 0 (OK) if successful, or –1 on error

SEE ALSO

LynxOS System Call - close()

3.3 ioctl()

NAME

ioctl() – I/O device control

SYNOPSIS

```
#include <ioctl.h>
#include <tpmc501.h>

int ioctl (int fd, int request, char *arg)
```

DESCRIPTION

ioctl provides a way of sending special commands to a device driver. The call sends the value of request and the pointer arg to the device associated with the descriptor fd.

The following ioctl codes are supported by the driver and are defined in *tpmc501.h*:

Symbol	Meaning
TPMC501_READ	Read state of the digital input lines
TPMC501_SEQSETUP	Setup sequencer configuration and start sequencer mode
TPMC501_SEQSTOP	Stop sequencer mode
TPMC501_SEQREAD	Read sequencer input data
TPMC501_GETINFO	Get board specific information

See behind for more detailed information on each control code.

RETURNS

ioctl returns 0 if successful, or –1 on error.

On error, *errno* will contain a standard error code (see also LynxOS System Call – ioctl).

SEE ALSO

LynxOS System Call - ioctl().

3.3.1 TPMC501_READ

NAME

TPMC501_READ – Executes an AD conversion and returns the current ADC input value

DESCRIPTION

This function starts an AD conversion on a specified channel and returns the input value. A pointer to the callers read buffer (*TPMC501_READ_BUFFER*) must be passed by the parameter *arg* to the device.

The function automatically decides if settling time is necessary or not. It will always execute as fast as possible.

```
typedef struct
{
        int             channel;
        int             gain;
        int             diffMode;
        int             pipeMode;
        int             corrMode;
        int             value;
} TPMC501_READ_BUFFER;
```

Members

mode

This argument specifies the ADC channel to use. Allowed values are 1 up to 32 for single-ended channels and 1 up to 16 for differential channels.

gain

This argument specifies the input gain that shall be used. Allowed values are 1, 2, 5 and 10 or 1, 2, 4 and 8 depending on the type of the used device.

diffMode

This argument specifies the input interface to be used. Symbols for the input interface mode are defined in tpmc501.h:

Define	Description
TPMC501_DIFF	This value specifies if differential input interface shall be used. (Only valid for channel 1 … 16)
TPMC501_SNGL	The value specifies that single-ended input interface shall be used.

pipeMode

This argument specifies if pipeline mode or the standard mode shall be used. Symbols for 'ON' and 'OFF' are defined in tpmc501.h:

Define	Description
TPMC501_OFF	If this value is specified, the ADC will be used in normal mode. The returned value will be the value of the current conversion.
TPMC501_ON	If this value is specified, the ADC will be used in pipeline mode. The returned value will be the value of the previous conversion.

corrMode

This argument specifies if input value correction will be used. Symbols for 'ON' and 'OFF' are defined in tpmc501.h:

Define	Description
TPMC501_OFF	This value specifies that the ADC raw value shall be returned.
TPMC501_ON	This value specifies that the ADC raw value shall be corrected with the factory stored correction data and the corrected value shall be returned.

value

This parameter is used to return the ADC input value. Returned values are between 0 and 65535 for unipolar module versions and between -32768 and 32767 for bipolar module versions.

EXAMPLE

```
#include <tpmc501.h>

int                     fd;
int                     result;
TPMC501_READ_BUFFER     rdBuf;

/* --- read current input value --- */
rdBuf.channel      = 4;               /* input channel 4 */
rdBuf.gain         = 2;               /* input gain 2 */
rdBuf.diffMode     = TPMC501_SNGL;    /* single-ended input */
rdBuf.pipeMode     = TPMC501_OFF;     /* no pipelining */
rdBuf.corrMode     = TPMC501_ON;      /* input value corrrection on */

…
```

…

```
result = ioctl(fd, TPMC501_READ, (char*)&rdBuf);
if (result >= 0)
{
     printf("ADC-INPUT: %d\n", rdBuf.value);
}
else
{
     /* Read failed */
}
```

ERRORS

EINTR	The function was cancelled.
ETIMEDOUT	The maximum allowed time to finish the ADC conversion failed. (no interrupts or HW-problem)
EINVAL	An unsupported input parameter has been specified. Check input parameters
EBUSY	The sequencer mode is active for the device and single channel conversions are not allowed.

Other returned error codes are system error conditions.

3.3.2 TPMC501_SEQSETUP

NAME

TPMC501_SEQSETUP – Configure sequencer mode and start execution

DESCRIPTION

This function configures channels and timing for sequencer mode and starts the execution. A pointer to the callers configuration buffer (*TPMC501_SEQSETUP_BUFFER*) must be passed by the parameter *arg* to the device.

```
typedef struct
{
        int             cycleTime;
        struct
        {
                int     enable;
                int     gain;
                int     diffMode;
                int     corrMode;
        }               chanCfg[MAX_NUM_CHANS];
} TPMC501_SEQSETUP_BUFFER, *PTPMC501_SEQSETUP_BUFFER;
```

Members

cycleTime

This argument specifies the cycle time for sequencer mode. The time is specified in steps of 100µs. Allowed values are 1 up to 32 for single-ended channels and 1 up to 16 for differential channels. A value of 0 specifies that the sequencer will work in continuous mode.

chanCfg[]

This argument is an array containing the channel configurations. The array element with index 0 specifies the configuration of channel 1, index 1 specifies the configuration of channel 2, and so on.

chanCfg[].enable

This array element specifies if the assigned channel shall be enabled for sequencer mode. Symbols for ‘ON’ and ‘OFF’ are defined in tpmc501.h:

Define	Description
TPMC501_OFF	If this value is specified, the channel will not be used.
TPMC501_ON	If this value is specified, the channel will be enabled in sequencer mode and converted value will be updated with every sequencer cycle.

chanCfg[].diffMode

This array element specifies the input interface to be used for the assigned channel. Symbols for the input interface mode are defined in tpmc501.h:

Define	Description
TPMC501_DIFF	This value specifies that the differential input interface shall be used for that channel. (Only valid for channel 1 … 16)
TPMC501_SNGL	The value specifies that the single-ended input interface shall be used.

chanCfg[].corrMode

This array element specifies if input value correction will be used for the assigned channel. Symbols for 'ON' and 'OFF' are defined in tpmc501.h:

Define	Description
TPMC501_OFF	This value specifies that values for the channel will be returned as a raw value.
TPMC501_ON	This value specifies that values for the channel will be returned as a corrected value.

EXAMPLE

```
#include <tpmc501.h>

int                     fd;
int                     chanIdx;
int                     result;
TPMC501_SEQSETUP_BUFFER seqBuf;

/* --- initialize structure --- */
for (chanIdx = 0; chanIdx < MAX_NUM_CHANS; chanIdx++)
{
     seqBuf.chanCfg[].enable = TPMC501_OFF;     /* disable channel */
}

/* --- configure and start sequencer --- */
seqBuf.cycleTime        = 5000;            /* cycle time: 0.5 sec */

/* Channel 1 */
seqBuf.chanCfg[0].enable     = TPMC501_ON;      /* enable channel */
seqBuf.chanCfg[0].gain       = 2;               /* input gain 2 */
seqBuf.chanCfg[0].diffMode   = TPMC501_SNGL;    /* single-ended input */
seqBuf.chanCfg[0].corrMode   = TPMC501_ON;      /* corrrection on */

…
```

…

```
/* Channel 10 */
seqBuf.chanCfg[9].enable     = TPMC501_ON;      /* enable channel */
seqBuf.chanCfg[9].gain       = 1;               /* input gain 1 */
seqBuf.chanCfg[9].diffMode   = TPMC501_DIFF;    /* differential input */
seqBuf.chanCfg[9].corrMode   = TPMC501_OFF;     /* corrrection off */

/* Channel 17 */
seqBuf.chanCfg[16].enable    = TPMC501_ON;      /* enable channel */
seqBuf.chanCfg[16].gain      = 2;               /* input gain 2 */
seqBuf.chanCfg[16].diffMode  = TPMC501_SNGL;    /* single-ended input */
seqBuf.chanCfg[16].corrMode  = TPMC501_OFF;     /* corrrection off */

result = ioctl(fd, TPMC501_SEQSETUP, (char*)&seqBuf);
if (result >= 0)
{
     /* Sequencer mode successfully started */
}
else
{
     /* Sequencer mode start failed */
}
```

ERRORS

EINVAL	An unsupported input parameter has been specified. Check input parameters
EBUSY	The sequencer mode is already active for the device. The sequencer must be stopped.

Other returned error codes are system error conditions.

3.3.3 TPMC501_SEQSTOP

NAME

TPMC501_SEQSTOP – Stop sequencer mode

DESCRIPTION

This function stops the sequencer mode. The function dependant parameter *arg* can be set to NULL.

EXAMPLE

```
#include <tpmc501.h>

int                     fd;
int                     result;

/* --- stop sequencer --- */
result = ioctl(fd, TPMC501_SEQSTOP, NULL);
if (result >= 0)
{
     /* Sequencer successfully stopped */
}
else
{
     /* Sequencer stop failed */
}
```

3.3.4 TPMC501_SEQREAD

NAME

TPMC501_SEQREAD – Read sequencer data

DESCRIPTION

This function reads a set of sequencer data if the sequencer is started. A pointer to the callers read buffer (*TPMC501_SEQREAD_BUFFER*) must be passed by the parameter *arg* to the device.

```
typedef struct
{
        int             waitMode;
        int             value[MAX_NUM_CHANS];
        unsigned int    status;
} TPMC501_SEQREAD_BUFFER, *PTPMC501_SEQREAD_BUFFER;
```

Members

waitMode

This argument specifies if the function will return immediately, returning the last converted values, or if function shall wait for completion of the current sequencer cycle and return the new values. Symbols for 'ON' and 'OFF' are defined in tpmc501.h:

Define	Description
TPMC501_OFF	The function returns immediately using the values of the last sequencer cycle.
TPMC501_ON	The function waits for completion of the current sequencer cycle and will return 'fresh' data.

value[]

This argument is an array returning the channels data. Only array elements of enabled channels will return values, the values of disabled channels will be undefined. The array element with index 0 specifies the configuration of channel 1, index 1 specifies the configuration of channel 2, and so on.

status

This parameter returns the status of the sequencer and the status of the data set. Symbols for the returned flags are defined in tpmc501.h, The status is a value of OR'ed flags:

Flag	Description
TPMC501_FL_HWOVERRUN	This TPMC501 signals a hardware overrun error. A set of data has not been handled before the next set is available. The TPMC501 will stop the sequencer and software should execute *TPMC501_SEQSTOP* to reset this error.
TPMC501_FL_TIMERERR	This TPMC501 signals and timer error. This error signals that the needed conversion time for enabled channels exceeds the time specified as sequencer cycle time. The TPMC501 will stop the sequencer and software should execute *TPMC501_SEQSTOP* to reset this error.
TPMC501_FL_INSTRAMERR	This TPMC501 signals an error in sequencer configuration. For example there is no enabled channel in sequencer mode. The TPMC501 will stop the sequencer and software should execute *TPMC501_SEQSTOP* to reset this error.
TPMC501_FL_SWOVERRUN	This flag signals, that at least one set of data has not been read by the application and the old data is overwritten. The sequencer continues execution.
TPMC501_FL_NOVALIDDATA	This flag signals, that there has been no sequencer cycle completed since the last read or the start of the sequencer. If the flag is set after a sequencer start before a cycle completed the returned data values are undefined. If the flag is set after a successful read, the data values of the last cycle will be returned again.

EXAMPLE

```
#include <tpmc501.h>

int                     fd;
int                     chanIdx;
int                     result;
TPMC501_SEQREAD_BUFFER  seqRdBuf;

/* --- configure and start sequencer --- */
seqRdBuf.waitMode  = TPMC501_ON;           /* wait for new data */

…
```

…

```
result = ioctl(fd, TPMC501_SEQREAD, (char*)&seqRdBuf);
if (result >= 0)
{
     /* Sequencer read successfully*/
     for (chanIdx = 0; chanIdx < MAX_NUM_CHANS; chanIdx++)
     {
          printf("ADC-INPUT [%d]: %d\n", chanIdx + 1, seqRdBuf.value);
     }
}
else
{
     /* Sequencer read failed */
}
```

ERRORS

EINTR	The function was cancelled.
ETIMEDOUT	The wait time has exceeded the maximum time of a sequencer cycle. (no interrupts or HW-problem)
EINVAL	An unsupported input parameter has been specified. Check input parameters.
EBUSY	The sequencer mode has not been started for the device. The sequencer must be started first.

Other returned error codes are system error conditions.

4 Debugging and Diagnostic

If the driver will not work properly, please enable debug outputs by defining the symbols *DEBUG, DEBUG_TPMC,* and *DEBUG_PCI* in file tpmc501.c.

The debug output should appear on the console. If not, please check the symbol *KKPF_PORT* in *uparam.h*. This symbol should be configured to a valid COM port (e.g. *SKDB_COM1*).

The debug output displays the device information data for the current major device like this.

```
TPMC501: Device Driver Install
Bus = 0  Dev = 16  Func = 0
[00] = 905010B5
[04] = 02800000
[08] = 11800001
[0C] = 00000000
[10] = 80003000
[14] = 00802001
[18] = 00803001
[1C] = 80004000
[20] = 00000000
[24] = 00000000
[28] = 00000000
[2C] = 01F51498
[30] = 00000000
[34] = 00000000
[38] = 00000000
[3C] = 00000109

PCI Base Address 0 (PCI_RESID_BAR0)

70603000 : 01 FF FF 0F 00 F8 FF 0F 00 00 00 00 00 00 00 00
70603010 : 00 00 00 00 01 00 00 00 01 08 00 00 00 00 00 00
70603020 : 00 00 00 00 00 00 00 00 42 A9 52 A9 C0 79 33 E9
70603030 : 00 00 00 00 00 00 00 00 00 00 00 00 41 00 00 00

PCI Base Address 1 (PCI_RESID_BAR1)

PCI Base Address 2 (PCI_RESID_BAR2)

70203000 : 0000 0000 0000 0000 0000 0000 0000 0000
70203010 : 0000 0000 0000 0000 0000 0000 0000 0000
70203020 : 0000 0000 0000 0000 0000 0000 0000 0000
70203030 : 0000 0000 0000 0000 0000 0000 0000 0000
70203040 : 0000 0000 0000 0000 0000 0000 0000 0000
70203050 : 0000 0000 0000 0000 0000 0000 0000 0000
70203060 : 0000 0000 0000 0000 0000 0000 0000 0000
```

```
70203070 : 0000 0000 0000 0000 0000 0000 0000 0000
70203080 : 0000 0000 0000 0000 0000 0000 0000 0000
70203090 : 0000 0000 0000 0000 0000 0000 0000 0000
702030A0 : 0000 0000 0000 0000 0000 0000 0000 0000
702030B0 : 0000 0000 0000 0000 0000 0000 0000 0000
702030C0 : 0000 0000 0000 0000 0000 0000 0000 0000
702030D0 : 0000 0000 0000 0000 0000 0000 0000 0000
702030E0 : 0000 0000 0000 0000 0000 0000 0000 0000
702030F0 : 0000 0000 0000 0000 0000 0000 0000 0000

PCI Base Address 3 (PCI_RESID_BAR3)

70604000 : 00 37 00 F8 00 39 00 F1 00 3E 00 EB 00 46 00 F0
TPMC501: Found TPMC501-xx on Bus 0, Device 16
    Correction data: Offset/Gain
                         248/55
                         241/57
                         235/62
                         240/70
```

The debug output above is only an example. Debug output on other systems may be different for addresses and data in some locations.