The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC501-SW-42

VxWorks Device Driver

32 Channel 16 Bit ADC

Version 1.3.x

User Manual

Issue 1.3.1

March 2005

powerBridge Computer

Ehlbeek 15a
30938 Burgwedel
fon 05139-9980-0
fax 05139-9980-49

www.powerbridge.de
info@powerbridge.de

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

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

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

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

TPMC501-SW-42

32 Channel 16 Bit ADC

VxWorks Device Driver

Issue	Description	Date
1.0	First Issue	April1999
1.1	New PCI Configuration	July 1999
1.2	Support for x86 target	June 2000
1.3	General Revision	November 2003
1.3.1	Release.txt added, Issue layout changed	March 8, 2005

Table of Contents

1 Introduction

The TPMC501-SW-42 VxWorks device driver software allows the operation of the TPMC501 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.

The TPMC501 driver includes following functions:

- read actual input value
- start and setup the input sequencer
- stop the input sequencer

2 Installation

The software is delivered on a 3½" HD diskette.

Following files are located on the diskette:

tp501drv.c	TPMC501 Driver Source
tp501exa.c	TPMC501 Device Driver Example
tpmc501.h	TPMC501 Driver Include File
Makefile	Makefile for TPMC501 Device Driver and Example
tp501pci.c	TPMC501 PCI MMU mapping for Intel x86 based targets
tpxxxhwdep.c	Collection of hardware dependent functions
tpxxxhwdep.h	include for hardware dependent functions
Release.txt	Information about the Device Driver Release

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

2.1 Install the driver to VxWorks system

To install the TPMC501 device driver to the VxWorks system following steps have to be done:

- Build the object code of the TPMC501 device driver
- Link or load the driver object file to the VxWorks system
- Call the *tp501Drv()* function to install the device driver.

2.2 Hardware dependent Configuration

The device driver supports the on board PMC slots of the Motorola MVME2600 by default.

The device driver software supports also TEWS PMC carrier boards and others. The system has to be setup to guarantee the following points:

- full access to the PMC I/O area of the card (register address space)
- full access to PMC memory area of the card (correction data address space)
- interrupt must be connected

2.3 Including the driver in VxWorks

How to include the device drive in the VxWorks system is described in the VxWorks and Tornado manuals.

2.4 Example application

The example application uses the MVME2600/3600 BSP. If an older version of the BSP (1.1/0 up to 1.1/2) is used, the value *_OLD_BSP_* in *TP501TST.C* must be defined. If this value is undefined the newer BSP will be used.

Using a Motorola PMC-span, the PCI/PCI Bridge has to be initialized on the span.

Using other carriers the initialization matching has to be adapted to the BSP.

The example code holds two functions setting and reading the PCI configuration registers. The first function (*PCIsetupTPMC501()*) sets up the PCI configuration registers, the TPMC501 registers will appear at the specified address.

The second function (*searchPCI()*) will search for the TPMC501 and read and calculate the modules registers and correction data address, which must be used, when installing the driver.

2.5 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***. If the contents of this macro are equal to *I80386*, *I80386* or *PENTIUM* 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 PCI 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.

Please examine the BSP documentation or contact the BSP Vendor whether the BSP perform automatic PCI and MMU configuration or not. If the PCI and MMU initialization is done by the BSP the function tp501PciInit() won't be included and the user can skip to the following steps.

The C source file **tp501pci.c** contains the function *tp501PciInit().* 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()*).

If the Tornado 2.0 project facility is used, the right place to call the function *tp501PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (can be opened from the project *Files* window).

If Tornado 1.0.1 compatibility tools are used insert the call to tp501PciInit() at the beginning of the root task (*usrRoot()*) in **usrConfig.c**.

Be sure that the function is called prior to MMU initialization otherwise the TPCM501 PCI spaces remains unmapped and an access fault occurs during driver initialization.

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

```
tp501PciInit();
```

To link the driver object modules to VxWorks, simply add all necessary driver files to the project. If Tornado 1.0.1 *Standard BSP Builds...* is used add the object modules to the macro *MACH_EXTRA* inside the BSP Makefile (MACH_EXTRA = tp501drv.o tp501pci.o ...).

The Function tp501PciInit() was designed for and tested on generic Pentium targets. If another BSP is used please refer to BSP documentation or contact the technical support for required adaptation.

If strange errors occur after system startup with the new build system please carry out a VxWorks *build clean* and *build all*.

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 tp501Drv()

NAME

tp501Drv() - installs the TPMC501 driver in the I/O system and initializes the driver.

SYNOPSIS

STATUS tp501Drv(void)

DESCRIPTION

This function installs the TPMC501 driver in the I/O system, allocates driver resources and initializes them.

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

RETURNS

OK or ERROR (if the driver cannot be installed)

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tp501DevCreate()

NAME

tp501DevCreate() - adds a TPMC501 device to the system and initializes device hardware.

SYNOPSIS

```
STATUS tp501DevCreate
    (
    char            *name,      /* name of the device to create                */
    unsigned long   RegAddr,    /* physical device register address            */
    unsigned long   CalAddr,    /* physical device calibration data address    */
    unsigned long   vector,     /* interrupt vector                            */
    unsigned long   level,      /* interrupt level                             */
    unsigned long   type        /* device type [TPMC501_nn]                    */
    )
```

DESCRIPTION

This routine is called to add a device to the system that will be serviced by the TPMC501 driver. This function must be called before performing any I/O request to this driver.

PARAMETER

The name of the device is selected by the string, which is deployed by this routine in the parameter **name**.

The argument **RegAddr** specifies the address of the modules registers (see TPMC501-DOC User Manual and PCI Configuration example).

The argument **CalAddr** specifies the address of the modules correction data memory (see TPMC501-DOC User Manual and PCI Configuration example).

The argument **vector** and **level** are board dependent. They specify the interrupt vector and the interrupt level.

The argument **type** specifies the module type mounted to the address. There are predefined symbols for this argument in *tpmc501.h*. Allowed values are *TPMC501_10*, *TPMC501_11*, *TPMC501_12*, *TPMC501_13*, *TPMC501_20*, *TPMC501_21*, *TPMC501_22* and *TPMC501_23*.

EXAMPLE

```
#include  "tpmc501.h"

...

/*-------------------------------------------------------
  Create the device "/tpmc501" with the registers at
  address 81000000h and the calibration data at C1000000h
  The mounted module is a TPMC501-10
  -------------------------------------------------------*/
status = tp501DevCreate ("/tpmc501",
                         0x81000000,
                         0xC1000000,
                         0xF,
                         0xF,
                         TPMC501_10);

...
```

RETURNS

OK or ERROR (if the driver is not installed or the device already exists or any other error occurred during the creation)

4 I/O interface functions

This chapter describes the interface to the basic I/O system.

4.1 open()

NAME

open() - opens a device or file.

SYNOPSIS

```
int open
      (
      const char  *name,          /* name of the device to open                */
      int         flags,          /* not used for TPMC501 driver, must be 0    */
      int         mode            /* not used for TPMC501 driver, must be 0    */
      )
```

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

The parameter **name** selects the device which shall be opened.

The parameters **flags** and **mode** are not used and must be 0.

EXAMPLE

```
...

/*------------------------------------------------------------
  Open the device named "/tpmc501" for I/O
  ------------------------------------------------------------*/
fd = open("/tpmc501", 0, 0);

...
```

RETURNS

A device descriptor number or ERROR (if the device does not exist or no device descriptors are available)

INCLUDES

ioLib.h, semLib.h

SEE ALSO

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

4.2 close()

NAME

close() - closes a device or file.

SYNOPSIS

```
int close
        (
        int     fd,                             /* descriptor to close                  */
        )
```

DESCRIPTION

This function closes opened devices.

EXAMPLE

```
int  retval;

...

/*------------------
  Close the device
  ------------------*/
retval = close(fd);

...
```

RETURNS

A device descriptor number or ERROR (if the device does not exist or no device descriptors are available)

INCLUDES

ioLib.h, semLib.h

SEE ALSO

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

4.3 read()

NAME

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

SYNOPSIS

```
int read
        (
        int         fd,         /* device descriptor from opened TPMC501 device   */
        char        *buffer,    /* pointer to an I/O buffer                       */
        size_t      maxbytes    /* not used                                       */
        )
```

DESCRIPTION

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

PARAMETER

The parameter **fd** is a file descriptor specifying the device which shall be used.

The parameter **buffer** points to the special I/O structure *TP501_IO_BUFFER* (refer to *tpmc501.h*).

The argument **maxbytes** is not used for the device driver.

data structure *TP501_IO_BUFFER*:

```
typedef struct
{
        int             Channel;        /* Channel number       */
        int             Gain;           /* Gain for channel     */
        unsigned long   flags;          /* Special flags    */
        long            value;          /* Return value    */
} TP501_IO_BUFFER;
```

The argument **Channel** specifies the channel to use.

The argument **Gain** specifies the gain, which shall be used.

The **flags** specify the mode to use. Allowed values are:

TP501_DIFFMODE	Enable Differential Mode
TP501_SNGLMODE	Enable Single Ended Mode
TP501_CORRENA	Enable Data Correction
TP501_CORRDIS	Disable Data Correction

The read value will be returned in **value**.

EXAMPLE

```
int                fd;
int                result;
TP501_IO_BUFFER    buf;

...

/*-------------------------------------------------------------
  Read the actual value of the input 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;
result = read (fd, &buf, 0);

...
```

RETURNS

ERROR if an error occurred

INCLUDE FILES

ioLib.h, semLib.h, tpmc501.h

SEE ALSO

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

4.4 ioctl()

NAME

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

SYNOPSIS

```
int ioctl
(
int     fd,          /* device descriptor from opened TPMC501 device      */
int     function,    /* function code                                     */
int     arg          /* optional function dependent argument              */
)
```

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls will be performed by calling the *ioctl()* function with a specific function code and an optional function dependent argument.

PARAMETER

The parameter **fd** specifies the device descriptor of the opened TPMC501 device.

The parameter **function** selects the action, which will be executed by the driver.

The structure **arg** depends on the function (see description below).

RETURNS

OK or ERROR (if the device descriptor does not exist or the function code is unknown or an error occurred)

INCULDES

ioLib.h, semLib.h, tpmc501.h

SEE ALSO

ioLib, basic I/O routine - *ioctl(),* VxWorks Programmer's Guide: I/O System

4.4.1 FIOSTARTSEQ Setup and start the sequencer

This function reads the values of the specified input lines and stores the input values into a user supplied FIFO. For this operation the argument **arg** points to a structure named *TP501_IOC_BUF*. This structure is defined in *tpmc501.h*.

Data structure *TP501_IOC_BUF*:

```
typedef struct
{
    unsigned short      cycletime;      /* sequencer cycletime  */
    unsigned long       act_channels;   /* number of active channels    */
    TP501_IO_BUFFER     *chan_setup;    /* channel configurations       */
    unsigned long       buf_size;       /* size of buffer */
    unsigned long       buf_stat;       /* buffer state / error     */
    unsigned long       putIdx;         /* Index to put data        */
    unsigned long       getIdx;         /* Index to read data       */
    long                *buffer;        /* pointer to buffer        */
} TP501_IOC_BUF
```

The **cycletime** argument specifies the length of a cycle. This value is specified in 100µs steps.

The argument **act_channels** specifies the number of active channels.

The pointer **chan_setup** points to an array of data structures specifying the channel setups. The used data structure is the same used by the read command (for more information refer to the *read* command).

The argument **buf_size** specifies the size of the input FIFO.

The actual state and errors will be shown in the **buf_stat** argument. The following bits are defined.

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.

The arguments **putIdx** and **getIdx** are used to specify the actual read and write pointer into the FIFO. **putIdx** shall be changed by the driver and just be read from the application. **getIdx** must be move by the application after reading a set of input values and is only read by the driver.

The pointer **buffer** points to the user supplied memory area where the sequencer input data will be stored.

4.4.2 FIOSTOPSEQ Stop the sequencer

This command stops a running sequence cycles. This command needs no argument.

4.4.3 EXAMPLE for control functions

```
#define   SBUF_SIZE 0x200

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

...

/*-------------------------------------------------------
  Start sequence 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;
result = ioctl(fd, FIOSTARTSEQ , (int)&cntrl_par);
if (result == OK)
{
     /* Sequencer started */
}
else
{
     /* Error when starting the sequencer */
}

...
```

```
/*-------------------------------------------------------
  Stop Sequencer
  -------------------------------------------------------*/
result = ioctl(fd, FIOSTOPSEQ , 0);
if (result == OK)
{
     /* Sequencer started */
}
else
{
     /* Error when starting the sequencer */
}
```

5 Appendix

This chapter describes the symbols which are defined in the file *tpmc501h*.

5.1 Predefined Symbols

Ioctl Function Codes

FIOSTARTSEQ	0x00000100	Start and setup the sequencer
FIOSTOPSEQ	0x00000101	Stop the sequencer

Module Types

TPMC501_10	10	TPMC501-10
TPMC501_11	11	TPMC501-11
TPMC501_12	12	TPMC501-12
TPMC501_13	13	TPMC501-13
TPMC501_20	20	TPMC501-20
TPMC501_21	21	TPMC501-21
TPMC501_22	22	TPMC501-22
TPMC501_23	23	TPMC501-23

5.2 Error Codes

If the device driver creates an error the error codes are stored in the *errno*. They can be read with the VxWorks function *errnoGet()* or *printErrno()*.

S_tp501Drv_ICHAN	0x05010001	Illegal channel number specified
S_tp501Drv_IGAIN	0x05010002	Illegal gain specified
S_tp501Drv_MODBUSY	0x05010003	Module is busy (the sequencer is running)
S_tp501Drv_TIMEOUT	0x05010004	Hardware timed out
S_tp501Drv_ICMD	0x05010005	Illegal I/O command