The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV004-SW-42

VxWorks Device Driver

Reconfigurable FPGA

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

TDRV004-SW-42

VxWorks Device Driver

Reconfigurable FPGA

Supported Modules:
TPMC630
TCP630

Issue	Description	Date
1.0.0	First Issue	May 04, 2006
1.1.0	New module support, tdrv004DevCreate() error handling enhanced, Interrupt functions added, New Address TEWS LLC	May 16, 2007
2.0.0	Support for VxBus and API description added	December 17, 2009
2.0.1	Legacy vs. VxBus Driver modified	March 26, 2010

Table of Contents

1 Introduction

The TRDV004-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 TDRV004-SW-42 device driver supports the following features:

- Program and reconfigure onboard FPGA
- Program onboard clock generator using the Serial Programming Interface (SPI)
- Read/write FPGA registers (32bit / 16bit / 8bit)
- Read/write EEPROM blocks located in clock device using the Serial Programming Interface (SPI)
- Read/write specific PLX9030 registers

The TDRV004-SW-42 supports the modules listed below:

TPMC630	Reconfigurable FPGA with 64 TTL I/O Lines or 32 Differential I/O Lines	(PMC)
TCP630	Reconfigurable FPGA with 64 TTL I/O Lines or 32 Differential I/O Lines	(CompactPCI)

In this document covers all supported modules and devices will be called TDRV004. Specials for certain devices will be advised.

To get more information about the features and use of TDRV004 devices it is recommended to read the manuals listed below.

TPMC630 product family User Manual

TPMC630 product family Engineering Manual

Cypress CY27EE16 User Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV004-SW-42':

TDRV004-SW-42-2.0.1.pdf	PDF copy of this manual
TDRV004-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TDRV004-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
fpgaexa.zip	FPGA example XSVF files
ChangeLog.txt	Release history
Release.txt	Release information

The archive TDRV004-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tdrv004':

tdrv004drv.c	TDRV004 device driver source
tdrv004def.h	TDRV004 driver include file
tdrv004.h	TDRV004 include file for driver and application
tdrv004api.c	TDRV004 API file
pf_micro.c	XSVF player functions (Platform Flash)
pf_micro.h	header file for XSVF player functions
pf_lenval.c	special functions for XSVF player
pf_lenval.h	header file for XSVF functions
pf_ports.c	hardware layer for XSVF player
pf_ports.h	header file for XSVF hardware layer
Makefile	Driver Makefile
40tdrv004.cdf	Component description file for VxWorks development tools
tdrv004.dc	Configuration stub file for direct BSP builds
tdrv004.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tdrv004exa.c	Example application

The archive TDRV004-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tdrv004':

tdrv004drv.c	TDRV004 device driver source
tdrv004def.h	TDRV004 driver include file
tdrv004.h	TDRV004 include file for driver and application
tdrv004api.c	TDRV004 API file
tdrv004pci.c	TDRV004 PCI MMU mapping for Intel x86 based targets
tdrv004exa.c	Example application
tdrv004init.c	Legacy driver initialization
pf_micro.c	XSVF player functions (Platform Flash)
pf_micro.h	header file for XSVF player functions
pf_lenval.c	special functions for XSVF player
pf_lenval.h	header file for XSVF functions
pf_ports.c	hardware layer for XSVF player
pf_ports.h	header file for XSVF hardware layer
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!)

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 TDRV004-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 TDRV004 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tdrv004.*

At this point the TDRV004 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/tdrv004

(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\tdrv004
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 TDRV004 driver with the VxWorks development tools (Workbench), the component configuration file *40tdrv004.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv004
C:> copy 40tdrv004.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 TDRV004 driver can be included in VxWorks projects by selecting the *"TEWS TDRV004 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 TDRV010 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\tdrv004
C:> copy tdrv004.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tdrv004.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 Tornado IDE project

For Including the TDRV004-SW-42 device driver into a Tornado IDE project follow the steps below:

(1) Extract all files from the archive TDRV004-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 tdrv004 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 Tornado User's Guide.

2.3.2 Special installation for Intel x86 based targets

The TDRV004 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 TDRV004 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tdrv004pci.c** contains the function *tdrv004PciInit().* This routine finds out all TDRV004 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 *tdrv004PciInit()* 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 TDRV004 PCI spaces remains unmapped and an access fault occurs during driver initialization.

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

```
tdrv004PciInit();
```

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	---	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 tdrv004Open()

Name

tdrv004Open() – opens a device.

Synopsis

```
TDRV004_DEV tdrv004Open
(
      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 TDRV004 device is named “/tdrv004/0”, the second device is named “/tdrv004/1” and so on.

Example

```
#include “tdrv004.h”
TDRV004_DEV     pDev;

/*
** open file descriptor to device
*/
pDev = tdrv004Open( “/tdrv004/0” );
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 tdrv004Close()

Name

tdrv004Close() – closes a device.

Synopsis

```
int tdrv004Close
(
      TDRV004_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 “tdrv004.h”
TDRV004_DEV pDev;
int result;

/*
** close file descriptor to device
*/
result = tdrv004Close( 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 tdrv004XsvfPlay()

Name

tdrv004XsvfPlay() – Play an XSVF file for FPGA programming

Synopsis

```
STATUS tdrv004XsvfPlay
(
        TDRV004_DEV             pDev,
        TDRV004_XSVF_BUF  *pXsvfBuf
);
```

Description

This function programs the FPGA with a supplied XSVF file. For information on building an XSVF file, please refer to the Engineering Documentation of the TDRV004 product family.

The device driver is not able to verify the supplied XSVF file content, so please make sure that the supplied XSVF is of a valid file format.

Parameters

pDev

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

pXsvfBuf

This parameter specifies a pointer to a TDRV004_XSVF_BUF structure:

```
typedef struct
{
        unsigned long       size;
        unsigned char       pData[1];     /* dynamically expandable */
} TDRV004_XSVF_BUF;
```

size

Specifies the total size of the supplied XSVF data.

pData

This dynamically expandable array holds the XSVF data. The data must be included inside the TDRV004_XSVF_BUF structure.

Programming Hints

Depending on the XSVF file, there might be a waiting period of approx. 15 seconds at the beginning of programming. The programming of the delivered FPGA example design XSVF file should not take much longer than 1 minute, depending on the system load.

If the programming fails, try to increase the used waitstates with control function FIO_TDRV004_SETWAITSTATES (refer to the corresponding section in this manual). Additionally, the CLK1 should not be lower than 10MHz for programming.

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;
TDRV004_XSVF_BUF   *pXsvfBuf;
int                bufsize;

/*
** allocate enough memory (about 3MB) to hold XSVF content
*/
bufsize = sizeof(TDRV004_XSVF_BUF) + <filesize> * sizeof(unsigned char);
pXsvfBuf = (TDRV004_XSVF_BUF*)malloc( bufsize );

/*
** read XSVF content from file and store it inside pXsvfBuf->pData[]
*/
pXsvfBuf->pData    = ...
pXsvfBuf->size     = ...

/*
** start FPGA programming
*/
result = tdrv004XsvfPlay(  pDev,
                           pXsvfBuf );
if (result == ERROR)
{
     /* handle error */
}
free( pXsvfBuf );
```

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	There was an error during XSVF processing.
EINTR	The function was cancelled.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.2 tdrv004XsvfPos()

Name

tdrv004XsvfPos() – Retrieve current play-position in XSVF file

Synopsis

```
STATUS tdrv004XsvfPos
(
        TDRV004_DEV         pDev,
        unsigned int        *pXsvfPos
);
```

Description

This function returns the number of the current processed byte in the XSVF file during programming. This control function can be used to monitor the programming progress.

Parameters

pDev

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

pXsvfPos

This parameter specifies a pointer to an unsigned int value which receives the current processed XSVF byte.

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;
unsigned int       XsvfPos;

/*
** Check XSVF position
*/
result = tdrv004XsvfPos(   pDev,
                           &XsvfPos );

if (result != ERROR)
{
     /* function succeeded */
     printf(“Current XSVF position: %ld\n”, XsvfPos);
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

This function returns no function specific error codes.

3.2.3 tdrv004XsvfLastCommand()

Name

tdrv004XsvfLastCommand() – Get the last executed XSVF command

Synopsis

```
STATUS tdrv004XsvfLastCommand
(
    TDRV004_DEV         pDev,
    unsigned int        *pXsvfLastCmd
);
```

Description

This function returns the number of the last executed XSVF command. This value can be used to find errors inside the supplied XSVF file. This value refers to the line inside the ASCII SVF file.

Parameters

pDev

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

pXsvfLastCmd

This parameter specifies a pointer to an unsigned int value which receives the last executed XSVF command (line number of ASCII SVF file).

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;
unsigned int       XsvfLastCmd;

/*
** Check Last XSVF command
*/
result = tdrv004XsvfLastCommand (     pDev,
                                      &XsvfLastCmd);

if (result != ERROR)
{
     /* function succeeded */
     printf(“Last XSVF command: %ld\n”, XsvfLastCmd);
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

This function returns no function specific error codes.

3.2.4 tdrv004Reconfigure()

Name

tdrv004Reconfigure() – Trigger FPGA reconfiguration process

Synopsis

```
STATUS tdrv004Reconfigure
(
     TDRV004_DEV         pDev
);
```

Description

This function starts the reconfiguration process of the FPGA. This control function must be called after the FPGA is programmed using tdrv004XsvfPlay(). The function returns after the reconfiguration is done, or an error occurred.

Parameters

pDev

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

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;

/*
** Reconfigure FPGA
*/
result = tdrv004Reconfigure ( pDev );

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
EIO	An error occurred during reconfiguration. This may be caused by an invalid FPGA content located inside the XSVF file.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.5 tdrv004SetWaitstates()

Name

tdrv004SetWaitstates() – Specify number of waitstates for programming

Synopsis

```
STATUS tdrv004SetWaitstates
(
    TDRV004_DEV     pDev,
    int             WaitStates
);
```

Description

This function configures the driver to use a number of waitstates during XSVF and SPI programming. This might be necessary, if the local clock (CLK1) of the onboard clock generator is configured to rather slow. The local programming interface is clocked with this frequency, which might result in errors during programming for low CLK1 frequencies and a small amount of waitstates. The system architecture (existing PCI-to-PCI bridges) might also have an impact.

Parameters

pDev

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

WaitStates

This parameter specifies the number of waitstates to be used for XSVF programming. Valid values are 1 to 1000.

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;
int                WaitStates;

/*
** Configure driver to use 3 waitstates
*/
WaitStates = 3;
result = tdrv004SetWaitstates ( pDev, WaitStates );

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

This function returns no function specific error codes.

3.2.6 tdrv004SetClock()

Name

tdrv004SetClock() – Set clock generator parameters

Synopsis

```
STATUS tdrv004SetClock
(
    TDRV004_DEV                 pDev,
    TDRV004_CLOCK_PARAM         *pClockParam
);
```

Description

This function configures the onboard clock generator.

Parameters

pDev

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

pClockParam

This parameter specifies a pointer to a TDRV004_CLOCK_PARAM structure:

```
typedef struct
{
    unsigned char    DeviceAddr;
    unsigned char    x09_ClkOE;
    unsigned char    x0C_DIV1SRCN;
    unsigned char    x10_InputCtrl;
    unsigned char    x40_CPumpPB;
    unsigned char    x41_CPumpPB;
    unsigned char    x42_POQcnt;
    unsigned char    x44_SwMatrix;
    unsigned char    x45_SwMatrix;
    unsigned char    x46_SwMatrix;
    unsigned char    x47_DIV2SRCN;
} TDRV004_CLOCK_PARAM;
```

DeviceAddr

Specifies the desired destination address. The CY27EE16 clock generator provides several EEPROM banks as well as SRAM. If TDRV004_CLKADR_SRAM (0x69) is specified, the values are directly stored inside the volatile RAM area and take effect immediately. If TDRV004_CLKADR_EEPROM (0x68) is specified, the values are stored in the non-volatile area of the clock generator, and the CY27EE16 loads it after the next power-up.

x09_ClkOE

Specifies which clock outputs shall be enabled.

x0C_DIV1SRCN

Specifies internal input source 1 and the corresponding frequency divider

x10_InputCtrl

Specifies value for the Input Pin Control register

x40_CPumpPB

Specifies value for Charge Pump and PB counter register

x41_CPumpPB

Specifies value for Charge Pump and PB counter register

x42_POQcnt

Specifies value for PO and Q counter register

x44_SwMatrix

Specifies value for Switching Matrix Register

x45_SwMatrix

Specifies value for Switching Matrix Register

x46_SwMatrix

Specifies value for Switching Matrix Register

x47_DIV2SRCN

Specifies internal input source 2 and the corresponding frequency divider

Please refer to the Cypress CY27EE16 user manual for detailed explanation of the above register values. Use Cypress' *CyberClocks* Version R3.10.00 to determine the correct values. This program is also part of the TPMC630 or TCP630 Engineering Documentation.

Example

```
#include “tdrv004.h”

TDRV004_DEV                pDev;
STATUS                     result;
TDRV004_CLOCK_PARAM        ClockParam;


/*
** Setup clock generator (SRAM):
**   CLK1: 50.0MHz       CLK2: 20.0MHz
**   CLK3: 10.0MHz       CLK4:  1.0MHz
**   CLK5:  0.2MHz       CLK6: -off-
*/
ClockParam.DeviceAddress   = TDRV004_CLKADR_SRAM;
ClockParam.x09_ClkOE       = 0x6f;
ClockParam.x0C_DIV1SRCN    = 0x64;
ClockParam.x10_InputCtrl   = 0x50;
ClockParam.x40_CPumpPB     = 0xc0;
ClockParam.x41_CPumpPB     = 0x03;
ClockParam.x42_POQcnt      = 0x81;
ClockParam.x44_SwMatrix    = 0x42;
ClockParam.x45_SwMatrix    = 0x9f;
ClockParam.x46_SwMatrix    = 0x3f;
ClockParam.x47_DIV2SRCN    = 0xe4;

/*
** start FPGA programming
*/
result = tdrv004SetClock(  pDev,
                           &ClockParam );
if (result != 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.

Error code	Description
EINVAL	It was tried to disable CLK1. This is not allowed.
EIO	An error occurred during SPI access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.7 tdrv004SpiWrite()

Name

tdrv004SpiWrite() – Write values to SPI storage

Synopsis

```
STATUS tdrv004SpiWrite
(
        TDRV004_DEV          pDev,
        TDRV004_SPI_BUF      *pSpiIoBuf
);
```

Description

This function writes up to 256 *unsigned char* (8bit) values to a specific sub-address of a Serial Programming Interface (SPI) address. The SPI storages are available in the clock generator device.

Do not use this control function to setup the clock generator. Please use API function tdrv004SpiWrite() instead.

Parameters

pDev

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

pSpiIoBuf

This parameter specifies a pointer to a TDRV004_SPI_BUF structure:

```
typedef struct
{
        unsigned char       SpiAddr;
        unsigned char       SubAddr;
        unsigned long       len;
        unsigned char       pData[1];     /* dynamically expandable */
} TDRV004_SPI_BUF;
```

SpiAddr

Specifies the Serial Programming Interface (SPI) address of the desired target. The following values are possible (refer to file *tdrv004.h*):

Symbol	Value	Description
TDRV004_CLKADDR_EEPROM	0x68	Clock Generator EEPROM (non-volatile)
TDRV004_CLKADDR_SRAM	0x69	Clock Generator SRAM (volatile)
TDRV004_CLKADDR_EEBLOCK1	0x40	EEPROM-Bank 1
TDRV004_CLKADDR_EEBLOCK2	0x41	EEPROM-Bank 2
TDRV004_CLKADDR_EEBLOCK3	0x42	EEPROM-Bank 3
TDRV004_CLKADDR_EEBLOCK4	0x43	EEPROM-Bank 4
TDRV004_CLKADDR_EEBLOCK5	0x44	EEPROM-Bank 5
TDRV004_CLKADDR_EEBLOCK6	0x45	EEPROM-Bank 6
TDRV004_CLKADDR_EEBLOCK7	0x46	EEPROM-Bank 7
TDRV004_CLKADDR_EEBLOCK8	0x47	EEPROM-Bank 8

SubAddr

Specifies the sub-address (starting offset).

len

This value specifies the amount of data items to write. A maximum of 256 is allowed.

pData

The values are copied from this buffer. It must be large enough to hold the specified amount of data. The data must be stored inside the structure, no pointer allowed.

Example

```
#include “tdrv004.h”

TDRV004_DEV                pDev;
STATUS                     result;
TDRV004_SPI_BUF            *pSpiBuf;
int                        BufferSize;

…
```

…

```
/*
** write 5 bytes to EEPROM block 1, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize = ( sizeof(TDRV004_SPI_BUF) + 5*sizeof(unsigned char) );
pSpiBuf = (TDRV004_SPI_BUF*)malloc( BufferSize );
pSpiBuf->SpiAddr   = TDRV004_CLKADDR_EEBLOCK1;
pSpiBuf->SubAddr   = 0x00;
pSpiBuf->len       = 5;
pSpiBuf->pData[0]  = 0x01;
pSpiBuf->pData[1]  = 0x02;
pSpiBuf->pData[2]  = 0x03;
pSpiBuf->pData[3]  = 0x04;
pSpiBuf->pData[4]  = 0x05;

result = tdrv004SpiWrite(  pDev,
                           pSpiBuf );
if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
free(pSpiBuf);
```

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	The specified SubAddr+len exceeds 256, or len is invalid
EIO	An error occurred during SPI access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.8 tdrv004SpiRead()

Name

tdrv004SpiRead() – Read values from SPI storage

Synopsis

```
STATUS tdrv004SpiRead
(
    TDRV004_DEV         pDev,
    TDRV004_SPI_BUF     *pSpiIoBuf
);
```

Description

This function reads up to 256 *unsigned char* (8bit) values from a specific sub-address of a Serial Programming Interface (SPI) address. The SPI storages are available in the clock generator device.

Parameters

pDev

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

pSpiIoBuf

This parameter specifies a pointer to a TDRV004_SPI_BUF structure:

```
typedef struct
{
    unsigned char       SpiAddr;
    unsigned char       SubAddr;
    unsigned long       len;
    unsigned char       pData[1];     /* dynamically expandable */
} TDRV004_SPI_BUF;
```

SpiAddr

Specifies the Serial Programming Interface (SPI) address of the desired target. The following values are possible (refer to file *tdrv004.h*):

Symbol	Value	Description
TDRV004_CLKADDR_EEPROM	0x68	Clock Generator EEPROM (non-volatile)
TDRV004_CLKADDR_SRAM	0x69	Clock Generator SRAM (volatile)
TDRV004_CLKADDR_EEBLOCK1	0x40	EEPROM-Bank 1
TDRV004_CLKADDR_EEBLOCK2	0x41	EEPROM-Bank 2
TDRV004_CLKADDR_EEBLOCK3	0x42	EEPROM-Bank 3
TDRV004_CLKADDR_EEBLOCK4	0x43	EEPROM-Bank 4
TDRV004_CLKADDR_EEBLOCK5	0x44	EEPROM-Bank 5
TDRV004_CLKADDR_EEBLOCK6	0x45	EEPROM-Bank 6
TDRV004_CLKADDR_EEBLOCK7	0x46	EEPROM-Bank 7
TDRV004_CLKADDR_EEBLOCK8	0x47	EEPROM-Bank 8

SubAddr

Specifies the sub-address (starting offset).

len

This value specifies the amount of data items to write. A maximum of 256 is allowed.

pData

The values are copied to this buffer. It must be large enough to hold the specified amount of data. The data space must be located inside the structure, no pointer allowed.

Example

```
#include “tdrv004.h”

TDRV004_DEV                pDev;
STATUS                     result;
TDRV004_SPI_BUF            *pSpiBuf;
int                        BufferSize;

…
```

…

```
/*
** read 5 bytes from EEPROM block 1, offset 0x00
** allocate enough memory to hold the data structure + read data
*/
BufferSize = ( sizeof(TDRV004_SPI_BUF) + 5*sizeof(unsigned char) );
pSpiBuf = (TDRV004_SPI_BUF*)malloc( BufferSize );
pSpiBuf->SpiAddr   = TDRV004_CLKADDR_EEBLOCK1;
pSpiBuf->SubAddr   = 0x00;
pSpiBuf->len       = 5;

result = tdrv004SpiRead(   pDev,
                           pSpiBuf );
if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
free(pSpiBuf);
```

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	The specified SubAddr+len exceeds 256, or len is invalid
EIO	An error occurred during SPI access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.9 tdrv004PlxWrite()

Name

tdrv004PlxWrite() – Write 16bit value to PLX PCI9030 EEPROM

Synopsis

```
STATUS tdrv004PlxWrite
(
    TDRV004_DEV         pDev,
    TDRV004_PLX_BUF     PlxBuf
);
```

Description

This function writes an *unsigned short* (16bit) value to a specific PLX PCI9030 EEPROM memory offset.

Please note that the PLX9030 reloads the new configuration from the EEPROM after a PCI reset, i.e. the system must be rebooted to make PLX PCI9030 dependent changes take effect.

Parameters

pDev

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

pPlxBuf

This parameter specifies a pointer to a TDRV004_PLX_BUF structure:

```
typedef struct
{
    unsigned long       Offset;
    unsigned short      Value;
} TDRV004_PLX_BUF;
```

Offset

Specifies the offset into the PLX9030 EEPROM, where the supplied data word should be written. The offset must be specified as even byte-address.
Following offsets are available:

Offset	Access
00h – 0Ch	R
0Eh	R / W
10h – 26h	R
28h – 36h	R / W
38h – 3Ah	R
3Ch – 4Ah	R / W
4Ch – 4Eh	R
50h – 5Eh	R / W
60h – 62h	R
64h – 7Eh	R / W
80h – 86h	R
88h - FEh	R / W

Refer to the PLX PCI9030 User Manual for detailed information on these registers.

Value

This value specifies a 16bit word that should be written to the specified offset.

Example

```
#include “tdrv004.h”

TDRV004_DEV                pDev;
STATUS                     result;
TDRV004_PLX_BUF            PlxBuf;


/*
** Change the Subsystem Vendor ID to TEWS TECHNOLOGIES (0x1498)
*/
PlxBuf.Offset = 0x0E;
PlxBuf.Value  = 0x1498;

result = tdrv004PlxWrite(  pDev,
                           &PlxBuf );
if (result == ERROR)
{
     /* 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.

Error code	Description
EINVAL	The specified offset is invalid, or read-only
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.10 tdrv004PlxRead()

Name

tdrv004PlxRead() – Read 16bit value from PLX PCI9030 EEPROM

Synopsis

```
STATUS tdrv004PlxRead
(
    TDRV004_DEV         pDev,
    TDRV004_PLX_BUF     PlxBuf
);
```

Description

This function reads an *unsigned short* (16bit) value from a specific PLX PCI9030 EEPROM memory offset.

Parameters

pDev

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

pPlxBuf

This parameter specifies a pointer to a TDRV004_PLX_BUF structure:

```
typedef struct
{
    unsigned long       Offset;
    unsigned short      Value;
} TDRV004_PLX_BUF;
```

Offset

Specifies the offset into the PLX PCI9030 EEPROM, where the supplied data word should be read. The offset must be specified as even byte-address.
Following offsets are available:

Offset	Access
00h – 0Ch	R
0Eh	R / W
10h – 26h	R
28h – 36h	R / W
38h – 3Ah	R
3Ch – 4Ah	R / W
4Ch – 4Eh	R
50h – 5Eh	R / W
60h – 62h	R
64h – 7Eh	R / W
80h – 86h	R
88h - FEh	R / W

Refer to the PLX PCI9030 User Manual for detailed information on these registers.

Value

This value holds the retrieved 16bit word.

Example

```
#include “tdrv004.h”

TDRV004_DEV                pDev;
STATUS                     result;
TDRV004_PLX_BUF            PlxBuf;

/*
** Read Subsystem ID
*/
PlxBuf.Offset = 0x0C;

result = tdrv004PlxRead(   pDev,
                           &PlxBuf );
if (result != ERROR)
{
     /* function succeeded */
     printf( “SubsystemID = 0x%04X\n”, PlxBuf.Value );
} 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.

Error code	Description
EINVAL	The specified offset is invalid, or read-only
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.11 tdrv004ReadU8()

Name

tdrv004ReadU8() – Read 8bit values from FPGA resource

Synopsis

```
STATUS tdrv004ReadU8
(
    TDRV004_DEV             pDev,
    TDRV004_MEMIO_BUF       *pMemIoBuf
);
```

Description

This function reads a number of *unsigned char* (8bit) values from a Memory or I/O area by using BYTE (8bit) accesses. The data buffer can be enlarged to the desired needs. The data section must be included inside the structure.

Parameters

pDev

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

pMemIoBuf

This parameter specifies a pointer to a TDRV004_MEMIO_BUF structure:

```
typedef struct
{
    TDRV004_RESOURCE  Resource;
    unsigned long     Offset;
    unsigned long     Size;
    unsigned char     pData[1];    /* dynamically expandable */
} TDRV004_MEMIO_BUF;
```

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX PCI9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TDRV004_RESOURCE
0	IO	*(reserved)*	TDRV004_RES_IO_1
1	MEM	*(reserved)*	TDRV004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TDRV004_RES_MEM_2
3	IO	*(not implemented by default)*	TDRV004_RES_IO_2
4	IO	*(not implemented by default)*	TDRV004_RES_IO_3
5	MEM	*(not implemented by default)*	TDRV004_RES_MEM_3

The PLX PCI9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the memory or I/O space specified by *Resource*.

Size

This value specifies the amount of data items to read.

pData

The received values are copied into this buffer. It must be large enough to hold the specified amount of data.

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;
TDRV004_MEMIO_BUF  pMemIoBuf;
unsigned char      *pValues;
int                BufferSize;

/*
** read 50 bytes from MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + read data
*/
BufferSize    = ( sizeof(TDRV004_MEMIO_BUF) + 50*sizeof(unsigned char) );
pMemIoBuf     = (TDRV004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size           = 50;
pMemIoBuf->Resource       = TDRV004_RES_MEM_2;
pMemIoBuf->Offset         = 0;

result = tdrv004ReadU8(    pDev,
                           pMemIoBuf );

…
```

…

```
if (result != ERROR)
{
     /* function succeeded */
     pValues = (unsigned char*)pMemIoBuf->pData;
} else {
     /* handle the error */
}
free( pMemIoBuf );
```

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	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.12 tdrv004ReadU16()

Name

tdrv004ReadU16() – Read 16bit values from FPGA resource

Synopsis

```
STATUS tdrv004ReadU16
(
    TDRV004_DEV             pDev,
    TDRV004_MEMIO_BUF       *pMemIoBuf
);
```

Description

This function reads a number of *unsigned short* (16bit) values from a Memory or I/O area by using WORD (16bit) accesses. The data buffer can be enlarged to the desired needs. The data section must be included inside the structure.

Parameters

pDev

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

pMemIoBuf

This parameter specifies a pointer to a TDRV004_MEMIO_BUF structure:

```
typedef struct
{
    TDRV004_RESOURCE  Resource;
    unsigned long     Offset;
    unsigned long     Size;
    unsigned char     pData[1];    /* dynamically expandable */
} TDRV004_MEMIO_BUF;
```

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX PCI9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type	TDRV004_RESOURCE
0	IO *(reserved)*	TDRV004_RES_IO_1
1	MEM *(reserved)*	TDRV004_RES_MEM_1
2	MEM *(used by VHDL Example)*	TDRV004_RES_MEM_2
3	IO *(not implemented by default)*	TDRV004_RES_IO_2
4	IO *(not implemented by default)*	TDRV004_RES_IO_3
5	MEM *(not implemented by default)*	TDRV004_RES_MEM_3

The PLX PCI9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the memory or I/O space specified by *Resource*.

Size

This value specifies the amount of data items to read.

pData

The received values are copied into this buffer. It must be large enough to hold the specified amount of data.

Example

```
#include “tdrv004.h”

TDRV004_DEV         pDev;
STATUS              result;
TDRV004_MEMIO_BUF   pMemIoBuf;
unsigned short      *pValues;
int                 BufferSize;

/*
** read 50 16bit words from MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + read data
*/
BufferSize    = ( sizeof(TDRV004_MEMIO_BUF) + 50*sizeof(unsigned short) );
pMemIoBuf     = (TDRV004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size           = 50;
pMemIoBuf->Resource       = TDRV004_RES_MEM_2;
pMemIoBuf->Offset         = 0;

result = tdrv004ReadU16(   pDev,
                           pMemIoBuf );

…
```

…

```
if (result != ERROR)
{
     /* function succeeded */
     pValues = (unsigned short*)pMemIoBuf->pData;
} else {
     /* handle the error */
}
free( pMemIoBuf );
```

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	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.13 tdrv004ReadU32()

Name

tdrv004ReadU32() – Read 32bit values from FPGA resource

Synopsis

```
STATUS tdrv004ReadU32
(
    TDRV004_DEV             pDev,
    TDRV004_MEMIO_BUF       *pMemIoBuf
);
```

Description

This function reads a number of *unsigned long* (32bit) values from a Memory or I/O area by using DWORD (32bit) accesses. The data buffer can be enlarged to the desired needs. The data section must be included inside the structure.

Parameters

pDev

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

pMemIoBuf

This parameter specifies a pointer to a TDRV004_MEMIO_BUF structure:

```
typedef struct
{
    TDRV004_RESOURCE  Resource;
    unsigned long     Offset;
    unsigned long     Size;
    unsigned char     pData[1];    /* dynamically expandable */
} TDRV004_MEMIO_BUF;
```

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX PCI9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type	TDRV004_RESOURCE
0	IO *(reserved)*	TDRV004_RES_IO_1
1	MEM *(reserved)*	TDRV004_RES_MEM_1
2	MEM *(used by VHDL Example)*	TDRV004_RES_MEM_2
3	IO *(not implemented by default)*	TDRV004_RES_IO_2
4	IO *(not implemented by default)*	TDRV004_RES_IO_3
5	MEM *(not implemented by default)*	TDRV004_RES_MEM_3

The PLX PCI9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the memory or I/O space specified by *Resource*.

Size

This value specifies the amount of data items to read.

pData

The received values are copied into this buffer. It must be large enough to hold the specified amount of data.

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;
TDRV004_MEMIO_BUF  pMemIoBuf;
unsigned int       *pValues;
int                BufferSize;

/*
** read 50 32bit dwords from MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + read data
*/
BufferSize    = ( sizeof(TDRV004_MEMIO_BUF) + 50*sizeof(unsigned int) );
pMemIoBuf     = (TDRV004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size           = 50;
pMemIoBuf->Resource       = TDRV004_RES_MEM_2;
pMemIoBuf->Offset         = 0;

result = tdrv004ReadU32(   pDev,
                           pMemIoBuf );

…
```

…

```
if (result != ERROR)
{
     /* function succeeded */
     pValues = (unsigned int*)pMemIoBuf->pData;
} else {
     /* handle the error */
}
free( pMemIoBuf );
```

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	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.14 tdrv004WriteU8()

Name

tdrv004WriteU8() – Write 8bit values to FPGA resource

Synopsis

```
STATUS tdrv004WriteU8
(
    TDRV004_DEV             pDev,
    TDRV004_MEMIO_BUF       *pMemIoBuf
);
```

Description

This function writes a number of *unsigned char* (8bit) values to a Memory or I/O area by using BYTE (8bit) accesses. The data buffer can be enlarged to the desired needs. The data section must be included inside the structure.

Parameters

pDev

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

pMemIoBuf

This parameter specifies a pointer to a TDRV004_MEMIO_BUF structure:

```
typedef struct
{
    TDRV004_RESOURCE  Resource;
    unsigned long     Offset;
    unsigned long     Size;
    unsigned char     pData[1];    /* dynamically expandable */
} TDRV004_MEMIO_BUF;
```

Resource

Specifies the desired PCI resource to write to. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX PCI9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type	TDRV004_RESOURCE
0	IO *(reserved)*	TDRV004_RES_IO_1
1	MEM *(reserved)*	TDRV004_RES_MEM_1
2	MEM *(used by VHDL Example)*	TDRV004_RES_MEM_2
3	IO *(not implemented by default)*	TDRV004_RES_IO_2
4	IO *(not implemented by default)*	TDRV004_RES_IO_3
5	MEM *(not implemented by default)*	TDRV004_RES_MEM_3

The PLX PCI9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the memory or I/O space specified by *Resource*.

Size

This value specifies the amount of data items to read.

pData

The values are copied from this buffer. It must be large enough to hold the specified amount of data.

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;
TDRV004_MEMIO_BUF  pMemIoBuf;
unsigned char      *pValues;
int                BufferSize;

/*
** write 10 byte to MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize    = ( sizeof(TDRV004_MEMIO_BUF) + 10*sizeof(unsigned char) );
pMemIoBuf     = (TDRV004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size           = 10;
pMemIoBuf->Resource       = TDRV004_RES_MEM_2;
pMemIoBuf->Offset         = 0;
pValues = (unsigned char*)pMemIoBuf->pData;
pValues[0] = 0x01;
pValues[1] = 0x02;
…
```

…

```
result = tdrv004WriteU8(   pDev,
                           pMemIoBuf );
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
free( pMemIoBuf );
```

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	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.15 tdrv004WriteU16()

Name

tdrv004WriteU16() – Write 16bit values to FPGA resource

Synopsis

```
STATUS tdrv004WriteU16
(
    TDRV004_DEV             pDev,
    TDRV004_MEMIO_BUF       *pMemIoBuf
);
```

Description

This function writes a number of *unsigned short* (16bit) values to a Memory or I/O area by using WORD (16bit) accesses. The data buffer can be enlarged to the desired needs. The data section must be included inside the structure.

Parameters

pDev

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

pMemIoBuf

This parameter specifies a pointer to a TDRV004_MEMIO_BUF structure:

```
typedef struct
{
    TDRV004_RESOURCE  Resource;
    unsigned long     Offset;
    unsigned long     Size;
    unsigned char     pData[1];    /* dynamically expandable */
} TDRV004_MEMIO_BUF;
```

Resource

Specifies the desired PCI resource to write to. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX PCI9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type	TDRV004_RESOURCE
0	IO *(reserved)*	TDRV004_RES_IO_1
1	MEM *(reserved)*	TDRV004_RES_MEM_1
2	MEM *(used by VHDL Example)*	TDRV004_RES_MEM_2
3	IO *(not implemented by default)*	TDRV004_RES_IO_2
4	IO *(not implemented by default)*	TDRV004_RES_IO_3
5	MEM *(not implemented by default)*	TDRV004_RES_MEM_3

The PLX PCI9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the memory or I/O space specified by *Resource*.

Size

This value specifies the amount of data items to read.

pData

The values are copied from this buffer. It must be large enough to hold the specified amount of data.

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;
TDRV004_MEMIO_BUF  pMemIoBuf;
unsigned short     *pValues;
int                BufferSize;

/*
** write 10 16bit words to MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize    = ( sizeof(TDRV004_MEMIO_BUF) + 10*sizeof(unsigned short) );
pMemIoBuf     = (TDRV004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size           = 10;
pMemIoBuf->Resource       = TDRV004_RES_MEM_2;
pMemIoBuf->Offset         = 0;
pValues = (unsigned short*)pMemIoBuf->pData;
pValues[0] = 0x0001;
pValues[1] = 0x0002;
…
```

…

```
result = tdrv004WriteU16(  pDev,
                           pMemIoBuf );
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
free( pMemIoBuf );
```

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	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.16 tdrv004WriteU32()

Name

tdrv004WriteU32() – Write 32bit values to FPGA resource

Synopsis

```
STATUS tdrv004WriteU32
(
    TDRV004_DEV             pDev,
    TDRV004_MEMIO_BUF       *pMemIoBuf
);
```

Description

This function writes a number of *unsigned long* (32bit) values to a Memory or I/O area by using DWORD (32bit) accesses. The data buffer can be enlarged to the desired needs. The data section must be included inside the structure.

Parameters

pDev

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

pMemIoBuf

This parameter specifies a pointer to a TDRV004_MEMIO_BUF structure:

```
typedef struct
{
    TDRV004_RESOURCE  Resource;
    unsigned long     Offset;
    unsigned long     Size;
    unsigned char     pData[1];    /* dynamically expandable */
} TDRV004_MEMIO_BUF;
```

Resource

Specifies the desired PCI resource to write to. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX PCI9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type	TDRV004_RESOURCE
0	IO *(reserved)*	TDRV004_RES_IO_1
1	MEM *(reserved)*	TDRV004_RES_MEM_1
2	MEM *(used by VHDL Example)*	TDRV004_RES_MEM_2
3	IO *(not implemented by default)*	TDRV004_RES_IO_2
4	IO *(not implemented by default)*	TDRV004_RES_IO_3
5	MEM *(not implemented by default)*	TDRV004_RES_MEM_3

The PLX PCI9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the memory or I/O space specified by *Resource*.

Size

This value specifies the amount of data items to read.

pData

The values are copied from this buffer. It must be large enough to hold the specified amount of data.

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;
TDRV004_MEMIO_BUF  pMemIoBuf;
unsigned int       *pValues;
int                BufferSize;

/*
** write 10 32bit dwords to MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize    = ( sizeof(TDRV004_MEMIO_BUF) + 10*sizeof(unsigned int) );
pMemIoBuf     = (TDRV004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size           = 10;
pMemIoBuf->Resource       = TDRV004_RES_MEM_2;
pMemIoBuf->Offset         = 0;
pValues = (unsigned int*)pMemIoBuf->pData;
pValues[0] = 0x00000001;
pValues[1] = 0x00000002;
…
```

…

```
result = tdrv004WriteU32(  pDev,
                           pMemIoBuf );
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
free( pMemIoBuf );
```

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	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

3.2.17 tdrv004ConfigureInterrupts()

Name

tdrv004ConfigureInterrupts() – Configure local interrupt source polarity

Synopsis

```
STATUS tdrv004SetWaitstates
(
    TDRV004_DEV         pDev,
    unsigned int        InterruptConfig
);
```

Description

This function configures the polarity of the local PLX PCI9030 interrupt sources.

Parameters

pDev

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

InterruptConfig

This value is an OR'ed value using the following definitions (only one value valid for each interrupt source):

Value	Description
TDRV004_LINT1_POLHIGH	Local Interrupt Source 1 HIGH active
TDRV004_LINT1_POLLOW	Local Interrupt Source 1 LOW active
TDRV004_LINT2_POLHIGH	Local Interrupt Source 2 HIGH active
TDRV004_LINT2_POLLOW	Local Interrupt Source 2 LOW active

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;
unsigned int       IntConfig;

/*
** Setup LINT1 to LOW polarity, and LINT2 to HIGH polarity
*/
IntConfig = TDRV004_LINT1_POLLOW | TDRV004_LINT2_POLHIGH;

result = tdrv004ConfigureInterrupts( pDev, IntConfig );

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
EINVAL	Invalid interrupt configuration specified.

3.2.18 tdrv004WaitForINT1()

Name

tdrv004WaitForINT1() – Wait for incoming Local Interrupt Source 1

Synopsis

```
STATUS tdrv004WaitForINT1
(
    TDRV004_DEV     pDev,
    int             Timeout
);
```

Description

This function enables the corresponding interrupt source, and waits for Local Interrupt Source 1 (LINT1) to arrive. After the interrupt has arrived, this specific local interrupt source is disabled.

The delay between an incoming interrupt and the return of the described function is system-dependent, and is most likely several microseconds.

For high interrupt load, a customized device driver should be used which serves the module-specific functionality directly on interrupt level.

Parameters

pDev

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

Timeout

This value specifies the amount of time to wait for the incoming interrupt event. The timeout value is specified in system ticks. To wait indefinitely, specify -1.

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;
int                Timeout;

/*
** Wait at least 5 seconds for incoming interrupt
*/
Timeout = 5 * sysClkRateGet();

result = tdrv004WaitForINT1( pDev, Timeout );

if (result != ERROR)
{
     /* function succeeded */
     /* acknowledge interrupt source in FPGA logic       */
     /* to clear the PLX PCI9030 Local Interrupt Source  */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
ETIME	The interrupt did not arrive before the specified timeout.
EBUSY	There is already a job waiting for this interrupt.

3.2.19 tdrv004WaitForINT2()

Name

tdrv004WaitForINT2() – Wait for incoming Local Interrupt Source 2

Synopsis

```
STATUS tdrv004WaitForINT2
(
    TDRV004_DEV     pDev,
    int             Timeout
);
```

Description

This function enables the corresponding interrupt source, and waits for Local Interrupt Source 2 (LINT2) to arrive. After the interrupt has arrived, this specific local interrupt source is disabled.

The delay between an incoming interrupt and the return of the described function is system-dependent, and is most likely several microseconds.

For high interrupt load, a customized device driver should be used which serves the module-specific functionality directly on interrupt level.

Parameters

pDev

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

Timeout

This value specifies the amount of time to wait for the incoming interrupt event. The timeout value is specified in system ticks. To wait indefinitely, specify -1.

Example

```
#include “tdrv004.h”

TDRV004_DEV        pDev;
STATUS             result;
int                Timeout;

/*
** Wait at least 5 seconds for incoming interrupt
*/
Timeout = 5 * sysClkRateGet();

result = tdrv004WaitForINT2( pDev, Timeout );

if (result != ERROR)
{
     /* function succeeded */
     /* acknowledge interrupt source in FPGA logic       */
     /* to clear the PLX PCI9030 Local Interrupt Source  */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
ETIME	The interrupt did not arrive before the specified timeout.
EBUSY	There is already a job waiting for this interrupt.

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 TDRV004 driver. For the VxBus-enabled TDRV004 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.

4.1 tdrv004Drv()

NAME

tdrv004Drv() - installs the TDRV004 driver in the I/O system

SYNOPSIS

#include “tdrv004.h”

STATUS tdrv004Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus and installs the TDRV004 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 "tdrv004.h”

/*-------------------
  Initialize Driver
  -------------------*/
status = tdrv004Drv();
if (status == 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 ALSO

VxWorks Programmer's Guide: I/O System

4.2 tdrv004DevCreate()

NAME

tdrv004DevCreate() – Add a TDRV004 device to the VxWorks system

SYNOPSIS

```
#include “tdrv004.h”

STATUS tdrv004DevCreate
(
    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.
If modules of the same type are installed the channel numbers will be advised in the order the VxWorks *pciFindDevice()* function will find the devices.

Example: A system with 2x TPMC630-10 and 1x TCP630-10 will assign the following device indices:

Module	Device Index
TPMC630-10 (1st)	0
TPMC630-10 (2nd)	1
TCP630-10	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 "tdrv004.h”

STATUS            result;

/*-------------------------------------------------------
  Create the device "/tdrv004/0" for the first TDRV004 device
  -------------------------------------------------------*/

result = tdrv004DevCreate( "/tdrv004/0",
                           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
ENXIO	No matching device found.
EEXIST	The device has already been created by a previous call to tdrv004DevCreate().
S_ioLib_NO_DRIVER	Device driver not installed.

SEE ALSO

VxWorks Programmer’s Guide: I/O System

4.3 tdrv004PciInit()

NAME

tdrv004PciInit() – Generic PCI device initialization

SYNOPSIS

void tdrv004PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC630 family PCI spaces (base address register) and to enable the TDRV004 device for access.

The global variable *tdrv004Status* 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 tdrv004Status is equal to the number of mapped PCI spaces
0	No TDRV004 device found
< 0	Initialization failed. The value of (tdrv004Status & 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 tdrv004PciInit();

…

tdrv004PciInit();

…
```

5 Basic I/O Functions

The VxWorks basic I/O interface functions are useable with the TDRV004 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 TDRV004 device, a file descriptor must be opened by invoking the basic I/O function *open().*

PARAMETER

name

Specifies the device which shall be opened

flags

Not used

mode

Not used

EXAMPLE

```
int  fd;

/*----------------------------------------
  Open the device named "/tdrv004/0" for I/O
  ----------------------------------------*/
fd = open("/tdrv004/0", 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

Error codes 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

Error codes 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 “tdrv004.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_TDRV004_XSVFPLAY	Play an XSVF file for FPGA programming
FIO_TDRV004_XSVFPOS	Retrieve current play-position in XSVF file
FIO_TDRV004_XSVFLASTCMD	Get the last executed XSVF command
FIO_TDRV004_RECONFIG	Trigger FPGA reconfiguration process
FIO_TDRV004_SETWAITSTATES	Specify number of waitstates for programming
FIO_TDRV004_SETCLOCK	Set clock generator parameters
FIO_TDRV004_SPIWRITE	Write values to clock generator
FIO_TDRV004_SPIREAD	Read values from clock generator
FIO_TDRV004_PLXWRITE	Write 16bit value to PLX PCI9030 EEPROM
FIO_TDRV004_PLXREAD	Read 16bit value from PLX PCI9030 EEPROM
FIO_TDRV004_READ_UCHAR	Read 8bit values from FPGA resource
FIO_TDRV004_READ_USHORT	Read 16bit values from FPGA resource
FIO_TDRV004_READ_ULONG	Read 32bit values from FPGA resource

FIO_TDRV004_WRITE_UCHAR	Write 8bit values to FPGA resource
FIO_TDRV004_WRITE_USHORT	Write 16bit values to FPGA resource
FIO_TDRV004_WRITE_ULONG	Write 32bit values to FPGA resource
FIO_TDRV004_CONFIGURE_INT	Configure local interrupt source polarity
FIO_TDRV004_WAIT_FOR_INT1	Wait for incoming Local Interrupt Source 1
FIO_TDRV004_WAIT_FOR_INT2	Wait for incoming Local Interrupt Source 2

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.

SEE ALSO

ioLib, basic I/O routine - ioctl()

5.3.1 FIO_TDRV004_XSVFPLAY

This I/O control function programs the FPGA with a supplied XSVF file. The function specific control parameter **arg** is a pointer to a *TDRV004_XSVF_BUF* structure. For information on building an XSVF file, please refer to the Engineering Documentation of the TDRV004 product family.

The device driver is not able to verify the supplied XSVF file content, so please make sure that the supplied XSVF is of a valid file format.

```
typedef struct
{
        unsigned long       size;
        unsigned char       pData[1];     /* dynamically expandable */
} TDRV004_XSVF_BUF;
```

size

Specifies the total size of the supplied XSVF data.

pData

This dynamically expandable array holds the XSVF data. The data must be included inside the TDRV004_XSVF_BUF structure.

Programming Hints

Depending on the XSVF file, there might be a waiting period of approx. 15 seconds at the beginning of programming. The programming of the delivered FPGA example design XSVF file should not take much longer than 1 minute, depending on the system load.

If the programming fails, try to increase the used waitstates with control function FIO_TDRV004_SETWAITSTATES (refer to the corresponding section in this manual). Additionally, the CLK1 should not be lower than 10MHz for programming.

EXAMPLE

```
#include “tdrv004.h”

int                 fd;
TDRV004_XSVF_BUF    *pXsvfBuf;
int                 bufsize;
int                 retval;

…
```

```
…

/*
** allocate enough memory (about 3MB) to hold XSVF content
*/
bufsize = sizeof(TDRV004_XSVF_BUF) + <filesize> * sizeof(unsigned char);
pXsvfBuf = (TDRV004_XSVF_BUF*)malloc( bufsize );

/*
** read XSVF content from file and store it inside pXsvfBuf->pData[]
*/
pXsvfBuf->pData    = ...
pXsvfBuf->size     = ...

/*
** start FPGA programming
*/
retval = ioctl(fd, FIO_TDRV004_XSVFPLAY, (int)pXsvfBuf);
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
free( pXsvfBuf );
```

ERROR CODES

Error code	Description
EINVAL	There was an error during XSVF processing.
EINTR	The function was cancelled.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.2 FIO_TDRV004_XSVFPOS

This I/O control function returns the number of the current processed byte in the XSVF file during programming with TDRV004_IOCS_XSVFPLAY. This control function can be used to monitor the programming progress. The function specific control parameter **arg** is a pointer to an unsigned int value.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
unsigned int       XsvfPos;
int                retval;


/*------------------------
  Execute ioctl() function
  ------------------------*/

retval = ioctl(fd, FIO_TDRV004_XSVFPOS, (int)&XsvfPos);
if (retval != ERROR)
{
     /* function succeeded */
     printf(“Current XSVF position: %ld\n”, XsvfPos);
} else {
     /* handle the error */
}
```

ERROR CODES

This ioctl function returns no function specific error codes.

5.3.3 FIO_TDRV004_XSVFLASTCMD

This I/O control function returns the number of the last executed XSVF command. This value can be used to find errors inside the supplied XSVF file. This value refers to the line inside the ASCII SVF file. The function specific control parameter **arg** is a pointer to an unsigned int value.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
unsigned int       XsvfLastCmd;
int                retval;


/*------------------------
  Execute ioctl() function
  ------------------------*/

retval = ioctl(fd, FIO_TDRV004_ XSVFLASTCMD, (int)&XsvfLastCmd);
if (retval != ERROR)
{
     /* function succeeded */
     printf(“Last XSVF command: %ld\n”, XsvfLastCmd);
} else {
     /* handle the error */
}
```

ERROR CODES

This ioctl function returns no function specific error codes.

5.3.4 FIO_TDRV004_RECONFIG

This I/O control function starts the reconfiguration process of the FPGA. This control function must be called after the FPGA is programmed using FIO_TDRV004_XSVFPLAY. The function returns after the reconfiguration is done, or an error occurred. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
int                retval;


/*------------------------
  Execute ioctl() function
  ------------------------*/

retval = ioctl(fd, FIO_TDRV004_RECONFIG, 0);
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EIO	An error occurred during reconfiguration. This may be caused by an invalid FPGA content located inside the XSVF file.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.5 FIO_TDRV004_SETWAITSTATES

This I/O control function configures the driver to use a number of waitstates during XSVF and SPI programming. This might be necessary, if the local clock (CLK1) of the onboard clock generator is configured to rather slow. The local programming interface is clocked with this frequency, which might result in errors during programming for low CLK1 frequencies and a small amount of waitstates. The system architecture (existing PCI-to-PCI bridges) might also have an impact. The function specific control parameter **arg** is a pointer to an unsigned int value. Valid values are 1 to 1000.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
unsigned long      WaitStates;
int                retval;


/*
**  configure driver to use 3 waitstates
*/
WaitStates = 3;

retval = ioctl(fd, FIO_TDRV004_SETWAITSTATES, (int)&WaitStates);
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
```

ERROR CODES

This ioctl function returns no function specific error codes.

5.3.6 FIO_TDRV004_SETCLOCK

This I/O control function configures the onboard clock generator. The function specific control parameter **arg** is a pointer to a *TDRV004_CLOCK_PARAM* structure.

```
typedef struct
{
        unsigned char       DeviceAddr;
        unsigned char       x09_ClkOE;
        unsigned char       x0C_DIV1SRCN;
        unsigned char       x10_InputCtrl;
        unsigned char       x40_CPumpPB;
        unsigned char       x41_CPumpPB;
        unsigned char       x42_POQcnt;
        unsigned char       x44_SwMatrix;
        unsigned char       x45_SwMatrix;
        unsigned char       x46_SwMatrix;
        unsigned char       x47_DIV2SRCN;
} TDRV004_CLOCK_PARAM;
```

DeviceAddr

Specifies the desired destination address. The CY27EE16 clock generator provides several EEPROM banks as well as SRAM. If TDRV004_CLKADR_SRAM is specified, the values are directly stored inside the volatile RAM area and take effect immediately. If TDRV004_CLKADR_EEPROM is specified, the values are stored in the non-volatile area of the clock generator, and the CY27EE16 loads it after the next power-up.

x09_ClkOE

Specifies which clock outputs shall be enabled.

x0C_DIV1SRCN

Specifies internal input source 1 and the corresponding frequency divider

x10_InputCtrl

Specifies value for the Input Pin Control register

x40_CPumpPB

Specifies value for Charge Pump and PB counter register

x41_CPumpPB

Specifies value for Charge Pump and PB counter register

x42_POQcnt

Specifies value for PO and Q counter register

x44_SwMatrix

Specifies value for Switching Matrix Register

x45_SwMatrix

Specifies value for Switching Matrix Register

x46_SwMatrix

Specifies value for Switching Matrix Register

x47_DIV2SRCN

Specifies internal input source 2 and the corresponding frequency divider

Please refer to the Cypress CY27EE16 user manual for detailed explanation of the above register values. Use Cypress' *CyberClocks* Version R3.10.00 to determine the correct values. This program is also part of the TPMC630 or TCP630 Engineering Documentation.

EXAMPLE

```
#include "tdrv004.h"

int                         fd;
TDRV004_CLOCK_PARAM         ClockParam;
int                         retval;


/*
** Setup clock generator (SRAM):
**   CLK1: 50.0MHz       CLK2: 20.0MHz
**   CLK3: 10.0MHz       CLK4:  1.0MHz
**   CLK5:  0.2MHz       CLK6: -off-
*/
ClockParam.DeviceAddress   = TDRV004_CLKADR_SRAM;
ClockParam.x09_ClkOE       = 0x6f;
ClockParam.x0C_DIV1SRCN    = 0x64;
ClockParam.x10_InputCtrl   = 0x50;
ClockParam.x40_CPumpPB     = 0xc0;
ClockParam.x41_CPumpPB     = 0x03;
ClockParam.x42_POQcnt      = 0x81;
ClockParam.x44_SwMatrix    = 0x42;
ClockParam.x45_SwMatrix    = 0x9f;
ClockParam.x46_SwMatrix    = 0x3f;
ClockParam.x47_DIV2SRCN    = 0xe4;

retval = ioctl(fd, FIO_TDRV004_SETCLOCK, (int)&ClockParam);
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EINVAL	It was tried to disable CLK1. This is not allowed.
EIO	An error occurred during SPI access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.7 FIO_TDRV004_SPIWRITE

This I/O control function writes up to 256 *unsigned char* values to a specific sub-address of a Serial Programming Interface (SPI) address. The function specific control parameter **arg** is a pointer to a *TDRV004_SPI_BUF* structure.

```
typedef struct
{
        unsigned char     SpiAddr;
        unsigned char     SubAddr;
        unsigned long     len;
        unsigned char     pData[1];          /* dynamically expandable */
} TDRV004_SPI_BUF;
```

SpiAddr

Specifies the Serial Programming Interface (SPI) address of the desired target. The following values are possible (refer to file *tdrv004.h*):

Symbol	Value	Description
TDRV004_CLKADDR_EEPROM	0x68	Clock Generator EEPROM (non-volatile)
TDRV004_CLKADDR_SRAM	0x69	Clock Generator SRAM (volatile)
TDRV004_CLKADDR_EEBLOCK1	0x40	EEPROM-Bank 1
TDRV004_CLKADDR_EEBLOCK2	0x41	EEPROM-Bank 2
TDRV004_CLKADDR_EEBLOCK3	0x42	EEPROM-Bank 3
TDRV004_CLKADDR_EEBLOCK4	0x43	EEPROM-Bank 4
TDRV004_CLKADDR_EEBLOCK5	0x44	EEPROM-Bank 5
TDRV004_CLKADDR_EEBLOCK6	0x45	EEPROM-Bank 6
TDRV004_CLKADDR_EEBLOCK7	0x46	EEPROM-Bank 7
TDRV004_CLKADDR_EEBLOCK8	0x47	EEPROM-Bank 8

SubAddr

Specifies the sub-address (starting offset).

len

This value specifies the amount of data items to write. A maximum of 256 is allowed.

pData

The values are copied from this buffer. It must be large enough to hold the specified amount of data. The data must be stored inside the structure, no pointer allowed.

Do not use this control function to setup the clockgenerator. Please use control function FIO_TDRV004_SETCLOCK instead.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
int                BufferSize;
TDRV004_SPI_BUF    *pSpiBuf;
int                retval;

/*
** write 5 bytes to EEPROM block 1, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize = ( sizeof(TDRV004_SPI_BUF) + 5*sizeof(unsigned char) );
pSpiBuf = (TDRV004_SPI_BUF*)malloc( BufferSize );
pSpiBuf->SpiAddr   = TDRV004_CLKADDR_EEBLOCK1;
pSpiBuf->SubAddr   = 0x00;
pSpiBuf->len       = 5;
pSpiBuf->pData[0]  = 0x01;
pSpiBuf->pData[1]  = 0x02;
pSpiBuf->pData[2]  = 0x03;
pSpiBuf->pData[3]  = 0x04;
pSpiBuf->pData[4]  = 0x05;

retval = ioctl(fd, FIO_TDRV004_SPIWRITE, (int)pSpiBuf);
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
free( pSpiBuf );
```

ERROR CODES

Error code	Description
EINVAL	The specified SubAddr+len exceeds 256, or len is invalid
EIO	An error occurred during SPI access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.8 FIO_TDRV004_SPIREAD

This I/O control function reads up to 256 *unsigned char* values from a specific sub-address of a Serial Programming Interface (SPI) address. The function specific control parameter **arg** is a pointer to a *TDRV004_SPI_BUF* structure.

```
typedef struct
{
        unsigned char     SpiAddr;
        unsigned char     SubAddr;
        unsigned long     len;
        unsigned char     pData[1];          /* dynamically expandable */
} TDRV004_SPI_BUF;
```

SpiAddr

Specifies the Serial Programming Interface (SPI) address of the desired target. The following values are possible (refer to file *tdrv004.h*):

Symbol	Value	Description
TDRV004_CLKADDR_EEPROM	0x68	Clock Generator EEPROM (non-volatile)
TDRV004_CLKADDR_SRAM	0x69	Clock Generator SRAM (volatile)
TDRV004_CLKADDR_EEBLOCK1	0x40	EEPROM-Bank 1
TDRV004_CLKADDR_EEBLOCK2	0x41	EEPROM-Bank 2
TDRV004_CLKADDR_EEBLOCK3	0x42	EEPROM-Bank 3
TDRV004_CLKADDR_EEBLOCK4	0x43	EEPROM-Bank 4
TDRV004_CLKADDR_EEBLOCK5	0x44	EEPROM-Bank 5
TDRV004_CLKADDR_EEBLOCK6	0x45	EEPROM-Bank 6
TDRV004_CLKADDR_EEBLOCK7	0x46	EEPROM-Bank 7
TDRV004_CLKADDR_EEBLOCK8	0x47	EEPROM-Bank 8

SubAddr

Specifies the sub-address (starting offset).

len

This value specifies the amount of data items to read. A maximum of 256 is allowed.

pData

The values are copied to this buffer. It must be large enough to hold the specified amount of data. The data space must be located inside the structure, no pointer allowed.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
int                BufferSize;
TDRV004_SPI_BUF    *pSpiBuf;
int                retval;

/*
** read 5 bytes from EEPROM block 1, offset 0x00
** allocate enough memory to hold the data structure + read data
*/
BufferSize = ( sizeof(TDRV004_SPI_BUF) + 5*sizeof(unsigned char) );
pSpiBuf = (TDRV004_SPI_BUF*)malloc( BufferSize );
pSpiBuf->SpiAddr   = TDRV004_CLKADDR_EEBLOCK1;
pSpiBuf->SubAddr   = 0x00;
pSpiBuf->len       = 5;

retval = ioctl(fd, FIO_TDRV004_SPIREAD, (int)pSpiBuf);
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
free( pSpiBuf );
```

ERROR CODES

Error code	Description
EINVAL	The specified SubAddr+len exceeds 256, or len is invalid
EIO	An error occurred during SPI access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.9 FIO_TDRV004_PLXWRITE

This I/O control function writes an *unsigned short* value to a specific PLX PCI9030 EEPROM memory offset. The function specific control parameter **arg** is a pointer to a *TDRV004_PLX_BUF* structure.

```
typedef struct
{
        unsigned long       Offset;
        unsigned short      Value;
} TDRV004_PLX_BUF;
```

Offset

Specifies the offset into the PLX9030 EEPROM, where the supplied data word should be written. The offset must be specified as even byte-address.
Following offsets are available:

Offset	Access
00h – 0Ch	R
0Eh	R / W
10h – 26h	R
28h – 36h	R / W
38h – 3Ah	R
3Ch – 4Ah	R / W
4Ch – 4Eh	R
50h – 5Eh	R / W
60h – 62h	R
64h – 7Eh	R / W
80h – 86h	R
88h - FEh	R / W

Refer to the PLX PCI9030 User Manual for detailed information on these registers.

Value

This value specifies a 16bit word that should be written to the specified offset.

Note that the PLX PCI9030 reloads the new configuration from the EEPROM after a PCI reset, i.e. the system must be rebooted to make PLX PCI9030 dependent changes take effect.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
TDRV004_PLX_BUF    PlxBuf;
int                retval;


/*
** Change the Subsystem Vendor ID to TEWS TECHNOLOGIES (0x1498)
*/
PlxBuf.Offset = 0x0E;
PlxBuf.Value  = 0x1498

retval = ioctl(fd, FIO_TDRV004_PLXWRITE, (int)&PlxBuf);
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EINVAL	The specified offset is invalid, or read-only
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.10 FIO_TDRV004_PLXREAD

This I/O control function reads an *unsigned short* value from a specific PLX PCI9030 EEPROM memory offset. The function specific control parameter **arg** is a pointer to a *TDRV004_PLX_BUF* structure.

```
typedef struct
{
        unsigned long       Offset;
        unsigned short      Value;
} TDRV004_PLX_BUF;
```

Offset

Specifies the offset into the PLX PCI9030 EEPROM, where the supplied data word should be read. The offset must be specified as even byte-address.
Following offsets are available:

Offset	**Access**
00h – 0Ch	R
0Eh	R / W
10h – 26h	R
28h – 36h	R / W
38h – 3Ah	R
3Ch – 4Ah	R / W
4Ch – 4Eh	R
50h – 5Eh	R / W
60h – 62h	R
64h – 7Eh	R / W
80h – 86h	R
88h - FEh	R / W

Refer to the PLX PCI9030 User Manual for detailed information on these registers.

Value

This value holds the retrieved 16bit word.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
TDRV004_PLX_BUF    PlxBuf;
int                retval;


/*
** Read Subsystem ID
*/
PlxBuf.Offset = 0x0C;

retval = ioctl(fd, FIO_TDRV004_PLXREAD, (int)&PlxBuf);
if (retval != ERROR)
{
     /* function succeeded */
     printf( “SubsystemID = 0x%04X\n”, PlxBuf.Value );
} else {
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EINVAL	The specified offset is invalid.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.11 FIO_TDRV004_READ_UCHAR

This I/O control function reads a number of *unsigned char* values from a Memory or I/O area by using BYTE (8bit) accesses. The function specific control parameter **arg** is a pointer to a *TDRV004_MEMIO_BUF* structure. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct
{
    TDRV004_RESOURCE  Resource;
    unsigned long     Offset;
    unsigned long     Size;
    unsigned char     pData[1];    /* dynamically expandable */
} TDRV004_MEMIO_BUF;
```

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TDRV004_RESOURCE
0	IO	*(reserved)*	TDRV004_RES_IO_1
1	MEM	*(reserved)*	TDRV004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TDRV004_RES_MEM_2
3	IO	*(not implemented by default)*	TDRV004_RES_IO_2
4	IO	*(not implemented by default)*	TDRV004_RES_IO_3
5	MEM	*(not implemented by default)*	TDRV004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the memory or I/O space specified by *Resource*.

Size

This value specifies the amount of data items to read.

pData

The received values are copied into this buffer. It must be large enough to hold the specified amount of data.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
unsigned long      BufferSize;
TDRV004_MEMIO_BUF  *pMemIoBuf;
unsigned char      *pValues;
int                retval;

/*
** read 50 bytes from MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + read data
*/
BufferSize    = ( sizeof(TDRV004_MEMIO_BUF) + 50*sizeof(unsigned char) );
pMemIoBuf     = (TDRV004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size            = 50;
pMemIoBuf->Resource        = TDRV004_RES_MEM_2;
pMemIoBuf->Offset          = 0;

retval = ioctl(fd, FIO_TDRV004_READ_UCHAR, (int)pMemIoBuf);
if (retval != ERROR)
{
     /* function succeeded */
     pValues = (unsigned char*)pMemIoBuf->pData;
} else {
     /* handle the error */
}
free( pMemIoBuf );
```

ERROR CODES

Error code	Description
EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.12 FIO_TDRV004_READ_USHORT

This I/O control function reads a number of *unsigned short* values from a Memory or I/O area by using WORD (16bit) accesses. The function specific control parameter **arg** is a pointer to a *TDRV004_MEMIO_BUF* structure. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct
{
    TDRV004_RESOURCE  Resource;
    unsigned long     Offset;
    unsigned long     Size;
    unsigned char     pData[1];    /* dynamically expandable */
} TDRV004_MEMIO_BUF;
```

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TDRV004_RESOURCE
0	IO	*(reserved)*	TDRV004_RES_IO_1
1	MEM	*(reserved)*	TDRV004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TDRV004_RES_MEM_2
3	IO	*(not implemented by default)*	TDRV004_RES_IO_2
4	IO	*(not implemented by default)*	TDRV004_RES_IO_3
5	MEM	*(not implemented by default)*	TDRV004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the memory or I/O space specified by *Resource*.

Size

This value specifies the amount of data items to read.

pData

The received values are copied into this buffer. It must be large enough to hold the specified amount of data. The data pointer is typecasted into an *unsigned short* pointer.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
unsigned long      BufferSize;
TDRV004_MEMIO_BUF  *pMemIoBuf;
unsigned short     *pValues;
int                retval;

/*
** read 50 16bit words from MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + read data
*/
BufferSize    = ( sizeof(TDRV004_MEMIO_BUF) + 50*sizeof(unsigned short) );
pMemIoBuf     = (TDRV004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size            = 50;
pMemIoBuf->Resource        = TDRV004_RES_MEM_2;
pMemIoBuf->Offset          = 0;

retval = ioctl(fd, FIO_TDRV004_READ_USHORT, (int)pMemIoBuf);
if (retval != ERROR)
{
     /* function succeeded */
     pValues = (unsigned short*)pMemIoBuf->pData;
} else {
     /* handle the error */
}
free( pMemIoBuf );
```

ERROR CODES

Error code	Description
EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.13 FIO_TDRV004_READ_ULONG

This I/O control function reads a number of *unsigned long* values from a Memory or I/O area by using DWORD (32bit) accesses. The function specific control parameter **arg** is a pointer to a *TDRV004_MEMIO_BUF* structure. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct
{
    TDRV004_RESOURCE  Resource;
    unsigned long     Offset;
    unsigned long     Size;
    unsigned char     pData[1];    /* dynamically expandable */
} TDRV004_MEMIO_BUF;
```

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TDRV004_RESOURCE
0	IO	*(reserved)*	TDRV004_RES_IO_1
1	MEM	*(reserved)*	TDRV004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TDRV004_RES_MEM_2
3	IO	*(not implemented by default)*	TDRV004_RES_IO_2
4	IO	*(not implemented by default)*	TDRV004_RES_IO_3
5	MEM	*(not implemented by default)*	TDRV004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the memory or I/O space specified by *Resource*.

Size

This value specifies the amount of data items to read.

pData

The received values are copied into this buffer. It must be large enough to hold the specified amount of data. The data pointer is typecasted into an *unsigned long* pointer.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
unsigned long      BufferSize;
TDRV004_MEMIO_BUF  *pMemIoBuf;
unsigned int       *pValues;
int                retval;

/*
** read 50 32bit dwords from MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + read data
*/
BufferSize    = ( sizeof(TDRV004_MEMIO_BUF) + 50*sizeof(unsigned int) );
pMemIoBuf     = (TDRV004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size            = 50;
pMemIoBuf->Resource        = TDRV004_RES_MEM_2;
pMemIoBuf->Offset          = 0;

retval = ioctl(fd, FIO_TDRV004_READ_ULONG, (int)pMemIoBuf);
if (retval != ERROR)
{
     /* function succeeded */
     pValues = (unsigned int*)pMemIoBuf->pData;
} else {
     /* handle the error */
}
free( pMemIoBuf );
```

ERROR CODES

Error code	Description
EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.14 FIO_TDRV004_WRITE_UCHAR

This I/O control function writes a number of *unsigned char* values to a Memory or I/O area by using BYTE (8bit) accesses. The function specific control parameter **arg** is a pointer to a *TDRV004_MEMIO_BUF* structure. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct
{
    TDRV004_RESOURCE  Resource;
    unsigned long     Offset;
    unsigned long     Size;
    unsigned char     pData[1];    /* dynamically expandable */
} TDRV004_MEMIO_BUF;
```

Resource

Specifies the desired PCI resource to write to. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TDRV004_RESOURCE
0	IO	*(reserved)*	TDRV004_RES_IO_1
1	MEM	*(reserved)*	TDRV004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TDRV004_RES_MEM_2
3	IO	*(not implemented by default)*	TDRV004_RES_IO_2
4	IO	*(not implemented by default)*	TDRV004_RES_IO_3
5	MEM	*(not implemented by default)*	TDRV004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the memory or I/O space specified by *Resource*.

Size

This value specifies the amount of data items to write.

pData

The values are copied from this buffer. It must be large enough to hold the specified amount of data.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
unsigned long      BufferSize;
TDRV004_MEMIO_BUF  *pMemIoBuf;
unsigned char      *pValues;
int                retval;

/*
** write 10 byte to MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize    = ( sizeof(TDRV004_MEMIO_BUF) + 10*sizeof(unsigned char) );
pMemIoBuf     = (TDRV004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size            = 10;
pMemIoBuf->Resource        = TDRV004_RES_MEM_2;
pMemIoBuf->Offset          = 0;
pValues = (unsigned char*)pMemIoBuf->pData;
pValues[0] = 0x01;
pValues[1] = 0x02;
…

retval = ioctl(fd, FIO_TDRV004_WRITE_UCHAR, (int)pMemIoBuf);
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
free( pMemIoBuf );
```

ERROR CODES

Error code	Description
EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.15 FIO_TDRV004_WRITE_USHORT

This I/O control function writes a number of *unsigned short* values to a Memory or I/O area by using WORD (16bit) accesses. The function specific control parameter **arg** is a pointer to a *TDRV004_MEMIO_BUF* structure. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct
{
    TDRV004_RESOURCE  Resource;
    unsigned long     Offset;
    unsigned long     Size;
    unsigned char     pData[1];    /* dynamically expandable */
} TDRV004_MEMIO_BUF;
```

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TDRV004_RESOURCE
0	IO	*(reserved)*	TDRV004_RES_IO_1
1	MEM	*(reserved)*	TDRV004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TDRV004_RES_MEM_2
3	IO	*(not implemented by default)*	TDRV004_RES_IO_2
4	IO	*(not implemented by default)*	TDRV004_RES_IO_3
5	MEM	*(not implemented by default)*	TDRV004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the memory or I/O space specified by *Resource*.

Size

This value specifies the amount of data items to write.

pData

The values are copied from this buffer. It must be large enough to hold the specified amount of data. The data pointer is typecasted into an *unsigned short* pointer.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
unsigned long      BufferSize;
TDRV004_MEMIO_BUF  *pMemIoBuf;
unsigned short     *pValues;
int                retval;

/*
** write 10 16bit words to MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize    = ( sizeof(TDRV004_MEMIO_BUF) + 10*sizeof(unsigned short) );
pMemIoBuf     = (TDRV004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size            = 10;
pMemIoBuf->Resource        = TDRV004_RES_MEM_2;
pMemIoBuf->Offset          = 0;
pValues = (unsigned char*)pMemIoBuf->pData;
pValues[0] = 0x0001;
pValues[1] = 0x0002;
…

retval = ioctl(fd, FIO_TDRV004_WRITE_USHORT, (int)pMemIoBuf);
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
free( pMemIoBuf );
```

ERROR CODES

Error code	Description
EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.16 FIO_TDRV004_WRITE_ULONG

This I/O control function writes a number of *unsigned long* values to a Memory or I/O area by using DWORD (32bit) accesses. The function specific control parameter **arg** is a pointer to a *TDRV004_MEMIO_BUF* structure. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct
{
    TDRV004_RESOURCE  Resource;
    unsigned long     Offset;
    unsigned long     Size;
    unsigned char     pData[1];    /* dynamically expandable */
} TDRV004_MEMIO_BUF;
```

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TDRV004_RESOURCE
0	IO	*(reserved)*	TDRV004_RES_IO_1
1	MEM	*(reserved)*	TDRV004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TDRV004_RES_MEM_2
3	IO	*(not implemented by default)*	TDRV004_RES_IO_2
4	IO	*(not implemented by default)*	TDRV004_RES_IO_3
5	MEM	*(not implemented by default)*	TDRV004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the memory or I/O space specified by *Resource*.

Size

This value specifies the amount of data items to write.

pData

The values are copied from this buffer. It must be large enough to hold the specified amount of data. The data pointer is typecasted into an *unsigned long* pointer.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
unsigned long      BufferSize;
TDRV004_MEMIO_BUF  *pMemIoBuf;
unsigned int       *pValues;
int                retval;

/*
** write 10 32bit dwords to MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize    = ( sizeof(TDRV004_MEMIO_BUF) + 10*sizeof(unsigned int) );
pMemIoBuf     = (TDRV004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size            = 10;
pMemIoBuf->Resource        = TDRV004_RES_MEM_2;
pMemIoBuf->Offset          = 0;
pValues = (unsigned int*)pMemIoBuf->pData;
pValues[0] = 0x00000001;
pValues[1] = 0x00000002;
…

retval = ioctl(fd, FIO_TDRV004_WRITE_ULONG, (int)pMemIoBuf);
if (retval != ERROR)
{
     /* function succeeded */
} else {
     /* handle the error */
}
free( pMemIoBuf );
```

ERROR CODES

Error code	Description
EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

5.3.17 FIO_TDRV004_CONFIGURE_INT

This TDRV004 control function configures the polarity of the PLX PCI9030 interrupt sources.

The function specific control parameter **arg** is a pointer to an *unsigned long* value. This value is an OR'ed value using the following definitions (only one value valid for each interrupt source):

Value	Description
TDRV004_LINT1_POLHIGH	Local Interrupt Source 1 HIGH active
TDRV004_LINT1_POLLOW	Local Interrupt Source 1 LOW active
TDRV004_LINT2_POLHIGH	Local Interrupt Source 2 HIGH active
TDRV004_LINT2_POLLOW	Local Interrupt Source 2 LOW active

EXAMPLE

```
#include "tdrv004.h"

int                fd;
int                retval;
unsigned int       IntConfig;

/*
** Setup LINT1 to LOW polarity, and LINT2 to HIGH polarity
*/
IntConfig = TDRV004_LINT1_POLLOW | TDRV004_LINT2_POLHIGH;

retval = ioctl(fd, FIO_TDRV004_CONFIGURE_INT, (int)&IntConfig);
if (retval == ERROR)
{
     /* handle the error */
}
```

ERROR CODES

This ioctl function returns no function specific error codes.

5.3.18 FIO_TDRV004_WAIT_FOR_INT1

This TDRV004 control function enables the corresponding interrupt source, and waits for Local Interrupt Source 1 (LINT1) to arrive. After the interrupt has arrived, this specific local interrupt source is disabled.

The function specific control parameter **arg** is a pointer to an *int* value containing the timeout in system ticks. To wait indefinitely, specify -1 as timeout parameter.

The delay between an incoming interrupt and the return of the described function is system-dependent, and is most likely several microseconds.

For high interrupt load, a customized device driver should be used which serves the module-specific functionality directly on interrupt level.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
int                retval;
int                Timeout;

/*
** Wait at least 5 seconds for incoming interrupt
*/
Timeout = 5 * sysClkRateGet();

retval = ioctl(fd, FIO_TDRV004_WAIT_FOR_INT1, (int)&Timeout);
if (retval == ERROR)
{
     /* acknowledge interrupt source in FPGA logic       */
     /* to clear the PLX PCI9030 Local Interrupt Source  */
} else {
     /* handle the error */
}
```

ERROR CODES

ETIME	The interrupt did not arrive before the specified timeout.
EBUSY	There is already a job waiting for this interrupt.

5.3.19 FIO_TDRV004_WAIT_FOR_INT2

This TDRV004 control function enables the corresponding interrupt source, and waits for Local Interrupt Source 2 (LINT2) to arrive. After the interrupt has arrived, this specific local interrupt source is disabled.

The function specific control parameter **arg** is a pointer to an *int* value containing the timeout in system ticks. To wait indefinitely, specify -1 as timeout parameter.

The delay between an incoming interrupt and the return of the described function is system-dependent, and is most likely several microseconds.

For high interrupt load, a customized device driver should be used which serves the module-specific functionality directly on interrupt level.

EXAMPLE

```
#include “tdrv004.h”

int                fd;
int                retval;
int                Timeout;

/*
** Wait at least 5 seconds for incoming interrupt
*/
Timeout = 5 * sysClkRateGet();

retval = ioctl(fd, FIO_TDRV004_WAIT_FOR_INT2, (int)&Timeout);
if (retval == OK)
{
     /* acknowledge interrupt source in FPGA logic       */
     /* to clear the PLX PCI9030 Local Interrupt Source  */
} else {
     /* handle the error */
}
```

ERROR CODES

ETIME	The interrupt did not arrive before the specified timeout.
EBUSY	There is already a job waiting for this interrupt.