The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV004-SW-72

LynxOS Device Driver

Reconfigurable FPGA

Version 1.1.x

User Manual

Issue 1.1.0

April 2008

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
25469 Halstenbek, Germany
www.tews.com
Phone: +49 (0) 4101 4058 0
Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com

TEWS TECHNOLOGIES LLC
9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com
Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

TDRV004-SW-72

LynxOS Device Driver

Reconfigurable FPGA

Supported Modules:
TPMC630
TCP630

Issue	Description	Date
1.0.0	First Issue	January 25, 2006
1.1.0	General revision, file list changed, interrupt feature added	April 22, 2008

Table of Contents

1 Introduction

The TDRV004-SW-72 LynxOS device driver allows the operation of the TPMC630 product family on LynxOS platforms with DRM based PCI interface.

This driver was successfully tested on a TEWS TECHNOLOGIES TVME8240 PowerPC board and on an Intel x86 native system.

The standard file (I/O) functions (open, close, ioctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and configuration operations.

The TDRV004 device driver includes the following functions:

- 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
- Wait for interrupt events on LINT1 or LINT2

The TDRV004-SW-72 supports the modules listed below:

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

In this document all supported modules and devices will be called TDRV004. Specials for a certain device 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 (or compatible) User manual

TPMC630 (or compatible) Engineering Manual

PLX PCI9030 User Manual

2 Installation

Following files are located in the directory TDRV004-SW-72 on the distribution media:

TDRV004-SW-72-1.1.0.pdf	This manual in PDF format
TDRV004-SW-72-SRC.tar	Device Driver and Example sources
fpgaexa.tar.gz	FPGA example XSVF
ChangeLog.txt	Release history
Release.txt	Information about the Device Driver Release

The TAR archive TDRV004-SW-72-SRC.tar contains the following files and directories:

tdrv004.c	Driver source code
tdrv004.h	Definitions and data structures for driver and application
tdrv004def.h	Definitions and data structures for the driver
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
tdrv004_info.c	Device information definition
tdrv004_info.h	Device information definition header
tdrv004.cfg	Driver configuration file include
tdrv004.import	Linker import file
Makefile	Device driver make file
example/tdrv004exa.c	Example application source

In order to perform a driver installation first extract the TAR file to a temporary directory, then copy the following files to their target directories:

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/tdrv004 or /sys/drivers.cpci_x86/tdrv004

2. Copy the following files to this directory:
   - tdrv004.c
   - tdrv004def.h
   - pf_micro.c, pf_micro.h
   - pf_lenval.c, pf_lenval.h
   - pf_ports.c, pf_ports.h
   - tdrv004.import
   - Makefile

3. Copy tdrv004.h to /usr/include/

4. Copy tdrv004_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tdrv004_info.h to /sys/dheaders/

6. Copy tdrv004.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform. For example: /sys/cfg.ppc or /sys/cfg.x86 ....

2.1 Device Driver Installation

The two methods of driver installation are as follows:

- Static Installation
- Dynamic Installation (only native LynxOS systems)

2.1.1 Static Installation

With this method, the driver object code is linked with the kernel routines and is installed during system start-up.

2.1.1.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tdrv004, where xxx represents the BSP that supports the target hardware.
2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

2.1.1.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).
2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = ... tdrv004_info.x
   ```

   And at the end of the Makefile

   ```
   tdrv004_info.o:$(DHEADERS)/tdrv004_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

2.1.1.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os/ respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry at the end of this file.

   ```
   I:tdrv004.cfg
   ```

2.1.1.4 Rebuild the Kernel

1. Change to the directory /sys/lynx.os/ (/sys/bsp.xxx)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

   The N flag instructs init to run mknod and create all the nodes mentioned in the new nodetab.

4. After reboot you should find the following new devices (depends on the device configuration): /dev/td004a, /dev/td004b, …

2.1.2 Dynamic Installation

This method allows you to install the driver after the operating system is booted. The driver object code is attached to the end of the kernel image and the operating system dynamically adds this driver to its internal structures. The driver can also be removed dynamically.

2.1.2.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tdrv004, where xxx represents the BSP that supports the target hardware.
2. To make the dynamic link-able driver enter :

   ```
   make dldd
   ```

2.1.2.2 Create Device Information Declaration

1. Change to the directory /sys/drivers.xxx/tdrv004, where xxx represents the BSP that supports the target hardware.
2. To create a device definition file for the major device (this works only on native system)

   ```
   make t004info
   ```

3. To install the driver enter:

   ```
   drinstall –c tdrv004.obj
   ```

   If successful, drinstall returns a unique <driver-ID>
4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> t004info
   ```

   The <driver-ID> is returned by the drinstall command
5. To create a node for the device enter:

   ```
   mknod /dev/td004a c <major_no> 0
   ```

   The <major_no> is returned by the devinstall command.

If all steps are successfully completed, the TDRV004 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TDRV004 device enter the following commands:

```
devinstall –u –c <device-ID>
drinstall –u <driver-ID>
```

2.1.3 Device Information Definition File

The device information definition contains information necessary to install the TDRV004 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tdrv004_info.h*.

This structure contains the following parameter:

PCIBusNumber Contains the PCI bus number at which the TDRV004 compatible device is connected. Valid bus numbers are in range from 0 to 255.

PCIDeviceNumber Contains the device number (slot) at which the TDRV004 compatible device is connected. Valid device numbers are in range from 0 to 31.

If both PCIBusNumber and PCIDeviceNumber are –1 then the driver will auto scan for the TPMC630 compatible device. The first device found in the scan order will be allocated by the driver for this major device.

Already allocated devices can't be allocated twice. This is important to know if there are more than one TDRV004 major devices.

A device information definition is unique for every TDRV004 major device. The file *tdrv004_info.c* on the distribution disk contains two device information declarations, **td004a_info** for the first major device and **td004b_info** for the second major device.

If the driver should support more than two major devices it is necessary to copy and paste an existing declaration and rename it with a unique name, for example **td004c_info**, **td004d_info** and so on.

It is also necessary to modify the device and driver configuration file, respectively the configuration include file *tdrv004.cfg*.

The following device declaration information uses the auto find method to detect a TDRV004 compatible device on the PCI bus.

```
TD004_INFO td004a_info = {

    -1,                 /*  Auto find the device on any PCI bus    */
    -1,

};
```

2.1.4 Configuration File: CONFIG.TBL

The device and driver configuration file CONFIG.TBL contains entries for device drivers and its major and minor device declarations. Each time the system is rebuild, the config utility read this file and produces a new set of driver and device configuration tables and a corresponding nodetab.

To install the TDRV004 driver and devices into the LynxOS system, the configuration include file tdrv004.cfg must be included in the CONFIG.TBL (see also chapter 2.1.1.3).

The file tdrv004.cfg on the distribution disk contains the driver entry (*C:tdrv004:*\....) and a major device entry (*D:TDRV004 1:t004a_info::*).

If the driver should support more than one major device, the following entries for major devices must be enabled by removing the comment character (#). By copy and paste an existing major and minor entries and renaming the new entries, it is possible to add any number of additional TDRV004 devices.

This example shows a driver entry with one major device and one minor device:

```
#    Format:
#    C:driver-name:open:close:read:write:select:control:install:uninstall
#    D:device-name:info-block-name:raw-partner-name
#    N:node-name:minor-dev

C:tdrv004:\
     :td004open:td004close:::\
     ::td004ioctl:td004install:td004uninstall
D:TDRV004 1:td004a_info::
N:td004a:0
#D:TDRV004 2:td004b_info::
#N:td004b:0
```

The configuration above creates the following node in the /dev directory.

```
/dev/td004a
```

3 TDRV004 Device Driver Programming

LynxOS system calls are all available directly to any C program. They are implemented as ordinary function calls to "glue" routines in the system library, which trap to the OS code.

Note that many system calls use data structures, which should be obtained in a program from appropriate header files. Necessary header files are listed with the system call synopsis.

3.1 open()

NAME

open() - open a file

SYNOPSIS

#include <sys/file.h>
#include <sys/types.h>
#include <fcntl.h>

int open (char *path, int oflags[, mode_t mode])

DESCRIPTION

Opens a file (TDRV004 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TDRV004 devices ***oflags*** must be set to **O_RDWR** to open the file for both reading and writing.

The ***mode*** argument is required only when a file is created. Because a TDRV004 device already exists this argument is ignored.

EXAMPLE

```
int fd

/* open the device named "/dev/td004a" for I/O */
fd = open ("/dev/td004a", O_RDWR);
if (!fd)
{
     /* handle error */
}
```

RETURNS

open returns a file descriptor number if successful, or –1 on error.

SEE ALSO

LynxOS System Call - open()

3.2 close()

NAME

close() – close a file

SYNOPSIS

int close(int fd)

DESCRIPTION

This function closes an opened device.

EXAMPLE

```
int  result;

/*
**   close the device
*/
result = close(fd);
if (result < 0)
{
     /* handle error */
}
```

RETURNS

close returns 0 (OK) if successful, or –1 on error

SEE ALSO

LynxOS System Call - close()

3.3 ioctl()

NAME

ioctl() – I/O device control

SYNOPSIS

#include <ioctl.h>
#include <tdrv004.h>

int ioctl (int fd, int request, char *arg)

DESCRIPTION

ioctl provides a way of sending special commands to a device driver. The call sends the value of request and the pointer arg to the device associated with the descriptor fd.

The following ioctl codes are supported by the driver and are defined in *tdrv004.h*:

Symbol	Meaning
TD004_C_XSVFPLAY	Play an XSVF file for FPGA programming
TD004_C_XSVFPOS	Retrieve current play-position in XSVF file
TD004_C_XSVFLASTCMD	Get the last executed XSVF command
TD004_C_RECONFIG	Trigger FPGA reconfiguration process
TD004_C_SETWAITSTATES	Specify number of waitstates for programming
TD004_C_SETCLOCK	Set clock generator parameters
TD004_C_SPIWRITE	Write values to clock generator
TD004_C_SPIREAD	Read values from clock generator
TD004_C_PLXWRITE	Write 16bit value to PLX9030 EEPROM
TD004_C_PLXREAD	Read 16bit value from PLX9030 EEPROM
TD004_C_READ_UCHAR	Read unsigned char values from FPGA resource
TD004_C_READ_USHORT	Read unsigned short values from FPGA resource
TD004_C_READ_ULONG	Read unsigned long values from FPGA resource
TD004_C_WRITE_UCHAR	Write unsigned char values to FPGA resource
TD004_C_WRITE_USHORT	Write unsigned short values to FPGA resource
TD004_C_WRITE_ULONG	Write unsigned long values to FPGA resource
TD004_C_CONFIGURE_INT	Configure local interrupt source polarity
TD004_C_WAIT_FOR_INT1	Wait for incoming Local Interrupt Source 1
TD004_C_WAIT_FOR_INT2	Wait for incoming Local Interrupt Source 2

See behind for more detailed information on each control code.

RETURNS

ioctl returns 0 if successful, or –1 on error.

On error, *errno* will contain a standard error code (see also LynxOS System Call – ioctl).

SEE ALSO

LynxOS System Call - ioctl().

3.3.1 TD004_C_XSVFPLAY

NAME

TD004_C_XSVFPLAY – Play an XSVF file for FPGA programming

DESCRIPTION

This TDRV004 control function programs the FPGA with a supplied XSVF file. The parameter *arg* passes a pointer to a TD004_XSVF_BUF buffer to the device driver, where the content of the XSVF file is stored. 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 XSVF file, 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 */
} TD004_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 TD004_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 wait states with control function TD004_C_SETWAITSTATES (refer to chapter 3.3.5 in this manual). Additionally, the CLK1 should not be lower than 10MHz for programming.

EXAMPLE

```
#include “tdrv004.h”

int                         fd;
int                         result;
int                         bufsize;
TD004_XSVF_BUF              *pXsvfBuf;

/*
** allocate enough memory (about 3MB) to hold XSVF content
*/
bufsize = sizeof(TD004_XSVF_BUF) + 3000000*sizeof(unsigned char);
pXsvfBuf = (TD004_XSVF_BUF*)malloc( bufsize );

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

/*
** start FPGA programming
*/
result = ioctl(fd, TD004_C_XSVFPLAY, (char*)pXsvfBuf);

if (result != OK) {
       /* handle ioctl error */
}
free( pXsvfBuf );
```

ERRORS

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.
ENOMEM	Error getting enough internal memory for XSVF data.

Other returned error codes are system error conditions.

3.3.2 TD004_C_XSVFPOS

NAME

TD004_C_XSVFPOS – Retrieve current play-position in XSVF file

DESCRIPTION

This TDRV004 control function returns the number of the current processed byte in the XSVF file during programming with TD004_IOCS_XSVFPLAY. This control function can be used to monitor the programming progress.

The parameter *arg* passes a pointer to an *unsigned long* buffer to the device driver, where the XSVF position is returned.

EXAMPLE

```
#include “tdrv004.h”

int                         fd;
int                         result;
unsigned long               XsvfPos;

/*
**  retrieve current position in XSVF file
*/
result = ioctl(fd, TD004_C_XSVFPOS, (char*)&XsvfPos);

if (result != OK 0) {
       /* handle ioctl error */
} else {
       printf(“Current XSVF position: %d\n”, XsvfPos);
}
```

ERRORS

Returned error codes are system error conditions.

3.3.3 TD004_C_XSVLASTCMD

NAME

TD004_C_XSVFLASTCMD – Get the last executed XSVF command

DESCRIPTION

This TDRV004 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 parameter *arg* passes a pointer to an *unsigned long* buffer to the device driver, where the number of the last executed command is returned.

EXAMPLE

```
#include “tdrv004.h”

int                         fd;
int                         result;
unsigned long               XsvfLastCmd;

/*
**  retrieve number of last executed command
*/
result = ioctl(fd, TD004_C_XSVFLASTCMD, (char*)&XsvfLastCmd);

if (result != OK 0) {
       /* handle ioctl error */
} else {
       printf(“Last XSVF command: %d\n”, XsvfLastCmd);
}
```

ERRORS

Returned error codes are system error conditions.

3.3.4 TD004_C_RECONFIG

NAME

TD004_C_RECONFIG – Trigger FPGA reconfiguration process

DESCRIPTION

This TDRV004 control function the reconfiguration process of the FPGA. This control function must be called after the FPGA is programmed using TD004_C_XSVFPLAY. The function returns after the reconfiguration is done, or an error occurred. No additional parameter is used for this function.

EXAMPLE

```
#include “tdrv004.h”

int                         fd;
int                         result;

/*
**  retrieve number of last executed command
*/
result = ioctl(fd, TD004_C_RECONFIG, 0);

if (result != OK 0) {
       /* handle ioctl error */
}
```

ERRORS

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.

3.3.5 TD004_C_SETWAITSTATES

NAME

TD004_C_SETWAITSTATES – Specify number of waitstates for programming

DESCRIPTION

This TDRV004 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 parameter *arg* passes a pointer to an *unsigned long* value to the device driver.

EXAMPLE

```
#include “tdrv004.h”

int                         fd;
int                         result;
unsigned long               WaitStates;

/*
**  configure driver to use 3 waitstates
*/
WaitStates = 3;
result = ioctl(fd, TD004_C_SETWAITSTATES, (char*)&WaitStates);

if (result != OK) {
   /* handle ioctl error */
}
```

ERRORS

EINVAL	The number of waitstates is larger than 1000.

Other returned error codes are system error conditions.

3.3.6 TD004_C_SETCLOCK

NAME

TD004_C_SETWAITSTATES – Specify Set clock generator parameters

DESCRIPTION

This TDRV004 control function configures the onboard clock generator. A pointer to the caller's data buffer (*TD004_CLOCK_PARAM*) is passed by the parameter *arg* to the driver. The necessary values must be calculated using the software tool *Cypress CycberClocks*.

```
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;
} TD004_CLOCK_PARAM;
```

DeviceAddr

Specifies the desired destination address. The CY27EE16 clock generator provides several EEPROM banks as well as SRAM. If TD004_CLKADR_SRAM is specified, the values are directly stored inside the volatile RAM area and take effect immediately. If TD004_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.

EXAMPLE

```
#include “tdrv004.h”

int                         fd;
int                         result;
TD004_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    = TD004_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;

…
```

```
…

/*
**  program clock settings
*/
result = ioctl(fd, TD004_C_SETCLOCK, (char*)&ClockParam);

if (result != OK) {
   /* handle ioctl error */
}
```

ERRORS

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.

3.3.7 TD004_C_SPIWRITE

NAME

TD004_C_SPIWRITE – Write values to clock generator

DESCRIPTION

This TDRV004 control function writes up to 256 *unsigned char* values to a specific sub-address of a Serial Programming Interface (SPI) address. A pointer to the caller's data buffer (*TD004_SPI_BUF*) is passed by the parameter *arg* to the driver. The data section must be included inside this structure.

```
typedef struct {
   unsigned char   SpiAddr;
   unsigned char   SubAddr;
   unsigned long   len;
   unsigned char   pData[1];       /* dynamically expandable */
} TD004_SPI_BUF;
```

SpiAddr

Specifies the Serial Programming Interface (SPI) address of the desired target. See file *tdrv004.h* for definitions.

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 clock generator. Please use control function TD004_C_SETCLOCK instead.

EXAMPLE

```
#include “tdrv004.h”

int                         fd;
int                         result;
int                         BufferSize;
TD004_SPI_BUF               *pSpiBuf;

/*
** write 5 bytes to EEPROM block 1, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize = ( sizeof(TD004_SPI_BUF) + 5*sizeof(unsigned char) );
pSpiBuf = (TD004_SPI_BUF*)malloc( BufferSize );
pSpiBuf->SpiAddr   = TD004_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 = ioctl(fd, TD004_C_SPIWRITE, (char*)pSpiBuf);

if (result != OK) {
   /* handle ioctl error */
}
free( pSpiBuf );
```

ERRORS

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.

3.3.8 TD004_C_SPIREAD

NAME

TD004_C_SPIREAD – Read values from clock generator

DESCRIPTION

This TDRV004 control function reads up to 256 *unsigned char* values from a specific sub-address of a Serial Programming Interface (SPI) address. A pointer to the caller's data buffer (*TD004_SPI_BUF*) is passed by the parameter *arg* to the driver. The data section must be included inside this structure.

```
typedef struct {
   unsigned char   SpiAddr;
   unsigned char   SubAddr;
   unsigned long   len;
   unsigned char   pData[1];        /* dynamically expandable */
} TD004_SPI_BUF;
```

SpiAddr

Specifies the Serial Programming Interface (SPI) address of the desired target. See file *tdrv004.h* for definitions.

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                         result;
int                         BufferSize;
TD004_SPI_BUF               *pSpiBuf;

/*
** read 5 bytes from EEPROM block 1, offset 0x00
** allocate enough memory to hold the data structure + read data
*/
BufferSize = ( sizeof(TD004_SPI_BUF) + 5*sizeof(unsigned char) );
pSpiBuf = (TD004_SPI_BUF*)malloc( BufferSize );
pSpiBuf->SpiAddr   = TD004_CLKADDR_EEBLOCK1;
pSpiBuf->SubAddr   = 0x00;
pSpiBuf->len       = 5;

result = ioctl(fd, TD004_C_SPIREAD, (char*)pSpiBuf);

if (result != OK) {
   /* handle ioctl error */
}
free( pSpiBuf );
```

ERRORS

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.

3.3.9 TD004_C_PLXWRITE

NAME

TD004_C_PLXWRITE – Write 16bit value to PLX9030 EEPROM

DESCRIPTION

This TDRV004 control function writes an *unsigned short* value to a specific PLX9030 EEPROM memory offset. A pointer to the caller's data buffer (*TD004_PLX_BUF*) is passed by the parameter *arg* to the driver.

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

```
typedef struct {
   unsigned long   Offset;
   unsigned short  Value;
} TD004_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 PLX9030 User Manual for detailed information on these registers.

Value

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

EXAMPLE

```
#include “tdrv004.h”

int                          fd;
int                          result;
TD004_PLX_BUF                PlxBuf;

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

result = ioctl(fd, TD004_C_PLXWRITE, (char*)&PlxBuf);

if (result != OK) {
   /* handle ioctl error */
}
```

ERRORS

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.

3.3.10 TD004_C_PLXREAD

NAME

TD004_C_PLXREAD – Read 16bit value from PLX9030 EEPROM

DESCRIPTION

This TDRV004 control function reads an *unsigned short* value from a specific PLX9030 EEPROM memory offset. A pointer to the caller's data buffer (*TD004_PLX_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct {
   unsigned long   Offset;
   unsigned short  Value;
} TD004_PLX_BUF;
```

Offset

Specifies the offset into the PLX9030 EEPROM, where the supplied data word should be retrieved. 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 PLX9030 User Manual for detailed information on these registers.

Value

This value holds the retrieved 16bit word.

EXAMPLE

```
#include “tdrv004.h”

int                          fd;
int                          result;
TD004_PLX_BUF                PlxBuf;

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

result = ioctl(fd, TD004_C_PLXREAD, (char*)&PlxBuf);

if (result == OK) {
   printf( “Subsystem-ID = 0x%04X \n”, PlxBuf.Value );
} else {
   /* handle ioctl error */
}
```

ERRORS

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.

3.3.11 TD004_C_READ_UCHAR

NAME

TD004_C_READ_UCHAR – Read unsigned char values from FPGA resource

DESCRIPTION

This TDRV004 control function reads a number of *unsigned char* values from a Memory or I/O area by using BYTE (8bit) accesses. A pointer to the caller's data buffer (*TD004_MEMIO_BUF*) is passed by the parameter *arg* to the driver. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct {
   TD004_RESOURCE  Resource;
   unsigned long   Offset;
   unsigned long   Size;
   unsigned char   pData[1];      /* dynamically expandable */
} TD004_MEMIO_BUF;
```

Resource

This parameter 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 BAR1.

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;
int                         result;
unsigned long               BufferSize;
TD004_MEMIO_BUF             *pMemIoBuf;
unsigned char               *pValues;

/*
** read 50 bytes from MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + read data
*/
BufferSize = ( sizeof(TD004_MEMIO_BUF) + 50*sizeof(unsigned char) );
pMemIoBuf = (TD004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size         = 50;
pMemIoBuf->Resource     = TD004_RES_MEM_2;
pMemIoBuf->Offset       = 0;

result = ioctl(fd, TD004_C_READ_UCHAR, (char*)pMemIoBuf);

if (result != OK) {
   /* handle ioctl error */
} else {
   pValues = (unsigned char*)pMemIoBuf->pData;
   printf( “Value[0] = 0x%02X \n”, pValues[0] );
}
free( pMemIoBuf );
```

ERRORS

EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.

Other returned error codes are system error conditions.

3.3.12 TD004_C_READ_USHORT

NAME

TD004_C_READ_USHORT – Read unsigned short values from FPGA resource

DESCRIPTION

This TDRV004 control function reads a number of *unsigned short* values from a Memory or I/O area by using WORD (16bit) accesses. A pointer to the caller's data buffer (*TD004_MEMIO_BUF*) is passed by the parameter *arg* to the driver. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct {
   TD004_RESOURCE  Resource;
   unsigned long   Offset;
   unsigned long   Size;
   unsigned char   pData[1];      /* dynamically expandable */
} TD004_MEMIO_BUF;
```

Resource

This parameter 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 BAR1.

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;
int                         result;
unsigned long               BufferSize;
TD004_MEMIO_BUF             *pMemIoBuf;
unsigned short              *pValues;

/*
** read 50 16bit words from MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + read data
*/
BufferSize = ( sizeof(TD004_MEMIO_BUF) + 50*sizeof(unsigned short) );
pMemIoBuf = (TD004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size         = 50;
pMemIoBuf->Resource     = TD004_RES_MEM_2;
pMemIoBuf->Offset       = 0;

result = ioctl(fd, TD004_C_READ_USHORT, (char*)pMemIoBuf);

if (result != OK) {
   /* handle ioctl error */
} else {
   pValues = (unsigned short*)pMemIoBuf->pData;
   printf( “Value[0] = 0x%04X \n”, pValues[0] );
}
free( pMemIoBuf );
```

ERRORS

EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.

Other returned error codes are system error conditions.

3.3.13 TD004_C_READ_ULONG

NAME

TD004_C_READ_ULONG – Read unsigned long values from FPGA resource

DESCRIPTION

This TDRV004 control function reads a number of *unsigned long* values from a Memory or I/O area by using DWORD (32bit) accesses. A pointer to the caller's data buffer (*TD004_MEMIO_BUF*) is passed by the parameter *arg* to the driver. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct {
   TD004_RESOURCE  Resource;
   unsigned long   Offset;
   unsigned long   Size;
   unsigned char   pData[1];      /* dynamically expandable */
} TD004_MEMIO_BUF;
```

Resource

This parameter 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 BAR1.

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;
int                         result;
unsigned long               BufferSize;
TD004_MEMIO_BUF             *pMemIoBuf;
unsigned long               *pValues;

/*
** read 50 32bit dwords from MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + read data
*/
BufferSize = ( sizeof(TD004_MEMIO_BUF) + 50*sizeof(unsigned long) );
pMemIoBuf = (TD004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size         = 50;
pMemIoBuf->Resource     = TD004_RES_MEM_2;
pMemIoBuf->Offset       = 0;

result = ioctl(fd, TD004_C_READ_ULONG, (char*)pMemIoBuf);

if (result != OK) {
   /* handle ioctl error */
} else {
   pValues = (unsigned long*)pMemIoBuf->pData;
   printf( “Value[0] = 0x%08lX \n”, pValues[0] );
}
free( pMemIoBuf );
```

ERRORS

EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.

Other returned error codes are system error conditions.

3.3.14 TD004_C_WRITE_UCHAR

NAME

TD004_C_WRITE_UCHAR – Write unsigned char values to FPGA resource

DESCRIPTION

This TDRV004 control function writes a number of *unsigned char* values to a Memory or I/O area by using BYTE (8bit) accesses. A pointer to the caller's data buffer (*TD004_MEMIO_BUF*) is passed by the parameter *arg* to the driver. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct {
   TD004_RESOURCE  Resource;
   unsigned long   Offset;
   unsigned long   Size;
   unsigned char   pData[1];      /* dynamically expandable */
} TD004_MEMIO_BUF;
```

Resource

This parameter 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 BAR1.

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”

int                         fd;
int                         result;
unsigned long               BufferSize;
TD004_MEMIO_BUF             *pMemIoBuf;
unsigned char               *pValues;

/*
** write 10 byte to MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize = ( sizeof(TD004_MEMIO_BUF) + 10*sizeof(unsigned char) );
pMemIoBuf = (TD004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size         = 10;
pMemIoBuf->Resource     = TD004_RES_MEM_2;
pMemIoBuf->Offset       = 0;
pValues = (unsigned char*)pMemIoBuf->pData;
pValues[0] = 0x01;
pValues[1] = 0x02;
…

result = ioctl(fd, TD004_C_WRITE_UCHAR, (char*)pMemIoBuf);

if (result != OK) {
   /* handle ioctl error */
}
free( pMemIoBuf );
```

ERRORS

EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.

Other returned error codes are system error conditions.

3.3.15 TD004_C_WRITE_USHORT

NAME

TD004_C_WRITE_USHORT – Write unsigned short values to FPGA resource

DESCRIPTION

This TDRV004 control function writes a number of *unsigned short* values to a Memory or I/O area by using WORD (16bit) accesses. A pointer to the caller's data buffer (*TD004_MEMIO_BUF*) is passed by the parameter *arg* to the driver. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct {
   TD004_RESOURCE  Resource;
   unsigned long   Offset;
   unsigned long   Size;
   unsigned char   pData[1];      /* dynamically expandable */
} TD004_MEMIO_BUF;
```

Resource

This parameter 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 BAR1.

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. The data pointer is typecasted into an *unsigned short* pointer.

EXAMPLE

```
#include “tdrv004.h”

int                         fd;
int                         result;
unsigned long               BufferSize;
TD004_MEMIO_BUF             *pMemIoBuf;
unsigned short              *pValues;

/*
** write 10 16bit words to MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize = ( sizeof(TD004_MEMIO_BUF) + 10*sizeof(unsigned short) );
pMemIoBuf = (TD004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size             = 10;
pMemIoBuf->Resource     = TD004_RES_MEM_2;
pMemIoBuf->Offset       = 0;
pValues = (unsigned short*)pMemIoBuf->pData;
pValues[0] = 0x0001;
pValues[1] = 0x0002;
…

result = ioctl(fd, TD004_C_WRITE_USHORT, (char*)pMemIoBuf);

if (result != OK) {
   /* handle ioctl error */
}
free( pMemIoBuf );
```

ERRORS

EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.

Other returned error codes are system error conditions.

3.3.16 TD004_C_WRITE_ULONG

NAME

TD004_C_WRITE_ULONG – Write unsigned long values to FPGA resource

DESCRIPTION

This TDRV004 control function writes a number of *unsigned long* values to a Memory or I/O area by using DWORD (32bit) accesses. A pointer to the caller's data buffer (*TD004_MEMIO_BUF*) is passed by the parameter *arg* to the driver. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct {
   TD004_RESOURCE  Resource;
   unsigned long   Offset;
   unsigned long   Size;
   unsigned char   pData[1];      /* dynamically expandable */
} TD004_MEMIO_BUF;
```

Resource

This parameter 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 BAR1.

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. The data pointer is typecasted into an *unsigned long* pointer.

EXAMPLE

```
#include “tdrv004.h”

int                         fd;
int                         result;
unsigned long               BufferSize;
TD004_MEMIO_BUF             *pMemIoBuf;
unsigned long               *pValues;

/*
** write 10 32bit dwords to MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize = ( sizeof(TD004_MEMIO_BUF) + 10*sizeof(unsigned long) );
pMemIoBuf = (TD004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Size         = 10;
pMemIoBuf->Resource     = TD004_RES_MEM_2;
pMemIoBuf->Offset       = 0;
pValues = (unsigned long*)pMemIoBuf->pData;
pValues[0] = 0x00000001;
pValues[1] = 0x00000002;
…

result = ioctl(fd, TD004_C_WRITE_ULONG, (char*)pMemIoBuf);

if (result != OK) {
   /* handle ioctl error */
}
free( pMemIoBuf );
```

ERRORS

EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.

Other returned error codes are system error conditions.

3.3.17 TD004_C_CONFIGURE_INT

NAME

TD004_C_CONFIGURE_INT – Configure local interrupt source polarity

DESCRIPTION

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

A pointer to the caller's data buffer (*unsigned long*) is passed by the parameter *argp* to the driver. This value is an OR'ed value using the following definitions (only one value valid for each interrupt source):

Value	Description
TD004_LINT1_POLHIGH	Local Interrupt Source 1 HIGH active
TD004_LINT1_POLLOW	Local Interrupt Source 1 LOW active
TD004_LINT2_POLHIGH	Local Interrupt Source 2 HIGH active
TD004_LINT2_POLLOW	Local Interrupt Source 2 LOW active

EXAMPLE

```
#include "tdrv004.h"

int           fd;
int           result;
unsigned long IntConfig;

/*
** Setup LINT1 to LOW polarity, and LINT2 to HIGH polarity
*/
IntConfig = TD004_LINT1_POLLOW | TD004_LINT2_POLHIGH;

result = ioctl(fd, TD004_C_CONFIGURE_INT, (char*)&IntConfig);

if (result != OK) {
   /* handle ioctl error */
}
```

ERRORS

EINVAL	The specified parameter is invalid.

Other returned error codes are system error conditions.

3.3.18 TD004_C_WAIT_FOR_INT1

NAME

TD004_C_WAIT_FOR_INT1 – Wait for incoming Local Interrupt Source 1

DESCRIPTION

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 inside the PLX9030.

A pointer to the caller's data buffer (*int*) is passed by the parameter *argp* to the driver. This value contains 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       result;
int       Timeout;

/*
** Wait at least 50 system ticks for incoming interrupt
*/
Timeout = 50;

result = ioctl(fd, TD004_C_WAIT_FOR_INT1, (char*)&Timeout);

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

ERRORS

EBUSY	Another job already waiting for this interrupt. Only one job is allowed at the same time.

Other returned error codes are system error conditions.

3.3.19 TD004_C_WAIT_FOR_INT2

NAME

TD004_C_WAIT_FOR_INT2 – Wait for incoming Local Interrupt Source 2

DESCRIPTION

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 inside the PLX9030.

A pointer to the caller's data buffer (*int*) is passed by the parameter *argp* to the driver. This value contains 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       result;
int       Timeout;

/*
** Wait at least 50 system ticks for incoming interrupt
*/
Timeout = 50;

result = ioctl(fd, TD004_C_WAIT_FOR_INT2, (char*)&Timeout);

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

ERRORS

EBUSY	Another job already waiting for this interrupt. Only one job is allowed at the same time.

Other returned error codes are system error conditions.

4 Debugging and Diagnostic

If the driver will not work properly, please enable debug outputs by defining the symbols *DEBUG, DEBUG_TPMC,* and *DEBUG_PCI.*

The debug output should appear on the console. If not, please check the symbol *KKPF_PORT* in *uparam.h*. This symbol should be configured to a valid COM port (e.g. *SKDB_COM1*).

The debug output displays the device information data for the current major device like this.

```
Bus = 1  Dev = 2  Func = 0
[00] = 02761498
[04] = 02800000
[08] = 11800000
[0C] = 00000000
[10] = 84000000
[14] = 00804001
[18] = 85000000
[1C] = 00000000
[20] = 00000000
[24] = 00000000
[28] = 00000000
[2C] = 000A1498
[30] = 00000000
[34] = 00000040
[38] = 00000000
[3C] = 0000010B
Found a TDRV004 compatible module, BusNo=1, DevNo=2
  MEM[0] = 0xB4600000 (size=0x00000080)
  MEM[1] = 0xB5600000 (size=0x01000000)
  I/O[0] = 0xB0204000 (size=0x00000080)
```

The debug output above is only an example. Debug output on other systems may be different for addresses and data in some locations.