The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV003-SW-42

VxWorks Device Driver

16 (8) Digital Inputs and 16 (8) Digital Outputs

Version 2.0.x

User Manual

Issue 2.0.0

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

TDRV003-SW-42

VxWorks Device Driver

16 (8) Digital Inputs and 16 (8) Digital Outputs

Supported Modules:
TPMC670
TPMC671

Issue	Description	Date
1.0.0	First Issue	October 14, 2005
1.0.1	New Address TEWS LLC, general revision	October 21, 2008
2.0.0	API documentation added	September 6, 2010

Table of Contents

1 Introduction

The TDRV003-SW-42 VxWorks device driver software allows the operation of the modules supported by TDRV003 conforming to the VxWorks I/O system specification.

The TDRV003-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) 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 TDRV003-SW-42 device driver supports the following features:

- read the actual input value
- write a new output value
- wait for selectable input events (match, high-, low-, any transition on the input line(s))
- enable and disable the output watchdog
- acknowledge watchdog errors
- configure, enable and disable input debouncing

The TDRV003-SW-42 supports the modules listed below:

TPMC670	16(8) Digital Input (24V) 16(8) Digital Output (24V, 0.5A) (50 pin connector)	PMC
TPMC671	16 Digital Input (24V) 16 Digital Output (24V, 0.5A) (64 pin connector)	PMC

In this document all supported modules and devices will be called TDRV003. Specials for certain devices will be advised.

To get more information about the features and use of TDRV003 devices, it is recommended to read the manuals listed below.

TPMC670/TPMC671 User manual
TPMC670/TPMC671 Engineering Manual

2 Installation

The following files are located on the distribution media:

Directory path 'TDRV003-SW-42':

TDRV003-SW-42-2.0.0.pdf	PDF copy of this manual
TDRV003-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TDRV003-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TDRV003-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tdrv003':

tdrv003drv.c	TDRV003 device driver source
tdrv003def.h	TDRV003 driver include file
tdrv003.h	TDRV003 include file for driver and application
tdrv003api.c	TDRV003 API file
Makefile	Driver Makefile
40tdrv003.cdf	Component description file for VxWorks development tools
tdrv003.dc	Configuration stub file for direct BSP builds
tdrv003.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tdrv003exa.c	Example application

The archive TDRV003-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tdrv003':

tdrv003drv.c	TDRV003 device driver source
tdrv003def.h	TDRV003 driver include file
tdrv003.h	TDRV003 include file for driver and application
tdrv003pci.c	TDRV003 device driver source for x86 based systems
tdrv003api.c	TDRV003 API file
tdrv003exa.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 TDRV003-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 TDRV003 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tdrv003.*

At this point the TDRV003 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/tdrv003

(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\tdrv003
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 TDRV003 driver with the VxWorks development tools (Workbench), the component configuration file *40tdrv003.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv003
C:> copy 40tdrv003.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 TDRV003 driver can be included in VxWorks projects by selecting the *"TEWS TDRV003 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 TDRV003 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\tdrv003
C:> copy tdrv003.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tdrv003.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 TDRV003-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 TDRV003-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 tdrv003 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 TDRV003 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 TDRV003 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tdrv003pci.c** contains the function *tdrv003PciInit().* This routine finds out all TDRV003 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 *tdrv003PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window):

```
tdrv003PciInit();
```

Be sure that the function is called prior to MMU initialization or otherwise the TDRV003 PCI spaces remain 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 ways 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.

2.3.4 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	~52 Bytes + (<max jobs> * ~12 Bytes)	~52 Bytes
Stack	< 200 Byte	---
Semaphores	0	<max jobs>

<max jobs> = *TDRV003_NUMJOBS*

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.

2.4 Special configurations

The value *TDRV003_NUMJOBS* (defined in *tdrv003def.h*) specifies the maximum number of jobs that can be handled at the same time for a device. System resources will be allocated while driver start to prevent extra time during normal application operation. This value may be changed to increase the maximum number of jobs or decreased to save system resources.

The driver must be build again after changing *TDRV003_NUMJOBS*.

3 API Documentation

3.1 General Functions

3.1.1 tdrv003Open()

Name

tdrv003Open() – opens a device.

Synopsis

```
TDRV003_DEV tdrv003Open
(
      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 TDRV003 device is named “/tdrv003/0”, the second device is named “/tdrv003/1” and so on.

Example

```
#include “tdrv003.h”

TDRV003_DEV   pDev;

/*
** open file descriptor to device
*/
pDev = tdrv003Open(“/tdrv003/0”);
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 tdrv003Close()

Name

tdrv003Close() – closes a device.

Synopsis

```
int tdrv003Close
(
     TDRV003_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 “tdrv003.h”

TDRV003_DEV   pDev;
int           result;

/*
** close file descriptor to device
*/
result = tdrv003Close(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 tdrv003InputRead

Name

tdrv003InputRead – read state of input lines

Synopsis

```
STATUS tdrv003InputRead
(
    TDRV003_DEV             pDev,
    unsigned short          *pInputBuf
)
```

Description

This function reads the current state of the input lines.

Parameters

pDev

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

pInputBuf

This argument points to a buffer where the value will be returned.

Example

```
#include “tdrv003.h”

TDRV003_DEV        pDev;
STATUS             result;
Unsigned short     in_value;

/*
** read current state of Input lines
*/
result = tdrv003InputRead(pDev, &in_value);
if (result == ERROR)
{
     /* handle error */
}
else
{
     printf(“value: 0x%04X\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. Function specific error codes will be described with the function.

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

3.2.2 tdrv003OutputRead

Name

tdrv003OutputRead – read state of output lines

Synopsis

```
STATUS tdrv003OutputRead
(
    TDRV003_DEV             pDev,
    unsigned short          *pOutputBuf
)
```

Description

This function reads the current state of the output lines.

Parameters

pDev

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

pOutputBuf

This argument points to a buffer where the value will be returned.

Example

```
#include “tdrv003.h”

TDRV003_DEV        pDev;
STATUS             result;
unsigned short     outputBuf;

/*
** read current state of Output lines
*/
result = tdrv003OutputRead(pDev, &outputBuf);
if (result == ERROR)
{
     /* handle error */
}
else
{
     printf(“value: 0x%04X\n”, outputBuf);
}
```

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
EINVAL	A NULL pointer is referenced for an output buffer
EBADF	The device handle is invalid

3.2.3 tdrv003OutputWrite

Name

tdrv003OutputWrite – set status of output lines

Synopsis

```
STATUS tdrv003OutputWrite
(
    TDRV003_DEV         pDev,
    unsigned short      OutputValue
)
```

Description

This function sets the output lines to the specified value.

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 new output value specifies the new output level of output line 1, bit 1 specifies the output level for output line 2 and so on.

Example

```
#include “tdrv003.h”

TDRV003_DEV        pDev;
STATUS             result;
unsigned short     outputValue;

/*
** Set status of Output lines
*/
result = tdrv003OutputWrite(pDev, outputValue);
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
ETIMEDOUT	The watchdog has timed out, all outputs set to OFF.

3.2.4 tdrv003OutputWriteMask

Name

tdrv003OutputWriteMask – set status of output lines with bitmask

Synopsis

```
STATUS tdrv003OutputWriteMask
(
    TDRV003_DEV             pDev,
    unsigned short          OutputValue,
    unsigned short          Mask
)
```

Description

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

Parameters

pDev

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

OutputValue

This parameter specifies a 16bit word which reflects the output lines. Bit 0 corresponds to the first output line, bit 1 corresponds to the seconds output line and so on.

mask

This parameter specifies a 16bit 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 seconds output line and so on.

Example

```
#include “tdrv003.h”

TDRV003_DEV        pDev;
STATUS             result;
unsigned short     OutputValue;
unsigned short     Mask;

/*-------------------------------------------------
  Set output line 1 and 8 (bit 0 and bit 7), and
  clear output line 16 (bit 15)
  -------------------------------------------------*/
OutputValue   = (1 <<  7) | (1 << 0);
Mask          = (1 << 15) | (1 << 7) | (1 << 0);

result = tdrv003OutputWriteMask(pDev, OutputValue, Mask);
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

3.2.5 tdrv003OutputSetBits

Name

tdrv003OutputSetBits – set specific output lines

Synopsis

```
STATUS tdrv003OutputSetBits
(
    TDRV003_DEV             pDev,
    unsigned short          OutputBits
)
```

Description

This function sets specified output lines to ‘1’.

Parameters

pDev

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

OutputBits

This parameter specifies the output lines to be set. Unset bits will be left unchanged for the output. Bit 0 corresponds to the first output line, bit 1 corresponds to the seconds output line and so on.

Example

```
#include “tdrv003.h”

TDRV003_DEV        pDev;
STATUS             result;
unsigned short     OutputBits;

/*-------------------------------------------------
  Set output lines 2, 3, and 16 (bits 1, 2 and 15)
  -------------------------------------------------*/
OutputBits = (1 << 15) | (1 << 2) | (1 << 1);

result = tdrv003OutputSetBits(pDev, OutputBits);
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
ETIMEDOUT	The watchdog has timed out, all outputs set to OFF.

3.2.6 tdrv003OutputClearBits

Name

tdrv003OutputClearBits – clear specific output lines

Synopsis

```
STATUS tdrv003OutputClearBits
(
    TDRV003_DEV             pDev,
    unsigned short          OutputBits
)
```

Description

This function clears specified output lines to ‘0’.

Parameters

pDev

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

OutputBits

This parameter specifies the output lines to be cleared. Unset bits will be left unchanged for the output. Bit 0 corresponds to the first output line, bit 1 corresponds to the seconds output line and so on.

Example

```
#include “tdrv003.h”

TDRV003_DEV        pDev;
STATUS             result;
unsigned short     OutputBits;

/*-------------------------------------------------
  Clear output lines 1 and 16 (bits 0 and 15)
  -------------------------------------------------*/
OutputBits = (1 << 15) | (1 << 0);

result = tdrv003OutputClearBits(pDev, OutputBits);
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
ETIMEDOUT	The watchdog has timed out, all outputs set to OFF.

3.2.7 tdrv003DebouncerEnable

Name

tdrv003DebouncerEnable – configure and enable debouncing circuit

Synopsis

```
STATUS tdrv003DebouncerEnable
(
    TDRV003_DEV             pDev,
    unsigned short          DebounceTimer
)
```

Description

This function configures and enables the debouncing circuit for the input lines.

Parameters

pDev

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

DebounceTimer

This parameter specifies the debouncer time. The debouncer time is specified in approx. 7µs steps. Please refer to the corresponding Hardware User Manual for a calculation formula and a table of values.

Example

```
#include “tdrv003.h”

TDRV003_DEV        pDev;
STATUS             result;

/*-------------------------------------
  Configure debouncer with delay of 1 ms
  -------------------------------------*/

result = tdrv003DebouncerEnable(pDev, 147);
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

3.2.8 tdrv003DebouncerDisable

Name

tdrv003DebouncerDisable – disable debouncing circuit

Synopsis

```
STATUS tdrv003DebouncerDisable
(
    TDRV003_DEV                 pDev
)
```

Description

This function disables the debouncing circuit for the input lines.

Parameters

pDev

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

Example

```
#include “tdrv003.h”

TDRV003_DEV        pDev;
STATUS             result;

/*-----------------
  Disable debouncer
  -----------------*/

result = tdrv003DebouncerDisable(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

3.2.9 tdrv003WatchdogEnable

Name

tdrv003WatchdogEnable – enable output watchdog

Synopsis

```
STATUS tdrv003WatchdogEnable
(
    TDRV003_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 *tdrv003WatchdogReset* 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 “tdrv003.h”

TDRV003_DEV        pDev;
STATUS             result;

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

result = tdrv003WatchdogEnable(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

3.2.10 tdrv003WatchdogDisable

Name

tdrv003WatchdogDisable – disable output watchdog

Synopsis

```
STATUS tdrv003WatchdogDisable
(
    TDRV003_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 “tdrv003.h”

TDRV003_DEV        pDev;
STATUS             result;

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

result = tdrv003WatchdogDisable(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

3.2.11 tdrv003WatchdogReset

Name

tdrv003WatchdogReset – reset output watchdog error

Synopsis

```
STATUS tdrv003WatchdogReset
(
    TDRV003_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 “tdrv003.h”

TDRV003_DEV        pDev;
STATUS             result;

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

result = tdrv003WatchdogReset(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

3.2.12 tdrv003EventWait

Name

tdrv003EventWait – wait for specific input event

Synopsis

```
STATUS tdrv003EventWait
(
    TDRV003_DEV         pDev
    unsigned long       EventMode,
    unsigned short      Mask,
    unsigned short      Match,
    long                timeout_ms
)
```

Description

This function waits for an input event.

Parameters

pDev

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

EventMode

This parameter specifies the kind of event to wait for. The following event types are defined in tdrv003.h:

Event type	Description
TDRV003_MATCH	Wait for a match event. This function will return if the specified bits match to the states at the input lines.
TDRV003_HIGH_TR	Wait for a low to high transition on a specified input pin.
TDRV003_LOW_TR	Wait for a high to high transition on a specified input pin.
TDRV003_ANY_TR	Wait for any transition on a specified input pin.

Mask

This parameter specifies the input pin(s) which shall be used for the event.
If the function shall wait for a match event only the marked lines will be compared.
If the function shall wait for a transition the mask specifies the input line the transition shall occur on. Only one bit shall be set in the mask, otherwise the occurred event cannot be determined exactly.
Bit 0 of the mask corresponds to the first input line, bit 1 corresponds to the second input line and so on.

Match

This parameter specifies the match value for match events. This parameter is only used for match events, for transition events this value is ignored.
Bit 0 corresponds to the first input line, bit 1 corresponds to the second input line and so on.

timeout_ms

This parameter specifies the maximum time to wait for the event. If this time is exceeded, the function will return with an error. The timeout time is specified in 1 ms steps. If the timeout value is specified 0 the function is used with no wait option. If the timeout value is specified with a negative value the function will wait forever (or until the event occurs).

Remember interrupt latency:

***TDRV003_MATCH* may miss very short events, because match check is made in the ISR. Therefore it is recommended not to use this event for fast changing input signals.**

Example

```
#include "tdrv003.h"

TDRV003_DEV        pDev;
STATUS             result;

/*---------------------------------------
  Wait until input line 1, 2 & 4 are high
  and input line 3 is low
  ---------------------------------------*/

result = tdrv003EventWait(pDev,
                  TDRV003_MATCH,
                  0x000F,       /* check bit 0..3           */
                  0x000B,       /* bit 0, 1, 3 must be set  */
                  1000,         /* Wait 1 second            */
                  );
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function succeeded */
}

…
```

```
/*-------------------------------------------------
  Wait for a high to low transition on input line 10
  -------------------------------------------------*/
result = tdrv003EventWait(pDev,
                  TDRV003_LOW_TR,
                  (1 << 9),      /* trans on bit 9 <- input line 10    */
                  0              /* unused                             */
                  -1             /* Wait forever                       */
                  );
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
ETIMEDOUT	The event has not occurred in the specified time. The wait timed out.

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 TDRV003 driver. For the VxBus-enabled TDRV003 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.

4.1 tdrv003Drv()

NAME

tdrv003Drv() - installs the TDRV003 driver in the I/O system

SYNOPSIS

#include “tdrv003.h”

STATUS tdrv003Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus, installs the TDRV003 driver in the I/O system.

The call of this function is the first thing the user has to do before adding any device to the system or performing any I/O request.

EXAMPLE

```
#include "tdrv003.h”

/*-------------------
  Initialize Driver
  -------------------*/
status = tdrv003Drv();
if (status == 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()*.

ENXIO	There was no module found supported by this driver

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.2 tdrv003DevCreate()

NAME

tdrv003DevCreate() – Add a TDRV003 device to the VxWorks system

SYNOPSIS

#include “tdrv003.h”

```
STATUS tdrv003DevCreate
(
    char        *name,
    int         devIdx,
    int         funcType,
    void        *pParam
)
```

DESCRIPTION

This routine adds the selected device to the VxWorks system. The device hardware will be setup and prepared for use.

This function must be called before performing any I/O request to this device.

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

devIdx

This index number specifies the device to add to the system.
The index number depends on the search priority of the modules. The modules will be searched in the following order:

- TPMC670-xx
- TPMC671-xx

If modules of the same type are installed, the index numbers will be assigned in the order the VxWorks *pciFindDevice()* function will find the devices.

Example: A system with 1 TPMC670-xx, and 2 TPMC671-xx will assign the following device indexes:

Module	Device Index
TPMC670-xx	0
TPMC671-xx (1st)	1
TPMC671-xx (2nd)	2

funcType

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

pParam

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

EXAMPLE

```
#include "tdrv003.h"

STATUS            result;

/*-----------------------------------------------------
  Create the device "/tdrv003/0" for the first device
  -----------------------------------------------------*/
result = tdrv003DevCreate(   "/tdrv003/0",
                             0,
                             0,
                             NULL);
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

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

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

S_ioLib_NO_DRIVER	Driver has not been installed
ENXIO	Specified device number is not available

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.3 tdrv003PciInit()

NAME

tdrv003PciInit() – Generic PCI device initialization

SYNOPSIS

void tdrv003PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TDRV003 PCI spaces (base address register) and to enable the TDRV003 device for access.

The global variable *tdrv003Status* 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 successfully completed. The value of tdrv003Status is equal to the number of mapped PCI spaces
0	No TDRV003 device found
< 0	Initialization failed. The value of (tdrv003Status & 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 tdrv003Init();

…

tdrv003PciInit();

…
```

4.4 tdrv003Init()

NAME

tdrv003Init() – initialize TDRV003 driver and devices

SYNOPSIS

#include “tdrv003.h”

STATUS tdrv003Init(void)

DESCRIPTION

This function is used by the TDRV003 example application to install the driver and to add all available devices to the VxWorks system.

See also 3.1.1 tdrv003Open() for the device naming convention for legacy devices.

After calling this function it is not necessary to call tdrv003Drv() and tdrv003DevCreate() explicitly.

EXAMPLE

```
#include "tdrv003.h”

STATUS    result;


result = tdrv003Init();

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

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 TDRV003 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 tdrv003DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int  fd;

/*----------------------------------------
  Open the device named "/tdrv003/0" for I/O
  ----------------------------------------*/
fd = open("/tdrv003/0", 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 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).

SEE ALSO

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

5.2 close()

NAME

close() – close a device or file

SYNOPSIS

```
int 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;
int  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 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).

SEE ALSO

ioLib, basic I/O routine - close()

5.3 ioctl()

NAME

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

SYNOPSIS

#include “tdrv003.h”

```
int ioctl
(
        int     fd,
        int     request,
        int     arg
)
```

DESCRIPTION

Special I/O operations 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_TDRV003_READ	Read actual state of input lines
FIO_TDRV003_WRITE	Set new output value
FIO_TDRV003_DEBENABLE	Enable and configure debouncer
FIO_TDRV003_DEBDISABLE	Disable debouncer
FIO_TDRV003_WDENABLE	Enable and configure watchdog timer
FIO_TDRV003_WDDISABLE	Disable watchdog timer
FIO_TDRV003_WDRESET	Reset watchdog error
FIO_TDRV003_EVWAIT	Wait for specified input event
FIO_TDRV003_OUTPUTGET	Read back output value
FIO_TDRV003_OUTPUTSETBITS	Set specific output lines
FIO_TDRV003_OUTPUTCLEARBITS	Clear specific output lines
FIO_TDRV003_WRITEMASK	Write output value in conjunction with bitmask

arg

This parameter depends on the selected function (request). How to use this parameter is described below with the function.

RETURNS

Function dependent value (described with the function) 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()*.

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.

ENOTSUP	Function code is not supported

SEE ALSO

ioLib, basic I/O routine - ioctl()

5.3.1 FIO_TDRV003_READ

This I/O control function reads actual state of the input lines. The function specific control parameter **arg** is a pointer on an unsigned short buffer where the input value will be returned in. Input Line 1 will be represented by bit 0 of the input value, Input Line 2 will be represented by bit 1 and so on.

EXAMPLE

```
#include “tdrv003.h”

int                fd;
unsigned short     inputBuf;
unsigned long      retval;

/*-------------------------
  Read value of input lines
  -------------------------*/
retval = ioctl(fd, FIO_TDRV003_READ, (int)&inputBuf);
if (retval != ERROR)
{
     /* function succeeded */
     printf(“Input lines: %04Xh\n”, inputBuf);
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

5.3.2 FIO_TDRV003_WRITE

This I/O control function sets the output value. The function specific control parameter **arg** is a pointer to the new output value. Bit 0 of the new output value specifies the new output level of output line 1, bit 1 specifies the output level for output line 2 and so on.

EXAMPLE

```
#include “tdrv003.h”

int                fd;
unsigned short     outputBuf;
unsigned long      retval;

/*-------------------------
  Set output lines 4 and 13
  -------------------------*/
outputBuf = 0x1008;

retval = ioctl(fd, FIO_TDRV003_WRITE, (int)&outputBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

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.

ETIMEDOUT	This error indicates that the watchdog has timed out.

5.3.3 FIO_TDRV003_DEBENABLE

This I/O control function configures and enable input debouncer. The function specific control parameter **arg** is a pointer to the value specifying the new debouncer time. The debouncer time is specified in 7µs steps (see Hardware Manual).

EXAMPLE

```
#include “tdrv003.h”

int                fd;
unsigned short     debBuf;
unsigned long      retval;

/*-------------------------------------
  Configure debouncer with delay of 1 ms
  -------------------------------------*/
debBuf = 147;

retval = ioctl(fd, FIO_TDRV003_DEBENABLE, (int)&debBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

5.3.4 FIO_TDRV003_DEBDISABLE

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

EXAMPLE

```
#include “tdrv003.h”

int                fd;
unsigned long      retval;

/*-----------------
  Disable debouncer
  -----------------*/
retval = ioctl(fd, FIO_TDRV003_DEBDISABLE, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

5.3.5 FIO_TDRV003_WDENABLE

This I/O control function enables the output watchdog. 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 *FIO_TDRV003_WDRESET* must be executed. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tdrv003.h”

int                fd;
unsigned long      retval;

/*----------------------
  Enable ouput watchdog
  ----------------------*/
retval = ioctl(fd, FIO_TDRV003_WDENABLE, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

5.3.6 FIO_TDRV003_WDDISABLE

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

EXAMPLE

```
#include “tdrv003.h”

int                fd;
unsigned long      retval;

/*------------------------
  Disable output watchdog
  ------------------------*/
retval = ioctl(fd, FIO_TDRV003_WDDISABLE, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

5.3.7 FIO_TDRV003_WDRESET

This I/O control function resets the watchdog state. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tdrv003.h”

int                fd;
unsigned long      retval;

/*--------------------------
  Reset watchdog error state
  --------------------------*/
retval = ioctl(fd, FIO_TDRV003_WDRESET, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

5.3.8 FIO_TDRV003_EVWAIT

This I/O control function waits for an input event. The function specific control parameter **arg** is a pointer on a *TDRV003_EVREAD_STRUCT* structure.

```
typedef struct
{
        unsigned long           mode;
        unsigned short          mask;
        unsigned short          match;
        long                    timeout;
} TDRV003_EVREAD_STRUCT;
```

mode

This parameter specifies the kind of event to wait for. The following event types are defined in tdrv003.h:

Event type	Description
TDRV003_MATCH	Wait for a match event. This function will return if the specified bits match to the states at the input lines.
TDRV003_HIGH_TR	Wait for a low to high transition on a specified input pin.
TDRV003_LOW_TR	Wait for a high to high transition on a specified input pin.
TDRV003_ANY_TR	Wait for any transition on a specified input pin.

mask

This parameter specifies the input pin(s) which shall be used for the event.
If the function shall wait for a match event only the marked lines will be compared.
If the function shall wait for a transition the mask specifies the input line the transition shall occur on. Only one bit shall be set in the mask, otherwise the occurred event cannot be determined exactly.
Bit 0 of the mask corresponds to the first input line, bit 1 corresponds to the second input line and so on.

match

This parameter specifies the match value for match events. This parameter is only used for match events, for transition events this value is ignored.
Bit 0 corresponds to the first input line, bit 1 corresponds to the second input line and so on.

timeout

This parameter specifies the maximum time to wait for the event. If this time is exceeded, the function will return with an error. The timeout time is specified in 1 ms steps. If the timeout value is specified 0 the function is used with no wait option. If the timeout value is specified with a negative value the function will wait forever (or until the event occurs).

Remember interrupt latency:

***TDRV003_MATCH* may miss very short events, because match check is made in the ISR. Therefore it is recommended not to use this event for fast changing input signals.**

EXAMPLE

```
#include “tdrv003.h”

int                     fd;
TDRV003_EVREAD_STRUCT   eventBuf;
unsigned long           retval;

/*--------------------------------------
  Wait until input line 1, 2 & 4 are high
  and input line 3 is low
  --------------------------------------*/
eventBuf.mode = TDRV003_MATCH;
eventBuf.mask = 0x000F;     /* check bit 0..3 */
eventBuf.match = 0x000B;    /* bit 0, 1, 3 must be set */
eventBuf.timeout = 1000;    /* Wait 1 second */

retval = ioctl(fd, FIO_TDRV003_EVWAIT, (int)&eventBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}

/*-------------------------------------------------
  Wait for a high to low transition on input line 10
  -------------------------------------------------*/
eventBuf.mode = TDRV003_LOW_TR;
eventBuf.mask = (1 << 9);        /* trans on bit 9 <- input line 10 */
eventBuf.timeout = WAIT_FOREVER; /* Wait forever for this event */

retval = ioctl(fd, FIO_TDRV003_EVWAIT, (int)&eventBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

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.

ENOTSUP	The specified event is invalid
ETIMEDOUT	The event has not occurred in the specified time. The wait timed out.

5.3.9 FIO_TDRV003_OUTPUTGET

This I/O control function reads back the current output value. The function specific control parameter **arg** is a pointer on an unsigned short buffer where the output value will be returned. Bit 0 corresponds to the first output line, bit 1 corresponds to the second output line and so on.

EXAMPLE

```
#include “tdrv003.h”

int                fd;
unsigned short     outputBuf;
unsigned long      retval;

/*--------------------------------------
  Read current state of output lines
  --------------------------------------*/
retval = ioctl(fd, FIO_TDRV003_OUTPUTGET, (int)&outputBuf);
if (retval != ERROR)
{
     /* function succeeded */
     printf(“Output lines: %04Xh\n”, outputBuf);
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

5.3.10 FIO_TDRV003_OUTPUTSETBITS

This I/O control function sets specific bits of the output value (set to 1). The function specific control parameter **arg** is a pointer to *unsigned short* buffer. Bit 0 corresponds to the first output line, bit 1 corresponds to the second output line and so on. Unset bits will be left unchanged for the output.

EXAMPLE

```
#include “tdrv003.h”

int                fd;
unsigned short     OutputBits;
unsigned long      retval;

/*-------------------------------------------------
  Set output lines 2, 3, and 16 (bits 1, 2 and 15)
  -------------------------------------------------*/
OutputBits = (1 << 15) | (1 << 2) | (1 << 1);

retval = ioctl(fd, FIO_TDRV003_OUTPUTSETBITS, (int)&OutputBits);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

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.

ETIMEDOUT	This error indicates that the watchdog has timed out.

5.3.11 FIO_TDRV003_OUTPUTCLEARBITS

This I/O control function clears some bits of the output value (set to 0). The function specific control parameter **arg** is a pointer to the new output value. Bit 0 corresponds to the first output line, bit 1 corresponds to the second output line and so on. Unset bits will be left unchanged for the output.

EXAMPLE

```
#include "tdrv003.h"

int                fd;
unsigned short     OutputBits;
unsigned long      retval;

/*-------------------------------------------------
  Clear output lines 1 and 16 (bits 0 and 15)
  -------------------------------------------------*/
OutputBits = (1 << 15) | (1 << 0);

retval = ioctl(fd, FIO_TDRV003_OUTPUTCLEARBITS, (int)&OutputBits);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

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.

ETIMEDOUT	This error indicates that the watchdog has timed out.

5.3.12 FIO_TDRV003_WRITEMASK

This I/O control function writes the output value in conjunction with a bitmask. Only specified bits will be changed. The function specific control parameter **arg** is a pointer to a *TDRV003_WRITEBUF* structure.

```
typedef struct
{
        unsigned short          value;
        unsigned short          mask;
} TDRV003_WRITEBUF;
```

value

This parameter specifies a 16bit word which reflects the output lines. Bit 0 corresponds to the first output line, bit 1 corresponds to the seconds output line and so on.

mask

This parameter specifies a 16bit mask. '1' means that the corresponding bit in *value* will be updated. '0' bits will be left unchanged. Bit 0 corresponds to the first output line, bit 1 corresponds to the seconds output line and so on.

EXAMPLE

```
#include "tdrv003.h"

int                 fd;
TDRV003_WRITEBUF    WriteBuf;
unsigned long       retval;

/*-------------------------------------------------
  Set output line 1 and 8 (bit 0 and bit 7), and
  clear output line 16 (bit 15)
  -------------------------------------------------*/
WriteBuf.value = (1 <<  7) | (1 << 0);
WriteBuf.mask  = (1 << 15) | (1 << 7) | (1 << 0);

retval = ioctl(fd, FIO_TDRV003_WRITEMASK, (int)&WriteBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

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.

ETIMEDOUT	This error indicates that the watchdog has timed out.