The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC600-SW-42

VxWorks Device Driver

32 (16) Digital Inputs

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

TPMC600-SW-42

VxWorks Device Driver

32 (16) Digital Inputs

Supported Modules:
TPMC600

Issue	Description	Date
1.0	First Issue	May 1, 2001
1.1	General Revision	November 2003
1.1.1	File-list changed	August 10, 2005
2.0.0	Functions tpmc600Drv(), tpmc600DevCreate modified Filelist changed	March 9, 2007
3.0.0	VxBus-Support added, driver API added “Application dependent adjustments” removed read() replaced by ioctl()-function FIO_TPMC600_READ	September 20, 2010

Table of Contents

1 Introduction

1.1 Device Driver

The TPMC600-SW-42 VxWorks device driver software allows the operation of the supported PMC conforming to the VxWorks I/O system specification.

The TPMC600-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.

Both drivers invoke a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

The TPMC600-SW-42 device driver supports the following features:

- read the actual input value
- wait for selectable input events (match, high-, low-, any transition on the input line(s))
- configure, start and stop input debouncing

The TPMC600-SW-42 supports the modules listed below:

TPMC600-x0	32 digital inputs	(PMC)
TPMC600-x1	16 digital inputs	(PMC)

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

TPMC600 User manual
TPMC600 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC600-SW-42':

TPMC600-SW-42-3.0.0.pdf	PDF copy of this manual
TPMC600-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TPMC600-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TPMC600-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tpmc600':

tpmc600drv.c	TPMC600 device driver source
tpmc600def.h	TPMC600 driver include file
tpmc600.h	TPMC600 include file for driver and application
tpmc600api.c	TPMC600 API file
Makefile	Driver Makefile
40tpmc600.cdf	Component description file for VxWorks development tools
tpmc600.dc	Configuration stub file for direct BSP builds
tpmc600.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tpmc600exa.c	Example application

The archive TPMC600-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tpmc600':

tpmc600drv.c	TPMC600 device driver source
tpmc600def.h	TPMC600 driver include file
tpmc600.h	TPMC600 include file for driver and application
tpmc600pci.c	TPMC600 device driver source for x86 based systems
tpmc600api.c	TPMC600 API file
tpmc600exa.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 TPMC600-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 TPMC600 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tpmc600.*

At this point the TPMC600 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/tpmc600

(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\tpmc600
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 TPMC600 driver with the VxWorks development tools (Workbench), the component configuration file *40tpmc600.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc600
C:> copy 40tpmc600.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 TPMC600 driver can be included in VxWorks projects by selecting the *"TEWS TPMC600 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 TPMC600 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\tpmc600
C:> copy tpmc600.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tpmc600.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 TPMC600-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 TPMC600-SW-42-LEGACY.zip to your project directory.

(2) Add the device driver's 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 tpmc600 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 TPMC600 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 an MMU mapping entry has to be added for the required TPMC600 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tpmc600pci.c** contains the function *tpmc600PciInit()*. This routine finds out all TPMC600 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 *tpmc600PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window):

```
tpmc600PciInit();
```

Be sure that the function is called prior to MMU initialization otherwise the TPMC600 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	---
Semaphores	<number of active event wait jobs> [1)]	1

[1)] The semaphore will be created dynamically if an event wait job is requested

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 tpmc600Open()

Name

tpmc600Open() – opens a device.

Synopsis

```
TPMC600_DEV tpmc600Open
(
      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 TPMC600 device is named “/tpmc600/0”, the second device is named “/tpmc600/1” and so on.

Example

```
#include “tpmc600.h”

TPMC600_DEV   pDev;

/*
** open file descriptor to device
*/
pDev = tpmc600Open(“/tpmc600/0”);
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 tpmc600Close()

Name

tpmc600Close() – closes a device.

Synopsis

```
int tpmc600Close
(
     TPMC600_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 “tpmc600.h”

TPMC600_DEV   pDev;
int           result;

/*
** close file descriptor to device
*/
result = tpmc600Close(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 tpmc600Read

Name

tpmc600Read – read input state of device

Synopsis

```
STATUS tpmc600Read
(
    TPMC600_DEV   pDev,
    unsigned int  *pDigInVal
)
```

Description

This function reads the current input state of the device.

Parameters

pDev

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

pDigInVal

This argument points to a buffer where the state of the input lines will be returned. Bit 0 of the returned value represents the state of IN1, bit 1 of IN2, and so on. If a module variant is used, supporting less than 32 input lines, the unused bits will be set to 0.

Example

```
#include “tpmc600.h”

TPMC600_DEV        pDev;
STATUS             result;
unsigned int       in_value;

/*
** read current state of I/O lines
*/
result = tpmc600Read(pDev, &in_value);
if (result == ERROR)
{
     /* handle error */
}
else
{
     printf(“input value: 0x%08X\n”, in_value);
}
```

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.

Error code	Description
EINVAL	A NULL pointer is referenced for an input value
EBADF	The device handle is invalid

3.2.2 tpmc600EnableDebouncer

Name

tpmc600EnableDebouncer – configure and enable input debouncer

Synopsis

```
STATUS tpmc600EnableDebouncer
(
    TPMC600_DEV     pDev,
    unsigned short  debValue
)
```

Description

This function configures the input debouncer, which shall prevent the bord to detect fast faulty signal changes.

Parameters

pDev

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

debValue

This argument specifies the debouncer timer value. Valid values are 0 for 7us and 65535 for 440ms. Please refer to the TPMC600 User Manual for a detailed description.

Example

```
#include “tpmc600.h”

TPMC600_DEV        pDev;
STATUS             result;

/*
** enable debouncer with a debounce time of ~1ms (143)
*/
result = tpmc600EnableDebouncer(pDev, 143);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function successfully completed */
}
```

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.

Error code	Description
EBADF	The device handle is invalid

3.2.3 tpmc600DisableDebouncer

Name

tpmc600DisableDebouncer – disable input debouncer

Synopsis

```
STATUS tpmc600DisableDebouncer
(
     TPMC600_DEV pDev
)
```

Description

This function disabled the input debouncer.

Parameters

pDev

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

Example

```
#include “tpmc600.h”

TPMC600_DEV        pDev;
STATUS             result;

/*
** disable debouncer
*/
result = tpmc600DisableDebouncer(pDev);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function successfully completed */
}
```

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.

Error code	Description
EINVAL	A NULL pointer is referenced for an input value

3.2.4 tpmc600WaitForAnyEvent

Name

tpmc600WaitForAnyEvent – wait for transition on input line

Synopsis

```
STATUS tpmc600WaitForAnyEvent
(
    TPMC600_DEV   pDev,
    int           inputLine,
    int           msTimeout
)
```

Description

This function waits for transition on the specified input line.

Parameters

pDev

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

inputLine

This argument specifies the input line. A value of 1 must be specified for IN1, 2 for IN2 and so on. The maximum valid input line depends on the used module variant.

msTimeout

This argument specifies the maximum time in milliseconds the function will wait for the specified event. If the time elapses without the event occurred, the function will return with an adequate error. A value of -1 specifies that the function never times out.

Example

```
#include “tpmc600.h”

TPMC600_DEV        pDev;
STATUS             result;

/*
** wait for a transition (any) on IN12
**     timeout after approximal 10 sec.
*/
result = tpmc600WaitForAnyEvent(pDev, 12, 10000);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* event occurred */
}
```

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.

Error code	Description
EINVAL	An invalid parameter value (inputLine) has been specified
EBADF	The device handle is invalid

3.2.5 tpmc600WaitForHighEvent

Name

tpmc600WaitForHighEvent – wait for low-to-high transition on input line

Synopsis

```
STATUS tpmc600WaitForHighEvent
(
    TPMC600_DEV   pDev,
    int           inputLine,
    int           msTimeout
)
```

Description

This function waits for a low-to-high transition on the specified input line.

Parameters

pDev

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

inputLine

This argument specifies the input line. A value of 1 must be specified for IN1, 2 for IN2 and so on. The maximum valid input line depends on the used module variant.

msTimeout

This argument specifies the maximum time in milliseconds the function will wait for the specified event. If the time elapses without the event occurred, the function will return with an adequate error. A value of -1 specifies that the function never times out.

Example

```
#include “tpmc600.h”

TPMC600_DEV        pDev;
STATUS             result;

/*
** wait for a low-to-high transition on IN7
**     timeout after approximal 0.5 sec.
*/
result = tpmc600WaitForHighEvent(pDev, 7, 500);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* event occurred */
}
```

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.

Error code	Description
EINVAL	An invalid parameter value (inputLine) has been specified
EBADF	The device handle is invalid

3.2.6 tpmc600WaitForLowEvent

Name

tpmc600WaitForLowEvent – wait for a high-to-low transition on input line

Synopsis

```
STATUS tpmc600WaitForLowEvent
(
    TPMC600_DEV   pDev,
    int           inputLine,
    int           msTimeout
)
```

Description

This function waits for a high-to-low transition on the specified input line.

Parameters

pDev

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

inputLine

This argument specifies the input line. A value of 1 must be specified for IN1, 2 for IN2 and so on. The maximum valid input line depends on the used module variant.

msTimeout

This argument specifies the maximum time in milliseconds the function will wait for the specified event. If the time elapses without the event occurred, the function will return with an adequate error. A value of -1 specifies that the function never times out.

Example

```
#include “tpmc600.h”

TPMC600_DEV        pDev;
STATUS             result;

/*
** wait for a high-to-low transition on IN5
**     never timeout
*/
result = tpmc600WaitForLowEvent(pDev, 5, -1);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* event occurred */
}
```

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.

Error code	Description
EINVAL	An invalid parameter value (inputLine) has been specified
EBADF	The device handle is invalid

3.2.7 tpmc600WaitForMultiAnyEvents

Name

tpmc600WaitForMultiAnyEvents – wait for 1st transition on set of input lines and return input state

Synopsis

```
STATUS tpmc600WaitForMultiAnyEvents
(
    TPMC600_DEV     pDev,
    unsigned int    lineMask,
    int             msTimeout,
    unsigned int    *pDigInVal
)
```

Description

This function waits for the first transition on any of the specified input lines. After detection of the transition the input state will be read and returned.

There is a delay between the transition and actual reading of the input state. This means that the returned value represents the input state a short time after the transition has occurred.

Parameters

pDev

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

lineMask

This argument specifies a mask of input lines that are active to wait for a transition. A set bit specifies an active input line. Input lines set to 0 will not be observed. Bit 0 is assigned to IN1, bit 1 to IN2, and so on.

msTimeout

This argument specifies the maximum time in milliseconds the function will wait for the specified event. If the time elapses without the event occurred, the function will return with an adequate error. A value of -1 specifies that the function never times out.

pDigInVal

This argument points to a buffer where the state of the input lines will be returned. Bit 0 of the returned value represents the state of IN1, bit 1 of IN2, and so on. If a module variant is used, supporting less than 32 input lines, the unused bits will be set to 0.

Example

```
#include “tpmc600.h”

TPMC600_DEV        pDev;
STATUS             result;
unsigned int       inVal;

/*
** read input state after 1st transition (any)
**     on IN1, IN2, IN3, IN4, or IN16
**     timeout after approx. 10 sec.
*/
result = tpmc600WaitForMultiAnyEvents(pDev,
                                      0x0000800F,
                                      10000,
                                      (int)&inVal);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* event occurred */
     printf(“Input Value after event: %08Xh\n”, inVal);
}
```

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.

Error code	Description
EINVAL	A NULL pointer is referenced for an input value
EBADF	The device handle is invalid

3.2.8 tpmc600WaitForMultiHighEvents

Name

tpmc600WaitForMultiHighEvents – wait for 1st low-to-high transition on set of input lines and return input state

Synopsis

```
STATUS tpmc600WaitForMultiHighEvents
(
    TPMC600_DEV     pDev,
    unsigned int    lineMask,
    int             msTimeout,
    unsigned int    *pDigInVal
)
```

Description

This function waits for the first low-to-high transition on any of the specified input lines. After detection of the transition the input state will be read and returned.

There is a delay between the transition and actual reading of the input state. This means that the returned value represents the input state a short time after the transition has occurred.

Parameters

pDev

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

lineMask

This argument specifies a mask of input lines that are active to wait for a transition. A set bit specifies an active input line, input lines set to 0 will not be observed. Bit 0 is assigned to IN1, bit 1 to IN2, and so on.

msTimeout

This argument specifies the maximum time in milliseconds the function will wait for the specified event. If the time elapses without the event occurred, the function will return with an adequate error. A value of -1 specifies that the function never times out.

pDigInVal

This argument points to a buffer where the state of the input lines will be returned. Bit 0 of the returned value represents the state of IN1, bit 1 of IN2, and so on. If a module variant is used, supporting less than 32 input lines, the unused bits will be set to 0.

Example

```
#include “tpmc600.h”

TPMC600_DEV        pDev;
STATUS             result;
unsigned int       inVal;

/*
** read input state after 1st low-to-high transition
**     on IN2, or IN16
**     timeout after approximal 1 sec.
*/
result = tpmc600WaitForMultiHighEvents(pDev,
                                       0x00008002,
                                       1000,
                                       (int)&inVal);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* high event occurred */
     printf(“Input Value after event: %08Xh\n”, inVal);
}
```

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.

Error code	Description
EINVAL	A NULL pointer is referenced for an input value
EBADF	The device handle is invalid

3.2.9 tpmc600WaitForMultiLowEvents

Name

tpmc600WaitForMultiLowEvents – wait for 1st high-to-low transition on set of input lines and return input state

Synopsis

```
STATUS tpmc600WaitForMultiLowEvents
(
    TPMC600_DEV     pDev,
    unsigned int    lineMask,
    int             msTimeout,
    unsigned int    *pDigInVal
)
```

Description

This function waits for the first high-to-low transition on any of the specified input lines. After detection of the transition the input state will be read and returned.

There is a delay between the transition and actual reading of the input state. This means that the returned value represents the input state a short time after the transition has occurred.

Parameters

pDev

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

lineMask

This argument specifies a mask of input lines that are active to wait for a transition. A set bit specifies an active input line, input lines set to 0 will not be observed. Bit 0 is assigned to IN1, bit 1 to IN2, and so on.

msTimeout

This argument specifies the maximum time in milliseconds the function will wait for the specified event. If the time elapses without the event occurred, the function will return with an adequate error. A value of -1 specifies that the function never times out.

pDigInVal

This argument points to a buffer where the state of the input lines will be returned. Bit 0 of the returned value represents the state of IN1, bit 1 of IN2, and so on. If a module variant is used, supporting less than 32 input lines, the unused bits will be set to 0.

Example

```
#include “tpmc600.h”

TPMC600_DEV        pDev;
STATUS             result;
unsigned int       inVal;

/*
** read input state after a high-to-low transition
**     on any input line
**     never timeout
*/
result = tpmc600WaitForMultiLowEvents(pDev,
                                      0xFFFFFFFF,
                                      -1,
                                      (int)&inVal);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* high-to-low event occurred */
     printf(“Input Value after event: %08Xh\n”, inVal);
}
```

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.

Error code	Description
EINVAL	A NULL pointer is referenced for an input value
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 TPMC600 driver. For the VxBus-enabled TPMC600 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.

4.1 tpmc600Drv()

NAME

tpmc600Drv() - installs the TPMC600 driver in the I/O system

SYNOPSIS

#include “tpmc600.h”

STATUS tpmc600Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus, installs the TPMC600 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 "tpmc600.h”

STATUS            result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tpmc600Drv();
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 TPMC600 found

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.2 tpmc600DevCreate()

NAME

tpmc600DevCreate() – Add a TPMC600 device to the VxWorks system

SYNOPSIS

```
#include “tpmc600.h”

STATUS tpmc600DevCreate
 (
       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 multiple devices are installed the index 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 "tpmc600.h"

STATUS            result;

/*-------------------------------------------------------------
  Create the device "/tpmc600/0" for the first TPMC600 device
..* Device specific parameters must be set up:
        xxx <- 10
        yyy <- 20
  -------------------------------------------------------------*/

result = tpmc600DevCreate(   "/tpmc600/0",
                             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	The driver has not been installed
ENXIO	Specified device not found

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.3 tpmc600PciInit()

NAME

tpmc600PciInit() – Generic PCI device initialization

SYNOPSIS

void tpmc600PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC600 PCI spaces (base address register) and to enable the TPMC600 device for access.

The global variable *tpmc600Status* 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 tpmc600Status is equal to the number of mapped PCI spaces
0	No TPMC600 device found
< 0	Initialization failed. The value of (tpmc600Status & 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 tpmc600PciInit();

tpmc600PciInit();
```

4.4 tpmc600Init()

NAME

tpmc600Init() – initialize TPMC600 driver and devices

SYNOPSIS

#include “tpmc600.h”

STATUS tpmc600Init(void)

DESCRIPTION

This function is used by the TPMC600 example application to install the driver and to add all available devices to the VxWorks system.

See also 3.1.1 tpmc600Open() for the device naming convention for legacy devices.

After calling this function it is not necessary to call tpmc600Drv() and tpmc600DevCreate() explicitly.

EXAMPLE

```
#include "tpmc600.h”

STATUS    result;

result = tpmc600Init();

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 tpmc600Drv() and 4.2 tpmc600DevCreate() for a description of possible error codes.

5 Basic I/O Functions

The VxWorks basic I/O interface functions are useable with the TPMC600 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 TPMC600 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 tpmc600DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*----------------------------------------
  Open the device named "/tpmc600/0" for I/O
  ----------------------------------------*/
fd = open("/tpmc600/0", 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 “tpmc600.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_TPMC600_READ	read input state
FIO_TPMC600_DEBENABLE	configure and enable debouncer function
FIO_TPMC600_DEBDISABLE	disable debouncer function
FIO_TPMC600_EVREAD	wait for input event and read back the input value

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_tpmc600Dev_ICMD	unknown ioctl() command code specified

SEE ALSO

ioLib, basic I/O routine - ioctl()

5.3.1 FIO_TPMC600_READ

This I/O control function reads the current input value. The function specific control parameter **arg** is a pointer to an unsigned int value where the input value will be returned. Bit 0 of the returned value represents the state of IN1, bit 1 of IN2, and so on. If a module variant is used, supporting less than 32 input lines, the unused bits will be set to 0.

EXAMPLE

```
#include “tpmc600.h”

int                fd;
unsigned int       inVal;
int                retval;

/*-------------------------
  Read state of input lines
  -------------------------*/
retval = ioctl(fd, FIO_TPMC600_READ, (int)&inVal);
if (retval != ERROR)
{
     /* function succeeded */
     printf(“Input: %08Xh\n”, inVal);
}
else
{
     /* handle the error */
}
```

5.3.2 FIO_TPMC600_DEBENABLE

This I/O control function configures and enables the debouncer function. The function specific control parameter **arg** defines the debouncer time. Valid values are 0 for 7us and 65535 for 440ms. Please refer to the TPMC600 User Manual for a detailed description.

EXAMPLE

```
#include “tpmc600.h”

int                fd;
int                retval;

/*-----------------------------------------
  Enable debouncer using the max. debounce time
  -----------------------------------------*/
retval = ioctl(fd, FIO_TPMC600_DEBENABLE, 65535);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

5.3.3 FIO_TPMC600_DEBDISABLE

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

EXAMPLE

```
#include “tpmc600.h”

int                fd;
int                retval;

/*--------------------------
  Disable debouncer function
  --------------------------*/
retval = ioctl(fd, FIO_TPMC600_DEBDISABLE, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

5.3.4 FIO_TPMC600_EVREAD

This I/O control function waits for a specified input event. The function specific control parameter **arg** is a pointer to a *TPMC600_EVREAD_STRUCT* structure.

```
typedef struct
{
        unsigned long       mode;
        unsigned long       mask;
        unsigned long       match;
        unsigned long       value;
        unsigned long       timeout;
} TPMC600_EVREAD_STRUCT;
```

mode

This parameter selects the type of the event. There are four different event types:

Value	Description
TPMC600_MATCH	In this mode the I/O request terminates, if all bits, which are masked in the call parameter mask, have the state defined in the parameter *match*. Note that this mode is not recommended for use anymore, runtime problems may hide matching events.
TPMC600_HIGH_TR	In this mode the I/O request terminates, if at least one of the specified input lines has a low to high transition. The parameter *match* will not be used.
TPMC600_LOW_TR	In this mode the I/O request terminates, if at least one of the specified input lines has a high to low transition. The parameter *match* will not be used.
TPMC600_ANY_TR	In this mode the I/O request terminates, if at least one of the specified input lines has a transition of any direction. The parameter *match* will not be used.

mask

This value specifies the bit(s) the driver has to observe for the event. (See also *mode* table)

match

This value specifies the matching pattern. This parameter is only used for *TPMC600_MATCH* events. (See also *mode* table)

value

This argument returns the input value shortly after the event occurred.

There is a delay between the specified event and the input value read. This bases on the system and OS dependent interrupt latency, so the value may not show the state at event time. The use of the event function and its value is not recommended for fast changing signals.

timeout

This value specifies the maximum time the function shall wait for the specified event. If the specified time has passed, the function will return with an error. The timeout is specified in ticks.

EXAMPLE

```
#include “tpmc600.h”

int                     fd;
TPMC600_EVREAD_STRUCT   param;
int                     retval;

/*-------------------------------------------------
  Wait until bit 3 the input port has a low to high
  transition or timeout after 100 ticks
-------------------------------------------------*/
param.mode    = TPMC600_HIGH_TR;
param.mask    = (1 << 3);
param.timeout = 100;

retval = ioctl(fd, FIO_TPMC600_EVREAD, (int)&param);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
S_tpmc600Dev_ILLMODE	Illegal event mode specified