The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV006-SW-42

VxWorks Device Driver

64 Digital Inputs/Outputs (Bit I/O)

Version 2.0.x

User Manual

Issue 2.0.1

March 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

TDRV006-SW-42

VxWorks Device Driver

64 Digital Inputs/Outputs (Bit I/O)

Supported Modules:
TPMC681

Issue	Description	Date
1.0.0	First Issue	July 7, 2006
1.0.1	New Address TEWS LLC, ChangeLog.txt added in file list	December 3, 2006
2.0.0	New ioctl() function: FIO_TDRV006_WRITE_MASKED, description for API and VxBus-support added, Address TEWS LLC removed	January 26, 2010
2.0.1	Legacy vs. VxBus Driver modified	March 26, 2010

Table of Contents

1 Introduction

The TDRV006-SW-42 VxWorks device driver software allows the operation of the TPMC681 PMC conforming to the VxWorks I/O system specification.

The TDRV006-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 TDRV006-SW-42 device driver supports the following features:

- configure direction of I/O lines
- set output value of output lines
- read value of I/O lines
- wait for input line events

The TDRV006-SW-42 supports the modules listed below:

TPMC681	64 bit digital I/O	PMC

In this document all supported modules and devices will be called TDRV006. Specials for certain devices will be advised.

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

TPMC681 User manual

TPMC681 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV006-SW-42':

TDRV006-SW-42-2.0.1.pdf	PDF copy of this manual
TDRV006-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TDRV006-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TDRV006-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tdrv006':

tdrv006drv.c	TDRV006 device driver source
tdrv006def.h	TDRV006 driver include file
tdrv006.h	TDRV006 include file for driver and application
tdrv006api.c	TDRV006 API file
Makefile	Driver Makefile
40tdrv006.cdf	Component description file for VxWorks development tools
tdrv006.dc	Configuration stub file for direct BSP builds
tdrv006.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tdrv006exa.c	Example application

The archive TDRV006-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tdrv006':

tdrv006drv.c	TDRV006 device driver source
tdrv006def.h	TDRV006 driver include file
tdrv006.h	TDRV006 include file for driver and application
tdrv006pci.c	TDRV006 device driver source for x86 based systems
tdrv006api.c	TDRV006 API file
tdrv006exa.c	Example application
tdrv006init.c	Legacy driver initialization
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 TDRV006-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 TDRV006 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tdrv006.*

At this point the TDRV006 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/tdrv006

(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\tdrv006
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 TDRV006 driver with the VxWorks development tools (Workbench), the component configuration file *40tdrv006.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv006
C:> copy 40tdrv006.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 TDRV006 driver can be included in VxWorks projects by selecting the *“TEWS TDRV006 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 TDRV006 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\tdrv006
C:> copy tdrv006.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tdrv006.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 TDRV006-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 TDRV006-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 tdrv006 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 TDRV006 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 TDRV006 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tdrv006pci.c** contains the function *tdrv006PciInit().* This routine finds out all TDRV006 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 *tdrv006PciInit()* 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 TDRV006 PCI spaces remains unmapped and an access fault occurs during driver initialization.

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

```
tdrv006PciInit();
```

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

2.3.3 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	0	up to 64

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 tdrv006Open()

Name

tdrv006Open() – opens a device.

Synopsis

```
TDRV006_DEV tdrv006Open
(
    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 TDRV006 device is named “/tdrv006/0”, the second device is named “/tdrv006/1” and so on.

Example

```
#include “tdrv006.h”

TDRV006_DEV     pDev;

/*
** open file descriptor to device
*/
pDev = tdrv006Open(“/tdrv006/0”);
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 tdrv006Close()

Name

tdrv006Close() – closes a device.

Synopsis

```
int tdrv006Close
(
        TDRV006_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 “tdrv006.h”

TDRV006_DEV    pDev;
int            result;

/*
** close file descriptor to device
*/
result = tdrv006Close(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 tdrv006Read

Name

tdrv006Read – read current input value of the I/O lines

Synopsis

```
STATUS tdrv006Read
(
        TDRV006_DEV    pDev,
        UINT32         *in31_0,
        UINT32         *in63_32
)
```

Description

This function reads the current input value of the I/O lines.

Parameters

pDev

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

in31_0

This argument points to a buffer where the current value of I/O lines 0 up to 31 will be returned. Bit 0 returns the value of I/O line 0, bit 1 the value of I/O line 1, and so on.

in63_32

This argument points to a buffer where the current value of I/O lines 32 up to 63 will be returned. Bit 0 returns the value of I/O line 32, bit 1 the value of I/O line 33, and so on.

Example

```
#include “tdrv006.h”

TDRV006_DEV         pDev;
STATUS              result;
UINT32              in_low;
UINT32              in_high;

/*
** read current state of I/O lines
*/
result = tdrv006Read(pDev, &in_low, &in_high);
if (result == ERROR)
{
     /* handle error */
}
else
{
     printf(“INPUT: 0x%08X%08X\n”, in_high, in_low);
}
```

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 tdrv006Write

Name

tdrv006Write – set output value

Synopsis

```
STATUS tdrv006Write
(
        TDRV006_DEV     pDev,
        UINT32          out31_0,
        UINT32          out63_32
)
```

Description

This function sets the output value.

The specified value will only appear on the I/O lines which are configured for output.

Parameters

pDev

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

out31_0

This argument specifies the output value for I/O lines 0 up to 31. Bit 0 specifies the value of I/O line 0, bit 1 the value of I/O line 1, and so on.

out63_32

This argument specifies the output value for I/O lines 32 up to 63. Bit 0 specifies the value of I/O line 32, bit 1 the value of I/O line 33, and so on.

Example

```
#include "tdrv006.h"

TDRV006_DEV         pDev;
STATUS              result;

/*
** Set output value (set I/O lines 0-15)
*/
result = tdrv006Write(pDev, 0x0000FFFF, 0x00000000);
if (result == ERROR)
{
     /* error handling */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

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

3.2.3 tdrv006WriteMasked

Name

tdrv006WriteMasked – set output value for specified I/O lines

Synopsis

```
STATUS tdrv006WriteMasked
(
      TDRV006_DEV    pDev,
      UINT32         out31_0,
      UINT32         out63_32,
      UINT32         mask31_0,
      UINT32         maks63_32
)
```

Description

This function sets the output value for specified I/O lines. The mask specifies which I/O bits shall be set to the specified output value and which shall keep the current value.

This specified value will only appear on the I/O lines which are configured for output.

Parameters

pDev

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

out31_0

This argument specifies the output value for I/O lines 0 up to 31. Bit 0 specifies the value of I/O line 0, bit 1 the value of I/O line 1, and so on.

out63_32

This argument specifies the output value for I/O lines 32 up to 63. Bit 0 specifies the value of I/O line 32, bit 1 the value of I/O line 33, and so on.

mask31_0

This argument specifies the output mask for output lines 0 up to 31. Bit 0 specifies the mask for I/O line 0, bit 1 the value for I/O line 1, and so on.
A set bit (1) means the bit shall be set to the value specified by *out_31_0*.
A reset bit (0) means that the old output value will not be changed.

mask63_32

This argument specifies the output mask for output lines 32 up to 63. Bit 0 specifies the mask for I/O line 32, bit 1 the value for I/O line 33, and so on.
A set bit (1) means the bit shall be set to the value specified by *out_63_32.*
A reset bit (0) means that the old output value will not be changed.

Example

```
#include “tdrv006.h”

TDRV006_DEV        pDev;
STATUS             result;

/*
** Set a part of the output value (set/reset I/O lines 0-15 and 48-63)
*/
result = tdrv006WriteMasked(pDev,
                            0x12345678, 0x87654321,
                            0x0000FFFF, 0xFFFF0000);
if (result == ERROR)
{
     /* error handling */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

Error code	Description
EBADF	The device handle is invalid

3.2.4 tdrv006SetOutputLine

Name

tdrv006SetOutputLine – set a specified output line

Synopsis

```
STATUS tdrv006SetOutputLine
(
        TDRV006_DEV     pDev,
        int             outputLine
)
```

Description

This function sets a single bit of the output value.

This specified value will only appear if the corresponding I/O line is configured for output.

pDev

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

outputLine

This argument specifies a data bit that shall be set. Allowed values are 0 up to 63.

Example

```
#include “tdrv006.h”

TDRV006_DEV         pDev;
STATUS              result;

/*
** Set I/O line 32
*/
result = tdrv006SetOutputLine(pDev, 32);
if (result == ERROR)
{
    /* error handling */
}
```

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	An invalid line number is specified
EBADF	The device handle is invalid

3.2.5 tdrv006ClearOutputLine

Name

tdrv006ClearOutputLine – reset a specified I/O line

Synopsis

```
STATUS tdrv006ClearOutputLine
(
    TDRV006_DEV    pDev,
    int            outputLine
)
```

Description

This function resets a single bit of output value.

This specified value will only appear if the corresponding I/O line is configured for output.

pDev

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

outputLine

This argument specifies data bit that shall be reset. Allowed values are 0 up to 63.

Example

```
#include “tdrv006.h”

TDRV006_DEV         pDev;
STATUS              result;

/*
** Clear I/O line 32
*/
result = tdrv006ClearOutputLine(pDev, 32);
if (result == ERROR)
{
    /* error handling */
}
```

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	An invalid line number is specified
EBADF	The device handle is invalid

3.2.6 tdrv006OutputEnable

Name

tdrv006OutputEnable – set the I/O line direction

Synopsis

```
STATUS tdrv006OutputEnable
(
        TDRV006_DEV    pDev,
        UINT32         enaout31_0,
        UINT32         enaout63_32
)
```

Description

This function sets the I/O line direction. The value specifies which I/O lines shall be configured for output and which I/O lines should be used for input.

Parameters

pDev

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

enaout31_0

This argument specifies the direction of I/O lines 0 up to 31. Bit 0 specifies the direction of I/O line 0, bit 1 the direction of I/O line 1, and so on. A set bit (1) configures the line for output, an unset bit (0) configures input (tri-state).

enaout63_32

This argument specifies the direction of I/O lines 32 up to 63. Bit 0 specifies the direction of I/O line 32, bit 1 the direction of I/O line 33, and so on. A set bit (1) configures the line for output, an unset bit (0) configures input (tri-state).

Example

```
#include “tdrv006.h”

TDRV006_DEV         pDev;
STATUS              result;

/*
** Enable I/O lines 0-8 for ouput
*/
result = tdrv006OutputEnable(pDev, 0x000001FF, 0x00000000);
if (result == ERROR)
{
     /* error handling */
}
```

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

3.2.7 tdrv006WaitForLowToHigh

Name

tdrv006WaitForLowToHigh – wait until a low to high transition occurs

Synopsis

```
STATUS tdrv006WaitForLowToHigh
(
        TDRV006_DEV    pDev,
        int            inputLine,
        int            timeout
)
```

Description

This function waits until a low to high transition occurs on the specified input line or the specified timeout time has passed.

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 which shall be observed for a low to high transition. Allowed values are 0 up to 63.

timeout

This argument specifies the time the function is willing to wait for the specified transition. If the specified time has passed the function will return with an error. The timeout is specified in ticks.

Example

```
#include “tdrv006.h”

TDRV006_DEV        pDev;
STATUS             result;

/*
** Wait for a low-to-high transition on input line 0
** Timeout after 600 ticks (10 seconds)
*/
result = tdrv006WaitForLowToHigh (pDev, 0, 600);
if (result == ERROR)
{
     /* error handling */
}
```

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	Invalid parameter specified
EBUSY	Another task is already waiting for a transition of the specified input line
EBADF	The device handle is invalid
S_objLib_OBJ_TIMEOUT	Timeout occurred.

3.2.8 tdrv006WaitForHighToLow

Name

tdrv006WaitForHighToLow – wait until a high to low transition occurs

Synopsis

```
STATUS tdrv006WaitHighToLow
(
        TDRV006_DEV    pDev,
        int            inputLine,
        int            timeout
)
```

Description

This function waits until a high to low transition occurs on the specified input line or the specified timeout time has passed.

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 which shall be observed for a high to low transition. Allowed values are 0 up to 63.

timeout

This argument specifies the time the function is willing to wait for the specified transition. If the specified time has passed the function will return with an error. This time is specified in ticks.

Example

```
#include “tdrv006.h”

TDRV006_DEV         pDev;
STATUS              result;

/*
** Wait for a high-to-low transition on input line 0
** Timeout after 600 ticks (10 seconds)
*/
result = tdrv006WaitForHighToLow (pDev, 0, 600);
if (result == ERROR)
{
     /* error handling */
}
```

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	Invalid parameter specified
EBUSY	Another task is already waiting for a transition of the specified input line
EBADF	The device handle is invalid
S_objLib_OBJ_TIMEOUT	Timeout occurred.

3.2.9 tdrv006WaitForAnyTrans

Name

tdrv006WaitForAnyTrans – wait until a transition occurs

Synopsis

```
STATUS tdrv006ForAnyTrans
(
        TDRV006_DEV    pDev,
        int            inputLine,
        int            timeout
)
```

Description

This function waits until a transition (high to low or low to high) occurs on the specified input line or the specified timeout time has passed.

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 which shall be observed for a transition. Allowed values are 0 up to 63.

timeout

This argument specifies the time the function is willing to wait for the specified transition. If the specified time has passed the function will return with an error. This time is specified in ticks.

Example

```
#include “tdrv006.h”

TDRV006_DEV         pDev;
STATUS              result;

/*
** Wait for a transition on input line 0
** Timeout after 600 ticks (10 seconds)
*/
result = tdrv006ForAnyTrans (pDev, 0, 600);
if (result == ERROR)
{
     /* error handling */
}
```

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	Invalid parameter specified
EBUSY	Another task is already waiting for a transition of the specified input line
EBADF	The device handle is invalid
S_objLib_OBJ_TIMEOUT	Timeout occurred.

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 TDRV006 driver. For the VxBus-enabled TDRV006 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.

4.1 tdrv006Drv()

NAME

tdrv006Drv() - installs the TDRV006 driver in the I/O system

SYNOPSIS

#include “tdrv006.h”

STATUS tdrv006Drv(void)

DESCRIPTION

This function searches for supported devices on the PCI bus, installs the TDRV006 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  "tdrv006.h”

STATUS             result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tdrv006Drv();
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**
ENOBUFS	Can't allocate memory buffers
ENXIO	No TDRV006 device found

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.2 tdrv006DevCreate()

NAME

tdrv006DevCreate() – Add a TDRV006 device to the VxWorks system

SYNOPSIS

```
#include "tdrv006.h"

STATUS tdrv006DevCreate
(
        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.

funcType

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

pParam

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

EXAMPLE

```
#include  "tdrv006.h"

STATUS              result;

/*-------------------------------------------------------
  Create the device "/tdrv006/0" for the first I/O device
  -------------------------------------------------------*/
result = tdrv006DevCreate(   "/tdrv006/0",
                             0,
                             0,
                             0);
if (result != OK)
{
     /* 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
ENXIO	No matching device found
EBUSY	Device has already been created

SEE ALSO

VxWorks Programmer’s Guide: I/O System

4.3 tdrv006PciInit()

NAME

tdrv006PciInit() – Generic PCI device initialization

SYNOPSIS

void tdrv006PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TDRV006 family PCI spaces (base address register) and to enable the TDRV006 device for access.

The global variable *tdrv006Status* 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 tdrv006Status is equal to the number of mapped PCI spaces
0	No TDRV006 device found
< 0	Initialization failed. The value of (tdrv006Status & 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 tdrv006PciInit();

tdrv006PciInit();
```

5 Basic I/O Functions

The VxWorks basic I/O interface functions are useable with the TDRV006 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 TDRV006 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 tdrv006DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int       fd;

/*------------------------------------------
  Open the device named "/tdrv006/0" for I/O
  ------------------------------------------*/
fd = open("/tdrv006/0", 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 “tdrv006.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_TDRV006_READ	Read value of I/O lines
FIO_TDRV006_WRITE	Write value of output lines
FIO_TDRV006_WRITE_MASKED	Write value of specified output lines
FIO_TDRV006_OUTPUT_ENABLE	Enable output
FIO_TDRV006_EVENT_WAIT	Wait for I/O transition events

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**
ENOSYS	Invalid request value specified

SEE ALSO

ioLib, basic I/O routine - ioctl()

5.3.1 FIO_TDRV006_READ

This I/O control function reads the value of the I/O lines. The function specific control parameter *arg* is a pointer on a *TDRV006_64BITBUFFER* structure.

```
typedef struct
{
        UINT32              val_31_0;
        UINT32              val_63_32;
} TDRV006_64BITBUFFER;
```

val_31_0

This argument returns the value of I/O lines 0 up to 31. Bit 0 returns the value of I/O line 0, bit 1 the value of I/O line 1, and so on.

val_63_32

This argument returns the value of I/O lines 32 up to 63. Bit 0 returns the value of I/O line 32, bit 1 the value of I/O line 33, and so on.

EXAMPLE

```
#include “tdrv006.h”

int                     fd;
TDRV006_64BITBUFFER     rBuf;
int                     retval;

/*--------------
  read I/O value
  --------------*/
retval = ioctl(fd, FIO_TDRV006_READ, (int)&rBuf);
if (retval != ERROR)
{
     /* function succeeded */
     printf(“value: %08lX %08lXh\n”, rBuf.val_63_32, rBuf.val_31_0);
}
else
{
     /* handle the error */
}
```

5.3.2 FIO_TDRV006_WRITE

This I/O control function sets the value of the output lines. The function specific control parameter *arg* is a pointer on a *TDRV006_64BITBUFFER* structure.

```
typedef struct
{
        UINT32          val_31_0;
        UINT32          val_63_32;
} TDRV006_64BITBUFFER;
```

val_31_0

This argument specifies the output value of I/O lines 0 up to 31. Bit 0 specifies the value of I/O line 0, bit 1 the value of I/O line 1, and so on.

val_63_32

This argument specifies the output value of I/O lines 32 up to 63. Bit 0 specifies the value of I/O line 32, bit 1 the value of I/O line 33, and so on.

EXAMPLE

```
#include “tdrv006.h”

int                     fd;
TDRV006_64BITBUFFER     wBuf;
int                     retval;

/*--------------------------
  set I/O line 0-7 and 56-61
  --------------------------*/
wBuf.val_31_0   = 0x000000FF;
wBuf.val_63_32  = 0x3F000000;
retval = ioctl(fd, FIO_TDRV006_WRITE, (int)&wBuf);
if (retval == ERROR)
{
     /* handle the error */
}
```

5.3.3 FIO_TDRV006_WRITE_MASKED

This I/O control function sets the value of specified output lines. The function specific control parameter *arg* is a pointer on a *TDRV006_64BITMASKBUFFER* structure.

```
typedef struct
{
          UINT32              val_31_0;
          UINT32              val_63_32;
          UINT32              mask_31_0;
          UINT32              mask_63_32;
} TDRV006_64BITMASKBUFFER;
```

val_31_0

This argument specifies the output value of output lines 0 up to 31. Bit 0 specifies the value of I/O line 0, bit 1 the value of I/O line 1, and so on.

val_63_32

This argument specifies the output value of output lines 32 up to 63. Bit 0 specifies the value of I/O line 32, bit 1 the value of I/O line 33, and so on.

mask_31_0

This argument specifies the output mask for output lines 0 up to 31. Bit 0 specifies the mask for I/O line 0, bit 1 the value of I/O line 1, and so on.
A set bit means the bit shall be set to the value specified by *val_31_0*.
A reset bit means that the old output value will not be changed.

mask_63_32

This argument specifies the output mask for output lines 32 up to 63. Bit 0 specifies the mask for I/O line 32, bit 1 the value of I/O line 33, and so on.
A set bit means the bit shall be set to the value specified by *val_63_32*.
A reset bit means that the old output value will not be changed.

EXAMPLE

```
#include “tdrv006.h”

int                         fd;
TDRV006_64BITMASKBUFFER     wmBuf;
int                         retval;

/*------------------------------------
  set I/O line 0-7 and 56-61
  change I/O lines 4-15 and 60-63 only
  ------------------------------------*/
wmBuf.val_31_0   = 0x000000FF;
wmBuf.val_63_32  = 0x3F000000;
wmBuf.mask_31_0   = 0x0000FFF0;
wmBuf.mask_63_32  = 0xF0000000;
retval = ioctl(fd, FIO_TDRV006_WRITE_MASKED, (int)&wmBuf);
if (retval == ERROR)
{
     /* handle the error */
}
```

5.3.4 FIO_TDRV006_OUTPUT_ENABLE

This I/O control function configures the direction of the I/O lines. The function specific control parameter *arg* is a pointer on a *TDRV006_64BITBUFFER* structure.

```
typedef struct
{
        UINT32          val_31_0;
        UINT32          val_63_32;
} TDRV006_64BITBUFFER;
```

val_31_0

This argument specifies the direction of I/O lines 0 up to 31. Bit 0 specifies the direction of I/O line 0, bit 1 the direction of I/O line 1, and so on. A set bit configures the line for output, an unset bit configures input (tri-state).

val_63_32

This argument specifies the direction of I/O lines 32 up to 63. Bit 0 specifies the direction of I/O line 32, bit 1 the direction of I/O line 33, and so on. A set bit configures the line for output, an unset bit configures input (tri-state).

EXAMPLE

```
#include “tdrv006.h”

int                     fd;
TDRV006_64BITBUFFER     dBuf;
int                     retval;

/*----------------------------------------------
  configure line 0-12 for output, other are input
  ----------------------------------------------*/
dBuf.val_31_0   = 0x00001FFF;
dBuf.val_63_32  = 0x00000000;
retval = ioctl(fd, FIO_TDRV006_OUTPUT_ENABLE, (int)&dBuf);
if (retval == ERROR)
{
    /* handle the error */
}
```

5.3.5 FIO_TDRV006_EVENT_WAIT

This I/O control function waits for an I/O line transition event. The function specific control parameter *arg* is a pointer on a *TDRV006_EVENTWAITBUFFER* structure.

```
typedef struct
{
        int             mode;
        int             inputLine;
        int             timeout;
} TDRV006_EVENTWAITBUFFER;
```

mode

This argument specifies the transition the function should wait for. The following values are valid:

definition	**event**
TDRV006_HIGH_TR	The function will return after a low to high transition has been detected
TDRV006_LOW_TR	The function will return after a high to low transition has been detected
TDRV006_ANY_TR	The function will return after a transition has been detected

inputLine

This argument specifies the input line the event shall occur. Valid values are 0 up to 63.

timeout

This argument specifies the maximum time the function shall wait for the event. After this specified time the function will return with an error. The timeout time is specified in ticks.

EXAMPLE

```
#include "tdrv006.h"

int                         fd;
TDRV006_EVENTWAITBUFFER     evBuf;
int                         retval;

…
```

```
…

/*------------------------------------
  Wait for any transition on I/O line 4
  ------------------------------------*/
evBuf.mode      = TDRV006_ANY_TR;
evBuf.inputLine = 4;
evBuf.timeout   = 600;      /* Ticks */
retval = ioctl(fd, FIO_TDRV006_EVENT_WAIT, (int)&evBuf);
if (retval != ERROR)
{
    /* handle the error */
}
```

ERROR CODES

Error code	Description
EINVAL	Invalid parameter specified
EBUSY	An other task is already waiting for a transition of the specified input line
S_objLib_OBJ_TIMEOUT	Timeout occurred.