The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV008-SW-42

VxWorks Device Driver

3x16bit I/O Ports with 512 Word FIFO and Handshake

Version 2.0.x

User Manual

Issue 2.0.0

September 2011

powerBridge Computer

Ehlbeek 15a
30938 Burgwedel
fon 05139-9980-0
fax 05139-9980-49

www.powerbridge.de
info@powerbridge.de

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7 25469 Halstenbek, Germany
49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
ail: info@tews.com www.tews.com

TDRV008-SW-42

VxWorks Device Driver

3x16bit I/O Ports with 512 Word FIFO and Handshake

Supported Modules:

TPMC682
TPMC680-50

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2005-2011 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	April 11, 2005
2.0.0	VxBus Support added, API Documentation added, new File List, IO-Interface Chapter removed	September 28, 2011

Table of Contents

1 Introduction

1.1 Device Driver

The TDRV008-SW-42 VxWorks device driver software allows the operation of the TDRV008 compatible PMC conforming to the VxWorks I/O system specification.

The TDRV008-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.

To prevent the application program from losing data, incoming messages will be stored in a message FIFO with a depth of 100 messages.

The TDRV008-SW-42 device driver supports the following features:

- buffered reading and writing data to handshake ports
- configuration of handshake ports
- flushing all FIFOs of a module
- setting value of output port 5
- reading value from input port 4

The TDRV008-SW-42 supports the modules listed below:

TPMC682	3 x 16 bit I/O Ports with 512 Word FIFO and Handshake	(PMC)
TPMC680-50	3 x 16 bit I/O Ports with 512 Word FIFO and Handshake	(PMC)

In this document all supported modules and devices will be called TDRV008. Specials for certain devices will be advised.

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

TPMC682/TPMC680-50 User Manual
TPMC682/TPMC680-50 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV008-SW-42':

TDRV008-SW-42-2.0.0.pdf	PDF copy of this manual
TDRV008-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TDRV008-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TDRV008-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tdrv008':

tdrv008drv.c	TDRV008 device driver source
tdrv008def.h	TDRV008 driver include file
tdrv008.h	TDRV008 include file for driver and application
tdrv008api.c	TDRV008 API file
Makefile	Driver Makefile
40tdrv008.cdf	Component description file for VxWorks development tools
tdrv008.dc	Configuration stub file for direct BSP builds
tdrv008.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tdrv008exa.c	Example application

The archive TDRV008-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tdrv008':

tdrv008drv.c	TDRV008 device driver source
tdrv008def.h	TDRV008 driver include file
tdrv008.h	TDRV008 include file for driver and application
tdrv008pci.c	TDRV008 device driver source for x86 based systems
tdrv008api.c	TDRV008 API file
tdrv008exa.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 TDRV008-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 TDRV008 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tdrv008.*

At this point the TDRV008 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/tdrv008

(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\tdrv008
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 TDRV008 driver with the VxWorks development tools (Workbench), the component configuration file *40tdrv008.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv008
C:> copy 40tdrv008.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 TDRV008 driver and API can be included in VxWorks projects by selecting the *"TEWS TDRV008 Driver" and "TEWS TDRV008 API"* components 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 TDRV008 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\tdrv008
C:> copy tdrv008.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tdrv008.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 TDRV008-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 TDRV008-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 tdrv008 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 TDRV008 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 TDRV008 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tdrv008pci.c** contains the function *tdrv008PciInit().* This routine finds out all TDRV008 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 *tdrv008PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window):

```
tdrv008PciInit();
```

Be sure that the function is called prior to MMU initialization otherwise the TDRV008 PCI spaces remains unmapped and an access fault occurs during driver initialization.

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

2.3.3 BSP Dependent Adjustments

The driver includes a file called *include/tdhal.h* which contains functions and definitions for BSP adaptation. It may be necessary to modify them for BSP specific settings. Most settings can be made automatically by conditional compilation set by the BSP header files, but some settings must be configured manually. There are two way of modification, first you can change the *include/tdhal.h* and define the corresponding definition and its value, or you can do it, using the command line option *–D*.

There are 3 offset definitions (*USERDEFINED_MEM_OFFSET*, *USERDEFINED_IO_OFFSET*, and *USERDEFINED_LEV2VEC*) that must be configured if a corresponding warning message appears during compilation. These definitions always need values. Definition values can be assigned by command line option *-D<definition>=<value>*.

Definition	Description
USERDEFINED_MEM_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI memory space access
USERDEFINED_IO_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI I/O space access
USERDEFINED_LEV2VEC	The value of this definition must be set to the difference of the interrupt vector (used to connect the ISR) and the interrupt level (stored to the PCI header)

Another definition allows a simple adaptation for BSPs that utilize a *pciIntConnect()* function to connect shared (PCI) interrupts. If this function is defined in the used BSP, the definition of *USERDEFINED_SEL_PCIINTCONNECT* should be enabled. The definition by command line option is made by *-D<definition>.*

Please refer to the BSP documentation and header files to get information about the interrupt connection function and the required offset values.

2.3.4 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	< 1 KB	< 1 KB
Stack	< 1 KB	---
Semaphores	---	3

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 tdrv008Open

NAME

tdrv008Open – opens a device.

SYNOPSIS

```
TDRV008_HANDLE tdrv008Open
(
      char          *DeviceName
)
```

DESCRIPTION

Before I/O can be performed to a device, a device 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 TDRV008 device is named “/tdrv008/0”, the second device is named “/tdrv008/1”, and so on.

EXAMPLE

```
#include      "tdrv008api.h"

TDRV008_HANDLE     hdl;

/*
** open the specified device
*/
hdl = tdrv008Open("/tdrv008/0");
if (hdl == NULL)
{
     /* handle open error */
}
```

RETURNS

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

ERROR CODES

The error codes are stored in *errno.*

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

3.1.2 tdrv008Close

NAME

tdrv008Close – closes a device.

SYNOPSIS

```
TDRV008_STATUS tdrv008Close
(
    TDRV008_HANDLE          hdl
)
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

hdl

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

EXAMPLE

```
#include      "tdrv008api.h"

TDRV008_HANDLE     hdl;
TDRV008_STATUS     result;

/*
** close the device
*/
result = tdrv008Close(hdl);
if (result != TDRV008_OK)
{
    /* handle close error */
}
```

RETURNS

On success, TDRV008_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV008_ERR_INVALID_HANDLE	The device handle is invalid

3.2 Device Access Functions

3.2.1 tdrv008WriteBuffer

NAME

tdrv008WriteBuffer – send data from buffer to port

SYNOPSIS

```
TDRV008_STATUS tdrv008WriteBuffer
(
    TDRV008_HANDLE      hdl,
    unsigned int        portNo,
    unsigned short      *buffer,
    unsigned int        bufferSize,
    unsigned int        *transmittedData,
    int                 timeout
)
```

DESCRIPTION

This function sends the content of a data buffer to the specified port. The data-words from the buffer will be transferred into the port's FIFO. The function returns if the last data-word is moved to the FIFO or the timeout occurs.

PARAMETERS

hdl

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

portNo

This argument specifies the port that shall be used. Allowed values are 0...2.

buffer

This argument points to a buffer where the data-words are stored that shall be transmitted.

bufferSize

This argument specifies the number of data values in the buffer. This is the amount of data-words that will be sent.

transmittedData

This argument points to an *unsigned int* value where the number of transferred data-words will be returned. This value may be helpful if a timeout occurs.

timeout

This argument specifies the maximum time the function will try to transfer data-words into the port's FIFO. The time must be specified in milliseconds. A value of '-1' specifies that the function shall never timeout.

EXAMPLE

```
#include "tdrv008api.h"

TDRV008_HANDLE     hdl;
TDRV008_STATUS     result;
unsigned int       transferred;
unsigned short     buffer[10] = {0x0000 ,0x1111, 0x2222, 0x3333, 0x4444,
                                 0x5555, 0x6666, 0x7777, 0x8888, 0x9999};

/*
** send 10 data-words via port 2
**    (timeout after 1000ms)
*/
result = tdrv008WriteBuffer(hdl, 2, buffer, 10, &transferred, 1000);
if (result != TDRV008_OK)
{
     if (result == TDRV008_ERR_TIMEOUT)
     {    /* just a timeout !!! */
          printf("%d data-words sent\n", transferred);
     }
     else
     {
          /* handle error */
     }
}
else
{
     printf("All data sent\n");
}
```

RETURN VALUE

On success, TDRV008_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV008_ERR_INVALID_HANDLE	The device handle is invalid
TDRV008_ERR_INVAL	A NULL pointer or invalid value is referenced for an input value
TDRV008_ERR_INVALDIR	Port is configured for input
TDRV008_ERR_BUSY	The port is busy
TDRV008_ERR_TIMEOUT	The transfer timed out, the specified time has passed

3.2.2 tdrv008ReadBuffer

NAME

tdrv008ReadBuffer – receive data from a specified port

SYNOPSIS

```
TDRV008_STATUS tdrv008ReadBuffer
(
    TDRV008_HANDLE      hdl,
    unsigned int        portNo,
    unsigned short      *buffer,
    unsigned int        bufferSize,
    unsigned int        *validData,
    int                 timeout
)
```

DESCRIPTION

This function receives data from a specified port. The data-words received on the specified port will be transferred from the port's FIFO into the specified buffer. The function returns if the buffer is filled completely or the timeout occurs.

PARAMETERS

hdl

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

portNo

This argument specifies the port that shall be used. Allowed values are 0...2.

buffer

This argument points to a buffer where the received data-words shall be stored.

bufferSize

This argument specifies the size of the buffer. This is the maximum amount of data-words that will be returned.

validData

This argument points to an *unsigned int* value which returns the number of data words received and stored in the specified buffer.

timeout

This argument specifies the maximum time the function will try to read data-words from the ports FIFO. The time must be specified in milliseconds. A value of '-1' specifies that the function shall never timeout. A value of '0' specifies that the function shall read all data that is currently stored in the port's FIFO, but the function will not wait for further incoming data.

EXAMPLE

```
#include "tdrv008api.h"

TDRV008_HANDLE     hdl;
TDRV008_STATUS     result;
unsigned int       received;
unsigned short     buffer[50];
int                i;

/*
** read data from port 0 (max. 50 data words)
**    (timeout after 5000ms)
*/
result = tdrv008ReadBuffer(hdl, 0, buffer, 50, &received, 5000);
if (result != TDRV008_OK)
{
     if (result == TDRV008_ERR_TIMEOUT)
     {    /* just a timeout !!! */
     }
     else
     {
          /* handle error */
          received = 0;
     }
}

printf("%d data-words received\n", received);
for (i = 0; i < received; i++)
{
          printf("    data[%d] = 0x%x\n", i, buffer[i]);
}
```

RETURN VALUE

On success, TDRV008_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV008_ERR_INVALID_HANDLE	The device handle is invalid
TDRV008_ERR_INVAL	A NULL pointer or invalid value is referenced for an input value
TDRV008_ERR_INVALDIR	Port is configured for output
TDRV008_ERR_BUSY	The port is busy
TDRV008_ERR_TIMEOUT	The transfer timed out, the specified time has passed

3.2.3 tdrv008GetPort

NAME

tdrv008GetPort – read current state of input port 4

SYNOPSIS

```
TDRV008_STATUS tdrv008GetPort
(
    TDRV008_HANDLE          hdl,
    unsigned char           *inValue
)
```

DESCRIPTION

This function reads the current state of input port 4.

PARAMETERS

hdl

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

inValue

This argument points to *an unsigned char* value where the state of the input lines will be written to. Only bits 3...7 are valid. Bits 0...2 will always be 0, these input lines are reserved for the input handshake lines of port 0…2.

EXAMPLE

```
#include "tdrv008api.h"

TDRV008_HANDLE     hdl;
TDRV008_STATUS     result;
unsigned char      inPort4;

/*
** read current state of port 4
*/
result = tdrv008GetPort(hdl, &inPort4);
if (result != TDRV008_OK)
{
     /* handle error */
}

printf("Port 4 = 0x%02x \n", inPort4);
```

RETURN VALUE

On success, TDRV008_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV008_ERR_INVALID_HANDLE	The device handle is invalid
TDRV008_ERR_INVAL	A NULL pointer or invalid value is referenced for an input value

3.2.4 tdrv008SetPort

NAME

tdrv008SetPort – set output port 5

SYNOPSIS

```
TDRV008_STATUS tdrv008SetPort
(
    TDRV008_HANDLE          hdl,
    unsigned char           outValue
)
```

DESCRIPTION

This function sets the output value of port 5.

PARAMETERS

hdl

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

outValue

This argument specifies the new output value of port 5. Only bits 3...7 are valid. Bits 0…2 will ignored, the output lines are reserved for the output handshake lines of port 0…2.

EXAMPLE

```
#include "tdrv008api.h"

TDRV008_HANDLE     hdl;
TDRV008_STATUS     result;

/*
** set all output lines of port 5
*/
result = tdrv008SetPort(hdl, 0xf8);
if (result != TDRV008_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV008_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV008_ERR_INVALID_HANDLE	The device handle is invalid

3.2.5 tdrv008ConfigurePort

NAME

tdrv008ConfigurePort – configure a handshake port

SYNOPSIS

```
TDRV008_STATUS tdrv008ConfigurePort
(
    TDRV008_HANDLE      hdl,
    unsigned int        portNo,
    unsigned int        flags,
    unsigned int        fifoTimeout,
    unsigned int        fifoThreshold
)
```

DESCRIPTION

This function configures the specified handshake port. It sets the direction, the handshake mode, threshold, and if necessary the receive timeout.

PARAMETERS

hdl

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

portNo

This argument specifies the port that shall be configured. Allowed values are 0...2.

flags

This argument specifies the mode of the port. The value must contain an ORed value of flags described below. There must be set one flag specifying the port direction and one flag specifying the output handshake mode. (For a more detailed description of the handshake modes, please refer to the User Manual of the specific module.)

Port Direction Flags	Description
TDRV008_CONF_PORTDIR_INPUT	Configure port for data input.
TDRV008_CONF_PORTDIR_OUTPUT	Configure port for data output.

Output Handshake Mode Flags	Description
TDRV008_CONF_HSOUT_HSNONE	No output handshake
TDRV008_CONF_HSOUT_HSINTERLOCKED	Interlocked output handshake
TDRV008_CONF_HSOUT_HSPULSED	Pulsed output handshake

fifoTimeout

This argument specifies the hardware FIFO timeout value. The value will be directly written to the module (TCPRx). Refer to the User Manual of your module for more information. This value is only used for input ports.

fifoThreshold

This argument specifies the FIFO threshold value. This value will be directly written to the module (FIFO_FTRx). Refer to the User Manual of your module for more information. This value must be set between 1 and 512.

EXAMPLE

```
#include "tdrv008api.h"

TDRV008_HANDLE     hdl;
TDRV008_STATUS     result;

/*
** Configure port 1 for data input (interlocked)
**   FIFO-Threshold: 50
**   Input Timeout:  100
*/
result = tdrv008ConfigurePort (  hdl,
                                 1,
                                 TDRV008_CONF_PORTDIR_INPUT |
                                 TDRV008_CONF_HSOUT_HSINTERLOCKED,
                                 100,
                                 50);
if (result != TDRV008_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV008_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV008_ERR_INVALID_HANDLE	The device handle is invalid
TDRV008_ERR_INVAL	An invalid value is referenced for an input value
TDRV008_ERR_BUSY	The port is busy

3.2.6 tdrv008FlushFifos

NAME

tdrv008FlushFifos – flush FIFOs of all handshake ports

SYNOPSIS

```
TDRV008_STATUS tdrv008 FlushFifos
(
      TDRV008_HANDLE              hdl
)
```

DESCRIPTION

This function flushes the FIFOs of all handshake ports (0…2).

PARAMETERS

hdl

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

EXAMPLE

```
#include "tdrv008api.h"

TDRV008_HANDLE     hdl;
TDRV008_STATUS     result;

/*
** flush handshake ports
*/
result = tdrv008FlushFifos(hdl);
if (result != TDRV008_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV008_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV008_ERR_INVALID_HANDLE	The device handle is invalid

4 Legacy I/O System Functions

This chapter describes the legacy driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

The legacy I/O system functions are only relevant for the legacy TDRV008 driver. For the VxBus-enabled TDRV008 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.

4.1 tdrv008Drv

NAME

tdrv008Drv - installs the TDRV008 driver in the I/O system

SYNOPSIS

STATUS tdrv008Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus and installs the TDRV008 driver in the I/O system.

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

EXAMPLE

```
#include      "tdrv008api.h"

STATUS        result;

/*--------------------
  Initialize Driver
  --------------------*/
result = tdrv008Drv();
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
TDRV008_ERR_NXIO	No supported device found, driver will not start

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.2 tdrv008DevCreate

NAME

tdrv008DevCreate – Add a TDRV008 device to the VxWorks system

SYNOPSIS

```
STATUS tdrv008DevCreate
(
    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:

- TPMC680-50
- TPMC682-xx

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

Example: A system with 1x TPMC680-50 and 2x TPMC682-10 will assign the following device indices:

Module	Device Index
TPMC680-50	0
TPMC682-10 (1st)	1
TPMC682-10 (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 "tdrv008api.h"

STATUS             result;

/*-------------------------------------------------------
  Create the device "/tdrv008/0" for the first device
  -------------------------------------------------------*/
result = tdrv008DevCreate(   "/tdrv008/0",
                             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

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

Error Code	Description
TDRV008_ERR_NXIO	Driver not started, or specified device index is invalid
TDRV008_ERR_EXISTS	Device already exists

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.3 tdrv008PciInit

NAME

tdrv008PciInit – Generic PCI device initialization

SYNOPSIS

void tdrv008PciInit()

DESCRIPTION

This function is only required for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TDRV008 device PCI spaces (base address registers) and to enable the TDRV008 device for access.

The global variable *tdrv008Status* 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 tdrv008Status is equal to the number of mapped PCI spaces
0	No TDRV008 device found
< 0	Initialization failed. The value of (tdrv008Status & 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 tdrv008PciInit();

tdrv008PciInit();
```

4.4 tdrv008Init

NAME

tdrv008Init – initialize TDRV008 driver and devices

SYNOPSIS

```
STATUS tdrv008Init
(
      void
)
```

DESCRIPTION

This function is used by the TDRV008 example application to install the driver and to add all available devices to the VxWorks system.

See also 3.1.1 tdrv008Open for the device naming convention for legacy devices.

After calling this function it is not necessary to call tdrv008Drv() and tdrv008DevCreate() explicitly.

EXAMPLE

```
#include  "tdrv008.h"

STATUS    result;

result = tdrv008Init();
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 tdrv008Drv and 4.2 tdrv008DevCreate for a description of possible error codes.