The Embedded I/O Company

TDRV004-SW-82

Linux Device Driver

Reconfigurable FPGA

Version 1.1.x

User Manual

Issue 1.1.1

April 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-82

Linux Device Driver

Reconfigurable FPGA

Supported Modules:
TPMC630
TCP630

Issue	Description	Date
1.0.0	First Issue	September 30, 2005
1.1.0	Interrupt Support functions added, New Address TEWS LLC	June 08, 2007
1.1.1	Address TEWS LLC removed	April 22, 2010

Table of Contents

1 Introduction

The TDRV004-SW-82 Linux device driver allows the operation of the TDRV004 compatible devices conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close()*,and *ioctl()* functions.

Special I/O operation that do not fit to the standard I/O calls will be performed by calling the *ioctl()* function with a specific function code and an optional function dependent argument.

The TDRV004-SW-82 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

<u>The TDRV004-SW-82 supports the modules listed below:</u>

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 supported 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

The directory TDRV004-SW-82 on the distribution media contains the following files:

TDRV004-SW-82-1.1.1.pdf	This manual in PDF format
TDRV004-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
fpgaexa.tar.gz	FPGA example XSVF
ChangeLog.txt	Release history
Release.txt	Information about the Device Driver Release

The GZIP compressed archive TDRV004-SW-82-SRC.tar.gz contains the following files and directories:

tdrv004.c	Driver source code
tdrv004def.h	Driver include file
tdrv004.h	Driver include file for application program
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	Device driver make file
makenode	Script for device node creation
include/config.h	Driver independent library header file
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
example/tdrv004exa.c	Example application
example/Makefile	Example application makefile

In order to perform an installation, extract all files of the archive TDRV004-SW-82.tar.gz to the desired target directory. Additionally, copy *tdrv004.h* into your include path.

2.1 Build and install the device driver

- Login as *root*
- Change to the target directory
- To create and install the driver in the module directory */lib/modules/<version>/misc* enter:

  # make install

- To update the device driver's module dependencies, enter:

  # **depmod -aq**

2.2 Uninstall the device driver

- Login as *root*
- Change to the target directory
- To remove the driver from the module directory */lib/modules/<version>/misc* enter:

```
# make uninstall
```

2.3 Install device driver into the running kernel

- To load the device driver into the running kernel, login as root and execute the following commands:

```
# modprobe tdrv004drv
```

- After the first build or if you are using dynamic major device allocation it is necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled a device file system (devfs or sysfs with udev) then you have to skip running the *makenode* script. Instead of creating device nodes from the script the driver itself takes creating and destroying of device nodes in its responsibility.

```
# sh makenode
```

On success the device driver will create a minor device for each TDRV004 device found. The first TDRV004 device can be accessed with device node /dev/tdrv004_0, the second module with device node /dev/tdrv004_1 and so on.

The assignment of device nodes to physical TDRV004 modules depends on the search order of the PCI bus driver.

2.4 Remove device driver from the running kernel

- To remove the device driver from the running kernel login as root and execute the following command:

```
# modprobe –r tdrv004drv
```

If your kernel has enabled devfs or sysfs (udev), all /dev/tdrv004_x nodes will be automatically removed from your file system after this.

Be sure that the driver isn't opened by any application program. If opened you will get the response "*tdrv004drv: Device or resource busy*" and the driver will still remain in the system until you close all opened files and execute *modprobe –r* again.

2.5 Change Major Device Number

This paragraph is only for Linux kernels without DEVFS installed. The TDRV004 driver uses dynamic allocation of major device numbers per default. If this isn't suitable for the application it is possible to define a major number for the driver.

To change the major number, edit the file tdrv004def.h, change the following symbol to appropriate value and enter `make install` to create a new driver.

TDRV004_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TDRV004_MAJOR   122
```

Be sure that the desired major number isn't used by other drivers. Please check *//proc/devices* to see which numbers are free.

3 Device Input/Output functions

This chapter describes the interface to the device driver I/O system.

3.1 open()

NAME

open() opens a file descriptor.

SYNOPSIS

#include <fcntl.h>

int open (*const char *filename, int flags*)

DESCRIPTION

The **open** function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file is to be opened. This is a bit mask. Create the value by the bitwise OR of the appropriate parameters (using the **|** operator in C). See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int fd;

fd = open(“/dev/tdrv004_0”, O_RDWR);
if (fd == -1)
{
  /* handle error condition */
}
```

RETURNS

The normal return value from **open** is a non-negative integer file descriptor. In case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

E_NODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during open. For more information about open error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.2 close()

NAME

close() closes a file descriptor.

SYNOPSIS

#include <unistd.h>

int **close** (int *filedes*)

DESCRIPTION

The **close** function closes the file descriptor *filedes*.

EXAMPLE

```
int fd;

. . .

if (close(fd) != 0) {
  /* handle close error conditions */
}
```

RETURNS

The normal return value from **close** is 0. In case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

E_NODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during close. For more information about close error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 ioctl()

NAME

ioctl() device control functions

SYNOPSIS

#include <sys/ioctl.h>

int **ioctl**(*int filedes, int request [, void *argp]*)

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *filedes*, causing the corresponding device to perform the requested operation.

The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *tdrv004.h*:

Function	**Description**
TD004_IOCS_XSVFPLAY	Play an XSVF file for FPGA programming
TD004_IOCG_XSVFPOS	Retrieve current play-position in XSVF file
TD004_IOCG_XSVFLASTCMD	Get the last executed XSVF command
TD004_IOC_RECONFIG	Trigger FPGA reconfiguration process
TD004_IOCT_SETWAITSTATES	Specify number of waitstates for programming
TD004_IOCS_SETCLOCK	Set clock generator parameters
TD004_IOCS_SPIWRITE	Write values to clock generator
TD004_IOCG_SPIREAD	Read values from clock generator
TD004_IOCS_PLXWRITE	Write 16bit value to PLX9030 EEPROM
TD004_IOCG_PLXREAD	Read 16bit value from PLX9030 EEPROM
TD004_IOCG_READ_UCHAR	Read unsigned char values from FPGA resource
TD004_IOCG_READ_USHORT	Read unsigned short values from FPGA resource
TD004_IOCG_READ_ULONG	Read unsigned long values from FPGA resource
TD004_IOCS_WRITE_UCHAR	Write unsigned char values to FPGA resource
TD004_IOCS_WRITE_USHORT	Write unsigned short values to FPGA resource
TD004_IOCS_WRITE_ULONG	Write unsigned long values to FPGA resource
TD004_IOCS_CONFIGURE_INT	Configure local interrupt source polarity
TD004_IOC_WAIT_FOR_INT1	Wait for incoming Local Interrupt Source 1
TD004_IOC_WAIT_FOR_INT2	Wait for incoming Local Interrupt Source 2

See below for more detailed information on each control code.

To use these TDRV004 specific control codes the header file *tdrv004.h* must be included in the application.

RETURNS

On success, zero is returned. In case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL	Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument *request*.

Other function dependant error codes will be described for each ioctl code separately. Note, the TDRV004 device driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TD004_IOCS_XSVFPLAY

NAME

TD004_IOCS_XSVFPLAY	Play an XSVF file for FPGA programming

DESCRIPTION

This ioctl function programs the FPGA with a supplied XSVF file. A pointer to the caller's data buffer (TD004_XSVF_BUF), where the content of the XSVF file is stored, is passed to the device driver. 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.

The *TD004_XSVF_BUF* structure has the following layout:

```
typedef struct {
    unsigned long   size;
    unsigned char   pData[1];        /* dynamically expandable */
} TD004_XSVF_BUF;
```

Members

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 waitstates with control function TD004_IOCT_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;
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_IOCS_XSVFPLAY, pXsvfBuf);

if (result < 0) {
    /* handle ioctl error */
}
free( pXsvfBuf );
```

ERRORS

EINVAL	There was an error during XSVF processing.
EINTR	The function was cancelled.
EFAULT	Error while copying data to or from user space.
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_IOCG_XSVFPOS

NAME

TD004_IOCG_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. A pointer to an unsigned long value is passed to the driver by the argument *argp*.

EXAMPLE

```
#include “tdrv004.h”

int           fd;
int           result;
unsigned long XsvfPos;


result = ioctl(fd, TD004_IOCG_XSVFPOS, &XsvfPos);

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

ERRORS

EFAULT	Error while copying data to user space.

Other returned error codes are system error conditions.

3.3.3 TD004_IOCG_XSVFLASTCMD

NAME

TD004_IOCG_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. A pointer to an unsigned long value is passed to the driver by the argument *argp*.

EXAMPLE

```
#include “tdrv004.h”

int           fd;
int           result;
unsigned long XsvfLastCmd;


result = ioctl(fd, TD004_IOCG_XSVFLASTCMD, &XsvfLastCmd);

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

ERRORS

EFAULT	Error while copying data to user space.

Other returned error codes are system error conditions.

3.3.4 TD004_IOC_RECONFIG

NAME

TD004_IOC_RECONFIG	Trigger FPGA reconfiguration process.

DESCRIPTION

This function starts the reconfiguration process of the FPGA. This control function must be called after the FPGA is programmed using DCMD_TD004_XSVFPLAY. The function returns after the reconfiguration is done, or an error occurred. No additional parameter is used for this function, so the optional argument can be omitted.

EXAMPLE

```
#include “tdrv004.h”

int fd;
int result;

result = ioctl(fd, TD004_IOC_RECONFIG);

if (result < 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_IOCT_SETWAITSTATES

NAME

TD004_IOCT_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.

A pointer to an unsigned long value must be passed to the driver by the parameter *argp*.

EXAMPLE

```
#include “tdrv004.h”

int           fd;
int           result;
unsigned long WaitStates;

/*
**  configure driver to use 3 waitstates
*/
WaitStates = 3;
result = ioctl(fd, TD004_IOCT_SETWAITSTATES, &WaitStates);

if (result < 0) {
   /* handle ioctl error */
}
```

ERRORS

This ioctl function returns no function specific error codes.

3.3.6 TD004_IOCS_SETCLOCK

NAME

TD004_IOCS_SETCLOCK	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 *argp* to the driver. The necessary values can be calculated using the 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 *x41_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;
int                 bufsize;
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;
```

```
/*
** start Clock Parameter programming
*/
result = ioctl(fd, TD004_IOCS_SETCLOCK, &ClockParam);

if (result < 0) {
    /* 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.
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.7 TD004_IOCS_SPIWRITE

NAME

TD004_IOCS_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 *argp* 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 clockgenerator. Please use control function TD004_IOCS_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_IOCS_SPIWRITE, pSpiBuf);
if (result < 0) {
    /* 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.
EFAULT	Error while copying data to or from user space.
ENOMEM	Error getting enough internal memory for SPI data.

Other returned error codes are system error conditions.

3.3.8 TD004_IOCG_SPIREAD

NAME

TD004_IOCG_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 *argp* 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_IOCG_SPIREAD, pSpiBuf);
if (result < 0) {
    /* 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.
EFAULT	Error while copying data to or from user space.
ENOMEM	Error getting enough internal memory for SPI data.

Other returned error codes are system error conditions.

3.3.9 TD004_IOCS_PLXWRITE

NAME

TD004_IOCS_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 *argp* 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 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 offset.

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.

EXAMPLE

```
#include “tdrv004.h”

int             fd;
int             result;
int             BufferSize;
TD004_PLX_BUF   PlxBuf;

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

result = ioctl(fd, TD004_IOCS_PLXWRITE, &PlxBuf);

if (result < 0) {
    /* 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.
EFAULT	Error while copying data from user space.

Other returned error codes are system error conditions.

3.3.10 TD004_IOCG_PLXREAD

NAME

TD004_IOCG_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 *argp* to the driver.

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

Offset

Specifies the offset into the PLX9030 EEPROM, from 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;
int                 BufferSize;
TD004_PLX_BUF       PlxBuf;

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

result = ioctl(fd, TD004_IOCG_PLXREAD, &PlxBuf);

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

ERRORS

EINVAL	The specified offset is invalid.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.

Other returned error codes are system error conditions.

3.3.11 TD004_IOCG_READ_UCHAR

NAME

TD004_IOCG_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 accesses. A pointer to the caller's data buffer (*TD004_MEMIO_BUF*) is passed by the parameter *argp* 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

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_IOCG_READ_UCHAR, pMemIoBuf);
if (result < 0) {
  /* handle ioctl error */
} else {
  pValues = (unsigned char*)pMemIoBuf->pData;
}
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.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

3.3.12 TD004_IOCG_READ_USHORT

NAME

TD004_IOCG_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 accesses. A pointer to the caller's data buffer (*TD004_MEMIO_BUF*) is passed by the parameter *argp* 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

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_IOCG_READ_USHORT, pMemIoBuf);
if (result < 0) {
  /* handle ioctl error */
} else {
  pValues = (unsigned short*)pMemIoBuf->pData;
}
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.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

3.3.13 TD004_IOCG_READ_ULONG

NAME

TD004_IOCG_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 accesses. A pointer to the caller's data buffer (*TD004_MEMIO_BUF*) is passed by the parameter *argp* 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

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_IOCG_READ_ULONG, pMemIoBuf);
if (result < 0) {
  /* handle ioctl error */
} else {
  pValues = (unsigned long*)pMemIoBuf->pData;
}
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.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

3.3.14 TD004_IOCS_WRITE_UCHAR

NAME

TD004_IOCS_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 accesses. A pointer to the caller's data buffer (*TD004_MEMIO_BUF*) is passed by the parameter *argp* 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

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 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;
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_IOCS_WRITE_UCHAR, pMemIoBuf);
if (result < 0) {
  /* 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.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

3.3.15 TD004_IOCS_WRITE_USHORT

NAME

TD004_IOCS_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 accesses. A pointer to the caller's data buffer (*TD004_MEMIO_BUF*) is passed by the parameter *argp* 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

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 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;
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 char*)pMemIoBuf->pData;
pValues[0]            = 0x0001;
pValues[1]            = 0x0002;
...

result = ioctl(fd, TD004_IOCS_WRITE_USHORT, pMemIoBuf);
if (result < 0) {
  /* 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.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

3.3.16 TD004_IOCS_WRITE_ULONG

NAME

TD004_IOCS_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 accesses. A pointer to the caller's data buffer (*TD004_MEMIO_BUF*) is passed by the parameter *argp* 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

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 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;
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_IOCS_WRITE_ULONG, pMemIoBuf);
if (result < 0) {
  /* 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.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

3.3.17 TD004_IOCS_CONFIGURE_INT

NAME

TD004_IOCS_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_IOCS_CONFIGURE_INT, &IntConfig);
if (result < 0) {
    /* handle ioctl error */
}
```

ERRORS

EFAULT	Error while copying data to or from user space.
EINVAL	Invalid parameter specified.

Other returned error codes are system error conditions.

3.3.18 TD004_IOC_WAIT_FOR_INT1

NAME

TD004_IOC_WAIT_FOR_INT	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 0 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;
unsigned long IntConfig;

/*
** Wait at least 5 seconds for incoming interrupt
*/
Timeout = 5 * HZ;

result = ioctl(fd, TD004_WAIT_FOR_INT1, &Timeout);
if (resul < 0)
{
  /* acknowledge interrupt source in FPGA logic       */
  /* to clear the PLX PCI9030 Local Interrupt Source  */
} else {
  /* handle the error */
}
```

ERRORS

EFAULT	Error while copying data to or from user space.
EINVAL	Invalid parameter specified (Timeout must be >= 0)
EBUSY	Another job already waiting for this interrupt. Only one job is allowed at the same time.
ETIME	The specified timeout occurred.

Other returned error codes are system error conditions.

3.3.19 TD004_IOC_WAIT_FOR_INT2

NAME

TD004_IOC_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 0 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;
unsigned long IntConfig;

/*
** Wait at least 5 seconds for incoming interrupt
*/
Timeout = 5 * HZ;

result = ioctl(fd, TD004_WAIT_FOR_INT2, &Timeout);
if (resul < 0)
{
  /* acknowledge interrupt source in FPGA logic       */
  /* to clear the PLX PCI9030 Local Interrupt Source  */
} else {
  /* handle the error */
}
```

ERRORS

EFAULT	Error while copying data to or from user space.
EINVAL	Invalid parameter specified (Timeout must be >= 0)
EBUSY	Another job already waiting for this interrupt. Only one job is allowed at the same time.
ETIME	The specified timeout occurred.

Other returned error codes are system error conditions.

4 Diagnostic

If the TDRV004 does not work properly it is helpful to get some status information from the driver respective kernel.

The Linux */proc* file system provides information about kernel, resources, driver, devices, and so on. The following screen dumps displays information of a correct running TDRV004 driver (see also the proc man pages).

```
# cat /proc/pci
  . . .
  Bus  0, device  11, function  0:
    Signal processing controller: PCI device 1498:0276 (TEWS Datentechnik
GmBH) (rev 0).
      IRQ 11.
      Non-prefetchable 32 bit memory at 0xec020000 [0xec02007f].
      I/O at 0xd400 [0xd47f].
      Non-prefetchable 32 bit memory at 0xeb000000 [0xebffffff].
```

```
# cat /proc/devices
Character devices:
  1 mem
  2 pty
  . . .
136 pts
162 raw
254 tdrv004drv
```

```
# cat /proc/ioports
  . . .
d000-d03f : Intel Corp. 82557/8/9 [Ethernet Pro 100]
  d000-d03f : e100
d400-d47f : PCI device 1498:0276 (TEWS Datentechnik GmBH)
  . . .
```

```
# cat /proc/iomem
00000000-0009f7ff : System RAM
  . . .
eb000000-ebffffff : PCI device 1498:0276 (TEWS Datentechnik GmBH)
ec000000-ec01ffff : Intel Corp. 82557/8/9 [Ethernet Pro 100]
  ec000000-ec01ffff : e100
ec020000-ec02007f : PCI device 1498:0276 (TEWS Datentechnik GmBH)
  . . .
```