The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC501-SW-42

VxWorks Device Driver

32 Channel 16-bit ADC PMC

Version 2.0.x

User Manual

Issue 2.0.1

January 2008

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
25469 Halstenbek, Germany
www.tews.com

Phone: +49 (0) 4101 4058 0
Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com

TEWS TECHNOLOGIES LLC
9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com

Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

TPMC501-SW-42

VxWorks Device Driver

32 Channel 16-bit ADC PMC

Supported Modules:
TPMC501

Issue	Description	Date
1.0	First Issue	April 15, 1999
1.1	New PCI Configuration	July 16, 1999
1.2	Support for x86 target	June 19, 2000
1.3	General Revision	November 24, 2003
1.3.1	Release.txt added, Issue layout changed	March 8, 2005
2.0.0	New driver startup functions, ChangeLog.txt added to file list, description for tpmc501PciInit changes	January 22, 2007
2.0.1	Function read(): description of parameter *maxbytes* changed	January 11, 2008

Table of Contents

1 Introduction

The TPMC501-SW-42 VxWorks device driver software allows the operation of the supported PMC conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open*(), *close(), read(),* and *ioctl()* functions.

This driver invokes a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

The TPMC501-SW-42 device driver supports the following features:

- start AD conversion and read data
- choosing gain, channel, input interface
- correction of input data with board-specific calibration data
- support of ADC sequencer mode

The TPMC501-SW-42 supports the modules listed below:

TPMC501	32(16) Channel - 16-bit ADC	(PMC)

In this document all supported modules and devices will be called TPMC501. Specials for a certain devices will be advised.

To get more information about the features and use of supported 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-42':

tpmc501drv.c	TPMC501 device driver source
tpmc501def.h	TPMC501 driver include file
tpmc501.h	TPMC501 include file for driver and application
tpmc501pci.c	TPMC501 PCI MMU mapping for Intel x86 based targets
tpmc501exa.c	Example application
include/tdhal.h	Hardware dependent interface functions and definitions
TPMC501-SW-42-2.0.1.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

2.1 Include device driver in Tornado IDE project

For including the TPMC501-SW-42 device driver into a Tornado IDE project follow the steps below:

(1) Copy the files from the distribution media into a subdirectory in your project path.
(For example: ./TPMC501)

(2) Add the device drivers C-files to your project.
Make a right click to your project in the 'Workspace' window and use the 'Add Files ...' topic.
A file select box appears, and the driver files can be selected.

(3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your Tornado User's Guide.

2.2 Special installation for Intel x86 based targets

The TPMC501 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro **CPU_FAMILY**. If the content of this macro is equal to *I80X86* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, the required device memory spaces can't be accessed.

To solve this problem a MMU mapping entry has to be added for the required TPMC501 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tpmc501pci.c** contains the function *tpmc501PciInit().* This routine finds out all TPMC501 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*).

The right place to call the function *tpmc501PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window).

Be sure that the function is called prior to MMU initialization otherwise the TPMC501 PCI spaces remains unmapped and an access fault occurs during driver initialization.

Please insert the following call at a suitable place in **sysLib.c**:

```
    tpmc501PciInit();
```

Modifying the sysLib.c file will change the sysLib.c in the BSP path. Remember this for future projects and recompilations.

2.3 BSP dependent adjustments

The driver includes a file called *include/tdhal.h* which contains functions and definitions for BSP adaptation. It may be necessary to modify them for BSP specific settings. Most settings can be made automatically by switches set by in the BSP header files, but some settings must be set manually.

There are 3 offset definitions (*USERDEFINED_MEM_OFFSET*, *USERDEFINED_IO_OFFSET*, and *USERDEFINED_LEV2VEC*) that must be set if a matching warning appears while compilation.

An other define allows a simple adaptation for BSPs that supports a *pciIntConnect()* function to connect shared (PCI) interrupts. If this function is defined in the used BSP, the definition of *USERDEFINED_SEL_PCIINTCONNECT* should be enabled.

Please refer to the BSP to get information about the interuupt connect function and the offset values.

2.4 System resource requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 1 KB
Stack	< 1 KB	---
Semaphores	---	3

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

3 I/O system functions

This chapter describes the driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

3.1 tpmc501Drv()

NAME

tpmc501Drv() - installs the TPMC501 driver in the I/O system

SYNOPSIS

#include “tpmc501.h”

STATUS tpmc501Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus, installs the TPMC501 driver in the I/O system.

A call to this function is the first thing the user has to do before adding any device to the system or performing any I/O request.

EXAMPLE

```
#include  "tpmc501.h”

…

STATUS             result;
…

/*-------------------
  Initialize Driver
  -------------------*/
result = tpmc501Drv();
if (result == ERROR)
{
     /* Error handling */
}

…
```

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
ENXIO	No TPMC501 module found

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tpmc501DevCreate()

NAME

tpmc501DevCreate() – Add a TPMC501 device to the VxWorks system

SYNOPSIS

```
#include "tpmc501.h"

STATUS tpmc501DevCreate
(
      char        *name,
      int         devIdx,
      int         funcType,
      void        *pParam
)
```

DESCRIPTION

This routine adds the selected device to the VxWorks system. The device hardware will be setup and prepared for use.

This function must be called before performing any I/O request to this device.

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

devIdx

This index number specifies the device to add to the system. The device numbers will be assigned in the order the VxWorks *pciFindDevice()* function will find the devices. A 0 selects the first device, a 1 the second, and so on.

funcType

This parameter is unused and should be set to *0*.

pParam

This parameter points to a structure (*TPMC501_CONF_BUFFER*) containing the default configuration of the device.

The structure (*TPMC501_CONF_BUFFER*) has the following layout and is defined in tpmc501.h:

```
typedef struct
{
        int     modelType;
} TPMC501CONF_BUFFER;
```

modelType

Specifies the model type of the selected device. The following model types are supported:

Model Type	**Module Name**	**Gains**	**Voltage Range**
10	TPMC501-10	1, 2, 5, 10	+/-10V
11	TPMC501-11	1, 2, 4, 8	+/-10V
12	TPMC501-12	1, 2, 5, 10	0..+10V
13	TPMC501-13	1, 2, 4, 8	0..+10V
20	TPMC501-20	1, 2, 5, 10	+/-10V
21	TPMC501-21	1, 2, 4, 8	+/-10V
22	TPMC501-22	1, 2, 5, 10	0..+10V
23	TPMC501-23	1, 2, 4, 8	0..+10V

EXAMPLE

```
#include  "tpmc501.h”

…

STATUS                  result;
TPMC501_CONF_BUFFER     tpmc501Conf;

…
```

```
…

/*------------------------------------------------------
  Create the device "/tpmc501/0" for the first ADC device
      Use TPMC501-11
  ------------------------------------------------------*/
tpmc501Conf.moduleType =    11;

result = tpmc501DevCreate(  "/tpmc501/0",
                            0,
                            0,
                            (void*)&tpmc501Conf);
if (result == OK)
{
    /* Device successfully created */
}
else
{
    /* Error occurred when creating the device */
}

…
```

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	**Description**
S_ioLib_NO_DRIVER	The driver has not been installed (call tpmc501Drv())
ENXIO	Specified device not found
EBUSY	The specified device has already been created.
ENOTSUP	The specified model type is not supported

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.3 tpmc501PciInit()

NAME

tpmc501PciInit() – Generic PCI device initialization

SYNOPSIS

void tpmc501PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC501 PCI spaces (base address register) and to enable the TPMC501 device for access.

The global variable *tpmc501Status* obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

Value	Meaning
> 0	Initialization successful completed. The value of tpmc501Status is equal to the number of mapped PCI spaces
0	No TPMC501 device found
< 0	Initialization failed. The value of (tpmc501Status & 0xFF) is equal to the number of mapped spaces until the error occurs. Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[]. Remedy: Add dummy entries as necessary (syslib.c).

EXAMPLE

```
extern void tpmc501PciInit();

…

tpmc501PciInit();

…
```

4 I/O Functions

4.1 open()

NAME

open() - open a device or file.

SYNOPSIS

```
int open
(
        const char   *name,
        int          flags,
        int          mode
)
```

DESCRIPTION

Before I/O can be performed to the TPMC501 device, a file descriptor must be opened by invoking the basic I/O function *open()*.

PARAMETER

name

Specifies the device which shall be opened, the name specified in tpmc501DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int       fd;

…

/*------------------------------------------
  Open the device named "/tpmc501/0" for I/O
  ------------------------------------------*/
fd = open("/tpmc501/0", 0, 0);
if (fd == ERROR)
{
    /* Handle error */
}

…
```

RETURNS

A device descriptor number or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - *open()*

4.2 close()

NAME

close() – close a device or file

SYNOPSIS

```
STATUS close
(
        int             fd
)
```

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

This file descriptor specifies the device to be closed. The file descriptor has been returned by the *open()* function.

EXAMPLE

```
int        fd;
STATUS     retval;

…

/*----------------
  close the device
  ----------------*/
retval = close(fd);
if (retval == ERROR)
{
     /* Handle error */
}

…
```

RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - close()

4.3 read()

NAME

read() – read a value from the specified TPMC501 device.

SYNOPSIS

```
int read
(
        int         fd,
        char        *buffer,
        size_t      maxbytes
)
```

DESCRIPTION

This function starts a conversion for one input channel and returns the value.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

buffer

This argument points to a user supplied special I/O buffer. (*TP501_IO_BUFFER*)

```
typedef struct
{
        int               Channel;
        int               Gain;
        unsigned long     flags;
        long              value;
} TP501_IO_BUFFER;
```

Channel

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

Gain

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

flags

This parameter specifies conversion flags setting the input mode. The value is an ORed value of the following flags:

TP501_DIFFMODE	If set differential input interface is used, if not set the input interface will be single-ended.
TP501_CORRENA	If this flag is set input data corrections is enabled and ADC data will be corrected with factory set correction data individual for every TPMC501. If the flag is not set the ADC value will be returned without correction.

value

This value returns the ADC input value. The possible range of the value depends on the device. If it is a bipolar version, input value is between -32768 (-10V) and 32767 (~+10V). unipolar versions return a value between 0 (0V) an 65635 (~+10V).

maxbytes

This parameter must be set to the buffer size in bytes.

EXAMPLE

```
#include “tpmc501.h”

int                  fd;
TP501_IO_BUFFER      buf;
int                  retval;

…

/*---------------------------------------------------
  Read the actual value of differential channel 1,
  the gain shall be 2 and the value shall be corrected
  ---------------------------------------------------*/
buf.Channel    = 1;
buf.Gain       = 2;
buf.flags      = TP501_CORRENA | TP501_DIFFMODE;
retval = read(fd, (char*)&buf, sizeof(TP501_IO_BUFFER));
if (retval != ERROR)
{
     printf(“ADC value: %ld”, buf.value);
}
else
{
     /* handle the read error */
}

…
```

RETURNS

Number of bytes read or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set error code described below.

Error code	**Description**
S_tp501Drv_ICHAN	Illegal channel number specified
S_tp501Drv_IGAIN	Illegal gain specified
S_tp501Drv_MODBUSY	The device is in use or the sequencer is active

SEE ALSO

ioLib, basic I/O routine - read()

4.4 ioctl()

NAME

ioctl() - performs an I/O control function.

SYNOPSIS

```
#include “tpmc501.h”

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

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls (read, write) will be performed by calling the ioctl() function.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

request

This argument specifies the function that shall be executed. Following functions are defined:

Function	**Description**
FIOSTARTSEQ	Set up and start sequencer mode
FIOSTOPSEQ	Stop sequencer mode

arg

This parameter depends on the selected function (request). How to use this parameter is described below with the function.

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual). Function specific error codes will be described with the function.

SEE ALSO

ioLib, basic I/O routine - ioctl()

4.4.1 FIOSTARTSEQ

This I/O control function sets up the sequencer channels and cycle time and afterwards starts the sequencer. The function specific control parameter **arg** is a pointer on a TP501_IOC_BUF structure that will be used while the sequencer is active.

```
typedef struct
{
        unsigned short          cycletime;
        unsigned long           act_channels;
        TP501_IO_BUFFER         *chan_setup;
        unsigned long           buf_size;
        unsigned long           buf_stat;
        unsigned long           putIdx;
        unsigned long           getIdx;
        long                    *buffer;
} TP501_IOC_BUF
```

cycletime

This argument specifies the length of one sequencer cycle. This value is specified in 100µs steps. Allowed values are between 1 and 65535. A specified value of 0 enable the continuous mode.

act_channels

This value specifies the number of active channels.

chan_setup

This parameter points to an array of data structures specifying the channel setup. The used data structure is the same used by the read command (for more information refer to the description of the read())

buf_size

This value specifies the size of the input FIFO. The value specifies the number of sequences that can be handled without reading before the buffer is filled.

buf_stat

This value specifies the current state and errors will be shown in this argument. The state is an ORed value of the following flags:

TP501_SEQ_BUF_OVERRUN	The user supplied FIFO is full and new data can not be stored
TP501_SEQ_DATA_OVERFLOW	Old data not read by the software when new values are ready
TP501_SEQ_TIMER_ERR	The specified cycle time is not long enough to convert the specified channels
TP501_SEQ_INST_RAM_ERR	The sequencer is started, but no channel has been selected.

putIdx

This parameter holds the current put index, which specifies the position in the FIFO where the next data will be written to. This index should just be read for information but never be changed by the application.

getIdx

This parameter holds the current get index, which specifies the position, where the application shall read the next value from FIFO. This index must be modified by the application after reading a value from the FIFO. This index is not changed by the driver.

buffer

This parameter points to the user supplied memory area where the sequencer input data will be stored in.

EXAMPLE

```
#include “tpmc501.h”

#define   SBUF_SIZE     0x200

int                 fd;
int                 retval;
TP501_IOC_BUF       seq_buf;
TP501_IO_BUFFER     seq_rw_par[2];
long                seq_data_buf[SBUF_SIZE];

…

/*------------------------------------------------
  Start sequencer using channel 1 and 3
    Channel 1:
        differential, data correction on, gain = 1
    Channel 3:
        single-ended, data correction off, gain = 5
  ------------------------------------------------*/
seq_buf.cycletime       = 10000;          /* Cycletime 1s */
seq_buf.buffer          = seq_data_buf;
seq_buf.act_channels    = 2;
seq_buf.buf_size        = SBUF_SIZE / seq_buf.act_channels;
seq_buf.chan_setup      = seq_rw_par;

…
```

```
…

seq_rw_par[0].Channel   = 1;
seq_rw_par[0].Gain      = 1;
seq_rw_par[0].flags     = TP501_CORRENA | TP501_DIFFMODE;
seq_rw_par[1].Channel   = 3;
seq_rw_par[1].Gain      = 5;
seq_rw_par[1].Mode      = TP501_CORRDIS | TP501_SNGLMODE;

retval = ioctl(fd, FIOSTARTSEQ, (int)&seq_buf);
if (retval != ERROR)
{
     /* sequencer started */
}
else
{
     /* handle the error */
}

…
```

ERROR CODES

Error code	Description
S_tp501Drv_MODBUSY	The device is currently in use

4.4.2 FIOSTOPSEQ

This I/O control function stops the sequencer. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tpmc501.h”

int                 fd;
int                 retval;

…

/*------------------------
  Execute ioctl() function
  ------------------------*/
retval = ioctl(fd, FIOSTOPSEQ, 0);
if (retval != ERROR)
{
     /* Sequencer is stopped */
}
else
{
     /* handle the error */
}

…
```

ERROR CODES

Error code	Description
S_tp501Drv_MODBUSY	The device is currently in use

5 Appendix

5.1 Additional Error Codes

Error code	Error value	Description
S_tp501Drv_ICHAN	0x05010001	Illegal channel number specified
S_tp501Drv_IGAIN	0x05010002	Illegal gain specified
S_tp501Drv_MODBUSY	0x05010003	The module is busy
S_tp501Drv_TIMEOUT	0x05010004	The conversion timed out (HW error?)
S_tp501Drv_ICMD	0x05010005	Illegal I/O command specified