

The Embedded I/O Company



TDRV004-SW-42

VxWorks Device Driver

Reconfigurable FPGA

Version 1.0.x

User Manual

Issue 1.0.0

May 2006

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
Phone: +49-(0)4101-4058-0
e-mail: info@tews.com

25469 Halstenbek / Germany
Fax: +49-(0)4101-4058-19
www.tews.com

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Phone: +1 (775) 686 6077
e-mail: usasales@tews.com

Reno, Nevada 89504 / USA
Fax: +1 (775) 686 6024
www.tews.com

TDRV004-SW-42

Reconfigurable FPGA

VxWorks Device Driver

Supported Modules:

TPMC630

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

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

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

©2006 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	May 04, 2006

Table of Content

1	INTRODUCTION.....	4
2	INSTALLATION	5
	2.1 Include device driver in Tornado IDE project	5
	2.2 Special installation for Intel x86 based targets.....	5
	2.3 System resource requirement	6
3	I/O SYSTEM FUNCTIONS.....	7
	3.1 tdrv004Drv().....	7
	3.2 tdrv004DevCreate()	9
	3.3 tdrv004Pcilnit().....	11
4	I/O FUNCTIONS	12
	4.1 open().....	12
	4.2 close().....	14
	4.3 ioctl().....	16
	4.3.1 FIO_TDRV004_XSVFPLAY	18
	4.3.2 FIO_TDRV004_XSVFPOS	20
	4.3.3 FIO_TDRV004_XSVFLASTCMD	21
	4.3.4 FIO_TDRV004_RECONFIG.....	22
	4.3.5 FIO_TDRV004_SETWAITSTATES	23
	4.3.6 FIO_TDRV004_SETCLOCK	24
	4.3.7 FIO_TDRV004_SPIWRITE.....	27
	4.3.8 FIO_TDRV004_SPIREAD.....	29
	4.3.9 FIO_TDRV004_PLXWRITE	31
	4.3.10 FIO_TDRV004_PLXREAD	33
	4.3.11 FIO_TDRV004_READ_UCHAR	35
	4.3.12 FIO_TDRV004_READ_USHORT	37
	4.3.13 FIO_TDRV004_READ_ULONG.....	39
	4.3.14 FIO_TDRV004_WRITE_UCHAR	41
	4.3.15 FIO_TDRV004_WRITE_USHORT.....	43
	4.3.16 FIO_TDRV004_WRITE_ULONG	45

1 Introduction

The TDRV004-SW-42 VxWorks device driver software allows the operation of the TPMC630 product family conforming to the VxWorks 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.

This driver invokes a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

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

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

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

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

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

- TPMC630 User Manual
- TPMC630 Engineering Manual
- Cypress CY27EE16 User Manual

2 Installation

Following files are located on the distribution media:

Directory path ‘.\TDRV004-SW-42\’:

tdrv004drv.c	TDRV004 device driver source
tdrv004def.h	TDRV004 driver include file
tdrv004.h	TDRV004 include file for driver and application
tdrv004pci.c	TDRV004 PCI MMU mapping for Intel x86 based targets
tdrv004exa.c	Example application
pf_micro.c	XSVF player functions (Platform Flash)
pf_micro.h	header file for XSVF player functions
pf_lenval.c	special functions for XSVF player
pf_lenval.h	header file for XSVF functions
pf_ports.c	hardware layer for XSVF player
pf_ports.h	header file for XSVF hardware layer
include/tdhal.h	Hardware dependent interface functions and definitions
fpgaexa.zip	FPGA example XSVF
TDRV004-SW-42-1.0.0.pdf	PDF copy of this manual
Release.txt	Release information

2.1 Include device driver in Tornado IDE project

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

- (1) Copy the files from the distribution media into a subdirectory in your project path.
(For example: ./TDRV004)
- (2) Add the device drivers C-files to your project.
Make a right click to your project in the ‘Workspace’ window and use the ‘Add Files ...’ topic.
A file select box appears, and the driver files can be selected.
- (3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your Tornado User’s Guide.

2.2 Special installation for Intel x86 based targets

The TDRV004 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro **CPU_FAMILY**. If the content of this macro is equal to *I80X86* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can’t be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP’s doesn’t map PCI memory spaces of devices which are not used by the BSP, the required device memory spaces can’t be accessed.

To solve this problem a MMU mapping entry has to be added for the required TDRV004 PCI memory spaces prior the MMU initialization (*usrMmulnit()*) is done.

The C source file **tdrv004pci.c** contains the function *tdrv004PciInit()*. This routine finds out all TDRV004 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmulnit()*).

The correct place to call the function `tdrv004PciInit()` is at the end of the function `sysHwInit()` in **sysLib.c** (it can be opened from the project *Files* window).

Be sure that the function is called prior to MMU initialization otherwise the TDRV004 PCI spaces remains unmapped and an access fault occurs during driver initialization.

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

```
tdrv004PciInit();
```

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

2.3 System resource requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 1 KB
Stack	< 1 KB	---
Semaphores	---	1

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

$$\langle \text{total requirement} \rangle = \langle \text{driver requirement} \rangle + (\langle \text{number of devices} \rangle * \langle \text{device requirement} \rangle)$$

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

3 I/O system functions

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

3.1 tdrv004Drv()

NAME

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

SYNOPSIS

```
#include "tdrv004.h"
```

```
STATUS tdrv004Drv(void)
```

DESCRIPTION

This function searches for devices on the PCI bus and installs the TDRV004 driver in the I/O system.

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

EXAMPLE

```
#include "tdrv004.h"

...

/*-----
   Initialize Driver
   -----*/
status = tdrv004Drv();
if (status == ERROR)
{
    /* Error handling */
}

...
```

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tdrv004DevCreate()

NAME

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

SYNOPSIS

```
#include "tdrv004.h"
```

```
STATUS tdrv004DevCreate
```

```
(
    char      *name,
    int       devIdx,
    int       funcType,
    void      *pParam
)
```

DESCRIPTION

This routine adds the selected device to the VxWorks system. The device hardware will be setup and prepared for use.

This function must be called before performing any I/O request to this device.

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

devIdx

This index number specifies the device to add to the system. If modules of the same type are installed the channel numbers will be advised in the order the VxWorks *pciFindDevice()* function will find the devices.

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

Module	Device Index
TPMC630-10 (1 st)	0
TPMC630-10 (2 nd)	1

funcType

This parameter is unused and should be set to 0.

pParam

This parameter is unused and should be set to *NULL*.

EXAMPLE

```
#include "tdrv004.h"

...

STATUS          result;

...

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

result = tdrv004DevCreate(  "/tdrv004/0",
                           0,
                           0,
                           NULL);

if (result == OK)
{
    /* Device successfully created */
}
else
{
    /* Error occurred when creating the device */
}

...
```

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

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

Error code	Description
ENXIO	No matching device found.
S_ioLib_NO_DRIVER	Device driver not installed.

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.3 tdrv004Pcilnit()

NAME

tdrv004Pcilnit() – Generic PCI device initialization

SYNOPSIS

```
void tdrv004Pcilnit()
```

DESCRIPTION

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

The global variable *tdrv004Status* obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

Value	Meaning
> 0	Initialization successful completed. The value of <i>tdrv004Status</i> is equal to the number of mapped PCI spaces
0	No TDRV004 device found
< 0	Initialization failed. The value of (<i>tdrv004Status</i> & 0xFF) is equal to the number of mapped spaces until the error occurs. Possible cause: Too few entries for dynamic mappings in <i>sysPhysMemDesc[]</i> . Remedy: Add dummy entries as necessary (<i>syslib.c</i>).

EXAMPLE

```
extern void tdrv004PciInit();
```

...

```
tdrv004PciInit();
```

...

4 I/O Functions

4.1 open()

NAME

open() - open a device or file.

SYNOPSIS

```
int open
(
    const char *name,
    int        flags,
    int        mode
)
```

DESCRIPTION

Before I/O can be performed to the TDRV004 device, a file descriptor must be opened by invoking the basic I/O function *open()*.

PARAMETER

name

Specifies the device which shall be opened, the name specified in *tdrv004DevCreate()* must be used

flags

Not used

mode

Not used

EXAMPLE

```
int        fd;

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

RETURNS

A device descriptor number or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

Error codes can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - *open()*

4.2 close()

NAME

close() – close a device or file

SYNOPSIS

```
STATUS close
(
    int      fd
)
```

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

This file descriptor specifies the device to be closed. The file descriptor has been returned by the *open()* function.

EXAMPLE

```
int      fd;
STATUS   retval;

/*-----
   close the device
   -----*/
retval = close(fd);
```

RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

Error codes can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - close()

4.3 ioctl()

NAME

ioctl() - performs an I/O control function.

SYNOPSIS

```
#include "tdrv004.h"
```

```
int ioctl
(
    int    fd,
    int    request,
    int    arg
)
```

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls (read, write) will be performed by calling the ioctl() function.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

request

This argument specifies the function that shall be executed. Following functions are defined:

Function	Description
FIO_TDRV004_XSVFPLAY	Play an XSVF file for FPGA programming
FIO_TDRV004_XSVFPOS	Retrieve current play-position in XSVF file
FIO_TDRV004_XSVFLASTCMD	Get the last executed XSVF command
FIO_TDRV004_RECONFIG	Trigger FPGA reconfiguration process
FIO_TDRV004_SETWAITSTATES	Specify number of waitstates for programming
FIO_TDRV004_SETCLOCK	Set clock generator parameters
FIO_TDRV004_SPIWRITE	Write values to clock generator
FIO_TDRV004_SPIREAD	Read values from clock generator
FIO_TDRV004_PLXWRITE	Write 16bit value to PLX9030 EEPROM
FIO_TDRV004_PLXREAD	Read 16bit value from PLX9030 EEPROM
FIO_TDRV004_READ_UCHAR	Read unsigned char values from FPGA resource
FIO_TDRV004_READ_USHORT	Read unsigned short values from FPGA resource
FIO_TDRV004_READ_ULONG	Read unsigned long values from FPGA resource

FIO_TDRV004_WRITE_UCHAR	Write unsigned char values to FPGA resource
FIO_TDRV004_WRITE_USHORT	Write unsigned short values to FPGA resource
FIO_TDRV004_WRITE_ULONG	Write unsigned long values to FPGA resource

arg

This parameter depends on the selected function (request). How to use this parameter is described below with the function.

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual). Function specific error codes will be described with the function.

SEE ALSO

ioLib, basic I/O routine - *ioctl()*

4.3.1 FIO_TDRV004_XSVFPLAY

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

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

TDRV004_XSVF_BUF STRUCTURE

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

size

Specifies the total size of the supplied XSVF data.

pData

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

Programming Hints

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

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

EXAMPLE

```
#include "tdrv004.h"

...

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

...
```

```
...

/*
** allocate enough memory (about 3MB) to hold XSVF content
*/
bufsize = sizeof(TDRV004_XSVF_BUF) + 3000000*sizeof(unsigned char);
pXsvfBuf = (TDRV004_XSVF_BUF*)malloc( bufsize );

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

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

...
```

ERROR CODES

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

Other returned error codes are system error conditions.

4.3.2 FIO_TDRV004_XSVFPOS

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

EXAMPLE

```
#include "tdrv004.h"

...

int          fd;
unsigned long XsvfPos;
int          retval;

...

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

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

...
```

ERROR CODES

This ioctl function returns no function specific error codes.

4.3.3 FIO_TDRV004_XSVFLASTCMD

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

EXAMPLE

```
#include "tdrv004.h"

...

int          fd;
unsigned long XsvfLastCmd;
int          retval;

...

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

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

...
```

ERROR CODES

This ioctl function returns no function specific error codes.

4.3.4 FIO_TDRV004_RECONFIG

This I/O control function the reconfiguration process of the FPGA. This control function must be called after the FPGA is programmed using FIO_TDRC004_XSVFPLAY. The function returns after the reconfiguration is done, or an error occurred. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include "tdrv004.h"

...

int          fd;
int          retval;

...

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

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

...
```

ERROR CODES

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

Other returned error codes are system error conditions.

4.3.5 FIO_TDRV004_SETWAITSTATES

This I/O control function configures the driver to use a number of waitstates during XSVF and SPI programming. This might be necessary, if the local clock (CLK1) of the onboard clock generator is configured to rather slow. The local programming interface is clocked with this frequency, which might result in errors during programming for low CLK1 frequencies and a small amount of waitstates. The function specific control parameter **arg** is a pointer to an unsigned long value.

EXAMPLE

```
#include "tdrv004.h"

...

int          fd;
unsigned long WaitStates;
int          retval;

...

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

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

...
```

ERROR CODES

This ioctl function returns no function specific error codes.

4.3.6 FIO_TDRV004_SETCLOCK

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

***TDRV004_CLOCK_PARAM* STRUCTURE**

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

DeviceAddr

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

x09_ClkOE

Specifies which clock outputs shall be enabled.

x0C_DIV1SRCN

Specifies internal input source 1 and the corresponding frequency divider

x10_InputCtrl

Specifies value for the Input Pin Control register

x40_CPumpPB

Specifies value for Charge Pump and PB counter register

x41_CPumpPB

Specifies value for Charge Pump and PB counter register

x42_POQcnt

Specifies value for PO and Q counter register

x44_SwMatrix

Specifies value for Switching Matrix Register

x45_SwMatrix

Specifies value for Switching Matrix Register

x46_SwMatrix

Specifies value for Switching Matrix Register

x47_DIV2SRCN

Specifies internal input source 2 and the corresponding frequency divider

Please refer to the Cypress CY27EE16 user manual for detailed explanation of the above register values.

EXAMPLE

```
#include "tdrv004.h"

...

int          fd;
TDRV004_CLOCK_PARAM  ClockParam;
int          retval;

...

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

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

...
```

ERROR CODES

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

Other returned error codes are system error conditions.

4.3.7 FIO_TDRV004_SPIWRITE

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

TDRV004_SPI_BUF STRUCTURE

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

SpiAddr

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

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

SubAddr

Specifies the sub-address (starting offset).

len

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

pData

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

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

EXAMPLE

```
#include "tdrv004.h"

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

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

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

ERROR CODES

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

Other returned error codes are system error conditions.

4.3.8 FIO_TDRV004_SPIREAD

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

TDRV004_SPI_BUF STRUCTURE

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

SpiAddr

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

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

SubAddr

Specifies the sub-address (starting offset).

len

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

pData

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

EXAMPLE

```
#include "tdrv004.h"

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

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

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

ERROR CODES

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

Other returned error codes are system error conditions.

4.3.9 FIO_TDRV004_PLXWRITE

This I/O control function writes an *unsigned short* value to a specific PLX9030 EEPROM memory offset. The function specific control parameter **arg** is a pointer to a *TDRV004_PLX_BUF* structure.

TDRV005_PLX_BUF STRUCTURE

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

Offset

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

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

Refer to the 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;
TDRV004_PLX_BUF PlxBuf;
int          retval;

...

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

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

...
```

ERROR CODES

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

Other returned error codes are system error conditions.

4.3.10 FIO_TDRV004_PLXREAD

This I/O control function reads an *unsigned short* value from a specific PLX9030 EEPROM memory offset. The function specific control parameter **arg** is a pointer to a *TDRV004_PLX_BUF* structure.

TDRV005_PLX_BUF STRUCTURE

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

Offset

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

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

Refer to the PLX9030 User Manual for detailed information on these registers.

Value

This value holds the retrieved 16bit word.

EXAMPLE

```
#include "tdrv004.h"

...

int          fd;
TDRV004_PLX_BUF PlxBuf;
int          retval;

...

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

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

...
```

ERROR CODES

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

Other returned error codes are system error conditions.

4.3.11 FIO_TDRV004_READ_UCHAR

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

TDRV004_MEMIO_BUF STRUCTURE

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

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.

The Base Address Register usage is programmable and can be changed by modifying the 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 <i>(reserved)</i>	TDRV004_RES_IO_1
1	MEM <i>(reserved)</i>	TDRV004_RES_MEM_1
2	MEM <i>(used by VHDL Example)</i>	TDRV004_RES_MEM_2
3	IO <i>(not implemented by default)</i>	TDRV004_RES_IO_2
4	IO <i>(not implemented by default)</i>	TDRV004_RES_IO_3
5	MEM <i>(not implemented by default)</i>	TDRV004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

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

EXAMPLE

```
#include "tdrv004.h"

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

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

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

ERROR CODES

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

Other returned error codes are system error conditions.

4.3.12 FIO_TDRV004_READ_USHORT

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

TDRV004_MEMIO_BUF STRUCTURE

typedef struct

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

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.

The Base Address Register usage is programmable and can be changed by modifying the 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 <i>(reserved)</i>	TDRV004_RES_IO_1
1	MEM <i>(reserved)</i>	TDRV004_RES_MEM_1
2	MEM <i>(used by VHDL Example)</i>	TDRV004_RES_MEM_2
3	IO <i>(not implemented by default)</i>	TDRV004_RES_IO_2
4	IO <i>(not implemented by default)</i>	TDRV004_RES_IO_3
5	MEM <i>(not implemented by default)</i>	TDRV004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

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

EXAMPLE

```
#include "tdrv004.h"

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

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

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

ERROR CODES

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

Other returned error codes are system error conditions.

4.3.13 FIO_TDRV004_READ_ULONG

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

TDRV004_MEMIO_BUF STRUCTURE

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

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.

The Base Address Register usage is programmable and can be changed by modifying the 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 <i>(reserved)</i>	TDRV004_RES_IO_1
1	MEM <i>(reserved)</i>	TDRV004_RES_MEM_1
2	MEM <i>(used by VHDL Example)</i>	TDRV004_RES_MEM_2
3	IO <i>(not implemented by default)</i>	TDRV004_RES_IO_2
4	IO <i>(not implemented by default)</i>	TDRV004_RES_IO_3
5	MEM <i>(not implemented by default)</i>	TDRV004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

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

EXAMPLE

```
#include "tdrv004.h"

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

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

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

ERROR CODES

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

4.3.14 FIO_TDRV004_WRITE_UCHAR

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

TDRV004_MEMIO_BUF STRUCTURE

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

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.

The Base Address Register usage is programmable and can be changed by modifying the 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 <i>(reserved)</i>	TDRV004_RES_IO_1
1	MEM <i>(reserved)</i>	TDRV004_RES_MEM_1
2	MEM <i>(used by VHDL Example)</i>	TDRV004_RES_MEM_2
3	IO <i>(not implemented by default)</i>	TDRV004_RES_IO_2
4	IO <i>(not implemented by default)</i>	TDRV004_RES_IO_3
5	MEM <i>(not implemented by default)</i>	TDRV004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to write.

pData

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

EXAMPLE

```
#include "tdrv004.h"

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

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

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

ERROR CODES

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

Other returned error codes are system error conditions.

4.3.15 FIO_TDRV004_WRITE_USHORT

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

TDRV004_MEMIO_BUF STRUCTURE

typedef struct

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

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.

The Base Address Register usage is programmable and can be changed by modifying the 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 <i>(reserved)</i>	TDRV004_RES_IO_1
1	MEM <i>(reserved)</i>	TDRV004_RES_MEM_1
2	MEM <i>(used by VHDL Example)</i>	TDRV004_RES_MEM_2
3	IO <i>(not implemented by default)</i>	TDRV004_RES_IO_2
4	IO <i>(not implemented by default)</i>	TDRV004_RES_IO_3
5	MEM <i>(not implemented by default)</i>	TDRV004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to write.

pData

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

EXAMPLE

```
#include "tdrv004.h"

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

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

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

ERROR CODES

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

4.3.16 FIO_TDRV004_WRITE_ULONG

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

TDRV004_MEMIO_BUF STRUCTURE

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

Resource

Specifies the desired PCI resource to read from. The TDRV004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TDRV004_RES_MEM_2, the second PCI-I/O space found is named TDRV004_RES_IO_2 and so on.

The Base Address Register usage is programmable and can be changed by modifying the 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 <i>(reserved)</i>	TDRV004_RES_IO_1
1	MEM <i>(reserved)</i>	TDRV004_RES_MEM_1
2	MEM <i>(used by VHDL Example)</i>	TDRV004_RES_MEM_2
3	IO <i>(not implemented by default)</i>	TDRV004_RES_IO_2
4	IO <i>(not implemented by default)</i>	TDRV004_RES_IO_3
5	MEM <i>(not implemented by default)</i>	TDRV004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to write.

pData

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

EXAMPLE

```
#include "tdrv004.h"

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

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

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

ERROR CODES

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

Other returned error codes are system error conditions.