The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC550-SW-42

VxWorks Device Driver

8/4 Channel 12 Bit D/A

Version 1.1.x

User Manual

Issue 1.1.1

August 2005

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

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Reno, Nevada 89504 / USA
Phone: +1 (775) 686 6077
Fax: +1 (775) 686 6024
e-mail: usasales@tews.com
www.tews.com

TPMC550-SW-42

8/4 Channel 12 Bit D/A

VxWorks Device Driver

Issue	Description	Date
1.0	First Issue	May 2001
1.1	General Revision	November 2003
1.1.1	File-list changed	August 8, 2005

Table of Contents

1 Introduction

The TPMC550-SW-42 VxWorks device driver software allows the operation of the TPMC550 PMC conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open*(), *close(), write(),* and *ioctl()* functions.

The TPMC550 driver includes following functions:

- write a new output value to a specified channel
- start and setup the output sequencer
- update sequencer output values
- stop the output sequencer
- read the module configuration

2 Installation

Following files are located in directory TPMC550-SW-42 on the distribution media:

tp550drv.c	TPMC550 Driver Source
tpmc550.h	TPMC550 Application and Driver Include File
tp550def.h	TPMC550 Driver Include File
tp550tst.c	TPMC550 Example Application
tp550pci.c	TPMC550 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 TPMC550 device driver to the VxWorks system following steps have to be done:

- Build the object code of the TPMC550 device driver
- Link or load the driver object file to the VxWorks system
- Call the *tp550Drv()* function to install the device driver.

2.2 Hardware dependent Configuration

The device driver software supports TEWS PMC carrier boards and others, Motorola MVME2600/3600 boards are supported by conditional compilation. 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 configuration values for the MVME2600/3600 BSP, if the value _MVME2600_3600 is defined. If an older version of the BSP (1.1/0 up to 1.1/2) is used, the value *_OLD_BSP_* in *TP550TST.C* must be defined too. 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 (*PCIsetupTPMC550()*) sets up the PCI configuration registers, the TPMC550 registers will appear at the specified address. This function is not used for Intel x86 targets because the PCI setup is done by BSP.

The second function (*searchPCI()*) will search for the TPMC550 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 TPMC550 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 TPMC550 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 *tp550PciInit()* won't be included and the user can skip to the following steps.

The C source file **tp550pci.c** contains the function *tp550PciInit().* This routine finds out all TPMC550 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 *tp550PciInit()* 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 *tp550PciInit()* at the beginning of the root task (*usrRoot()*) in **usrConfig.c**.

Be sure that the function is called prior to MMU initialization otherwise the TPCM550 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**:

```
tp550PciInit();
```

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 = tp550drv.o tp550pci.o ...).

The function *tp550PciInit()* 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 tp550Drv()

NAME

tp550Drv() - installs the TPMC550 driver in the I/O system and initializes the driver.

SYNOPSIS

STATUS tp550Drv(void)

DESCRIPTION

This function installs the TPMC550 driver in the I/O system.

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 tp550DevCreate()

NAME

tp550DevCreate() - adds a TPMC550 device to the system and initializes device hardware.

SYNOPSIS

```
STATUS tp550DevCreate
    (
    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   IntVector,  /* interrupt vector                            */
    unsigned long   IntLevel    /* interrupt level                             */
    )
```

DESCRIPTION

This routine is called to add a device to the system that will be serviced by the TPMC550 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 TPMC550-DOC User Manual and PCI Configuration example).

The argument **CalAddr** specifies the address of the modules correction data memory (see TPMC550-DOC User Manual and PCI Configuration example).

The argument **IntVector** and **IntLevel** are board dependent. They specify the interrupt vector and the interrupt level. The value of this parameter depends on the used hardware.

EXAMPLE

```
#include  "tpmc550.h"

...

/*-------------------------------------------------------
  Create the device "/tpmc550" with the registers at
  address 0xfe000000 and the calibration data at 0xfd000000
  using interrupt level and vector 0x19
  -------------------------------------------------------------*/
status = tp550DevCreate ("/tpmc550",
                         0xFE000000,
                         0xFD000000,
                         0x19,
                         0x19);

...
```

RETURNS

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

Include Files

tpmc550.h

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 TPMC550 driver, must be 0     */
      int         mode            /* not used for TPMC550 driver, must be 0     */
      )
```

DESCRIPTION

Before I/O can be performed to the TPMC550 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 "/tpmc550" for I/O
  -------------------------------------------------------*/
fd = open("/tpmc550", 0, 0);

...
```

RETURNS

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

INCLUDES

tpmc550.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

tpmc550.h

SEE ALSO

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

4.3 write()

NAME

write() – writes new output value to the specified TPMC550 device.

SYNOPSIS

```
int write
        (
        int         fd,         /* device descriptor from opened TPMC550 device    */
        char        *buffer,    /* pointer to the write structure                  */
        size_t      nbytes      /* not used                                        */
        )
```

DESCRIPTION

This function writes a new value to a specified channel.

PARAMETER

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

The parameter **buffer** points to a data structure *TP550_RW_ARGS* (see below).

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

data structure *TP550_RW_ARGS*:

```
typedef struct
{
        unsigned long       Flags;
        long                Channel;
        long                Value;
} TP550_RW_ARGS;
```

The parameter **Flags** specifies how to make the conversion. The following values are allowed for this function.

TP550_CORR	This flag specifies, that the output value shall be corrected with the board dependent correction data.
TP550_LATCHED	The DAC will be loaded in latched mode.
TP550_SIMCONV	This flag starts a simultaneous conversion.

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

The **Value** specifies the new output data.

EXAMPLE

```
int           fd;
int           retval;
TP550_RW_ARGS rw_par;

...

/*-------------------------------------------------------------
  Set channel 3 to 0x600 and use correction data
-------------------------------------------------------------*/
rw_par.Channel= 3;
rw_par.Flags  = TP550_CORR;
rw_par.Value  = 0x600;
retval = write (fd, &rw_par, 0);
if (retval == ERROR)
{
     /* Error writing new output value */
}

...
```

RETURNS

ERROR or number of bytes written

INCLUDE FILES

tpmc550.h

SEE ALSO

ioLib, tyWrite, basic I/O routine - *write()*

4.4 ioctl()

NAME

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

SYNOPSIS

```
int ioctl
(
  int   fd,          /* device descriptor from opened TPMC550 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 TPMC550 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

tpmc550.h

SEE ALSO

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

4.4.1 FIO_TP550_READ_CONV Read the module configuration

This command will read the configuration of the TPMC550. This configuration is made by the hardware (see TPMC550-DOC User Manual). The function dependent argument **arg** points to a data structure *TP550_CONF_ARGS*. The configuration of the board will be returned in this structure.

data structure *TP550_CONF_ARGS*:

```
typedef struct
{
    unsigned long       Channels;
    unsigned long       Voltage_1_4;
    unsigned long       Voltage_5_8;
} TP550_CONF_ARGS;
```

The parameter **Channels** will return the number of implemented channels on the board. This value will be 8 for TPMC550-10/-20 and it will be 4 for TPMC550-11/-21.

The parameters **Voltage_1_4** returns the selected voltage range of channel 1, 2, 3 and 4. Possible values are:

TP550_0_10	This value specifies the channel are configured for a voltage range between 0V and +10V.
TP550_10_10	This value specifies the channel are configured for a voltage range between –10V and +10V.

The parameters **Voltage_5_8** returns the selected voltage range of channel 5, 6, 7 and 8. Possible values are:

TP550_0_10	This value specifies the channel are configured for a voltage range between 0V and +10V.
TP550_10_10	This value specifies the channel are configured for a voltage range between –10V and +10V.

4.4.2 FIO_TP550_SEQ_START Start sequencer mode

This function will setup and start the TPMC550 to work in sequencer mode. The function dependent argument **arg** points to a data structure called *TP550_SEQ_START_ARGS*. The data structure holds the values for the first and the second cycle.

data structure *TP550_SEQ_START_ARGS*:

```
typedef struct
{
    unsigned short          Time;
    TP550_CHANNEL_ARGS      ChannelA[8];
    TP550_CHANNEL_ARGS      ChannelB[8];
} TP550_SEQ_START_ARGS;
```

The argument **Time** specifies the cycle time of the sequencer. The time is scaled to 100µs steps.

The array **ChannelA** specifies the values for the first cycle. The array **ChannelB** specifies the values for the second cycle. Both arrays are arrays of the data structure *TP550_CHANNEL_ARGS*.

data structure *TP550_CHANNEL_ARGS*:

```
typedef struct
{
    unsigned long           Flags;
    long                    Value;
} TP550_CHANNEL_ARGS;
```

The parameter **Flags** specifies the settings for this channel. Allowed Flags are:

TP550_CORR	This flag specifies, that the output value shall be corrected with the board dependent correction data.
TP550_UPDATE	This flag must be set to allow a new conversion for the channel. If this flag is not set, the output of the channel will not change. This value is only used for ChannelB data.
TP550_ENABLE	This flag enables this channel to be used by the sequencer. This flag is only valid for ChannelA data. The value of a channel which is not enabled will be never changed after starting the sequencer.

The parameter **Value** specifies the output value.

4.4.3 FIO_TP550_SEQ_WRITE Update sequencer output data

This function will update the output data. The changes will be used for the next sequencer cycle. The function dependent argument **arg** points to a data structure called *TP550_SEQ_ARGS*. The data structure holds the values for the next cycle.

data structure *TP550_SEQ_ARGS*:

```
typedef struct
{
    TP550_CHANNEL_ARGS          Channel[8];
} TP550_SEQ_ARGS;
```

The array **Channel** specifies the values for the next cycle. The array is an array of the data structure *TP550_CHANNEL_ARGS*.

data structure *TP550_CHANNEL_ARGS*:

```
typedef struct
{
    unsigned long               Flags;
    long                        Value;
} TP550_CHANNEL_ARGS;
```

The parameter **Flags** specifies the settings for this channel. Allowed Flags are:

TP550_CORR	This flag specifies, that the output value shall be corrected with the board dependent correction data.
TP550_UPDATE	This flag must be set to allow a new conversion for this channel. If this flag is not set, the output of the channel will not change.

The parameter **Value** specifies the new output value.

4.4.4 FIO_TP550_SEQ_STOP Stop sequencer mode

This command stops the sequencer. The function dependent argument **arg** is not used for this function.

4.4.5 EXAMPLE for control functions

```
int                     fd;
STATUS                  retval;
TP550_CONF_ARGS         conf_par;
TP550_SEQ_START_ARGS    seq_st_par;
TP550_SEQ_ARGS          seq_par;

...

/*****************************/
/* Read module configuration */
/*****************************/
retval = ioctl(fd, FIO_TP550_READ_CONV, (int)&conf_par);
if (retval == ERROR)
{
     /* Handle error */
}

...

/*********************************/
/* Start the sequencer:          */
/* Use channel 1 and channel 3   */
/* Cycle time: 0.5 sec           */
/* channel 1: 0x100 -> 0x200     */
/*            use data correction */
/* channel 3: 0x400 -> 0x700     */
/*********************************/
/* Channel 1 */
seq_st_par.ChannelA[0].Flags = TP550_ENABLE | TP550_CORR;
seq_st_par.ChannelA[0].Value = 0x100;
seq_st_par.ChannelB[0].Flags = TP550_UPDATE | TP550_CORR;
seq_st_par.ChannelB[0].Value = 0x200;

/* Channel 3 */
seq_st_par.ChannelA[2].Flags = TP550_ENABLE;
seq_st_par.ChannelA[2].Value = 0x400;
seq_st_par.ChannelB[2].Flags = TP550_UPDATE;
seq_st_par.ChannelB[2].Value = 0x700;
```

```
/* Disable the other channels */
seq_st_par.ChannelA[1].Flags = 0;
seq_st_par.ChannelA[3].Flags = 0;
seq_st_par.ChannelA[4].Flags = 0;
seq_st_par.ChannelA[5].Flags = 0;
seq_st_par.ChannelA[6].Flags = 0;
seq_st_par.ChannelA[7].Flags = 0;

/* Setup the cycle time */
seq_st_par.Time    = 5000;

retval = ioctl(fd, FIO_TP550_SEQ_START, (int)&seq_st_par);
if (retval == ERROR)
{
     /* Handle the error */
}

...

/*********************************/
/* Write new data to sequencer:   */
/* Use channel 1 and channel 3    */
/* channel 1: xxxx -> 0x300       */
/*            use data correction */
/* channel 3: xxxx -> 0x123       */
/*********************************/
/* Channel 1 */
seq_par.Channel[0].Flags    = TP550_UPDATE | TP550_CORR;
seq_par.Channel[0].Value    = 0x300;

seq_par.Channel[2].Flags    = TP550_UPDATE;
seq_par.Channel[2].Value    = 0x123;

retval = ioctl(fd, FIO_TP550_SEQ_WRITE, (int)&seq_par);
if (retval == ERROR)
{
     /* Handle the error */
}

...

/**********************/
/* Stop the sequencer */
/**********************/
retval = ioctl(fd, FIO_TP550_SEQ_STOP, 0);
```

```
if (retval == ERROR)
{
     /* Handle the error */
}
...
```

5 Appendix

This chapter describes the symbols which are defined in the file *tpmc550h*.

5.1 Predefined Symbols

Ioctl Function Codes

FIO_TP550_SEQ_START	0x05500100	Start and setup the sequencer
FIO_TP550_SEQ_WRITE	0x05500101	Write a new output value for the next sequencer cycle
FIO_TP550_SEQ_STOP	0x05500102	Stop the sequencer
FIO_TP550_READ_CONV	0x05500103	Read module configuration

Conversion flags

TP550_CORR	(1 << 0)	Correct the output value using the board dependent correction data
TP550_LATCHED	(1 << 1)	Load DAC in latched mode
TP550_SIMCONV	(1 << 2)	Start simultaneous conversion
TP550_UPDATE	(1 << 8)	Update the channel with the next cycle
TP550_ENABLE	(1 << 9)	Enable this channel for sequencer

Voltage Ranges

TP550_0_10	0	Channels are configured in unipolar mode (0V ... +10V)
TP550_10_10	1	Channels are configured in bipolar mode (-10V ... +10V)

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_tp550Drv_NOERR	0x00000000	No error, operation successful
S_tp550Drv_IDEVICE	0x05500001	Illegal device number
S_tp550Drv_NOMEM	0x05500002	Not enough memory, driver can not allocate memory
S_tp550Drv_NOSEM	0x05500003	Driver can not create a semaphore
S_tp550Drv_CHANERR	0x05500005	Illegal channel number specified
S_tp550Drv_MODBUSY	0x05500006	Module is busy, sequencer is running, or another access is still active
S_tp550Drv_TIMEOUT	0x05500007	Waiting for completion of conversion timed out
S_tp550Drv_ICMD	0x05500008	Illegal command
S_tp550Drv_SEQOFF	0x0550000A	Sequencer is disabled, command only valid for running sequencer
S_tp550Drv_SEQERR	0x0550000B	Sequencer hardware detected an error