The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC700-SW-42

VxWorks Device Driver

32 (16) Digital Outputs

Version 4.0.x

User Manual

Issue 4.0.0

April 2016

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
1.0	First Issue	1999
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
4.0.0	New implementation of API, support of VxWorks 6.9 (SMP and 64-bit)	April 25, 2016

Table of Contents

1 Introduction

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 releases and mandatory for VxWorks SMP systems.

Both drivers, legacy and VxBus, share the same application programming interface (API).

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

- Set Output Lines
- Start and Stop Output Watchdog
- Reset 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 TPMC700 devices it is recommended to read the manuals listed below.

TPMC700 User Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC700-SW-42':

TPMC700-SW-42-4.0.0.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.h	TPMC700 API include file
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
tpmc700api.h	TPMC700 API include file
tpmc700api.c	TPMC700 API file
tpmc700pci.c	TPMC700 PCI MMU mapping for Intel x86 based targets
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!) ▪ 64-bit systems (only the VxBus driver is 64-bit compatible)

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 processer (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 build SMP-enabled libraries, the argument VXBUILD=SMP must be added to the command line

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

To build 64-bit libraries, the argument VXBUILD=LP64 must be added to the command line

```
> make TOOL=gnu CPU=CORE VXBUILD=LP64
```

For 64-bit SMP-enabled libraries a build command may look like this

```
> make TOOL=gnu CPU=CORE VXBUILD="LP64 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" and the "TEWS TPMC700 API"* 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 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 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 Targets

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).

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.

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

```
tpmc700PciInit();
```

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

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_HANDLE tpmc700Open
(
        char            *DeviceName
)
```

DESCRIPTION

Before I/O can be performed to a device, a device handle must be opened by a call to this function. If the legacy TPMC700 driver is used, this function will also install the legacy driver and create devices with the first call. The VxBus TPMC700 driver will be installed automatically by the VxBus system.

The tpmc700Open function can be called multiple times (e.g. in different tasks)

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 “tpmc700api.h”

TPMC700_HANDLE     hdl;

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

RETURNS

A device handle, 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

```
TPMC700_STATUS tpmc700Close
(
     TPMC700_HANDLE       hdl
)
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tpmc700api.h”

TPMC700_HANDLE     hdl;
TPMC700_STATUS     result;

/*
** close file descriptor to device
*/
result = tpmc700Close( hdl );
if (result != TPMC700_OK)
{
     /* handle close error */
}
```

RETURNS

On success, TPMC700_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC700_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.3 tpmc700GetPciInfo

NAME

tpmc700GetPciInfo – Get PCI information of the module

SYNOPSIS

```
TPMC700_STATUS tpmc700GetPciInfo
(
    TPMC700_HANDLE          hdl,
    TPMC700_PCIINFO_BUF     *pPciInfoBuf
)
```

DESCRIPTION

This function returns information about the module's PCI header as well as the PCI localization.

PARAMETERS

hdl

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

pPciInfoBuf

This argument is a pointer to the structure TPMC700_PCIINFO_BUF that receives information of the module PCI header.

```
typedef struct
{
    unsigned short      vendorId;
    unsigned short      deviceId;
    unsigned short      subSystemId;
    unsigned short      subSystemVendorId;
    int                 pciBusNo;
    int                 pciDevNo;
    int                 pciFuncNo;
} TPMC700_PCIINFO_BUF;
```

vendorId

PCI module vendor ID.

deviceId

PCI module device ID

subSystemId

PCI module sub system ID

subSystemVendorId

PCI module sub system vendor ID

pciBusNo

Number of the PCI bus, where the module resides.

pciDevNo

PCI device number

pciFuncNo

PCI function number

EXAMPLE

```
#include “tpmc700api.h”

TPMC700_HANDLE          hdl;
TPMC700_STATUS          result;
TPMC700_PCIINFO_BUF     pciInfoBuf;

/*
** get module PCI information
*/
result = tpmc700GetPciInfo( hdl, &pciInfoBuf );

if (result == TPMC700_OK)
{
     printf( “PCI Localization (Bus:Dev.Func): %d:%d.%d\n”,
               pciInfoBuf.pciBusNo,
               pciInfoBuf.pciDevNo,
               pciInfoBuf.pciFuncNo );
}
else
{
     /* handle error */
}
```

RETURN VALUE

On success, TPMC700_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC700_ERR_INVALID_HANDLE	The specified device handle is invalid
TPMC700_ERR_INVAL	Specified pointer is invalid.

3.2 Output Functions

3.2.1 tpmc700Write

NAME

tpmc700Write – Write Output Value

SYNOPSIS

```
TPMC700_STATUS tpmc700Write
(
    TPMC700_HANDLE          hdl,
    unsigned int            OutputValue
)
```

DESCRIPTION

This function writes the specified output value (32bit) to the specific module.

PARAMETERS

hdl

This value specifies the device handle 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 the first output line, bit 1 corresponds to the second output line, and so on.

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

EXAMPLE

```
#include “tpmc700api.h”

TPMC700_HANDLE      hdl;
TPMC700_STATUS      result;

…
```

…

```
/*-----------------------------
  Set output lines to 0x12345678
  -----------------------------*/
result = tpmc700Write(  hdl,
                        0x12345678 );
if (result != TPMC700_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TPMC700_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC700_ERR_INVALID_HANDLE	The specified device handle is invalid.
TPMC700_ERR_TIMEOUT	Watchdog Timeout error occurred.

3.2.2 tpmc700WriteMask

NAME

tpmc700WriteMask – Write Output Value with Bitmask

SYNOPSIS

```
TPMC700_STATUS tpmc700WriteMask
(
    TPMC700_HANDLE      hdl,
    unsigned int        OutputValue,
    unsigned int        Mask
)
```

DESCRIPTION

This function sets the output lines to the specified value. Only output lines specified by the bitmask are affected.

PARAMETERS

hdl

This value specifies the device handle 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 the first output line, bit 1 corresponds to the second output line, and so on.

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

Mask

This parameter specifies a 32bit mask. ‘1’ means that the corresponding bit in *OutputValue* will be updated. ‘0’ bits will be left unchanged. Bit 0 corresponds to the first output line, bit 1 corresponds to the second output line and so on.

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

EXAMPLE

```
#include “tpmc700api.h”

TPMC700_HANDLE     hdl;
TPMC700_STATUS     result;
unsigned int       OutputValue;
unsigned int       Mask;

/*-------------------------------------------------
  Set output line 1 and 8 (bit 0 and bit 7), and
  clear output line 32 (bit 31)
  -------------------------------------------------*/
OutputValue   = (1 <<  7) | (1 << 0);
Mask          = (1 << 31) | (1 << 7) | (1 << 0);

result = tpmc700WriteMask(   hdl,
                             OutputValue,
                             Mask );
if (result != TPMC700_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TPMC700_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC700_ERR_INVALID_HANDLE	The specified device handle is invalid.
TPMC700_ERR_TIMEOUT	Watchdog Timeout error occurred.

3.2.3 tpmc700OutputLineSet

NAME

tpmc700OutputLineSet – Set the specific Output Line

SYNOPSIS

```
TPMC700_STATUS tpmc700OutputLineSet
(
    TPMC700_HANDLE          hdl,
    int                     OutputLine
)
```

DESCRIPTION

This function sets the specified output line to ‘1’.

PARAMETERS

hdl

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

OutputLine

This argument specifies the output line number which shall be set. Valid values are 1 to 32. For TPMC700-x1 modules, values higher than 16 are ignored.

EXAMPLE

```
#include “tpmc700api.h”

TPMC700_HANDLE     hdl;
TPMC700_STATUS     result;

…
```

…

```
/*-------------------------------------------------
  Set output line 4
  -------------------------------------------------*/
result = tpmc700OutputLineSet( hdl, 4 );
if (result != TPMC700_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TPMC700_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC700_ERR_INVALID_HANDLE	The specified device handle is invalid.
TPMC700_ERR_TIMEOUT	Watchdog Timeout error occurred.
TPMC700_ERR_INVAL	Specified output line is invalid.

3.2.4 tpmc700OutputLineClear

NAME

tpmc700OutputLineClear – Clear the specific Output Line

SYNOPSIS

```
TPMC700_STATUS tpmc700OutputLineClear
(
      TPMC700_HANDLE          hdl,
      int                     OutputLine
)
```

DESCRIPTION

This function clears the specified output line to ‘0’.

PARAMETERS

hdl

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

OutputLine

This argument specifies the output line number which shall be cleared. Valid values are 1 to 32. For TPMC700-x1 modules, values higher than 16 are ignored.

EXAMPLE

```
#include “tpmc700api.h”

TPMC700_HANDLE     hdl;
TPMC700_STATUS     result;

/*-------------------------------------------------
  Clear output line 32
  -------------------------------------------------*/
result = tpmc700OutputLineClear( hdl, 32 );
if (result != TPMC700_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TPMC700_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC700_ERR_INVALID_HANDLE	The specified device handle is invalid.
TPMC700_ERR_TIMEOUT	Watchdog Timeout error occurred.
TPMC700_ERR_INVAL	Specified output line is invalid.

3.2.5 tpmc700OutputStatus

NAME

tpmc700OutputStatus – Read Status of Output Lines and Watchdog

SYNOPSIS

```
TPMC700_STATUS tpmc700OutputStatus
(
    TPMC700_HANDLE      hdl,
    unsigned int        *pOutputValue,
    unsigned int        *pWatchdogStatus
)
```

DESCRIPTION

This function reads the status of the output lines and also the watchdog facility.

PARAMETERS

hdl

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

pOutputValue

This argument is a pointer to an *unsigned int* (32bit) value where the output line status is returned. Bit 0 of the output word corresponds to the first output line, bit 1 corresponds to the second output line, and so on. For TPMC700-x1, the upper 16bits shall be ignored.

pWatchdogStatus

This argument is a pointer to an *unsigned int* (32bit) value where the watchdog status is returned. The following values are possible:

Value	Description
TPMC700_WD_ENABLED	The Watchdog is enabled.
TPMC700_WD_DISABLED	The Watchdog is disabled.
TPMC700_WD_FAILURE	The Watchdog has recognized a failure and has disabled all output channels.

EXAMPLE

```
#include “tpmc700api.h”

TPMC700_HANDLE     hdl;
TPMC700_STATUS     result;
unsigned int       OutputValue;
unsigned int       WatchdogStatus;

/*-------------------------------------------------
  Read output status
  -------------------------------------------------*/

result = tpmc700OutputStatus(    hdl,
                                 &OutputValue,
                                 &WatchdogStatus );
if (result == TPMC700_OK)
{
     if (WatchdogStatus != TPMC700_WD_FAILURE)
     {
          printf(“Output Status: 0x%08X\n”, OutputValue);
     }
     else
     {
          printf(“Output disabled by Watchdog\n”);
     }
}
else
{
     /* handle error */
}
```

RETURN VALUE

On success, TPMC700_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC700_ERR_INVALID_HANDLE	The specified device handle is invalid.

3.3 Watchdog Functions

3.3.1 tpmc700WatchdogEnable

NAME

tpmc700WatchdogEnable – Enable Output Watchdog

SYNOPSIS

```
TPMC700_STATUS tpmc700WatchdogEnable
(
      TPMC700_HANDLE              hdl
)
```

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

hdl

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

EXAMPLE

```
#include “tpmc700api.h”

TPMC700_HANDLE     hdl;
TPMC700_STATUS     result;

…
```

…

```
/*-----------------
  Enable Watchdog
  -----------------*/
result = tpmc700WatchdogEnable( hdl );
if (result != TPMC700_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TPMC700_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC700_ERR_INVALID_HANDLE	The specified device handle is invalid.

3.3.2 tpmc700WatchdogDisable

NAME

tpmc700WatchdogDisable – Disable Output Watchdog

SYNOPSIS

```
TPMC700_STATUS tpmc700WatchdogDisable
(
     TPMC700_HANDLE              hdl
)
```

DESCRIPTION

This function disables the watchdog timer for the output lines.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tpmc700api.h”

TPMC700_HANDLE     hdl;
TPMC700_STATUS     result;

/*-----------------
  Disable Watchdog
  -----------------*/
result = tpmc700WatchdogDisable( hdl );
if (result != TPMC700_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TPMC700_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC700_ERR_INVALID_HANDLE	The specified device handle is invalid.

3.3.3 tpmc700WatchdogReset

NAME

tpmc700WatchdogReset – Reset Output Watchdog Error

SYNOPSIS

```
TPMC700_STATUS tpmc700WatchdogReset
(
     TPMC700_HANDLE              hdl
)
```

DESCRIPTION

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

PARAMETERS

hdl

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

EXAMPLE

```
#include “tpmc700api.h”

TPMC700_HANDLE     hdl;
TPMC700_STATUS     result;

/*-----------------
  Reset Watchdog
  -----------------*/
result = tpmc700WatchdogReset( hdl );
if (result != TPMC700_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TPMC700_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC700_ERR_INVALID_HANDLE	The specified 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 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();
```

5 Debugging and Diagnostic

The TPMC700 device driver provides a function and debug statements to display versatile information of the driver installation and status on the debugging console.

If the VxBus driver is used, the TPMC700 show routine is included in the driver by default and can be called from the VxWorks shell. If this function is not needed or program space is rare the function can be removed from the code by un-defining the macro INCLUDE_TPMC700_SHOW in tpmc700drv.c

The tpmc700Show function (only if VxBus is used) displays detailed information about probed modules, assignment of devices respective device names to probed TPMC700 modules and device statistics.

If TPMC700 modules were probed but no devices were created it may be helpful to enable debugging code inside the driver code by defining the macro TPMC700_DEBUG in tpmc700drv.c.

In contrast to VxBus TPMC700 devices, legacy TPMC700 devices must be created "manually". This will be done with the first call to the tpmc700Open API function.

```
-> tpmc700Show
Probed Modules:
    [0] TPMC700-10: Bus=4, Dev=2, DevId=0x0353, VenId=0x1498, Init=OK, vxDev=0x5380

Associated Devices:
    [0] TPMC700-10: /tpmc700/0

Device Statistics:
    /tpmc700/0:
        open count            = 0
        Current Output Value  = 0x00000000
        Watchdog disabled
```