The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV004-SW-65

Windows Device Driver

Reconfigurable FPGA

Version 2.0.x

User Manual

Issue 2.0.0

November 2011

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-65

Windows Device Driver

Reconfigurable FPGA

Supported Modules:
TPMC630
TCP630

Issue	Description	Date
1.0.0	First Issue	July 11, 2005
1.0.1	Detailed description of TD004_RESOURCE added. Manual filename changed in installation section.	March 31, 2006
1.1.0	Interrupt features added, filelist changed	August 10, 2006
1.1.1	New Address TEWS LLC	October 25, 2006
1.1.2	Files moved to subdirectory	June 23, 2008
2.0.0	Driver supports Windows7, Description of API Functions added, Description of I/O Functions removed, General Revision	November 18, 2011

Table of Contents

1 Introduction

The TDRV004-SW-65 Windows device driver is a kernel mode driver which allows the operation of supported hardware modules on an Intel or Intel-compatible Windows operating system. Supported Windows versions are:

- Windows 2000
- Windows XP
- Windows XP Embedded
- Windows 7 (32bit and 64bit)

The TDRV004-SW-65 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 EEPROM registers

The TDRV004-SW-65 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 TDRV004 devices it is recommended to read the manuals listed below.

TPMC630 / TCP630 User manual
TPMC630 / TCP630 Engineering Manual

2 Installation

Following files are located in directory TDRV004-SW-65 on the distribution media:

i386\	Directory containing driver files for 32bit Windows versions
amd64\	Directory containing driver files for 64bit Windows versions
installer_32bit.exe	Installation tool for 32bit systems (Windows XP or later)
installer_64bit.exe	Installation tool for 64bit systems (Windows XP or later)
tdrv004.inf	Windows installation script
tdrv004.h	Header file with IOCTL codes and structure definitions
example\tdrv004exa.c	Example application
example\fpgaexa.zip	Example FPGA design (XSVF files) as a ZIP archive
api\tdrv004api.c	Application Programming Interface source
api\tdrv004api.h	Application Programming Interface header
TDRV004-SW-65-2.0.0.pdf	This document
Release.txt	Information about the Device Driver Release
ChangeLog.txt	Release history

2.1 Software Installation

2.1.1 Windows 2000

This section describes how to install the TDRV004 Device Driver on a Windows 2000 operating system.

After installing the TDRV004 card(s) and boot-up your system, Windows 2000 setup will show a "***New hardware found***" dialog box.

1. The "***Upgrade Device Driver Wizard***" dialog box will appear on your screen.
   Click "***Next***" button to continue.
2. In the following dialog box, choose "***Search for a suitable driver for my device***".
   Click "***Next***" button to continue.
3. Insert the TDRV004 driver media; select "***Disk Drive***" in the dialog box.
   Click "***Next***" button to continue.
4. Now the driver wizard should find a suitable device driver on the media.
   Click "***Next***" button to continue.
5. Complete the upgrade device driver and click "***Finish***" to take all the changes effect.
6. Now copy all needed files (tdrv004.h and API files) to the desired target directories.

After successful installation the TDRV004 device driver will start immediately and creates devices (TDRV004_1, TDRV004_2 ...) for all recognized TDRV004 modules.

2.1.2 Windows 7 / XP

This section describes how to install the TDRV004-SW-65 Device Driver on a Windows 7 (32bit or 64bit) or Windows XP (32-bit) operating system.

Depending on the operating system type, execute the installer binaries for either 32bit or 64bit systems. This will install all required driver files using an installation wizard.

Copy needed files (tdrv004.h and API files) to desired target directory.

After successful installation a device is created for each module found (TDRV004_1, TDRV004_2 ...).

2.2 Confirming Driver Installation

To confirm that the driver has been properly loaded, perform the following steps:

1. Open the Windows Device Manager:

   a. For Windows 2000 / XP, open the "***Control Panel***" from "***My Computer***" and click the "***System***" icon and choose the "***Hardware***" tab, and then click the "***Device Manager***" button.

   b. For Windows 7, open the "***Control Panel***" from "***My Computer***" and then click the "***Device Manager***" entry.

2. Click the "**+**" in front of "***Embedded I/O***".
   The driver "***TEWS TECHNOLOGIES – TDRV004 (Reconfigurable FPGA) (<Modulename>)***" should appear for each installed device.

3 API Documentation

3.1 General Functions

3.1.1 tdrv004Open

NAME

tdrv004Open – Opens a Device

SYNOPSIS

```
TDRV004_HANDLE tdrv004Open
(
      char          *DeviceName
);
```

DESCRIPTION

Before I/O can be performed to a device, a file descriptor must be opened by a call to this function.

PARAMETERS

DeviceName

This parameter points to a null-terminated string that specifies the name of the device.

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE     hdl;

/*
** open file descriptor to device
*/
hdl = tdrv004Open(“\\\\.\\TDRV004_1”);
if (hdl == NULL)
{
     /* handle open error */
}
```

RETURNS

A device handle, or NULL if the function fails. To get extended error information, call ***GetLastError***.

ERROR CODES

All error codes are standard error codes set by the I/O system.

3.1.2 tdrv004Close

NAME

tdrv004Close – Closes a Device

SYNOPSIS

```
TDRV004_STATUS tdrv004Close
(
      TDRV004_HANDLE          hdl
);
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE hdl;
TDRV004_STATUS result;

/*
** close file descriptor to device
*/
result = tdrv004Close(hdl);
if (result != TDRV004_OK)
{
     /* handle close error */
}
```

RETURNS

On success TDRV004_OK, or an appropriate error code.

ERROR CODES

All error codes are standard error codes set by the I/O system.

3.2 Device Access Functions

3.2.1 tdrv004XsvfPlay

NAME

tdrv004XsvfPlay – Play an XSVF file for FPGA programming

SYNOPSIS

```
TDRV004_STATUS tdrv004XsvfPlay
(
    TDRV004_HANDLE      hdl,
    unsigned char       *pXsvfContent,
    unsigned int        XsvfSize
);
```

DESCRIPTION

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

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

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.

If the programming fails, try to increase the used waitstates with control function tdrv004SetWaitStates() (refer to the corresponding section in this manual). Additionally, the CLK1 should not be lower than 10MHz for programming.

PARAMETERS

hdl

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

pXsvfContent

This argument points to an unsigned char array containing the XSVF data.

XsvfSize

This argument holds the size of the supplied XSVF data area.

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE    hdl;
TDRV004_STATUS    result;
unsigned char     *pXsvfContent;
unsigned long     XsvfFileSize;

/*
** Play an XSVF file to program the FPGA.
** The filecontent must be available in a local buffer,
** the size of the file must be stored in XsvfFileSize.
*/
result = tdrv004XsvfPlay (   hdl,
                             pXsvfContent,
                             XsvfFileSize );
if (result != TDRV004_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_BUSY	The device is already busy with XSVF
TDRV004_ERR_INVAL	An error occurred during XSVF operation

3.2.2 tdrv004XsvfPos

NAME

tdrv004XsvfPos – Retrieve current play-position in XSVF file

SYNOPSIS

```
TDRV004_STATUS tdrv004XsvfPos
(
    TDRV004_HANDLE      hdl,
    int                 *pXsvfPos
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

pXsvfPos

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE      hdl;
TDRV004_STATUS      result;
int                 XsvfPos;

/*
** Check XSVF position
*/
result = tdrv004XsvfPos(   hdl,
                           &XsvfPos );
if (result == TDRV004_OK)
{
     printf(“Current XSVF position: %d\n”, XsvfPos);
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.

3.2.3 tdrv004XsvfLastCommand

NAME

tdrv004XsvfLastCommand – Get the last executed XSVF command

SYNOPSIS

```
TDRV004_STATUS tdrv004XsvfLastCommand
(
    TDRV004_HANDLE      hdl,
    int                 *pXsvfLastCmd
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

pXsvfLastCmd

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE      hdl;
TDRV004_STATUS      result;
int                 XsvfLastCmd;

/*
** Check Last XSVF command
*/
result = tdrv004XsvfLastCommand( hdl,
                                 &XsvfLastCmd );
if (result == TDRV004_OK)
{
     printf(“Last XSVF command: %d\n”, XsvfLastCmd);
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.

3.2.4 tdrv004Reconfigure

NAME

tdrv004Reconfigure – Trigger FPGA reconfiguration process

SYNOPSIS

```
TDRV004_STATUS tdrv004Reconfigure
(
      TDRV004_HANDLE        hdl
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE   hdl;
TDRV004_STATUS   result;

/*
** Reconfigure FPGA
*/
result = tdrv004Reconfigure( hdl );
if (result != TDRV004_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_BUSY	The device is already busy with XSVF, Reconfig or SPI action.
TDRV004_ERR_NOT_READY	The DONE signal of the FPGA refused to change state, the reconfiguration might be invalid. An error occurred during reconfiguration. This may be caused by an invalid FPGA content located inside the XSVF file.

3.2.5 tdrv004SetWaitstates

NAME

tdrv004SetWaitstates – Specify number of waitstates for programming

SYNOPSIS

```
TDRV004_STATUS tdrv004SetWaitstates
(
    TDRV004_HANDLE      hdl,
    int                 WaitStates
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

WaitStates

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE  hdl;
TDRV004_STATUS  result;

/*
** Configure driver to use 3 waitstates
*/
WaitStates = 3;
result = tdrv004SetWaitstates( hdl,
                               WaitStates );
if (result != TDRV004_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.

3.2.6 tdrv004SetClock

NAME

tdrv004SetClock – Set clock generator parameters

SYNOPSIS

```
TDRV004_STATUS tdrv004SetClock
(
    TDRV004_HANDLE          hdl,
    TDRV004_CLOCK_PARAM     *pClockParam
);
```

DESCRIPTION

This function configures the onboard clock generator.

PARAMETERS

hdl

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

pClockParam

This parameter specifies a pointer to a TDRV004_CLOCK_PARAM structure:

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

DeviceAddr

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

x09_ClkOE

Specifies which clock outputs shall be enabled.

x0C_DIV1SRCN

Specifies internal input source 1 and the corresponding frequency divider

x10_InputCtrl

Specifies value for the Input Pin Control register

x40_CPumpPB

Specifies value for Charge Pump and PB counter register

x41_CPumpPB

Specifies value for Charge Pump and PB counter register

x42_POQcnt

Specifies value for PO and Q counter register

x44_SwMatrix

Specifies value for Switching Matrix Register

x45_SwMatrix

Specifies value for Switching Matrix Register

x46_SwMatrix

Specifies value for Switching Matrix Register

x47_DIV2SRCN

Specifies internal input source 2 and the corresponding frequency divider

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE        hdl;
TDRV004_STATUS        result;
TDRV004_CLOCK_PARAM   ClockParam;


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

result = tdrv004SetClock(  hdl,
                           &ClockParam );
if (result != TDRV004_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_BUSY	The device is already busy with XSVF, Reconfig or SPI action.
TDRV004_ERR_INVAL	It was tried to disable CLK1. This is not allowed.
TDRV004_ERR_NOT_READY	A device error occurred during programming.

3.2.7 tdrv004SpiWrite

NAME

tdrv004SpiWrite – Write values to SPI storage

SYNOPSIS

```
TDRV004_STATUS tdrv004SpiWrite
(
    TDRV004_HANDLE      hdl,
    TDRV004_SPI_BUF     *pSpiIoBuf
);
```

DESCRIPTION

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

Do not use this control function to setup the clock generator. Please use API function tdrv004SetClock() instead.

PARAMETERS

hdl

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

pSpiIoBuf

This parameter specifies a pointer to a TDRV004_SPI_BUF structure:

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

SpiAddr

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

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

SubAddr

Specifies the sub-address (starting offset).

len

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

pData

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE    hdl;
TDRV004_STATUS    result;
TDRV004_SPI_BUF   *pSpiBuf;
int               BufferSize;
unsigned char     *pu8Ptr;

/*
** 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;
pu8Ptr             = pSpiBuf->pData;
```

```
pu8Ptr[0]     = 0x01;
pu8Ptr[1]     = 0x02;
pu8Ptr[2]     = 0x03;
pu8Ptr[3]     = 0x04;
pu8Ptr[4]     = 0x05;

result = tdrv004SpiWrite(  hdl,
                           pSpiBuf );
if (result != TDRV004_OK)
{
     /* handle error */
}
free(pSpiBuf);
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_BUSY	The device is already busy with XSVF, Reconfig or SPI action.
TDRV004_ERR_NOT_READY	A device error occurred during programming.
TDRV004_ERR_INVAL	The specified SubAddr+len exceeds 256, or *len* is invalid.

3.2.8 tdrv004SpiRead

NAME

tdrv004SpiRead – Read values from SPI storage

SYNOPSIS

```
TDRV004_STATUS tdrv004SpiRead
(
    TDRV004_HANDLE      hdl,
    TDRV004_SPI_BUF     *pSpiIoBuf
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

pSpiIoBuf

This parameter specifies a pointer to a TDRV004_SPI_BUF structure:

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

SpiAddr

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

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

SubAddr

Specifies the sub-address (starting offset).

len

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

pData

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE    hdl;
TDRV004_STATUS    result;
TDRV004_SPI_BUF   *pSpiBuf;
int               BufferSize;

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

```
result = tdrv004SpiRead( hdl,
                         pSpiBuf );
if (result != TDRV004_OK)
{
     /* handle error */
}
free(pSpiBuf);
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_BUSY	The device is already busy with XSVF, Reconfig or SPI action.
TDRV004_ERR_NOT_READY	A device error occurred during programming.
TDRV004_ERR_INVAL	The specified SubAddr+len exceeds 256, or *len* is invalid.

3.2.9 tdrv004PlxWrite

NAME

tdrv004PlxWrite – Write 16bit value to PLX PCI9030 EEPROM

SYNOPSIS

```
TDRV004_STATUS tdrv004PlxWrite
(
    TDRV004_HANDLE      hdl,
    TDRV004_PLX_BUF     *pPlxBuf
);
```

DESCRIPTION

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

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

PARAMETERS

hdl

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

pPlxBuf

This parameter specifies a pointer to a TDRV004_PLX_BUF structure:

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

Offset

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

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

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

Value

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE    hdl;
TDRV004_STATUS    result;
TDRV004_PLX_BUF   PlxBuf;


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

result = tdrv004PlxWrite(  hdl,
                           &PlxBuf );
if (result != TDRV004_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_BUSY	The device is already busy with XSVF, Reconfig or SPI action.
TDRV004_ERR_INVAL	The specified offset is invalid, or read-only.

3.2.10 tdrv004PlxRead

NAME

tdrv004PlxRead – Read 16bit value from PLX PCI9030 EEPROM

SYNOPSIS

```
TDRV004_STATUS tdrv004PlxRead
(
    TDRV004_HANDLE      hdl,
    TDRV004_PLX_BUF     *pPlxBuf
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

pPlxBuf

This parameter specifies a pointer to a TDRV004_PLX_BUF structure:

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

Offset

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

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

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

Value

This value holds the retrieved 16bit word.

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE    hdl;
TDRV004_STATUS    result;
TDRV004_PLX_BUF   PlxBuf;


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

result = tdrv004PlxRead( hdl,
                         &PlxBuf );
if (result == TDRV004_OK)
{
     printf( “SubsystemID = 0x%04X\n”, PlxBuf.Value );
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_BUSY	The device is already busy with XSVF, Reconfig or SPI action.
TDRV004_ERR_INVAL	The specified offset is invalid.

3.2.11 tdrv004ReadU8

NAME

tdrv004ReadU8 – Read 8bit values from FPGA resource

SYNOPSIS

```
TDRV004_STATUS tdrv004ReadU8
(
    TDRV004_HANDLE      hdl,
    TDRV004_MEMIO_BUF   *pMemIoBuf
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

pMemIoBuf

This parameter specifies a pointer to a TDRV004_MEMIO_BUF structure:

```
typedef struct
{
    TDRV004_RESOURCE   Resource;
    unsigned int       Offset;
    unsigned int       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 (BAR) usage is programmable and can be changed by modifying the PLX PCI9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

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

The PLX PCI9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE     hdl;
TDRV004_STATUS     result;
TDRV004_MEMIO_BUF  pMemIoBuf;
unsigned char      *pValues;
int                BufferSize;

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

…

```
result = tdrv004ReadU8( hdl, pMemIoBuf );
if (result == TDRV004_OK)
{
     pValues = (unsigned char*)pMemIoBuf->pData;
} else {
     /* handle error */
}
free( pMemIoBuf );
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_ACCESS	The specified Resource is not available for access.
TDRV004_ERR_INVAL	The specified Offset+Size exceeds the available memory or I/O space.

3.2.12 tdrv004ReadU16

NAME

tdrv004ReadU16 – Read 16bit values from FPGA resource

SYNOPSIS

```
TDRV004_STATUS tdrv004ReadU16
(
    TDRV004_HANDLE      hdl,
    TDRV004_MEMIO_BUF   *pMemIoBuf
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

pMemIoBuf

This parameter specifies a pointer to a TDRV004_MEMIO_BUF structure:

```
typedef struct
{
    TDRV004_RESOURCE   Resource;
    unsigned int       Offset;
    unsigned int       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 (BAR) usage is programmable and can be changed by modifying the PLX PCI9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

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

The PLX PCI9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE     hdl;
TDRV004_STATUS     result;
TDRV004_MEMIO_BUF  pMemIoBuf;
unsigned short     *pValues;
int                BufferSize;

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

…

```
result = tdrv004ReadU16( hdl, pMemIoBuf );
if (result == TDRV004_OK)
{
     pValues = (unsigned short*)pMemIoBuf->pData;
} else {
     /* handle error */
}
free( pMemIoBuf );
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_ACCESS	The specified Resource is not available for access.
TDRV004_ERR_INVAL	The specified Offset+Size exceeds the available memory or I/O space.

3.2.13 tdrv004ReadU32

NAME

tdrv004ReadU32 – Read 32bit values from FPGA resource

SYNOPSIS

```
TDRV004_STATUS tdrv004ReadU32
(
    TDRV004_HANDLE      hdl,
    TDRV004_MEMIO_BUF   *pMemIoBuf
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

pMemIoBuf

This parameter specifies a pointer to a TDRV004_MEMIO_BUF structure:

```
typedef struct
{
    TDRV004_RESOURCE   Resource;
    unsigned int       Offset;
    unsigned int       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 (BAR) usage is programmable and can be changed by modifying the PLX PCI9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

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

The PLX PCI9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE     hdl;
TDRV004_STATUS     result;
TDRV004_MEMIO_BUF  pMemIoBuf;
unsigned int       *pValues;
int                BufferSize;

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

…

```
result = tdrv004ReadU32( hdl, pMemIoBuf );
if (result == TDRV004_OK)
{
     pValues = (unsigned int*)pMemIoBuf->pData;
} else {
     /* handle error */
}
free( pMemIoBuf );
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_ACCESS	The specified Resource is not available for access.
TDRV004_ERR_INVAL	The specified Offset+Size exceeds the available memory or I/O space.

3.2.14 tdrv004WriteU8

NAME

tdrv004WriteU8 – Write 8bit values to FPGA resource

SYNOPSIS

```
TDRV004_STATUS tdrv004WriteU8
(
    TDRV004_HANDLE      hdl,
    TDRV004_MEMIO_BUF   *pMemIoBuf
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

pMemIoBuf

This parameter specifies a pointer to a TDRV004_MEMIO_BUF structure:

```
typedef struct
{
    TDRV004_RESOURCE   Resource;
    unsigned int       Offset;
    unsigned int       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 (BAR) usage is programmable and can be changed by modifying the PLX PCI9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

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

The PLX PCI9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to write.

pData

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE      hdl;
TDRV004_STATUS      result;
TDRV004_MEMIO_BUF   pMemIoBuf;
unsigned char       *pValues;
int                 BufferSize;

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

…

```
result = tdrv004WriteU8( hdl, pMemIoBuf );
if (result != TDRV004_OK)
{
     /* handle error */
}
free( pMemIoBuf );
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_ACCESS	The specified Resource is not available for access.
TDRV004_ERR_INVAL	The specified Offset+Size exceeds the available memory or I/O space.

3.2.15 tdrv004WriteU16

NAME

tdrv004WriteU16 – Write 16bit values to FPGA resource

SYNOPSIS

```
TDRV004_STATUS tdrv004WriteU16
(
    TDRV004_HANDLE      hdl,
    TDRV004_MEMIO_BUF   *pMemIoBuf
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

pMemIoBuf

This parameter specifies a pointer to a TDRV004_MEMIO_BUF structure:

```
typedef struct
{
    TDRV004_RESOURCE   Resource;
    unsigned int       Offset;
    unsigned int       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 (BAR) usage is programmable and can be changed by modifying the PLX PCI9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

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

The PLX PCI9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to write.

pData

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE     hdl;
TDRV004_STATUS     result;
TDRV004_MEMIO_BUF  pMemIoBuf;
unsigned short     *pValues;
int                BufferSize;

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

…

```
result = tdrv004WriteU16( hdl, pMemIoBuf );
if (result != TDRV004_OK)
{
     /* handle error */
}
free( pMemIoBuf );
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_ACCESS	The specified Resource is not available for access.
TDRV004_ERR_INVAL	The specified Offset+Size exceeds the available memory or I/O space.

3.2.16 tdrv004WriteU32

NAME

tdrv004WriteU32 – Write 32bit values to FPGA resource

SYNOPSIS

```
TDRV004_STATUS tdrv004WriteU32
(
    TDRV004_HANDLE      hdl,
    TDRV004_MEMIO_BUF   *pMemIoBuf
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

pMemIoBuf

This parameter specifies a pointer to a TDRV004_MEMIO_BUF structure:

```
typedef struct
{
    TDRV004_RESOURCE   Resource;
    unsigned int       Offset;
    unsigned int       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 (BAR) usage is programmable and can be changed by modifying the PLX PCI9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

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

The PLX PCI9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to write.

pData

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE     hdl;
TDRV004_STATUS     result;
TDRV004_MEMIO_BUF  pMemIoBuf;
unsigned int       *pValues;
int                BufferSize;

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

…

```
result = tdrv004WriteU32( hdl, pMemIoBuf );
if (result != TDRV004_OK)
{
     /* handle error */
}
free( pMemIoBuf );
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_ACCESS	The specified Resource is not available for access.
TDRV004_ERR_INVAL	The specified Offset+Size exceeds the available memory or I/O space.

3.2.17 tdrv004ConfigureInterrupts

NAME

tdrv004ConfigureInterrupts – Configure local interrupt source polarity

SYNOPSIS

```
TDRV004_STATUS tdrv004ConfigureInterrupts
(
    TDRV004_HANDLE      hdl,
    unsigned int        InterruptConfig
);
```

DESCRIPTION

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

PARAMETERS

hdl

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

InterruptConfig

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

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE  hdl;
TDRV004_STATUS  result;
unsigned int    IntConfig;

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

result = tdrv004ConfigureInterrupts( hdl,
                                     IntConfig );
if (result != TDRV004_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_INVAL	Invalid interrupt configuration specified.

3.2.18 tdrv004WaitForINT1

NAME

tdrv004WaitForINT1 – Wait for incoming Local Interrupt Source 1

SYNOPSIS

```
TDRV004_STATUS tdrv004WaitForINT1
(
    TDRV004_HANDLE      hdl,
    int                 TimeoutMS
);
```

DESCRIPTION

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

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

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

PARAMETERS

hdl

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

TimeoutMS

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE   hdl;
TDRV004_STATUS   result;
int              TimeoutMS;

/*
** Wait at least 5 seconds for incoming interrupt
*/
TimeoutMS = 5000;

result = tdrv004WaitForINT1(  hdl,
                              TimeoutMS );
if (result == TDRV004_OK)
{
     /* function succeeded */
     /* acknowledge interrupt source in FPGA logic       */
     /* to clear the PLX PCI9030 Local Interrupt Source  */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_BUSY	There is already a job waiting for this interrupt.
TDRV004_ERR_TIMEOUT	The specified timeout has passed
TDRV004_ERR_ABORTED	The waiting has been aborted by a power down of the system

3.2.19 tdrv004WaitForINT2

NAME

tdrv004WaitForINT2 – Wait for incoming Local Interrupt Source 2

SYNOPSIS

```
TDRV004_STATUS tdrv004WaitForINT2
(
    TDRV004_HANDLE      hdl,
    int                 TimeoutMS
);
```

DESCRIPTION

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

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

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

PARAMETERS

hdl

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

TimeoutMS

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

EXAMPLE

```
#include “tdrv004api.h”

TDRV004_HANDLE   hdl;
TDRV004_STATUS   result;
int              TimeoutMS;

/*
** Wait at least 5 seconds for incoming interrupt
*/
TimeoutMS = 5000;

result = tdrv004WaitForINT2(  hdl,
                              TimeoutMS );
if (result == TDRV004_OK)
{
     /* function succeeded */
     /* acknowledge interrupt source in FPGA logic       */
     /* to clear the PLX PCI9030 Local Interrupt Source  */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV004_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV004_ERR_INVALID_HANDLE	The specified TDRV004_HANDLE is invalid.
TDRV004_ERR_BUSY	There is already a job waiting for this interrupt.
TDRV004_ERR_TIMEOUT	The specified timeout has passed
TDRV004_ERR_ABORTED	The waiting has been aborted by a power down of the system

4 Hibernate Mode

The driver supports the “Hibernate”-mode of Windows. The driver will remember the device configuration and it will restore the configuration immediately after leaving the hibernation mode.

If there are pending requests, (e.g. waiting for an Interrupt) and the system wants to enter hibernation mode, the request will be aborted with an appropriate error code. The application may now decide how to handle this situation.