The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV004-SW-95

QNX6 - Neutrino Device Driver

Reconfigurable FPGA

Version 1.1.x

User Manual

Issue 1.1.1

October 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-95

QNX6 - Neutrino Device Driver

Reconfigurable FPGA

Issue	Description	Date
1.0.0	First Issue	September 19, 2005
1.1.0	Interrupt functionality added, General Revision	September 3, 2008
1.1.1	General revision	October 1, 2010

Table of Contents

1 Introduction

The TDRV004-SW-95 QNX-Neutrino device driver allows the operation of the TPMC630 product family on Intel-x86 based QNX-Neutrino operating systems.

The TDRV004 device driver is basically implemented as a user installable Resource Manager. The standard file (I/O) functions (open, close and devctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control 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 local interrupts

The TDRV004-SW-95 device driver supports the modules listed below:

TPMC630	User Programmable FPGA	(PMC)
TCP630	User Programmable FPGA	(cPCI)

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

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

TPMC630 / TCP630 User Manual
TPMC630 / TCP630 Engineering Manual

2 Installation

The distribution media contains the following files:

TDRV004-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TDRV004-SW-95-1.1.1.pdf	this manual
fpgaexa.tar.gz	FPGA example XSVF
Release.txt	Information about the Device Driver Release
ChangeLog.txt	Release history

The GZIP compressed archive TDRV004-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tdrv004':

driver/tdrv004.c	Driver source code
driver/tdrv004.h	Definitions and data structures for driver and application
driver/tdrv004def.h	Device driver include
driver/pf_micro.c	XSVF player functions (Platform Flash)
driver/pf_micro.h	header file for XSVF player functions
driver/pf_lenval.c	special functions for XSVF player
driver/pf_lenval.h	header file for XSVF functions
driver/pf_ports.c	hardware layer for XSVF player
driver/pf_ports.h	header file for XSVF hardware layer
driver/node.c	Queue management source code
driver/node.h	Queue management definitions
driver/nto/*	Build path
example/tdrv004exa.c	Example application
example/nto/*	Build path

For installation copy the tar-archive into the /usr/src directory and unpack it (e.g. `tar –xzvf TDRV004-SW-95-SRC.tar.gz`). After that the necessary directory structure for the automatic build and the source files are available underneath the new directory called *tdrv004*.

It is absolutely important to extract the TDRV004 tar archive in the /usr/src directory. Otherwise the automatic build with make will fail.

2.1 Build the device driver

Change to the */usr/src/tdrv004/driver* directory

Execute the Makefile:

```
# make install
```

After successful completion the driver binary (*tdrv004*) will be installed in the /bin directory.

Build the example application

Change to the */usr/src/tdrv004/example* directory

Execute the Makefile:

```
# make install
```

After successful completion the example binary (*tdrv004exa*) will be installed in the /bin directory.

2.2 Start the driver process

To start the TDRV004 device driver, you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tdrv004 [-v] &
```

The TDRV004 Resource Manager registers created devices in the QNX-Neutrino pathname space under following names.

```
/dev/tdrv004_0
/dev/tdrv004_1
…
/dev/tdrv004_x
```

This pathname must be used in the application program to open a path to the desired TDRV004 device.

```
fd = open("/dev/tdrv004_0", O_RDWR);
```

For debugging, you can start the TDRV004 Resource Manager with the –v option. Now the Resource Manager will print versatile information about TDRV004 configuration and command execution on the terminal window.

```
tdrv004 –v &
```

Make sure that only one instance of the device driver process is started.

3 Device Input/Output functions

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

3.1 open()

NAME

open() - open a file descriptor

SYNOPSIS

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

int open (const char *pathname, int flags)

DESCRIPTION

The *open* function creates and returns a new file descriptor for the TDRV004 named by pathname. The flags argument controls how the file is to be opened. TDRV004 devices must be opened O_RDWR.

EXAMPLE

```
int fd;

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

RETURNS

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

ERRORS

Returns only Neutrino specific error codes, see Neutrino Library Reference.

SEE ALSO

Library Reference - open()

3.2 close()

NAME

close() – close 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 the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

Returns only Neutrino specific error code, see Neutrino Library Reference.

SEE ALSO

Library Reference - close()

3.3 devctl()

NAME

devctl() – device control functions

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>
#include <devctl.h>

int devctl
(
        int             filedes,
        int             dcmd,
        void            *data_ptr,
        size_t          n_bytes,
        int             *dev_info_ptr
)
```

DESCRIPTION

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

The argument *dcmd* specifies the control code for the operation.

The arguments *data_ptr* and *n_bytes* depends on the command and will be described for each command in detail later in this chapter. Usually *data_ptr* points to a buffer that passes data between the user task and the driver and *n_bytes* defines the size of this buffer.

The argument *dev_info_ptr* is unused for the TDRV004 driver and should be set to NULL.

The following devctl command codes are defined in *tdrv004.h*:

Value	Description
DCMD_TD004_XSVFPLAY	Play an XSVF file for FPGA programming
DCMD_TD004_XSVFPOS	Retrieve current play-position in XSVF file
DCMD_TD004_XSVFLASTCMD	Get the last executed XSVF command
DCMD_TD004_RECONFIG	Trigger FPGA reconfiguration process
DCMD_TD004_SETWAITSTATES	Specify number of waitstates for programming
DCMD_TD004_SETCLOCK	Set clock generator parameters
DCMD_TD004_SPIWRITE	Write values to clock generator
DCMD_TD004_SPIREAD	Read values from clock generator
DCMD_TD004_PLXWRITEWORD	Write 16bit value to PLX9030 EEPROM
DCMD_TD004_PLXREADWORD	Read 16bit value from PLX9030 EEPROM

DCMD_TD004_READ_UCHAR	Read unsigned char values from FPGA resource
DCMD_TD004_READ_USHORT	Read unsigned short values from FPGA resource
DCMD_TD004_READ_ULONG	Read unsigned long values from FPGA resource
DCMD_TD004_WRITE_UCHAR	Write unsigned char values to FPGA resource
DCMD_TD004_WRITE_USHORT	Write unsigned short values to FPGA resource
DCMD_TD004_WRITE_ULONG	Write unsigned long values to FPGA resource
DCMD_TD004_CONFIGURE_INT	Configure local interrupt source polarity
DCMD_TD004_WAIT_FOR_INT1	Wait for incoming Local Interrupt Source 1
DCMD_TD004_WAIT_FOR_INT2	Wait for incoming Local Interrupt Source 2

See behind 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, EOK is returned. In the case of an error, the appropriate error code is returned by the function (not in errno!).

ERRORS

Returns only Neutrino specific error codes, see Neutrino Library Reference.

Other function dependent error codes will be described for each devctl code separately.

The TDRV004 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TD004_XSVFPLAY

NAME

DCMD_TD004_XSVFPLAY – Play an XSVF file for FPGA programming

DESCRIPTION

This TDRV004 control function programs the FPGA with a supplied XSVF file. The XSVF programming data is passed to the driver using a shared memory buffer. A pointer to the callers data buffer (*TD004_XSVF_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

```
typedef struct
{
        char            shMemName[TD004_MAXNAME_LEN];
        unsigned long   size;
} TD004_XSVF_BUF;
```

shMemName

This value specifies the name of the shared memory section. The maximum length of the name is limited to TD004_MAXNAME_LEN, which is defined in the file *tdrv004.h*.

size

This value specifies the total size of the mapped memory region for the specified shared memory object.

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 DCMD_TD004_SETWAITSTATES (refer to the corresponding section in this manual). Additionally, the CLK1 should not be lower than 10MHz for programming.

Due to the high PCI bus load during XSVF programming it is not possible to program more than one module at the same time.

EXAMPLE

```
#include <tdrv004.h>

#define SHARED_MEMORY_NAME   “/xsvfbuffer”
#define MEM_ALLOC_SIZE       3000000

int                fd;
int                result;
int                filesize;
int                sharedmemfd;
TD004_XSVF_BUF     XsvfBuf;

/*
** init shared-memory buffer
*/
     sharedmemfd = shm_open(SHARED_MEMORY_NAME, O_RDWR | O_CREAT, 0777);
     if (sharedmemfd == -1)
     {
          fprintf(stderr, "Open of shared memory failed: %s\n",
                  strerror(errno));
     }
     /* set size of shared memory */
filesize = MEM_ALLOC_SIZE;
if (ftruncate(sharedmemfd, filesize) == -1)
{
     fprintf(stderr, "error ftruncate: %s\n", strerror(errno));
}

/* map memory area */
filecontent = mmap(     0,
                        filesize,
                        PROT_READ | PROT_WRITE,
                        MAP_SHARED, sharedmemfd, 0);
if (filecontent == MAP_FAILED)
{
     fprintf(stderr, "mmap failed: %s\n", strerror(errno));
     pucPtr = NULL;
}
memset( filecontent, 0, MEM_ALLOC_SIZE);

/* init TD004_XSVF_BUF structure */
sprintf( XsvfBuf.shMemName, SHARED_MEMORY_NAME );
XsvfBuf.size = filesize;
```

```
/*
** read XSVF file content and copy it into the SharedMemory buffer
*/

/* start XSVF processing */
result = devctl(   fd,
                   DCMD_TD004_XSVFPLAY,
                   &XsvfBuf,
                   sizeof(TD004_XSVF_BUF),
                   NULL);
if (result != EOK)
{
     /* process devctl() error */
}

/* unmap and release the shared memory object */
munmap( filecontent, filesize );
close( sharedmemfd );
shm_unlink( SHARED_MEMORY_NAME );
```

ERRORS

EINVAL	An error occurred during XSVF processing.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
ENOBUFS	The specified shared memory object cannot be opened.
ENOSPC	The specified shared memory object is too small for the specified size.
EACCES	Mapping of the specified shared memory object failed.

All other returned error codes are system error conditions.

3.3.2 DCMD_TD004_XSVFPOS

NAME

DCMD_TD004_XSVFPOS – Retrieve current play-position in XSVF file

DESCRIPTION

This function returns the current position in the XSVF file during processing with DCMD_TD004_XSVFPLAY. A pointer to a caller's buffer (*unsigned long*) and the size of this buffer are passed by the parameters *data_ptr* and *n_bytes* to the device. The returned value is only valid during an XSVF processing.

EXAMPLE

```
#include <tdrv004.h>

int                     fd;
int                     result;
unsigned long           FilePos;

/*
** Retrieve current position in XSVF file
*/
result = devctl(   fd,
                   DCMD_TD004_XSVFPOS,
                   &FilePos,
                   sizeof(unsigned long),
                   NULL);
if (result == EOK)
{
     printf("FilePos = %d\n", FilePos);
} else {
     /* process devctl() error */
}
```

ERRORS

All returned error codes are system error conditions.

3.3.3 DCMD_TD004_XSVFLASTCMD

NAME

DCMD_TD004_XSVFLASTCMD – Get the last executed XSVF command

DESCRIPTION

This function returns the number of the last executed XSVF command. This value can be used to find errors inside the supplied XSVF file. This value refers to the line inside the ASCII SVF file. A pointer to a caller's buffer (*unsigned long*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

EXAMPLE

```
#include <tdrv004.h>

int                     fd;
int                     result;
unsigned long           LastCmd;

/*
** Retrieve the last executed XSVF command
*/
result = devctl(   fd,
                   DCMD_TD004_XSVFLASTCMD,
                   &LastCmd,
                   sizeof(unsigned long),
                   NULL);
if (result == EOK)
{
     printf("LastCmd = %d\n", LastCmd);
} else {
     /* process devctl() error */
}
```

ERRORS

All returned error codes are system error conditions.

3.3.4 DCMD_TD004_RECONFIG

NAME

DCMD_TD004_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. No additional parameter is used for this function.

EXAMPLE

```
#include <tdrv004.h>

int                fd;
int                result;

/*
** Start FPGA reconfiguration process
*/
result = devctl(   fd,
                   DCMD_TD004_RECONFIG,
                   NULL,
                   0,
                   NULL);
if (result != EOK)
{
     /* process devctl() error */
     break;
}
```

ERRORS

EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EIO	An error occurred during reconfiguration. The programmed XSVF content might be faulty.

All other returned error codes are system error conditions.

3.3.5 DCMD_TD004_SETWAITSTATES

NAME

DCMD_TD004_SETWAITSTATES – Specify number of waitstates for programming

DESCRIPTION

This function configures the driver to use a number of waitstates during XSVF and SPI programming. This might be necessary, if the local clock (CLK1) of the onboard clock generator is configured to rather slow. The local programming interface is clocked with this frequency, which might result in errors during programming for low CLK1 frequencies and a small amount of waitstates. By default no waitstate is used. The maximum allowed value is 100. A pointer to a caller's buffer (*unsigned long*) and the size of this buffer are passed by the parameters *data_ptr* and *n_bytes* to the device.

EXAMPLE

```
#include <tdrv004.h>

int                fd;
int                result;
unsigned long      WaitStates;

/*
** Setup driver to use 3 waitstates
*/
WaitStates = 3;
result = devctl(   fd,
                   DCMD_TD004_SETWAITSTATES,
                   &WaitStates,
                   sizeof(unsigned long),
                   NULL);
if (result != EOK)
{
     /* process devctl() error */
}
```

ERRORS

EINVAL	The supplied value is out of range (max. 100).

All other returned error codes are system error conditions.

3.3.6 DCMD_TD004_SETCLOCK

NAME

DCMD_TD004_SETCLOCK – Set clock generator parameters

DESCRIPTION

This function configures the onboard clock generator. A pointer to a caller's buffer (*TD004_CLOCK_PARAM*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

```
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;
```

Members

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;
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;
```

```
result = devctl(   fd,
                   DCMD_TD004_SETCLOCK,
                   &ClockParam,
                   sizeof(TD004_CLOCK_PARAM),
                   NULL);
if (result != EOK)
{
     /* process devctl() error */
}
```

ERRORS

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

All other returned error codes are system error conditions.

3.3.7 DCMD_TD004_SPIWRITE

NAME

DCMD_TD004_SPIWRITE – Write values to clock generator

DESCRIPTION

This function writes up to 256 *unsigned char* values to a specific sub-address of a Serial Programming Interface (SPI) address. A pointer to a caller's buffer (*TD004_SPI_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

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

Members

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 the control function DCMD_TD004_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[0] = 0x02;
pSpiBuf->pData[0] = 0x03;
pSpiBuf->pData[0] = 0x04;
pSpiBuf->pData[0] = 0x05;

result = devctl(   fd,
                   DCMD_TD004_SPIWRITE,
                   pSpiBuf,
                   BufferSize,
                   NULL);
if (result != EOK)
{
     /* process devctl() error */
}
```

ERRORS

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

All other returned error codes are system error conditions.

3.3.8 DCMD_TD004_SPIREAD

NAME

DCMD_TD004_SPIREAD – Read values from clock generator

DESCRIPTION

This function reads up to 256 *unsigned char* values from a specific sub-address of a Serial Programming Interface (SPI) address. A pointer to a caller's buffer (*TD004_SPI_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

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

Members

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 into 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 = devctl(   fd,
                   DCMD_TD004_SPIREAD,
                   pSpiBuf,
                   BufferSize,
                   NULL);
if (result != EOK)
{
     /* process devctl() error */
}
```

ERRORS

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

All other returned error codes are system error conditions.

3.3.9 DCMD_TD004_PLXWRITEWORD

NAME

DCMD_TD004_PLXWRITEWORD – Write 16bit value to PLX9030 EEPROM

DESCRIPTION

This function writes an *unsigned short* value to a specific PLX9030 memory offset. A pointer to a caller's buffer (*TD004_PLX_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

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;
```

Members

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.

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 = devctl(   fd,
                   DCMD_TD004_PLXWRITEWORD,
                   &PlxBuf,
                   sizeof(TD004_PLX_BUF),
                   NULL);
if (result != EOK)
{
     /* process devctl() error */
}
```

ERRORS

EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

All other returned error codes are system error conditions.

3.3.10 DCMD_TD004_PLXREADWORD

NAME

DCMD_TD004_PLXWRITEWORD – Read 16bit value from PLX9030 EEPROM

DESCRIPTION

This function reads an *unsigned short* value from a specific PLX9030 memory offset. A pointer to a caller's buffer (*TD004_PLX_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

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

Members

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;
TD004_PLX_BUF           PlxBuf;

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

result = devctl(   fd,
                   DCMD_TD004_PLXREADWORD,
                   &PlxBuf,
                   sizeof(TD004_PLX_BUF),
                   NULL);
if (result == EOK)
{
     printf( “SubsystemID = 0x%04X\n”, PlxBuf.Value );
} else {
     /* process devctl() error */
}
```

ERRORS

EBUSY	The device is already busy with XSVF, Reconfig or SPI action.

All other returned error codes are system error conditions.

3.3.11 DCMD_TD004_READ_UCHAR

NAME

DCMD_TD004_READ_UCHAR – Read unsigned char values from FPGA resource

DESCRIPTION

This function reads a number of *unsigned char* values from a Memory or I/O area by using BYTE (8bit) accesses. A pointer to a callers buffer (*TD004_MEMIO_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device. The data buffer can be enlarged to the desired needs. Due to restrictions of the I/O-Manager, 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;
```

Members

Resource

Specifies the desired PCI resource to read from. The TD004_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 TD004_RES_MEM_2, the second PCI-I/O space found is named TD004_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		TD004_RESOURCE
0	IO	*(reserved)*	TD004_RES_IO_1
1	MEM	*(reserved)*	TD004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TD004_RES_MEM_2
3	IO	*(not implemented by default)*	TD004_RES_IO_2
4	IO	*(not implemented by default)*	TD004_RES_IO_3
5	MEM	*(not implemented by default)*	TD004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

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

EXAMPLE

```
#include <tdrv004.h>

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

/*
** 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->Resource     = TD004_RES_MEM_2;
pMemIoBuf->Offset       = 0;
pMemIoBuf->Size         = 50;

result = devctl(   fd,
                   DCMD_TD004_READ_UCHAR,
                   pMemIoBuf,
                   BufferSize,
                   NULL);
if (result == EOK)
{
     pValues = (unsigned char*)pMemIoBuf->pData;
} else {
     /* process devctl() error */

}
```

ERRORS

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

All other returned error codes are system error conditions.

3.3.12 DCMD_TD004_READ_USHORT

NAME

DCMD_TD004_READ_USHORT – Read unsigned short values from FPGA resource

DESCRIPTION

This function reads a number of *unsigned short* values from a Memory or I/O area by using WORD (16bit) accesses. A pointer to a callers buffer (*TD004_MEMIO_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device. The data buffer can be enlarged to the desired needs. Due to restrictions of the I/O-Manager, 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;
```

Members

Resource

Specifies the desired PCI resource to read from. The TD004_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 TD004_RES_MEM_2, the second PCI-I/O space found is named TD004_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		TD004_RESOURCE
0	IO	*(reserved)*	TD004_RES_IO_1
1	MEM	*(reserved)*	TD004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TD004_RES_MEM_2
3	IO	*(not implemented by default)*	TD004_RES_IO_2
4	IO	*(not implemented by default)*	TD004_RES_IO_3
5	MEM	*(not implemented by default)*	TD004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

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

EXAMPLE

```
#include <tdrv004.h>

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

/*
** 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->Resource     = TD004_RES_MEM_2;
pMemIoBuf->Offset       = 0;
pMemIoBuf->Size         = 50;

result = devctl(   fd,
                   DCMD_TD004_READ_USHORT,
                   pMemIoBuf,
                   BufferSize,
                   NULL);
if (result == EOK)
{
     pValues = (unsigned short*)pMemIoBuf->pData;
} else {
     /* process devctl() error */

}
```

ERRORS

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

All other returned error codes are system error conditions.

3.3.13 DCMD_TD004_READ_ULONG

NAME

DCMD_TD004_READ_ULONG – Read unsigned long values from FPGA resource

DESCRIPTION

This function reads a number of *unsigned long* values from a Memory or I/O area by using DWORD (32bit) accesses. A pointer to a callers buffer (*TD004_MEMIO_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device. The data buffer can be enlarged to the desired needs. Due to restrictions of the I/O-Manager, 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;
```

Members

Resource

Specifies the desired PCI resource to read from. The TD004_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 TD004_RES_MEM_2, the second PCI-I/O space found is named TD004_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		TD004_RESOURCE
0	IO	*(reserved)*	TD004_RES_IO_1
1	MEM	*(reserved)*	TD004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TD004_RES_MEM_2
3	IO	*(not implemented by default)*	TD004_RES_IO_2
4	IO	*(not implemented by default)*	TD004_RES_IO_3
5	MEM	*(not implemented by default)*	TD004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

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

EXAMPLE

```
#include <tdrv004.h>

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

/*
** 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 short) );
pMemIoBuf = (TD004_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->Resource     = TD004_RES_MEM_2;
pMemIoBuf->Offset       = 0;
pMemIoBuf->Size         = 50;

result = devctl(   fd,
                   DCMD_TD004_READ_ULONG,
                   pMemIoBuf,
                   BufferSize,
                   NULL);
if (result == EOK)
{
     pValues = (unsigned long*)pMemIoBuf->pData;
} else {
     /* process devctl() error */

}
```

ERRORS

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

All other returned error codes are system error conditions.

3.3.14 DCMD_TD004_WRITE_UCHAR

NAME

DCMD_TD004_WRITE_UCHAR – Write unsigned char values to FPGA resource

DESCRIPTION

This function writes a number of *unsigned char* values to a Memory or I/O area by using BYTE (8bit) accesses. A pointer to a callers buffer (*TD004_MEMIO_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device. The data buffer can be enlarged to the desired needs. Due to restrictions of the I/O-Manager, 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;
```

Members

Resource

Specifies the desired PCI resource to write to. The TD004_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 TD004_RES_MEM_2, the second PCI-I/O space found is named TD004_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		TD004_RESOURCE
0	IO	*(reserved)*	TD004_RES_IO_1
1	MEM	*(reserved)*	TD004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TD004_RES_MEM_2
3	IO	*(not implemented by default)*	TD004_RES_IO_2
4	IO	*(not implemented by default)*	TD004_RES_IO_3
5	MEM	*(not implemented by default)*	TD004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to write.

pData

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

EXAMPLE

```
#include <tdrv004.h>

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

/*
** 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 );
pValues = (unsigned char*)pMemIoBuf->pData;
pValues[0] = 0x01;
pValues[1] = 0x02;
...
pMemIoBuf->Resource     = TD004_RES_MEM_2;
pMemIoBuf->Offset       = 0;
pMemIoBuf->Size         = 10;

result = devctl(   fd,
                   DCMD_TD004_WRITE_UCHAR,
                   pMemIoBuf,
                   BufferSize,
                   NULL);
if (result != EOK)
{
     /* process devctl() error */

}
```

ERRORS

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

All other returned error codes are system error conditions.

3.3.15 DCMD_TD004_WRITE_USHORT

NAME

DCMD_TD004_WRITE_USHORT – Write unsigned short values to FPGA resource

DESCRIPTION

This function writes a number of *unsigned short* values to a Memory or I/O area by using WORD (16bit) accesses. A pointer to a callers buffer (*TD004_MEMIO_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device. The data buffer can be enlarged to the desired needs. Due to restrictions of the I/O-Manager, 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;
```

Members

Resource

Specifies the desired PCI resource to write to. The TD004_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 TD004_RES_MEM_2, the second PCI-I/O space found is named TD004_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		TD004_RESOURCE
0	IO	*(reserved)*	TD004_RES_IO_1
1	MEM	*(reserved)*	TD004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TD004_RES_MEM_2
3	IO	*(not implemented by default)*	TD004_RES_IO_2
4	IO	*(not implemented by default)*	TD004_RES_IO_3
5	MEM	*(not implemented by default)*	TD004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to write.

pData

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

EXAMPLE

```
#include <tdrv004.h>

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

/*
** 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 );
pValues = (unsigned short*)pMemIoBuf->pData;
pValues[0] = 0x0001;
pValues[1] = 0x0002;

pMemIoBuf->Resource     = TD004_RES_MEM_2;
pMemIoBuf->Offset       = 0;
pMemIoBuf->Size         = 10;

result = devctl(   fd,
                   DCMD_TD004_WRITE_USHORT,
                   pMemIoBuf,
                   BufferSize,
                   NULL);
if (result != EOK)
{
     /* process devctl() error */

}
```

ERRORS

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

All other returned error codes are system error conditions.

3.3.16 DCMD_TD004_WRITE_ULONG

NAME

DCMD_TD004_WRITE_ULONG – Write unsigned long values to FPGA resource

DESCRIPTION

This function writes a number of *unsigned long* values to a Memory or I/O area by using DWORD (32bit) accesses. A pointer to a callers buffer (*TD004_MEMIO_BUF*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device. The data buffer can be enlarged to the desired needs. Due to restrictions of the I/O-Manager, 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;
```

Members

Resource

Specifies the desired PCI resource to write to. The TD004_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 TD004_RES_MEM_2, the second PCI-I/O space found is named TD004_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		TD004_RESOURCE
0	IO	*(reserved)*	TD004_RES_IO_1
1	MEM	*(reserved)*	TD004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TD004_RES_MEM_2
3	IO	*(not implemented by default)*	TD004_RES_IO_2
4	IO	*(not implemented by default)*	TD004_RES_IO_3
5	MEM	*(not implemented by default)*	TD004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to write.

pData

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

EXAMPLE

```
#include <tdrv004.h>

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

/*
** 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 );
pValues = (unsigned long*)pMemIoBuf->pData;
pValues[0] = 0x00000001;
pValues[1] = 0x00000002;
…
pMemIoBuf->Resource     = TD004_RES_MEM_2;
pMemIoBuf->Offset       = 0;
pMemIoBuf->Size         = 10;

result = devctl(   fd,
                   DCMD_TD004_WRITE_ULONG,
                   pMemIoBuf,
                   BufferSize,
                   NULL);
if (result != EOK)
{
     /* process devctl() error */
}
```

ERRORS

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

All other returned error codes are system error conditions.

3.3.17 DCMD_TD004_CONFIGURE_INT

NAME

DCMD_TD004_CONFIGURE_INT – Configure local interrupt source polarity

DESCRIPTION

This function configures the polarity of the PLX PCI9030 interrupt sources. A pointer to a caller's buffer (*unsigned long*) and the size of this buffer are passed by the parameters *data_ptr* and *n_bytes* to the device. 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 = devctl(   fd,
                   DCMD_TD004_CONFIGURE_INT,
                   &IntConfig,
                   sizeof(unsigned long),
                   NULL);
if (result != EOK)
{
     /* process devctl() error */
}
```

ERRORS

EINVAL	The supplied value is invalid.

All other returned error codes are system error conditions.

3.3.18 DCMD_TD004_WAIT_FOR_INT1

NAME

DCMD_TD004_WAIT_FOR_INT1 – Wait for incoming Local Interrupt Source 1

DESCRIPTION

This function enables the corresponding interrupt source, and waits for Local Interrupt Source 1 (LINT1) to arrive. After the interrupt has arrived, this specific local interrupt source is disabled. A pointer to a caller's buffer (*int*) and the size of this buffer are passed by the parameters *data_ptr* and *n_bytes* to the device. This value contains the timeout in seconds. 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 5 seconds for incoming interrupt LINT1
*/
Timeout = 5;
result = devctl(    fd,
                    DCMD_TD004_WAIT_FOR_INT1,
                    &Timeout,
                    sizeof(int),
                    NULL);
if (result != EOK)
{
     /* process devctl() error */
}
```

ERRORS

EBUSY	The device is already busy waiting for this interrupt.
ETIMEDOUT	The interrupt has not arrived during the specified timeout.

All other returned error codes are system error conditions.

3.3.19 DCMD_TD004_WAIT_FOR_INT2

NAME

DCMD_TD004_WAIT_FOR_INT2 – Wait for incoming Local Interrupt Source 2

DESCRIPTION

This function enables the corresponding interrupt source, and waits for Local Interrupt Source 2 (LINT2) to arrive. After the interrupt has arrived, this specific local interrupt source is disabled. A pointer to a caller's buffer (*int*) and the size of this buffer are passed by the parameters *data_ptr* and *n_bytes* to the device. This value contains the timeout in seconds. 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 indefinitely for incoming interrupt LINT2
*/
Timeout = -1;
result = devctl(    fd,
                    DCMD_TD004_WAIT_FOR_INT2,
                    &Timeout,
                    sizeof(int),
                    NULL);
if (result != EOK)
{
     /* process devctl() error */
}
```

ERRORS

EBUSY	The device is already busy waiting for this interrupt.
ETIMEDOUT	The interrupt has not arrived during the specified timeout.

All other returned error codes are system error conditions.