The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC551-SW-42

VxWorks Device Driver

8/4 Channel 16 Bit D/A

Version 2.0.x

User Manual

Issue 2.0.0

March 2007

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

TPMC551-SW-42

VxWorks Device Driver

8/4 Channel 16 Bit D/A

Supported Modules:
TPMC551

Issue	Description	Date
1.0	First Issue	July 15, 1999
1.1	Support for Intel x86 based targets	June 19, 2000
1.2	General Revision	November 28, 2003
1.3.0	File list changed (Release.txt added)	August 10, 2005
2.0.0	New Parameter Interfaces tpmc551Drv(), tpmc551DevCreate() File list changed (ChangeLog.txt added)	March 7, 2007

Table of Contents

1 Introduction

The TPMC551-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(), write(),* and *ioctl()* functions.

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

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

The TPMC551-SW-42 device driver supports the following features:

- Setting DAC output value
- Configure, start, and stop DAC-sequencer
- Use of data correction for simple conversion and in sequencer mode
- Reading TPMC551 configuration (number of channels and uni-/bipolar output)

The TPMC551-SW-42 supports the modules listed below:

TPMC551-x0	8 channel 16-bit D/A	(PMC)
TPMC551-x1	4 channel 16-bit D/A	(PMC)

In this document all supported modules and devices will be called TPMC551. Specials for 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.

TPMC551User manual

TPMC551 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC551-SW-42':

tpmc551drv.c	TPMC551 device driver source
tpmc551def.h	TPMC551 driver include file
tpmc551.h	TPMC551 include file for driver and application
tpmc551pci.c	TPMC551 PCI MMU mapping for Intel x86 based targetst
tpmc551exa.c	Example application
include/tdhal.h	Hardware dependent interface functions and definitions
TPMC551-SW-42-2.0.0.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 TPMC551-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: ./TPMC551)

(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 TPMC551 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 TPMC551 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tpmc551pci.c** contains the function *tpmc551PciInit()*. This routine finds out all TPMC551 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 *tpmc551PciInit()* 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 TPMC551 PCI spaces remain unmapped and an access fault occurs during driver initialization.

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

```
tpmc551PciInit();
```

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 conditional compilation set by the BSP header files, but some settings must be configured manually. There are two way of modification, first you can change the *include/tdhal.h* and define the corresponding definition and its value, or you can do it, using the command line option *–D*.

There are 3 offset definitions (*USERDEFINED_MEM_OFFSET*, *USERDEFINED_IO_OFFSET*, and *USERDEFINED_LEV2VEC*) that must be configured if a corresponding warning message appears during compilation. These definitions always need values. Definition values can be assigned by command line option *-D<definition>=<value>*.

definition	**description**
USERDEFINED_MEM_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI memory space access
USERDEFINED_IO_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI I/O space access
USERDEFINED_LEV2VEC	The value of this definition must be set to the difference of the interrupt vector (used to connect the ISR) and the interrupt level (stored to the PCI header)

Another definition allows a simple adaptation for BSPs that utilize 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. The definition by command line option is made by *-D<definition>*.

Please refer to the BSP documentation and header files to get information about the interrupt connection function and the required 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	---	2

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 tpmc551Drv()

NAME

tpmc551Drv() - installs the TPMC551 driver in the I/O system

SYNOPSIS

#include “tpmc551.h”

STATUS tpmc551Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus, installs the TPMC551 driver in the I/O system. Found devices are initialized.

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 "tpmc551.h”

STATUS            result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tpmc551Drv();
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 TPMC551 device found

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tpmc551DevCreate()

NAME

tpmc551DevCreate() – Add a TPMC551 device to the VxWorks system

SYNOPSIS

```
#include “tpmc551.h”

STATUS tpmc551DevCreate
(
        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.

funcType

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

pParam

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

EXAMPLE

```
#include "tpmc551.h"

STATUS        result;

/*-----------------------------------------------------
  Create the device "/tpmc551/0" for the first device
  -----------------------------------------------------*/
result = tpmc551DevCreate(   "/tpmc551/0",
                             0,
                             0,
                             0);
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	Driver not installed, tpmc551Drv() has not been executed
ENXIO	Specified module not installed
EBUSY	The device has been created.

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.3 tpmc551PciInit()

NAME

tpmc551PciInit() – Generic PCI device initialization

SYNOPSIS

void tpmc551PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC551 PCI spaces (base address register) and to enable the TPMC551 device for access.

The global variable *tpmc551Status* 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 tpmc551Status is equal to the number of mapped PCI spaces
0	No TPMC551 device found
< 0	Initialization failed. The value of (tpmc551Status & 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 tpmc551PciInit();

…

tpmc551PciInit();
```

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 TPMC551 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 tpmc551DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*----------------------------------------
  Open the device named "/tpmc551/0" for I/O
  ----------------------------------------*/
fd = open("/tpmc551/0", 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 write()

NAME

write() – write new output value to the specified TPMC551 device

SYNOPSIS

```
int write
(
        int             fd,
        char            *buffer,
        size_t          nbytes
)
```

DESCRIPTION

This function writes a new value to a specified channel.

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 buffer (*TP551_RW_ARGS*) specifying channel and the new output value.

```
typedef struct
{
        unsigned long       Flags;
        long                Channel;
        long                Value;
} TP551_RW_ARGS;
```

Flags

This parameter specifies how to make the conversion. The following flags are allowed for this function and can be ORed binary.

TP551_CORR	This flag specifies, that the output value shall be corrected with the board dependent correction data.
TP551_LATCHED	The DAC will be loaded in latched mode.
TP551_SIMCONV	This flag starts a simultaneous conversion.

Channel

This parameter specifies the channel number. The first channel is specified with 1, the second with 2 and so on.

Value

This argument specifies the new output value. Allowed values are 0…65535 for channels configured in unipolar mode (0..10V), and -32768…32767 in bipolar mode (+/-10V).

nbytes

This parameter is not used.

EXAMPLE

```
#include “tpmc551.h”

int                fd;
int                retval;
TP551_RW_ARGS      dac_par;

/*-------------------------------------------------------------
  Set channel 3 to 0x6000 and use correction data
-------------------------------------------------------------*/
dac_par.Channel    = 3;
dac_par.Flags = TP551_CORR;
dac_par.Value = 0x6000;

retval = write (fd, &dac_par, 0);
if (retval != ERROR)
{
     printf(“New DAC value set\n”, retval);
}
else
{
     /* handle the write error */
}
```

RETURNS

Number of bytes written (2) 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 code described below.

Error code	Description
S_tp551Drv_CHANERR	An unsupported channel number has been specified
S_tp551Drv_MODBUSY	The TPMC551 is busy, sequencer is running
S_tp551Drv_TIMEOUT	The sequencer has not finished conversion in the expected time

SEE ALSO

ioLib, basic I/O routine - write()

4.4 ioctl()

NAME

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

SYNOPSIS

#include “tpmc551.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
FIO_TP551_READ_CONV	Read module configuration
FIO_TP551_SEQ_START	Start sequencer mode
FIO_TP551_SEQ_WRITE	Update sequencer output data
FIO_TP551_SEQ_STOP	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 FIO_TP551_READ_CONV

This I/O control function returns the hardware configuration of the TPMC551. The function specific control parameter **arg** is a pointer on a *TP551_CONF_ARGS* structure.

```
typedef struct
{
        unsigned long       Channels;
        unsigned long       Voltage_1_4;
        unsigned long       Voltage_5_8;
} TP551_CONF_ARGS;
```

Channels

Returns the number of channels supported by the specified device.

Voltage_1_4

Returns the output mode of channel 1 up to channel 4. Possible values are:

TP551_0_10	This value specifies the channel are configured for a voltage range between 0V and +10V.
TP551_10_10	This value specifies the channel are configured for a voltage range between –10V and +10V.

Voltage_5_8

Returns the output mode of channel 5 up to channel 8. Possible values are:

TP551_0_10	This value specifies the channel are configured for a voltage range between 0V and +10V.
TP551_10_10	This value specifies the channel are configured for a voltage range between –10V and +10V.

EXAMPLE

```
#include “tpmc551.h”

int                 fd;
TP551_CONF_ARGS     confBuf;
int                 retval;

…
```

```
…

/*---------------------------
  Read hardware configuration
  ---------------------------*/
retval = ioctl(fd, FIO_TP551_READ_CONV, (int)&confBuf);
if (retval != ERROR)
{
     printf("Number of channels: %d\n", confBuf.Channels);
}
else
{
     /* handle the error */
}
```

4.4.2 FIO_TP551_SEQ_START

This I/O control function sets up and start the TPMC551 to work in sequencer mode. The function specific control parameter **arg** is a pointer on a *TP551_SEQ_START_ARGS* structure.

```
typedef struct
{
    unsigned short          Time;
    TP551_CHANNEL_ARGS      ChannelA[8];
    TP551_CHANNEL_ARGS      ChannelB[8];
} TP551_SEQ_START_ARGS;
```

Time

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

ChannelA[]

The array specifies the values for the first cycle. The array is an array of the data structure *TP551_CHANNEL_ARGS*. The array index specifies the channel, index 0 for channel 1, index 1 for channel 2, and so on.

ChannelB[]

The array specifies the values for the second cycle. The array is an array of the data structure *TP551_CHANNEL_ARGS*. The array index specifies the channel, index 0 for channel 1, index 1 for channel 2, and so on.

```
typedef struct
{
    unsigned long       Flags;
    long                Value;
} TP551_CHANNEL_ARGS;
```

Flags

The parameter specifies the flags for this channel that can be ORed. Allowed Flags are:

TP551_CORR	This flag specifies, that the output value shall be corrected with the board dependent correction data.
TP551_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.
TP551_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.

Value

This parameter specifies the new output value of the channel.

EXAMPLE

```
#include “tpmc551.h”

int                     fd;
TP551_SEQ_START_ARGS    seqBuf;
int                     retval;

/*-------------------------------------
  Start sequencer: cycle time: 0.5 sec
  -------------------------------------*/
seqBuf.Time = 5000;     /* 5000 * 100µs */

/* Channel 1: correction enabled */
seqBuf.ChannelA[0].Flags    = TP551_ENABLE | TP551_CORR;
seqBuf.ChannelA[0].Value    = 0x100;
seqBuf.ChannelB[0].Flags    = TP551_UPDATE | TP551_CORR;
seqBuf.ChannelB[0].Value    = 0x200;

/* Channel 3: correction disabled */
seqBuf.ChannelA[2].Flags    = TP551_ENABLE;
seqBuf.ChannelA[2].Value    = 0x400;
seqBuf.ChannelB[2].Flags    = TP551_UPDATE;
seqBuf.ChannelB[2].Value    = 0x700;

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

retval = ioctl(fd, FIO_TP551_SEQ_START, (int)&seqBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
S_tp551Drv_MODBUSY	The sequencer is already in use

4.4.3 FIO_TP551_SEQ_STOP

This I/O control function stops the TPMC551 sequencer mode. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tpmc551.h”

int                     fd;
int                     retval;

/*--------------
  Stop sequencer
  --------------*/
retval = ioctl(fd, FIO_TP551_SEQ_STOP, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

4.4.4 FIO_TP551_SEQ_WRITE

This I/O control function updates the output data. The specified values will be used for the next sequencer cycle. The function specific control parameter **arg** is a pointer on a *TP551_SEQ_ ARGS* structure.

```
typedef struct
{
        TP551_CHANNEL_ARGS      Channel[8];
} TP551_SEQ_ ARGS;
```

Channel[]

The array specifies the values for the next cycle. The array is an array of the data structure *TP551_CHANNEL_ARGS.* The array index specifies the channel, index 0 for channel 1, index 1 for channel 2, and so on.

```
typedef struct
{
        unsigned long       Flags;
        long                Value;
} TP551_CHANNEL_ARGS;
```

Flags

The parameter specifies the flags for this channel that can be ORed. Allowed Flags are:

TP551_CORR	This flag specifies, that the output value shall be corrected with the board dependent correction data.
TP551_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.

Value

This parameter specifies the new output value of the channel.

EXAMPLE

```
#include “tpmc551.h”

int                 fd;
TP551_SEQ_ ARGS     seqBuf;
int                 retval;

…
```

```
…

/*---------------------
  Update sequencer data
  ---------------------*/
/* Channel 1: correction enabled */
seqBuf.Channel[0].Flags = TP551_UPDATE | TP551_CORR;
seqBuf.Channel[0].Value = 0x300;

/* Channel 3: correction disabled */
seqBuf.Channel[2].Flags = TP551_UPDATE;
seqBuf.Channel[2].Value = 0x1234;

retval = ioctl(fd, FIO_TP551_SEQ_WRITE, (int)&seqBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
S_tp551Drv_SEQOFF	The sequencer is not started

5 Appendix

5.1 Additional Error Codes

Error code	Error value	Description
S_tp551Drv_CHANERR	0x05510005	An unsupported channel number has been specified
S_tp551Drv_MODBUSY	0x05510006	The sequencer is already in use, TPMC551 is busy, sequencer is running
S_tp551Drv_TIMEOUT	0x05510007	The sequencer has not finished conversion in the expected time
S_tp551Drv_ICMD	0x05510008	An illegal ioctl() command has been specified
S_tp551Drv_SEQOFF	0x0551000A	The sequencer is disabled