The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV004-SW-65

Windows 2000/XP Device Driver

Reconfigurable FPGA

Version 1.1.x

User Manual

Issue 1.1.2

June 2008

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

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

TDRV004-SW-65

Windows 2000/XP 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

Table of Content

1 Introduction

The TDRV004-SW-65 Windows WDM (Windows Driver Model) device driver is a kernel mode driver which allows the operation of the TDRV004 product family on an Intel or Intel-compatible x86 Windows 2000 or Windows XP operating system.

The standard file and device (I/O) functions (CreateFile, CloseHandle and DeviceIoControl) provide the basic interface for opening and closing a resource handle and for performing device I/O control operations.

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 registers

The TDRV004-SW-65 device driver supports the modules listed below:

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

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

TPMC630 / TCP630 User manual

TPMC630 / TCP630 Engineering Manual

2 Installation

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

tdrv004.sys	Windows WDM driver binary
tdrv004.inf	Windows WDM installation script
tdrv004.h	Header file with IOCTL codes and structure definitions
TDRV004-SW-65-1.1.2.pdf	This document
\example\tdrv004exa.c	Example application
\example\fpgaexa.zip	Example FPGA design (XSVF file) as a ZIP archive
Release.txt	Information about the Device Driver Release
ChangeLog.txt	Release history

2.1 Software Installation

2.1.1 Windows 2000 / XP

This section describes how to install the TDRV004-SW-65 Device Driver on a Windows 2000 / XP operating system.

After installing the hardware and boot-up your system, Windows 2000 / XP 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. In Drive A, insert the TDRV001 driver disk; 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 diskette.
   Click "***Next***" button to continue.
5. Complete the upgrade device driver and click "***Finish***" to take all the changes effect.
6. Repeat the steps above for each found module of the TDRV004 product family.
7. Copy needed files (tdrv004.h, TDRV004-SW-65.pdf) to desired target directory.

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

2.1.2 Confirming Windows 2000 / XP Installation

To confirm that the driver has been properly loaded in Windows 2000 / XP, perform the following steps:

1. From Windows 2000 / XP, open the "***Control Panel***" from "***My Computer***".
2. Click the "***System***" icon and choose the "***Hardware***" tab, and then click the "***Device Manager***" button.
3. Click the "**+**" in front of "***Other Devices***".
   The driver "***TEWS TECHNOLOGIES TDRV004 Reconfigurable FPGA***" should appear for each installed device.

3 Device Driver Programming

The TDRV004-SW-65 Windows WDM device driver is a kernel mode device driver using Direct I/O.

The standard file and device (I/O) functions (CreateFile, CloseHandle and DeviceIoControl) provide the basic interface for opening and closing a resource handle and for performing device I/O control operations.

All of these standard Win32 functions are described in detail in the Windows Platform SDK Documentation (Windows base services / Hardware / Device Input and Output).

For details refer to the Win32 Programmers Reference of your used programming tools (C++, Visual Basic etc.)

3.1 Files and I/O Functions

The following section doesn't contain a full description of the Win32 functions for interaction with the TDRV004 device driver. Only the required parameters are described in detail.

3.1.1 Opening a Device

Before you can perform any I/O the TDRV004 device, it must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the TDRV004 device.

```
HANDLE CreateFile(
      LPCTSTR   lpFileName,
      DWORD     dwDesiredAccess,
      DWORD     dwShareMode,
      LPSECURITY_ATTRIBUTES  lpSecurityAttributes,
      DWORD     dwCreationDistribution,
      DWORD     dwFlagsAndAttributes,
      HANDLE    hTemplateFile
);
```

PARAMETERS

lpFileName

Points to a null-terminated string, which specifies the name of the TDRV004 to open. The *lpFileName* string should be of the form **\\.\TDRV004_*x*** to open the device *x.* The ending *x* is a one-based number. The first device found by the driver is \\.\TDRV004_1, the second device \\.\TDRV004_2 and so on.

dwDesiredAccess

Specifies the type of access to the TDRV004.
For the TDRV004, this parameter must be set to read-write access (GENERIC_READ | GENERIC_WRITE)

dwShareMode

Set of bit flags that specify how the object can be shared. Set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for TDRV004 devices.

dwCreationDistribution

Specifies which action to take on files that exist, and which action to take when files do not exist. TDRV004 devices must be always opened OPEN_EXISTING.

dwFlagsAndAttributes

Specifies the file attributes and flags for the file. If overlapped I/O shall be used, this value may be set to FILE_FLAG_OVERLAPPED.

hTemplateFile

This value must be NULL for TDRV004 devices.

RETURN VALUE

If the function succeeds, the return value is an open handle to the specified TDRV004 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call ***GetLastError***.

EXAMPLE

```
HANDLE    hDevice;

hDevice = CreateFile(
     “\\\\.\\TDRV004_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,                    // no security attrs
     OPEN_EXISTING,           // TDRV004 device always open existing
     FILE_FLAG_OVERLAPPED,    // overlapped I/O
     NULL
);

if (hDevice == INVALID_HANDLE_VALUE) {
     ErrorHandler( "Could not open device" ); // process error
}
```

SEE ALSO

CloseHandle(), Win32 documentation CreateFile()

3.1.2 Closing a Device

The **CloseHandle** function closes an open TDRV004 handle.

BOOL CloseHandle(
HANDLE *hDevice;*
);

PARAMETERS

hDevice

Identifies an open TDRV004 handle.

RETURN VALUE

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call ***GetLastError*.**

EXAMPLE

```
HANDLE  hDevice;

if( !CloseHandle( hDevice ) ) {
     ErrorHandler( "Could not close device" ); // process error
}
```

SEE ALSO

CreateFile (), Win32 documentation CloseHandle ()

3.1.3 Device I/O Control Functions

The **DeviceIoControl** function sends a control code directly to a specified device driver, causing the corresponding device to perform the specified operation.

```
BOOL DeviceIoControl(
    HANDLE      hDevice,
    DWORD       dwIoControlCode,
    LPVOID      lpInBuffer,
    DWORD       nInBufferSize,
    LPVOID      lpOutBuffer,
    DWORD       nOutBufferSize,
    LPDWORD     lpBytesReturned,
    LPOVERLAPPED  lpOverlapped
);
```

PARAMETERS

hDevice

Handle to the TDRV004 device that is to perform the operation.

dwIoControlCode

Specifies the control code for the operation. This value identifies the specific operation to be performed. The following values are defined in *tdrv004.h*:

Value	Meaning
IOCTL_TD004_XSVFPLAY	Play an XSVF file for programming
IOCTL_TD004_XSVFPOS	Retrieve current play-position in XSVF file
IOCTL_TD004_XSVFLASTCMD	Get the last executed XSVF command
IOCTL_TD004_RECONFIG	Trigger FPGA reconfiguration process
IOCTL_TD004_SETWAITSTATES	Specify number of waitstates for programming
IOCTL_TD004_SETCLOCK	Set clock generator parameters
IOCTL_TD004_SPIWRITE	Write values to clock generator
IOCTL_TD004_SPIREAD	Read values from clock generator
IOCTL_TD004_PLXWRITEWORD	Write 16bit value to PLX9030 EEPROM
IOCTL_TD004_PLXREADWORD	Read 16bit value from PLX9030 EEPROM
IOCTL_TD004_READ_UCHAR	Read unsigned char values from FPGA
IOCTL_TD004_READ_USHORT	Read unsigned short values from FPGA
IOCTL_TD004_READ_ULONG	Read unsigned long values from FPGA
IOCTL_TD004_WRITE_UCHAR	Write unsigned char values to FPGA
IOCTL_TD004_WRITE_USHORT	Write unsigned short values to FPGA
IOCTL_TD004_WRITE_ULONG	Write unsigned long values to FPGA

IOCTL_TD004_CONFIGURE_INT	Configure local interrupt source polarity
IOCTL_TD004_WAIT_FOR_INT1	Wait for incoming Local Interrupt Source 1
IOCTL_TD004_WAIT_FOR_INT2	Wait for incoming Local Interrupt Source 2

See behind for more detailed information on each control code.

lpInBuffer

Pointer to a buffer that contains the data required to perform the operation.

nInBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpInBuffer*.

lpOutBuffer

Pointer to a buffer that receives the operation's output data.

nOutBufferSize

Specifies the size, in bytes, of the buffer pointed to by *lpOutBuffer*.

lpBytesReturned

Pointer to a variable that receives the size, in bytes, of the data stored into the buffer pointed to by *lpOutBuffer*. A valid pointer is required.

lpOverlapped

Pointer to an *Overlapped* structure.

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

RETURN VALUE

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call ***GetLastError***.

Note that the TDRV004 device driver always returns standard Win32 error codes on failure. Please refer to the Windows Platform SDK Documentation for a detailed description of the returned error codes.

SEE ALSO

Win32 documentation DeviceIoControl ()

3.1.3.1 IOCTL_TD004_XSVFPLAY

This TDRV004 control function programs the FPGA with a supplied XSVF file. A pointer to the caller's data buffer, where the content of the XSVF file is stored, is passed by the parameter *lpInBuffer* to the driver. This control function may be called in Overlapped operation mode. During programming, the progress can be monitored using IOCTL_TD004_XSVFPOS (see below). In non-overlapped mode, this function will block until XSVF programming is finished. For information on building an XSVF file, please refer to the Engineering Documentation of the TDRV004 product family.

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

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 IOCTL_TD004_SETWAITSTATES (refer to the corresponding section in this manual). Additionally, the CLK1 should not be lower than 10MHz for programming.

EXAMPLE

```
#include "tdrv004.h"

HANDLE            hDevice;
BOOLEAN           success;
ULONG             NumBytes;
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.
*/
success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_XSVFPLAY,       // control code
     pXsvfContent,
     XsvfFileSize,
     NULL,
     0,
     &NumBytes,                  // number of bytes transferred
     NULL
);
```

```
if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_BUSY	The device is already busy with XSVF.
ERROR_INVALID_PARAMETER	An error occurred during XSVF operation.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.2 IOCTL_TD004_XSVFPOS

This TDRV004 control function returns the current byte in the XSVF file during programming with IOCTL_TD004_XSVFPLAY. A pointer to an *int* value is passed by the parameter *lpOutBuffer* to the driver. This control function can be used to monitor the programming progress.

EXAMPLE

```
#include “tdrv004.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
int                 XsvfPos;

/*
** Get XSVF position to monitor progress
*/
success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_XSVFPOS,        // control code
     NULL,
     0,
     &XsvfPos,
     sizeof(int),
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER The size of the supplied output buffer is too small.
All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.3 IOCTL_TD004_XSVFLASTCMD

This TDRV004 control function returns the number of the last executed XSVF command. This value can be used to find errors inside the supplied XSVF file. This value refers to the line inside the ASCII SVF file. A pointer to an *int* value is passed by the parameter *lpOutBuffer* to the driver.

EXAMPLE

```
#include “tdrv004.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
int                 XsvfLastCmd;

/*
** Get XSVF position to monitor progress
*/
success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_XSVFLASTCMD,    // control code
     NULL,
     0,
     &XsvfLastCmd,
     sizeof(int),
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied output buffer is too small.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.4 IOCTL_TD004_RECONFIG

This TDRV004 control function starts the reconfiguration process of the FPGA. This control function must be called after the FPGA is programmed using IOCTL_TD004_XSVFPLAY.No parameter is used for this function.

EXAMPLE

```
#include “tdrv004.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;

/*
** Issue Reconfiguration command
*/
success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_RECONFIG,       // control code
     NULL,
     0,
     NULL,
     0,
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_BUSY	The device is already busy with XSVF or Reconfig.
ERROR_NOT_READY	The DONE signal of the FPGA refused to change state, the reconfiguration might be invalid.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.5 IOCTL_TD004_SETWAITSTATES

This TDRV004 control function configures the driver to use a number of waitstates during XSVF and SPI programming. This might be necessary, if the local clock (CLK1) of the onboard clock generator is configured to rather slow. The local programming interface is clocked with this frequency, which might result in errors during programming for low CLK1 frequencies and a small amount of waitstates.

A pointer to waitstates (*int* value) is passed to the driver by the parameter *lpInBuffer*.

EXAMPLE

```
#include “tdrv004.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
int                 WaitStates;

/*
** Setup 5 waitstates for CLK1 < 20MHz
*/
WaitStates = 5;
success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_SETWAITSTATES,  // control code
     &WaitStates,
     sizeof(int),
     NULL,
     0,
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied input buffer is too small.
ERROR_INVALID_PARAMETER	The specified waitstates are invalid (<0).

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.6 IOCTL_TD004_SETCLOCK

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

The *TD004_CLOCK_PARAM* structure has the following layout:

```
typedef struct {
   unsigned char   DeviceAddr;
   unsigned char   x09_ClkOE;
   unsigned char   x0C_DIV1SRCN;
   unsigned char   x10_InputCtrl;
   unsigned char   x40_CPumpPB;
   unsigned char   x41_CPumpPB;
   unsigned char   x42_POQcnt;
   unsigned char   x44_SwMatrix;
   unsigned char   x45_SwMatrix;
   unsigned char   x46_SwMatrix;
   unsigned char   x47_DIV2SRCN;
} TD004_CLOCK_PARAM;
```

DeviceAddr

Specifies the desired destination address. The CY27EE16 clock generator provides several EEPROM banks as well as SRAM. If TD004_CLKADR_SRAM is specified, the values are directly stored inside the volatile RAM area and take effect immediately. If TD004_CLKADR_EEPROM is specified, the values are stored in the non-volatile area of the clock generator, and the CY27EE16 loads it after the next power-up.

x09_ClkOE

Specifies which clock outputs shall be enabled.

x0C_DIV1SRCN

Specifies internal input source 1 and the corresponding frequency divider

x10_InputCtrl

Specifies value for the Input Pin Control register

x40_CPumpPB

Specifies value for Charge Pump and PB counter register

x41_CPumpPB

Specifies value for Charge Pump and PB counter register

x41_POQcnt

Specifies value for PO and Q counter register

x44_SwMatrix

Specifies value for Switching Matrix Register

x45_SwMatrix

Specifies value for Switching Matrix Register

x46_SwMatrix

Specifies value for Switching Matrix Register

x47_DIV2SRCN

Specifies internal input source 2 and the corresponding frequency divider

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

EXAMPLE

```
#include “tdrv004.h”

HANDLE             hDevice;
BOOLEAN            success;
ULONG              NumBytes;
TD004_CLOCK_PARAM  ClockParam;

/*
** Setup clock generator (SRAM):
**   CLK1: 50.0MHz       CLK2: 20.0MHz
**   CLK3: 10.0MHz       CLK4:  1.0MHz
**   CLK5:  0.2MHz       CLK6: -off-
*/
ClockParam.DeviceAddress    = TD004_CLKADR_SRAM;
ClockParam.x09_ClkOE        = 0x6f;
ClockParam.x0C_DIV1SRCN     = 0x64;
ClockParam.x10_InputCtrl    = 0x50;
ClockParam.x40_CPumpPB      = 0xc0;
ClockParam.x41_CPumpPB      = 0x03;
ClockParam.x42_POQcnt       = 0x81;
ClockParam.x44_SwMatrix     = 0x42;
ClockParam.x45_SwMatrix     = 0x9f;
ClockParam.x46_SwMatrix     = 0x3f;
ClockParam.x47_DIV2SRCN     = 0xe4;
success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_SETCLOCK,       // control code
     &ClockParam,                // input buffer
     sizeof(TD004_CLOCK_PARAM),
     NULL,
     0,
```

```
     &NumBytes,                  // number of bytes transferred
     NULL
);

if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied input buffer is too small.
ERROR_BUSY	The device is already busy with an SPI operation.
ERROR_NOT_READY	A device error occurred during programming.
ERROR_INVALID_PARAMETER	Tried to disable CLK1. This is not allowed.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.7 IOCTL_TD004_SPIWRITE

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

The *TD004_SPI_BUF* structure has the following layout:

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

SpiAddr

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

SubAddr

Specifies the sub-address (starting offset).

len

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

pData

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

Do not use this control function to setup the clockgenerator. Please use control function IOCTL_TD004_SETCLOCK instead.

EXAMPLE

```
#include “tdrv004.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
unsigned long       BufferSize;
TD004_SPI_BUF       *pSpiBuf;

/*
** write 5 bytes to EEPROM block 1, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize = ( sizeof(TD004_SPI_BUF) + 5*sizeof(unsigned char) );
pSpiBuf = (TD004_SPI_BUF*)malloc( BufferSize );
pSpiBuf->SpiAddr = TD004_CLKADDR_EEBLOCK1;
pSpiBuf->SubAddr = 0x00;
pSpiBuf->len      = 5;
pSpiBuf->pData[0] = 0x01;
pSpiBuf->pData[0] = 0x02;
pSpiBuf->pData[0] = 0x03;
pSpiBuf->pData[0] = 0x04;
pSpiBuf->pData[0] = 0x05;

success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_SPIWRITE,       // control code
     pSpiBuf,                    // input buffer
     BufferSize,
     NULL,
     0,
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied input buffer is too small.
ERROR_BUSY	The device is already busy with an SPI operation.
ERROR_NOT_READY	A device error occurred during programming.
ERROR_INVALID_PARAMETER	The specified SubAddr+len exceeds 256, or *len* is invalid.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.8 IOCTL_TD004_SPIREAD

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

The *TD004_SPI_BUF* structure has the following layout:

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

SpiAddr

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

SubAddr

Specifies the sub-address (starting offset).

len

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

pData

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

EXAMPLE

```
#include "tdrv004.h"

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
unsigned long       BufferSize;
TD004_SPI_BUF       *pSpiBuf;

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

```
pSpiBuf->len      = 5;

success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_SPIREAD,        // control code
     pSpiBuf,                    // input buffer
     BufferSize,
     NULL,
     0,
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied input buffer is too small.
ERROR_BUSY	The device is already busy with an SPI operation.
ERROR_NOT_READY	A device error occurred during programming.
ERROR_INVALID_PARAMETER	The specified SubAddr+len exceeds 256, or *len* is invalid.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.9 IOCTL_TD004_PLXWRITE

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

The *TD004_PLX_BUF* structure has the following layout:

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

Offset

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

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

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

Value

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

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

EXAMPLE

```
#include “tdrv004.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
TD004_PLX_BUF       PlxBuf;

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

success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_PLXWRITE,       // control code
     &PlxBuf,                    // input buffer
     sizeof(TD004_PLX_BUF),
     NULL,
     0,
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied input buffer is too small.
ERROR_BUSY	The device is busy with XSVF or Reconfig.
ERROR_INVALID_PARAMETER	The specified Offset is not valid, or read-only.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.10 IOCTL_TD004_PLXREAD

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

The *TD004_PLX_BUF* structure has the following layout:

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

Offset

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

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

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

Value

This value holds the retrieved 16bit word.

EXAMPLE

```
#include “tdrv004.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
TD004_PLX_BUF       PlxBuf;

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

success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_PLXREAD,        // control code
     &PlxBuf,                    // input buffer
     sizeof(TD004_PLX_BUF),
     NULL,
     0,
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( success ) {
     printf( “SubsystemVendorID = 0x%04X\n”, PlxBuf.Value );
} else {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied input buffer is too small.
ERROR_BUSY	The device is busy with XSVF or Reconfig.
ERROR_INVALID_PARAMETER	The specified Offset is not valid.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.11 IOCTL_TD004_READ_UCHAR

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

The *TD004_MEMIO_BUF* structure has the following layout:

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

Resource

Specifies the desired PCI resource to read from. The TD004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TD004_RES_MEM_2, the second PCI-I/O space found is named TD004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TD004_RESOURCE
0	IO	*(reserved)*	TD004_RES_IO_1
1	MEM	*(reserved)*	TD004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TD004_RES_MEM_2
3	IO	*(not implemented by default)*	TD004_RES_IO_2
4	IO	*(not implemented by default)*	TD004_RES_IO_3
5	MEM	*(not implemented by default)*	TD004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the PCI-Memory or PCI-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”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
unsigned long       BufferSize;
TD004_MEMIO_BUF     *pMemIoBuf;
unsigned char       *pValues;

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

success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_READ_UCHAR,     // control code
     NULL,
     0,
     pMemIoBuf,                  // buffer which receives the data
     BufferSize,
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( success ) {
     // Process data
     pValues = (unsigned char*)pMemIoBuf->pData;
} else {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied input buffer is too small.
ERROR_ACCESS_DENIED	The specified Resource is not available for access.
ERROR_INVALID_PARAMETER	The specified Offset+Size exceeds the available memory or I/O space.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.12 IOCTL_TD004_READ_USHORT

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

The *TD004_MEMIO_BUF* structure has the following layout:

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

Resource

Specifies the desired PCI resource to read from. The TD004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TD004_RES_MEM_2, the second PCI-I/O space found is named TD004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TD004_RESOURCE
0	IO	*(reserved)*	TD004_RES_IO_1
1	MEM	*(reserved)*	TD004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TD004_RES_MEM_2
3	IO	*(not implemented by default)*	TD004_RES_IO_2
4	IO	*(not implemented by default)*	TD004_RES_IO_3
5	MEM	*(not implemented by default)*	TD004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

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

EXAMPLE

```
#include “tdrv004.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
unsigned long       BufferSize;
TD004_MEMIO_BUF     *pMemIoBuf;
unsigned short      *pValues;

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

success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_READ_USHORT,    // control code
     NULL,
     0,
     pMemIoBuf,                  // buffer which receives the data
     BufferSize,
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( success ) {
     // Process data
     pValues = (unsigned short*)pMemIoBuf->pData;
} else {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied input buffer is too small.
ERROR_ACCESS_DENIED	The specified Resource is not available for access.
ERROR_INVALID_PARAMETER	The specified Offset+Size exceeds the available memory or I/O space.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.13 IOCTL_TD004_READ_ULONG

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

The *TD004_MEMIO_BUF* structure has the following layout:

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

Resource

Specifies the desired PCI resource to read from. The TD004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TD004_RES_MEM_2, the second PCI-I/O space found is named TD004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TD004_RESOURCE
0	IO	*(reserved)*	TD004_RES_IO_1
1	MEM	*(reserved)*	TD004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TD004_RES_MEM_2
3	IO	*(not implemented by default)*	TD004_RES_IO_2
4	IO	*(not implemented by default)*	TD004_RES_IO_3
5	MEM	*(not implemented by default)*	TD004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

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

EXAMPLE

```
#include “tdrv004.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
unsigned long       BufferSize;
TD004_MEMIO_BUF     *pMemIoBuf;
unsigned long       *pValues;

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

success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_READ_ULONG,     // control code
     NULL,
     0,
     pMemIoBuf,                  // buffer which receives the data
     BufferSize,
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( success ) {
     // Process data
     pValues = (unsigned long*)pMemIoBuf->pData;
} else {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied input buffer is too small.
ERROR_ACCESS_DENIED	The specified Resource is not available for access.
ERROR_INVALID_PARAMETER	The specified Offset+Size exceeds the available memory or I/O space.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.14 IOCTL_TD004_WRITE_UCHAR

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

The *TD004_MEMIO_BUF* structure has the following layout:

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

Resource

Specifies the desired PCI resource to read from. The TD004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TD004_RES_MEM_2, the second PCI-I/O space found is named TD004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TD004_RESOURCE
0	IO	*(reserved)*	TD004_RES_IO_1
1	MEM	*(reserved)*	TD004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TD004_RES_MEM_2
3	IO	*(not implemented by default)*	TD004_RES_IO_2
4	IO	*(not implemented by default)*	TD004_RES_IO_3
5	MEM	*(not implemented by default)*	TD004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to write.

pData

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

EXAMPLE

```
#include “tdrv004.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
unsigned long       BufferSize;
TD004_MEMIO_BUF     *pMemIoBuf;
unsigned char       *pValues;

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

success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_WRITE_UCHAR,    // control code
     pMemIoBuf,                  // pointer to data buffer
     BufferSize,
     NULL,
     0,
     &NumBytes,
     NULL
);
if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied input buffer is too small.
ERROR_ACCESS_DENIED	The specified Resource is not available for access.
ERROR_INVALID_PARAMETER	The specified Offset+Size exceeds the available memory or I/O space.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.15 IOCTL_TD004_WRITE_USHORT

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

The *TD004_MEMIO_BUF* structure has the following layout:

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

Resource

Specifies the desired PCI resource to read from. The TD004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TD004_RES_MEM_2, the second PCI-I/O space found is named TD004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TD004_RESOURCE
0	IO	*(reserved)*	TD004_RES_IO_1
1	MEM	*(reserved)*	TD004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TD004_RES_MEM_2
3	IO	*(not implemented by default)*	TD004_RES_IO_2
4	IO	*(not implemented by default)*	TD004_RES_IO_3
5	MEM	*(not implemented by default)*	TD004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

The 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”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
unsigned long       BufferSize;
TD004_MEMIO_BUF     *pMemIoBuf;
unsigned short      *pValues;

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

success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_WRITE_USHORT,   // control code
     pMemIoBuf,                  // pointer to data buffer
     BufferSize,
     NULL,
     0,
     &NumBytes,
     NULL
);
if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied input buffer is too small.
ERROR_ACCESS_DENIED	The specified Resource is not available for access.
ERROR_INVALID_PARAMETER	The specified Offset+Size exceeds the available memory or I/O space.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.16 IOCTL_TD004_WRITE_ULONG

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

The *TD004_MEMIO_BUF* structure has the following layout:

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

Resource

Specifies the desired PCI resource to read from. The TD004_RESOURCE enumeration contains values for all possible memory and I/O areas. Both first PCI-Memory and PCI-I/O areas of the TDRV004 module are restricted and cannot be used by the application. The second found PCI-Memory area is named TD004_RES_MEM_2, the second PCI-I/O space found is named TD004_RES_IO_2 and so on.
The Base Address Register usage is programmable and can be changed by modifying the PLX9030 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type		TD004_RESOURCE
0	IO	*(reserved)*	TD004_RES_IO_1
1	MEM	*(reserved)*	TD004_RES_MEM_1
2	MEM	*(used by VHDL Example)*	TD004_RES_MEM_2
3	IO	*(not implemented by default)*	TD004_RES_IO_2
4	IO	*(not implemented by default)*	TD004_RES_IO_3
5	MEM	*(not implemented by default)*	TD004_RES_MEM_3

The PLX9030 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to read.

pData

The 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”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
unsigned long       BufferSize;
TD004_MEMIO_BUF     *pMemIoBuf;
unsigned long       *pValues;

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

success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_WRITE_ULONG,    // control code
     pMemIoBuf,                  // pointer to data buffer
     BufferSize,
     NULL,
     0,
     &NumBytes,
     NULL
);
if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_USER_BUFFER	The size of the supplied input buffer is too small.
ERROR_ACCESS_DENIED	The specified Resource is not available for access.
ERROR_INVALID_PARAMETER	The specified Offset+Size exceeds the available memory or I/O space.

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.17 IOCTL_TD004_CONFIGURE_INT

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

A pointer to an *unsigned long* value containing the new interrupt configuration is passed to the driver by the parameter *lpInBuffer*. This value is an OR'ed value using the following definitions (only one value valid for each interrupt source):

value	description
TD004_LINT1_POLHIGH	Local Interrupt Source 1 HIGH active
TD004_LINT1_POLLOW	Local Interrupt Source 1 LOW active
TD004_LINT2_POLHIGH	Local Interrupt Source 2 HIGH active
TD004_LINT2_POLLOW	Local Interrupt Source 2 LOW active

EXAMPLE

```
#include "tdrv004.h"

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
unsigned long       IntConfig;

/*
** Setup LINT1 to LOW polarity, and LINT2 to HIGH polarity
*/
IntConfig = TD004_LINT1_POLLOW | TD004_LINT2_POLHIGH;
success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_CONFIGURE_INT,  // control code
     &IntConfig,
     sizeof(unsigned long),
     NULL,
     0,
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( !success ) {
     // Process DeviceIoControl() error
}
```

ERROR CODES

ERROR_INVALID_PARAMETER The specified parameter is invalid.
All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.18 IOCTL_TD004_WAIT_FOR_INT1

This TDRV004 control function enables the corresponding interrupt source, and waits for Local Interrupt Source 1 (LINT1) to arrive. After the interrupt has arrived, this specific local interrupt source is disabled.

A pointer to an *int* value containing the timeout in seconds is passed to the driver by the parameter *lpInBuffer*. To wait indefinitely, specify -1 as timeout parameter.

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

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

EXAMPLE

```
#include “tdrv004.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
int                 Timeout;

/*
** Wait at least 5 seconds for incoming interrupt
*/
Timeout = 5;
success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_WAIT_FOR_INT1,  // control code
     &Timeout,
     sizeof(int),
     NULL,
     0,
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( success ) {
     // acknowledge interrupt source in FPGA logic
     // to clear the PLX PCI9030 Local Interrupt Source
} else {
     // Process DeviceIoControl() error
}
```

ERROR CODES

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

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()

3.1.3.19 IOCTL_TD004_WAIT_FOR_INT2

This TDRV004 control function enables the corresponding interrupt source, and waits for Local Interrupt Source 2 (LINT2) to arrive. After the interrupt has arrived, this specific local interrupt source is disabled.

A pointer to an *int* value containing the timeout in seconds is passed to the driver by the parameter *lpInBuffer*. To wait indefinitely, specify -1 as timeout parameter.

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

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

EXAMPLE

```
#include “tdrv004.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
int                 Timeout;

/*
** Wait at least 5 seconds for incoming interrupt
*/
Timeout = 5;
success = DeviceIoControl (
     hDevice,                    // TDRV004 handle
     IOCTL_TD004_WAIT_FOR_INT2,  // control code
     &Timeout,
     sizeof(int),
     NULL,
     0,
     &NumBytes,                  // number of bytes transferred
     NULL
);
if( success ) {
     // acknowledge interrupt source in FPGA logic
     // to clear the PLX PCI9030 Local Interrupt Source
} else {
     // Process DeviceIoControl() error
}
```

ERROR CODES

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

All other returned error codes are system error conditions.

SEE ALSO

Win32 documentation DeviceIoControl()