The Embedded I/O Company

TPMC700-SW-42

VxWorks Device Driver

32 (16) Digital Outputs

Version 3.0.x

User Manual

Issue 3.0.0

September 2010

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

TPMC700-SW-42

VxWorks Device Driver

32 (16) Digital Outputs

Supported Modules:
TPMC700

Issue	Description	Date
1.1	Intel X86 support added	March 11, 2002
1.2	General Revision	November 28, 2003
1.2.1	Filelist changed	August 11, 2005
2.0.0	Functions tpmc700Drv(), tpmc700DevCreate modified Filelist changed	March 8, 2007
2.0.1	Modified parameter description for write()	February 6, 2009
3.0.0	VxBus support and API added, general revision	September 17, 2010

Table of Contents

1 Introduction

1.1 Device Driver

The TPMC700-SW-42 VxWorks device driver software allows the operation of the supported PMC conforming to the VxWorks I/O system specification.

The TPMC700-SW-42 release contains independent driver sources for the old legacy (pre-VxBus) and the new VxBus-enabled driver model. The VxBus-enabled driver is recommended for new developments with later VxWorks 6.x release and mandatory for VxWorks SMP systems.

Both drivers, legacy and VxBus, share the same application programming interface (API) and device-independent basic I/O interface with open(), close() and ioctl() functions. The basic I/O interface is only for backward compatibility with existing applications and should not be used for new developments.

The TPMC700-SW-42 device driver supports the following features:

- set the output lines
- start and stop the output watchdog
- reset the watchdog error flag

The TPMC700-SW-42 supports the modules listed below:

TPMC700-x0	32 digital outputs	(PMC)
TPMC700-x1	16 digital outputs	(PMC)

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

TPMC700 User Manual
TPMC700 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC700-SW-42':

TPMC700-SW-42-3.0.0.pdf	PDF copy of this manual
TPMC700-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TPMC700-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TPMC700-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tpmc700':

tpmc700drv.c	TPMC700 device driver source
tpmc700def.h	TPMC700 driver include file
tpmc700.h	TPMC700 include file for driver and application
tpmc700api.c	TPMC700 API file
Makefile	Driver Makefile
40tpmc700.cdf	Component description file for VxWorks development tools
tpmc700.dc	Configuration stub file for direct BSP builds
tpmc700.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tpmc700exa.c	Example application

The archive TPMC700-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tpmc700':

tpmc700drv.c	TPMC700 device driver source
tpmc700def.h	TPMC700 driver include file
tpmc700.h	TPMC700 include file for driver and application
tpmc700pci.c	TPMC700 device driver source for x86 based systems
tpmc700api.c	TPMC700 API file
tpmc700exa.c	Example application
include/tdhal.h	Hardware dependent interface functions and definitions

2.1 Legacy vs. VxBus Driver

In later VxWorks 6.x releases, the old VxWorks 5.x legacy device driver model was replaced by VxBus-enabled device drivers. Legacy device drivers are tightly coupled with the BSP and the board hardware. The VxBus infrastructure hides all BSP and hardware differences under a well defined interface, which improves the portability and reduces the configuration effort. A further advantage is the improved performance of API calls by using the method interface and bypassing the VxWorks basic I/O interface.

VxBus-enabled device drivers are the preferred driver interface for new developments.

The checklist below will help you to make a decision which driver model is suitable and possible for your application:

Legacy Driver	VxBus Driver
▪ VxWorks 5.x releases ▪ VxWorks 6.5 and earlier releases ▪ VxWorks 6.x releases without VxBus PCI bus support	▪ VxWorks 6.6 and later releases with VxBus PCI bus ▪ SMP systems (only the VxBus driver is SMP safe!)

TEWS TECHNOLOGIES recommends not using the VxBus Driver before VxWorks release 6.6. In previous releases required header files are missing and the support for 3rd-party drivers may not be available.

2.2 VxBus Driver Installation

Because Wind River doesn't provide a standard installation method for 3rd party VxBus device drivers the installation procedure needs to be done manually.

In order to perform a manual installation extract all files from the archive TPMC700-SW-42-VXBUS.zip to the typical 3rd party directory *installDir/vxworks-6.x/target/3rdparty* (whereas *installDir* must be substituted by the VxWorks installation directory).

After successful installation the TPMC700 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tpmc700.*

At this point the TPMC700 driver is not configurable and cannot be included with the kernel configuration tool in a Wind River Workbench project. To make the driver configurable the driver library for the desired processor (CPU) and build tool (TOOL) must be built in the following way:

(1) Open a VxWorks development shell (e.g. C:\WindRiver\wrenv.exe -p vxworks-6.7)

(2) Change into the driver installation directory
installDir/vxworks-6.x/target/3rdparty/tews/tpmc700

(3) Invoke the build command for the required processor and build tool
make CPU=cpuName TOOL=tool

For Windows hosts this may look like this:

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc700
C:> make CPU=PENTIUM4 TOOL=diab
```

To compile SMP-enabled libraries, the argument VXBUILD=SMP must be added to the command line

```
C:> make CPU=PENTIUM4 TOOL=diab VXBUILD=SMP
```

To integrate the TPMC700 driver with the VxWorks development tools (Workbench), the component configuration file *40tpmc700.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc700
C:> copy 40tpmc700.cdf \Windriver\vxworks-6.7\target\config\comps\vxWorks
```

In VxWorks 6.7 and newer releases the kernel configuration tool scans the CDF file automatically and updates the *CxrCat.txt* cache file to provide component parameter information for the kernel configuration tool as long as the timestamp of the copied CDF file is newer than the one of the *CxrCat.txt*. If your copy command preserves the timestamp, force to update the timestamp by a utility, such as *touch*.

In earlier VxWorks releases the CxrCat.txt file may not be updated automatically. In this case, remove or rename the original *CxrCat.txt* file and invoke the make command to force recreation of this file.

```
C:> cd \Windriver\vxworks-6.7\target\config\comps\vxWorks
C:> del CxrCat.txt
C:> make
```

After successful completion of all steps above and restart of the Wind River Workbench, the TPMC700 driver can be included in VxWorks projects by selecting the *“TEWS TPMC700 Driver“* component in the *“hardware (default) - Device Drivers”* folder with the kernel configuration tool.

2.2.1 Direct BSP Builds

In development scenarios with the direct BSP build method without using the Workbench or the vxprj command-line utility, the TPMC700 configuration stub files must be copied to the directory *installDir/vxworks-6.x/target/config/comps/src/hwif.* Afterwards the *vxbUsrCmdLine.c* file must be updated by invoking the appropriate make command.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc700
C:> copy tpmc700.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tpmc700.dr \Windriver\vxworks-6.7\target\config\comps\src\hwif

C:> cd \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> make vxbUsrCmdLine.c
```

2.3 Legacy Driver Installation

2.3.1 Include device driver in VxWorks projects

For including the TPMC700-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Extract all files from the archive TPMC700-SW-42-LEGACY.zip to your project directory.

(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 in the tpmc700 directory 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 VxWorks User's Guide (e.g. Tornado, Workbench, etc.)

2.3.2 Special installation for Intel x86 based

The TPMC700 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 TPMC700 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tpmc700pci.c** contains the function *tpmc700PciInit().* This routine finds out all TPMC700 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 *tpmc700PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window):

```
tpmc700PciInit();
```

Be sure that the function is called prior to MMU initialization otherwise the TPMC700 PCI spaces remains unmapped and an access fault occurs during driver initialization.

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

2.3.3 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	---

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 API Documentation

3.1 General Functions

3.1.1 tpmc700Open()

Name

tpmc700Open() – opens a device.

Synopsis

```
TPMC700_DEV tpmc700Open
(
        char            *DeviceName
)
```

Description

Before I/O can be performed to a device, a file descriptor must be opened by a call to this function.

Parameters

DeviceName

This parameter points to a null-terminated string that specifies the name of the device. The first TPMC700 device is named "/tpmc700/0", the second device is named "/tpmc700/1" and so on.

Example

```
#include "tpmc700.h"

TPMC700_DEV   pDev;

/*
** open file descriptor to device
*/
pDev = tpmc700Open("/tpmc700/0");
if (pDev == NULL)
{
     /* handle open error */
}
```

RETURNS

A device descriptor pointer, or NULL if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno.*

The error code is a standard error code set by the I/O system.

3.1.2 tpmc700Close()

Name

tpmc700Close() – closes a device.

Synopsis

```
int tpmc700Close
(
    TPMC700_DEV   pDev
)
```

Description

This function closes previously opened devices.

Parameters

pDev

This value specifies the file descriptor pointer to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tpmc700.h”

Tpmc700_DEV    pDev;
int            result;

/*
** close file descriptor to device
*/
result = tpmc700Close(pDev);
if (result < 0)
{
     /* handle close error */
}
```

RETURNS

Zero, or -1 if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno*.

The error code is a standard error code set by the I/O system.

3.2 Device Access Functions

3.2.1 tpmc700Write

Name

tpmc700Write – write output value

Synopsis

```
STATUS tpmc700Write
(
        TPMC700_DEV    pDev,
        UINT32         OutputValue
)
```

Description

This function writes a long word to the output register of the specified module.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

OutputValue

This argument specifies the new output value. Bit 0 of the output word corresponds to output line 1, bit 1 corresponds to output line 2, and so on.

Bit 16 up to 32 will be ignored for TPMC700-x1 (16 output lines).

Example

```
#include “tpmc700.h”

TPMC700_DEV          pDev;
STATUS               result;

/*------------------------------
  Set output lines to 0x12345678
  ------------------------------*/
result = tpmc700Write(pDev, 0x12345678);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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. Function specific error codes will be described with the function.

Error code	Description
EBADF	The device handle is invalid
S_tpmc700Dev_WDTIMEOUT	A watchdog error occurred (, if watchdog is active)

3.2.2 tpmc700WatchdogEnable

Name

tpmc700WatchdogEnable – enable output watchdog

Synopsis

```
STATUS tpmc700WatchdogEnable
(
        TPMC700_DEV    pDev
)
```

Description

This function enables the watchdog timer for the output lines. The watchdog function is activated after the next write operation to the device. Please remember that if the watchdog is enabled and no write access occurs within 120 ms, all outputs go into the OFF state. To unlock the output register and leave the OFF state the function *tpmc700WatchdogReset* must be executed.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tpmc700.h”

TPMC700_DEV         pDev;
STATUS              result;

/*-----------------
  Enable Watchdog
  -----------------*/

result = tpmc700WatchdogEnable(pDev);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function succeeded */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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. Function specific error codes will be described with the function.

Error code	Description
EBADF	The device handle is invalid

3.2.3 tpmc700WatchdogDisable

Name

tpmc700WatchdogDisable – disable output watchdog

Synopsis

```
STATUS tpmc700WatchdogDisable
(
    TPMC700_DEV   pDev
)
```

Description

This function disables the watchdog timer for the output lines.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tpmc700.h”

TPMC700_DEV        pDev;
STATUS             result;

/*-----------------
  Disable Watchdog
  -----------------*/

result = tpmc700WatchdogDisable(pDev);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function succeeded */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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. Function specific error codes will be described with the function.

Error code	**Description**
EBADF	The device handle is invalid

3.2.4 tpmc700WatchdogReset

Name

tpmc700WatchdogReset – reset output watchdog error

Synopsis

```
STATUS tpmc700WatchdogReset
(
    TPMC700_DEV   pDev
)
```

Description

This function resets the watchdog status and clears an occurred error.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tpmc700.h”

TPMC700_DEV        pDev;
STATUS             result;

/*-----------------
  Reset Watchdog
  -----------------*/

result = tpmc700WatchdogReset(pDev);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function succeeded */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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. Function specific error codes will be described with the function.

Error code	**Description**
EBADF	The device handle is invalid

4 Legacy I/O system functions

This chapter describes the legacy 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.

The legacy I/O system functions are only relevant for the legacy TPMC700 driver. For the VxBus-enabled TPMC700 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.

4.1 tpmc700Drv()

NAME

tpmc700Drv() - installs the TPMC700 driver in the I/O system

SYNOPSIS

#include “tpmc700.h”

STATUS tpmc700Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus, installs the TPMC700 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  "tpmc700.h”

STATUS             result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tpmc700Drv();
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 TPMC700 found

SEE ALSO

VxWorks Programmer’s Guide: I/O System

4.2 tpmc700DevCreate()

NAME

tpmc700DevCreate() – Add a TPMC700 device to the VxWorks system

SYNOPSIS

```
#include "tpmc700.h"

STATUS tpmc700DevCreate
(
        char            *name,
        int             devIdx,
        int             funcType
)
```

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.

If more than one module are installed the channel 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*.

EXAMPLE

```
#include  "tpmc700.h”

STATUS             result;

/*-------------------------------------------------------
  Create the device "/tpmc700/0" for the first TPMC700 device
  -------------------------------------------------------*/
result = tpmc700DevCreate(   "/tpmc700/0",
                             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, tpmc700 has not been called
ENXIO	specified device not found
EBUSY	device has already been created

SEE ALSO

VxWorks Programmer’s Guide: I/O System

4.3 tpmc700PciInit()

NAME

tpmc700PciInit() – Generic PCI device initialization

SYNOPSIS

void tpmc700PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC700 PCI spaces (base address register) and to enable the TPMC700 device for access.

The global variable *tpmc700Status* 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 tpmc700Status is equal to the number of mapped PCI spaces
0	No TPMC700 device found
< 0	Initialization failed. The value of (tpmc700Status & 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 tpmc700PciInit();

tpmc700PciInit();
```

4.4 tpmc700Init()

NAME

tpmc700Init() – initialize TPMC700 driver and devices

SYNOPSIS

#include “tpmc700.h”

STATUS tpmc700Init(void)

DESCRIPTION

This function is used by the TPMC700 example application to install the driver and to add all available devices to the VxWorks system.

See also 3.1.1 tpmc700Open() for the device naming convention for legacy devices.

After calling this function it is not necessary to call tpmc700Drv() and tpmc700DevCreate() explicitly.

EXAMPLE

```
#include  "tpmc700.h”

STATUS    result;


result = tpmc700Init();

if (result == ERROR)
{
     /* Error handling */
}
```

RETURNS

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

ERROR CODES

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

See 4.1 and 4.2 for a description of possible error codes.

5 Basic I/O Functions

The VxWorks basic I/O interface functions are useable with the TPMC700 legacy and VxBus-enabled driver in a uniform manner.

5.1 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 TPMC700 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 tpmc700DevCreate() must be used.

flags

Not used

mode

Not used

EXAMPLE

```
int       fd;

/*-----------------------------------------
  Open the device named "/tpmc700/0" for I/O
  -----------------------------------------*/
fd = open("/tpmc700/0", 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()*

5.2 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()

5.3 ioctl()

NAME

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

SYNOPSIS

```
#include “tpmc700.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_TPMC700_WRITE	Write output value
FIO_TPMC700_WDENABLE	Enable watchdog
FIO_TPMC700_WDDISABLE	Disable watchdog
FIO_TPMC700_WDRESET	Reset watchdog error

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.

Error code	**Description**
S_tpmc700Dev_ICMD	Illegal function code specified

SEE ALSO

ioLib, basic I/O routine - ioctl()

5.3.1 FIO_TPMC700_WRITE

This I/O control function writes a new output value to the specified TPMC700 device. The function specific control parameter **arg** passes a UINT32 value to the driver.

Bit 16 up to 32 will be ignored for TPMC700-x1 (16 output lines).

EXAMPLE

```
#include “tpmc700.h”

int                 fd;
int                 retval;

/*------------------------------
  Set output lines to 0x12345678
  ------------------------------*/
retval = ioctl(fd, FIO_TPMC700_WRITE, 0x12345678);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

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_tpmc700Dev_WDTIMEOUT	A watchdog error occurred (if watchdog is active)

5.3.2 FIO_TPMC700_WDENABLE

This I/O control function enables the output watchdog timer function of the TPMC700. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tpmc700.h”

int                 fd;
int                 retval;

/*---------------------
  Enable watchdog timer
  ---------------------*/
retval = ioctl(fd, FIO_TPMC700_WDENABLE, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

5.3.3 FIO_TPMC700_WDDISABLE

This I/O control function disables the output watchdog timer function of the TPMC700. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tpmc700.h”

int                 fd;
int                 retval;

/*---------------------
  Disable watchdog timer
  ---------------------*/
retval = ioctl(fd, FIO_TPMC700_WDDISABLE, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

5.3.4 FIO_TPMC700_WDRESET

This I/O control function resets a pending watchdog error state. This must be done every time a watchdog error has occurred. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tpmc700.h”

int                 fd;
int                 retval;

/*---------------------
  Disable watchdog timer
  ---------------------*/
retval = ioctl(fd, FIO_TPMC700_WDRESET, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```