The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV014-SW-65

Windows 2000/XP Device Driver

Reconfigurable FPGA

Version 1.1.x

User Manual

Issue 1.1.1

October 2010

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7 25469 Halstenbek, Germany

Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19

e-mail: info@tews.com www.tews.com

TDRV014-SW-65

Windows 2000/XP Device Driver

Reconfigurable FPGA

Supported Modules:

TCP631

Issue	Description	Date
1.0.0	First Issue	September 15, 2009
1.1.0	DMA functions added	September 16, 2010
1.1.1	Filelist modified, API FileDescriptor type and error codes corrected	October 7, 2010

Table of Contents

1 Introduction

The TDRV014-SW-65 Windows WDM (Windows Driver Model) device driver is a kernel mode driver which allows the operation of the TCP631 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 TDRV014-SW-65 device driver supports the following features:

- Program and reconfigure onboard FPGA
- Program onboard clock generator
- Read/write FPGA registers (32bit / 16bit / 8bit)
- Read/write specific PLX PCI9056 EEPROM registers
- DMA transfers to/from FPGA resource

The TDRV014-SW-65 device driver supports the modules listed below:

TCP631	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.

TCP631 User manual
TCP631 Engineering Manual
PLX PCI9056 User Manual

2 Installation

Following files are located in directory TDRV014-SW-65 on the distribution media:

tdrv014.sys	Windows WDM driver binary
tdrv014.inf	Windows WDM installation script
tdrv014.h	Header file with IOCTL codes and structure definitions
EmbeddedIoDeviceClass.dll	Windows WDM device class library
TDRV014-SW-65-1.1.1.pdf	This document
api\tdrv014api.h	API include file
api\tdrv014api.c	API source file
svf_parser\svf_parser.h	SVF parser header file
svf_parser\svf_parser.c	SVF parser source file
example\tdrv014exa.c	Example application
example\fpgaexa.zip	Example FPGA design (SVF 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 TDRV014-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 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 TDRV014 product family.
7. Copy needed files (tdrv014.h, API and parser files) to desired target directory.

After successful installation a device is created for each found module (TDRV014_1, TDRV014_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 "***Embedded I/O***".
   The driver "***TEWS TECHNOLOGIES TDRV014 Reconfigurable FPGA (TCP631)***" should appear for each installed device.

3 Device Driver Programming

The TDRV014-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.)

4 API Documentation

4.1 General Functions

4.1.1 tdrv014open()

Name

tdrv014open() – opens a device.

Synopsis

```
TDRV014_HANDLE tdrv014open
(
     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 “tdrv014api.h”
TDRV014_HANDLE FileDescriptor;

/*
** open file descriptor to device
*/
FileDescriptor = tdrv014open(“\\\\.\\TDRV014_1” );
if (FileDescriptor == INVALID_HANDLE_VALUE)
{
     /* handle open error */
}
```

RETURNS

A device descriptor number, or INVALID_HANDLE_VALUE if the function fails. To get extended error information, call ***GetLastError***.

ERROR CODES

The error code is a standard error code set by the I/O system.

4.1.2 tdrv014close()

Name

tdrv014close() – closes a device.

Synopsis

```
int tdrv014close
(
     TDRV014_HANDLE     FileDescriptor
);
```

Description

This function closes previously opened devices.

Parameters

FileDescriptor

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

Example

```
#include “tdrv014api.h”
TDRV014_HANDLE FileDescriptor;
int result;

/*
** close file descriptor to device
*/
result = tdrv014close( FileDescriptor );
if (result < 0)
{
     /* handle close error */
}
```

RETURNS

Zero, or a negative error code.

ERROR CODES

The inverted error code is a standard error code set by the I/O system.

4.2 SVF Parser Functions

The SVF parser functions can be used to create the SVF command structure required by other TDRV014 API functions to execute JTAG commands.

4.2.1 svfparser_openfile()

Name

svfparser_openfile() – Open an SVF file

Synopsis

```
FILE* svfparser_openfile
(
     char*     filename
);
```

Description

This function opens an SVF file for parsing.

Parameters

filename

This value specifies the filename of the SVF file as a null-terminated character string.

Example

```
#include “svf_parser.h”

int                    result;
FILE*                  file;

/*
** Open an SVF file
*/
file = svfparser_openfile( “svffile.svf” );
if (!file)
{
     /* handle error */
}
```

RETURNS

On success, a file handle is returned. In the case of an error, zero is returned.

ERROR CODES

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

4.2.2 svfparser_closefile()

Name

svfparser_closefile() – Close an open SVF file

Synopsis

```
int svfparser_closefile
(
     FILE*     file
);
```

Description

This function closes a previously opened SVF file.

Parameters

file

This value specifies the filehandle which has been retrieved by a call to svfparser_openfile.

Example

```
#include “svf_parser.h”

FILE*                   file;
int                     result;

/*
** Close an SVF file
*/
result = svfparser_closefile( file );
if (result != 0)
{
     /* handle error */
}
```

RETURNS

On success, zero is returned. In the case of an error, EOF is returned.

ERROR CODES

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

4.2.3 svfparser_fetchcommand()

Name

svfparser_fetchcommand() – Fetches an SVF command from an SVF file

Synopsis

```
SVFCMD* svfparser_fetchcommand
(
     FILE*     file
);
```

Description

This function fetches an SVF command from a previously opened SVF file. The function allocates an SVFCMD structure and fills in all SVF command data. The application has to take care of the allocated memory. The SVFCMD structure is compatible to the TDRV014_SVFCMD structure.

Parameters

file

This value specifies the filehandle which has been retrieved by a call to svfparser_openfile.

Example

```
#include “svf_parser.h”

FILE*                   file;
SVFCMD*                 pSvfCmd;

/*
** fetch an SVF command from an SVF file
*/
pSvfCmd = svfparser_fetchcommand( file );
if (!pSvfCmd)
{
     /* handle error */
}
```

RETURNS

On success, a pointer to an allocated SVFCMD structure is returned. In the case of an error, NULL is returned.

ERROR CODES

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

4.2.4 svfparser_freecommand()

Name

svfparser_freecommand() – free the allocated memory of an SVF command

Synopsis

```
void svfparser_freecommand
(
     SVFCMD*        pSvfCmd
);
```

Description

This function frees the previously allocated memory of an SVF command.

Parameters

pSvfCmd

This value specifies a pointer to an SVF command which memory has to be freed again.

Example

```
#include “svf_parser.h”

SVFCMD*       pSvfCmd;

/*
** free memory of an allocated SVF command
*/
svfparser_freecommand( pSvfCmd );
```

4.3 Device Access Functions

4.3.1 tdrv014JtagEnable()

Name

tdrv014JtagEnable() – Enable access to JTAG controller CPLD

Synopsis

```
int tdrv014JtagEnable
(
     TDRV014_HANDLE     FileDescriptor
);
```

Description

This function enables access to the onboard JTAG controller CPLD of the specific device. The FPGA is put into reset to prevent access violations on the local bus.

Parameters

FileDescriptor

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

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;

/*
** Enable JTAG controller CPLD and put FPGA into reset
*/
result = tdrv014JtagEnable( FileDescriptor );
if (result < 0)
{
     /* handle error */
}
```

RETURNS

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

ERROR CODES

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

4.3.2 tdrv014JtagDisable()

Name

tdrv014JtagDisable() – Disable access to JTAG controller CPLD

Synopsis

```
int tdrv014JtagDisable
(
     TDRV014_HANDLE     FileDescriptor
);
```

Description

This function disables access to the onboard JTAG controller CPLD of the specific device. Depending on the configuration, the FPGA will start fetching its new configuration either from the Platform Flash or via the JTAG interface.

Parameters

FileDescriptor

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

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;

/*
** Disable JTAG controller CPLD and cause FPGA to configure
*/
result = tdrv014JtagDisable( FileDescriptor );
if (result < 0)
{
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_BUSY	The device is busy with SVF action.

Other returned error codes are system error conditions.

4.3.3 tdrv014PlaySvf()

Name

tdrv014PlaySvf() – Play a complete SVF file

Synopsis

```
int tdrv014PlaySvf
(
     TDRV014_HANDLE     FileDescriptor,
     char*              Filename,
     long*              currFilePos,
     long*              filesize
);
```

Description

This function can be used to program the FPGA or the clock generator. This function opens an SVF file, and reads and executes the contained single SVF commands using the SVF parser. This function does not affect the state of the JTAG controller CPLD. The function blocks until all SVF commands are executed properly, or an error occurred.

Before this function can be used, the access to the JTAG controller CPLD has to be enabled using the API function tdrv014JtagEnable(). The FPGA will not work before the JTAG controller CPLD has been disabled using API function tdrv014JtagDisable().

Parameters

FileDescriptor

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

Filename

This value specifies the filename of the SVF file as a null-terminated character string.

curFilePos

This value updates the current position in the SVF file. This value can be used to monitor the programming progress in a separate thread. Supply NULL if progress monitoring is not required.

filesize

This value updates the filesize of the supplied SVF file, and can be used as a basis to monitor the programming progress. Supply NULL if progress monitoring is not required.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;

/* enable JTAG controller */
result = tdrv014JtagEnable( FileDescriptor );
/* handle error */

/*
** Play SVF file “fpgacontent.svf” for FPGA programming
*/
result = tdrv014PlaySvf(     FileDescriptor,
                             “fpgacontent.svf”,
                             NULL,
                             NULL );
if (result < 0)
{
     /* handle error */
}

/* disable JTAG controller */
result = tdrv014JtagDisable( FileDescriptor );
/* handle error */
```

RETURNS

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

ERROR CODES

The error codes are set by the used functions for enabling and disabling the JTAG controller CPLD and executing an SVF command (refer to the corresponding function descriptions). Other error codes are standard error codes set by the I/O system.

4.3.4 tdrv014SvfCommand()

Name

tdrv014SvfCommand() – Execute a single SVF command

Synopsis

```
int tdrv014playSvf
(
     TDRV014_HANDLE     FileDescriptor,
     TDRV014_SVFCMD     *pSvfCommand
);
```

Description

This function executes a single SVF command. The function blocks until the SVF command is executed completely, or an error occurred.

Before executing this function, the JTAG controller CPLD must be enabled.

Parameters

FileDescriptor

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

pSvfCommand

This parameter supplies a pointer to an SVF structure containing all required data for this specific SVF command. Use the provided SVF parser to create the structure and fill in the SVF command data.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;
FILE*              file;
TDRV014_SVFCMD     *pSvfCmd;

/*
** open SVF file with SVF parser and get an SVF command
*/
file = svfparser_openfile( „svf_file_name.svf” );
pSvfCmd = (TDRV014_SVFCMD*)svfparser_fetchcommand( file );

if (pSvfCmd)
{
     result = tdrv014SvfCommand( FileDescriptor, pSvfCmd );
     if (result < 0)
     {
          /* handle error */
     }
     svfparser_freecommand( (SVFCMD*)pSvfCmd );
}

result = svfparser_closefile( file );
```

RETURNS

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

ERROR CODES

ERROR_ACCESS_DENIED	Access to JTAG controller CPLD is not enabled.
ERROR_INVALID_PARAMETER	There was an error during SVF processing.
ERROR_OPERATION_ABORTED	The function was cancelled.
ERROR_BUSY	The device is already busy with SVF action.
ERROR_NO_SYSTEM_RESOURCES	Error getting enough internal memory for SVF data.

Other returned error codes are system error conditions.

4.3.5 tdrv014DoneStatus()

Name

tdrv014DoneStatus() – Read DONE status of FPGA.

Synopsis

```
int tdrv014DoneStatus
(
     TDRV014_HANDLE     FileDescriptor
);
```

Description

This function reads the current FPGA's DONE status of the specified device.

The result of the function is on the return value. Following values are possible:

Value	Description
TDRV014_DONESTATUS_HIGH	The status of the FPGA's DONE signal is HIGH (configuration successful)
TDRV014_DONESTATUS_LOW	The status of the FPGA's DONE signal is LOW (error during configuration)

Parameters

FileDescriptor

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

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;

/*
** read DONE status
*/
result = tdrv014DoneStatus( FileDescriptor );
if (result >= 0)
{
       printf(“DONE Status: %s\n”,
              (result == TDRV014_DONESTATUS_HIGH) ? “HIGH” : “LOW”);
} else {
     /* handle error */
}
```

RETURNS

On success, a positive value is returned. In the case of an error, the appropriate negative error code is returned by the function.

ERROR CODES

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

4.3.6 tdrv014PowerStatus()

Name

tdrv014PowerStatus() – Read current status of onboard power supplies.

Synopsis

```
int tdrv014PowerStatus
(
     TDRV014_HANDLE     FileDescriptor
);
```

Description

This function returns the current status of the onboard power supplies.

The result of the function is on the return value. Following binary OR'ed values are possible:

Value	Description
TDRV014_POWERSTAT_2V5OK	Onboard 2.5V power supply is available
TDRV014_POWERSTAT_1V8OK	Onboard 1.8V power supply is available
TDRV014_POWERSTAT_1V2OK	Onboard 1.2V power supply is available
TDRV014_POWERSTAT_0V9OK	Onboard 0.8V power supply is available

Before executing this function, the JTAG controller CPLD must be enabled.

Parameters

FileDescriptor

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

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;

/*
** read Power Status
*/
result = tdrv014PowerStatus( FileDescriptor );
if (result >= 0)
{
       printf(“2.5V: %s\n”,
              (result & TDRV014_POWERSTAT_2V5OK) ? “OK” : “ERROR”);
       printf(“1.8V: %s\n”,
              (result & TDRV014_POWERSTAT_1V8OK) ? “OK” : “ERROR”);
       printf(“1.2V: %s\n”,
              (result & TDRV014_POWERSTAT_1V2OK) ? “OK” : “ERROR”);
       printf(“0.8V: %s\n”,
              (result & TDRV014_POWERSTAT_0V8OK) ? “OK” : “ERROR”);
} else {
     /* handle error */
}
```

RETURNS

On success, a positive value is returned. In the case of an error, the appropriate negative error code is returned by the function.

ERROR CODES

ERROR_ACCESS_DENIED	Access to JTAG controller CPLD is not enabled.

Other returned error codes are system error conditions.

4.3.7 tdrv014JtagChainConfigGet()

Name

tdrv014JtagChainConfigGet() – Read current configuration of onboard JTAG chain.

Synopsis

```
int tdrv014JtagChainConfigGet
(
     TDRV014_HANDLE     FileDescriptor
);
```

Description

This function returns the current configuration of the onboard JTAG chain. Specific JTAG chain segments can be disabled.

The result is on the return value. Following values (binary OR'ed) are possible:

Value	Description
TDRV014_CHAIN_BYPASS_FPGA	JTAG segment of FPGA is disabled
TDRV014_CHAIN_BYPASS_CLOCK	JTAG segment of clock generator is disabled
TDRV014_CHAIN_BYPASS_PIM	JTAG segment of local PIM slot is disabled
TDRV014_CHAIN_BYPASS_J2	JTAG segment of J2 rear I/O is disabled

Before executing this function, the JTAG controller CPLD must be enabled.

Parameters

FileDescriptor

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

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;

/*
** read Power Status
*/
result = tdrv014JtagChainConfigGet( FileDescriptor );
if (result >= 0)
{
     printf(“FPGA Segment: %s\n”,
          (result & TDRV014_CHAIN_BYPASS_FPGA) ? “BYPASS” : “in chain”);
     printf(“Clock Segment: %s\n”,
          (result & TDRV014_CHAIN_BYPASS_CLOCK) ? “BYPASS” : “in chain”);
     printf(“PIM Segment: %s\n”,
          (result & TDRV014_CHAIN_BYPASS_PIM) ? “BYPASS” : “in chain”);
     printf(“J2 Segment: %s\n”,
          (result & TDRV014_CHAIN_BYPASS_J2) ? “BYPASS” : “in chain”);
} else {
     /* handle error */
}
```

RETURNS

On success, a positive value is returned. In the case of an error, the appropriate negative error code is returned by the function.

ERROR CODES

ERROR_ACCESS_DENIED	Access to JTAG controller CPLD is not enabled.

Other returned error codes are system error conditions.

4.3.8 tdrv014JtagChainConfigSet()

Name

tdrv014JtagChainConfigSet() – Configure onboard JTAG chain.

Synopsis

```
int tdrv014JtagChainConfigSet
(
     TDRV014_HANDLE     FileDescriptor,
     unsigned char      ChainCfg
);
```

Description

This function configures the onboard JTAG chain. Specific JTAG chain segments can be disabled.

Before executing this function, the JTAG controller CPLD must be enabled.

Parameters

FileDescriptor

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

ChainCfg

This parameter specifies the JTAG segments which should be disabled. Following values (binary OR'ed) are possible:

Value	Description
TDRV014_CHAIN_BYPASS_FPGA	JTAG segment of FPGA is disabled
TDRV014_CHAIN_BYPASS_CLOCK	JTAG segment of clock generator is disabled
TDRV014_CHAIN_BYPASS_PIM	JTAG segment of local PIM slot is disabled
TDRV014_CHAIN_BYPASS_J2	JTAG segment of J2 rear I/O is disabled

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;
unsigned char      ChainCfg;

/*
** Configure JTAG chain
** bypass FPGA, PIM slot and J2, only leaving clock generator in chain
*/
ChainCfg  =    TDRV014_CHAIN_BYPASS_FPGA  |
               TDRV014_CHAIN_BYPASS_PIM   |
               TDRV014_CHAIN_BYPASS_J2;

result = tdrv014JtagChainConfigSet( FileDescriptor, ChainCfg );
if (result < 0)
{
     /* handle error */
}
```

RETURNS

On success, a zero value is returned. In the case of an error, the appropriate negative error code is returned by the function.

ERROR CODES

ERROR_ACCESS_DENIED	Access to JTAG controller CPLD is not enabled.
ERROR_INVALID_PARAMETER	Invalid parameter specified.

Other returned error codes are system error conditions.

4.3.9 tdrv014SetFpgaMode()

Name

tdrv014SetFpgaMode() – Configure FPGA configuration source.

Synopsis

```
int tdrv014SetFpgaMode
(
     TDRV014_HANDLE     FileDescriptor,
     unsigned char      FpgaMode
);
```

Description

This function configures the configuration source of the FPGA.

Before executing this function, the JTAG controller CPLD must be enabled.

Parameters

FileDescriptor

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

FpgaMode

This value specifies the new configuration source for the FPGA. Following values are possible:

Value	Description
TDRV014_FPGAMODE_FLASH	Cause the FPGA to load from Platform Flash
TDRV014_FPGAMODE_JTAG	Cause the FPGA to load content via JTAG

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;

/*
** configure FPGA to load from Platform Flash
*/
result = tdrv014SetFpgaMode( FileDescriptor, TDRV014_FPGAMODE_FLASH );
if (result < 0)
{
     /* handle error */
}
```

RETURNS

On success, a zero value is returned. In the case of an error, the appropriate negative error code is returned by the function.

ERROR CODES

ERROR_ACCESS_DENIED	Access to JTAG controller CPLD is not enabled.
ERROR_INVALID_PARAMETER	Invalid parameter specified.

Other returned error codes are system error conditions.

4.3.10 tdrv014PlxEepromRead()

Name

tdrv014PlxEepromRead() – Read 16bit value from PLX PCI9056 EEPROM.

Synopsis

```
int tdrv014PlxEepromRead
(
     TDRV014_HANDLE     FileDescriptor,
     int                Offset,
     unsigned short     *pusValue
);
```

Description

This function reads a 16bit value from a specific PLX PCI9056 EEPROM memory offset.

Parameters

FileDescriptor

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

Offset

Specifies the offset into the PLX PCI9056 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 Method
00h – 0Ah	R
0Ch – 22h	R / W
24h – 2Ah	R
2Ch – 42h	R / W
44h	R
46h – 52h	R / W
54h – 56h	R
58h	R / W
5Ch – 62h	R
64h – FEh	R / W

Refer to the PLX PCI9056 User Manual for detailed information on these registers.

pusValue

This parameter is a pointer to an *unsigned short* (16bit) value where the retrieved EEPROM value will be stored.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;
unsigned short     SubsysId;

/*
** read EEPROM value at offset 0x0C (Subsystem ID)
*/
result = tdrv014PlxEepromRead( FileDescriptor, 0x0C, &SubsysId );
if (result >= 0)
{
     printf( “SubsystemID = 0x%04X\n”, SubsysId );
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_PARAMETER	The specified offset is invalid.
ERROR_BUSY	The device is busy with SVF action.

Other returned error codes are system error conditions.

4.3.11 tdrv014PlxEepromWrite()

Name

tdrv014PlxEepromWrite() – Write 16bit value to PLX PCI9056 EEPROM.

Synopsis

```
int tdrv014PlxEepromRead
(
     TDRV014_HANDLE     FileDescriptor,
     int                Offset,
     unsigned short     usValue
);
```

Description

This function writes a 16bit value to a specific PLX PCI9056 EEPROM memory offset.

Parameters

FileDescriptor

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

Offset

Specifies the offset into the PLX PCI9056 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 Method
00h – 0Ah	R
0Ch – 22h	R / W
24h – 2Ah	R
2Ch – 42h	R / W
44h	R
46h – 52h	R / W
54h – 56h	R
58h	R / W
5Ch – 62h	R
64h – FEh	R / W

Refer to the PLX PCI9056 User Manual for detailed information on these registers.

usValue

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

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;

/*
** Change the Subsystem Vendor ID to TEWS TECHNOLOGIES (0x1498)
*/
result = tdrv014PlxEepromRead( FileDescriptor, 0x0E, 0x1498 );
if (result < 0)
{
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_PARAMETER	The specified offset is invalid, or read-only
ERROR_BUSY	The device is busy with SVF action.

Other returned error codes are system error conditions.

4.3.12 tdrv014Read8()

Name

tdrv014Read8() – Read 8bit values from FPGA resource.

Synopsis

```
int tdrv014Read8
(
     TDRV014_HANDLE     FileDescriptor,
     int                PciResource,
     int                Offset,
     int                NumItems,
     unsigned char      *pData
);
```

Description

This function reads a number of 8bit values from a PCI Memory or PCI I/O area by using BYTE (8bit) accesses.

Parameters

FileDescriptor

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

PciResource

This parameter specifies the desired PCI Memory or PCI I/O resource to be used for this access. The PLX PCI9056 target chip supports up to four base address registers. Following values are possible:

Value	Description
TDRV014_RES_MEM_1	First found PCI Memory area. *Reserved.*
TDRV014_RES_MEM_2	Second found PCI Memory area.
TDRV014_RES_MEM_3	Third found PCI Memory area.
TDRV014_RES_IO_1	First found PCI I/O area. *Reserved*.
TDRV014_RES_IO_2	Second found PCI I/O area.
TDRV014_RES_IO_3	Third found PCI I/O area.

The Base Address Register usage is programmable and can be changed by modifying the PLX PCI9056 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type	TDRV004_RESOURCE
0	MEM *(reserved)*	TDRV014_RES_MEM_1
1	I/O *(reserved)*	TDRV014_RES_IO_1
2	MEM *(used by VHDL Example)*	TDRV014_RES_MEM_2
3	MEM *(used by VHDL Example)*	TDRV014_RES_MEM_3

The PLX PCI9056 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the PCI Memory or PCI I/O area specified by *PciResource*.

NumItems

This value specifies the amount of data items to read.

pData

The received 8bit values are copied into this buffer. It must be large enough to hold the specified amount of data.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;
unsigned char      Data[50];

/*
** read 50 bytes from MemorySpace 2, offset 0x00
*/
result = tdrv014Read8(  FileDescriptor,
                        TDRV014_RES_MEM_2,
                        0x00,
                        50,
                        &Data );
if (result < 0)
{
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_PARAMETER	Specified Offset+NumItems exceeds the available memory or I/O space.
ERROR_ACCESS_DENIED	The specified Resource is not available for access.
ERROR_BUSY	The device is already busy with SVF action.

Other returned error codes are system error conditions.

4.3.13 tdrv014Read16()

Name

tdrv014Read16() – Read 16bit values from FPGA resource.

Synopsis

```
int tdrv014Read8
(
     TDRV014_HANDLE     FileDescriptor,
     int                PciResource,
     int                Offset,
     int                NumItems,
     unsigned short     *pData
);
```

Description

This function reads a number of 16bit values from a PCI Memory or PCI I/O area by using WORD (16bit) accesses. Little-endian access is used.

Parameters

FileDescriptor

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

PciResource

This parameter specifies the desired PCI Memory or PCI I/O resource to be used for this access. The PLX PCI9056 target chip supports up to four base address registers. Following values are possible:

Value	Description
TDRV014_RES_MEM_1	First found PCI Memory area. *Reserved.*
TDRV014_RES_MEM_2	Second found PCI Memory area.
TDRV014_RES_MEM_3	Third found PCI Memory area.
TDRV014_RES_IO_1	First found PCI I/O area. *Reserved*.
TDRV014_RES_IO_2	Second found PCI I/O area.
TDRV014_RES_IO_3	Third found PCI I/O area.

The Base Address Register usage is programmable and can be changed by modifying the PLX PCI9056 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type	TDRV004_RESOURCE
0	MEM *(reserved)*	TDRV014_RES_MEM_1
1	I/O *(reserved)*	TDRV014_RES_IO_1
2	MEM *(used by VHDL Example)*	TDRV014_RES_MEM_2
3	MEM *(used by VHDL Example)*	TDRV014_RES_MEM_3

The PLX PCI9056 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the PCI Memory or PCI I/O area specified by *PciResource*.

NumItems

This value specifies the amount of data items to read.

pData

The received 16bit values are copied into this buffer. It must be large enough to hold the specified amount of data.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;
unsigned short     Data[20];

/*
** read 20 16bit words from MemorySpace 2, offset 0x00
*/
result = tdrv014Read16( FileDescriptor,
                        TDRV014_RES_MEM_2,
                        0x00,
                        20,
                        &Data );
if (result < 0)
{
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_PARAMETER	Specified Offset+NumItems exceeds the available memory or I/O space.
ERROR_ACCESS_DENIED	The specified Resource is not available for access.
ERROR_BUSY	The device is already busy with SVF action.

Other returned error codes are system error conditions.

4.3.14 tdrv014Read32()

Name

tdrv014Read32() – Read 32bit values from FPGA resource.

Synopsis

```
int tdrv014Read8
(
     TDRV014_HANDLE     FileDescriptor,
     int                PciResource,
     int                Offset,
     int                NumItems,
     uint32_t           *pData
);
```

Description

This function reads a number of 32bit values from a PCI Memory or PCI I/O area by using DWORD (32bit) accesses. Little-endian access is used.

Parameters

FileDescriptor

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

PciResource

This parameter specifies the desired PCI Memory or PCI I/O resource to be used for this access. The PLX PCI9056 target chip supports up to four base address registers. Following values are possible:

Value	Description
TDRV014_RES_MEM_1	First found PCI Memory area. *Reserved.*
TDRV014_RES_MEM_2	Second found PCI Memory area.
TDRV014_RES_MEM_3	Third found PCI Memory area.
TDRV014_RES_IO_1	First found PCI I/O area. *Reserved*.
TDRV014_RES_IO_2	Second found PCI I/O area.
TDRV014_RES_IO_3	Third found PCI I/O area.

The Base Address Register usage is programmable and can be changed by modifying the PLX PCI9056 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type	TDRV004_RESOURCE
0	MEM *(reserved)*	TDRV014_RES_MEM_1
1	I/O *(reserved)*	TDRV014_RES_IO_1
2	MEM *(used by VHDL Example)*	TDRV014_RES_MEM_2
3	MEM *(used by VHDL Example)*	TDRV014_RES_MEM_3

The PLX PCI9056 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the PCI Memory or PCI I/O area specified by *PciResource*.

NumItems

This value specifies the amount of data items to read.

pData

The received 32bit values are copied into this buffer. It must be large enough to hold the specified amount of data.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;
uint32_t           Data[10];

/*
** read 10 32bit dwords from MemorySpace 2, offset 0x00
*/
result = tdrv014Read32( FileDescriptor,
                        TDRV014_RES_MEM_2,
                        0x00,
                        10,
                        &Data );
if (result < 0)
{
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_PARAMETER	Specified Offset+NumItems exceeds the available memory or I/O space.
ERROR_ACCESS_DENIED	The specified Resource is not available for access.
ERROR_BUSY	The device is already busy with SVF action.

Other returned error codes are system error conditions.

4.3.15 tdrv014Write8()

Name

tdrv014Write8() – Write 8bit values to FPGA resource.

Synopsis

```
int tdrv014Write8
(
     TDRV014_HANDLE     FileDescriptor,
     int                PciResource,
     int                Offset,
     int                NumItems,
     unsigned char      *pData
);
```

Description

This function writes a number of 8bit values to a PCI Memory or PCI I/O area by using BYTE (8bit) accesses.

Parameters

FileDescriptor

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

PciResource

This parameter specifies the desired PCI Memory or PCI I/O resource to be used for this access. The PLX PCI9056 target chip supports up to four base address registers. Following values are possible:

Value	Description
TDRV014_RES_MEM_1	First found PCI Memory area. *Reserved.*
TDRV014_RES_MEM_2	Second found PCI Memory area.
TDRV014_RES_MEM_3	Third found PCI Memory area.
TDRV014_RES_IO_1	First found PCI I/O area. *Reserved*.
TDRV014_RES_IO_2	Second found PCI I/O area.
TDRV014_RES_IO_3	Third found PCI I/O area.

The Base Address Register usage is programmable and can be changed by modifying the PLX PCI9056 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type	TDRV004_RESOURCE
0	MEM *(reserved)*	TDRV014_RES_MEM_1
1	I/O *(reserved)*	TDRV014_RES_IO_1
2	MEM *(used by VHDL Example)*	TDRV014_RES_MEM_2
3	MEM *(used by VHDL Example)*	TDRV014_RES_MEM_3

The PLX PCI9056 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the PCI Memory or PCI I/O area specified by *PciResource*.

NumItems

This value specifies the amount of data items to read.

pData

The 8bit values are copied from this buffer. It must be large enough to hold the specified amount of data.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;
unsigned char      Data[10];

/*
** write 10 bytes to MemorySpace 2, offset 0x00
*/
Data[0] = 0x41;
Data[1] = 0x42;
...
result = tdrv014Write8( FileDescriptor,
                        TDRV014_RES_MEM_2,
                        0x00,
                        10,
                        &Data );
if (result < 0)
{
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_PARAMETER	The specified Offset+NumItems exceeds the available memory or I/O space.
ERROR_ACCESS_DENIED	The specified Resource is not available for access.
ERROR_BUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

4.3.16 tdrv014Write16()

Name

tdrv014Write16() – Write 16bit values to FPGA resource.

Synopsis

```
int tdrv014Write8
(
     TDRV014_HANDLE     FileDescriptor,
     int                PciResource,
     int                Offset,
     int                NumItems,
     unsigned short     *pData
);
```

Description

This function writes a number of 16bit values to a PCI Memory or PCI I/O area by using WORD (16bit) accesses. Little-endian access is used.

Parameters

FileDescriptor

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

PciResource

This parameter specifies the desired PCI Memory or PCI I/O resource to be used for this access. The PLX PCI9056 target chip supports up to four base address registers. Following values are possible:

Value	Description
TDRV014_RES_MEM_1	First found PCI Memory area. *Reserved.*
TDRV014_RES_MEM_2	Second found PCI Memory area.
TDRV014_RES_MEM_3	Third found PCI Memory area.
TDRV014_RES_IO_1	First found PCI I/O area. *Reserved*.
TDRV014_RES_IO_2	Second found PCI I/O area.
TDRV014_RES_IO_3	Third found PCI I/O area.

The Base Address Register usage is programmable and can be changed by modifying the PLX PCI9056 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type	TDRV004_RESOURCE
0	MEM *(reserved)*	TDRV014_RES_MEM_1
1	I/O *(reserved)*	TDRV014_RES_IO_1
2	MEM *(used by VHDL Example)*	TDRV014_RES_MEM_2
3	MEM *(used by VHDL Example)*	TDRV014_RES_MEM_3

The PLX PCI9056 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the PCI Memory or PCI I/O area specified by *PciResource*.

NumItems

This value specifies the amount of data items to read.

pData

The 16bit values are copied from this buffer. It must be large enough to hold the specified amount of data.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;
unsigned short     Data[10];

/*
** write 10 16bit words to MemorySpace 2, offset 0x00
*/
Data[0] = 0x1234;
Data[1] = 0x5678;
...
result = tdrv014Write16(     FileDescriptor,
                             TDRV014_RES_MEM_2,
                             0x00,
                             10,
                             &Data );
if (result < 0)
{
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_PARAMETER	The specified Offset+NumItems exceeds the available memory or I/O space.
ERROR_ACCESS_DENIED	The specified Resource is not available for access.
ERROR_BUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

4.3.17 tdrv014Write32()

Name

tdrv014Write32() – Write 32bit values to FPGA resource.

Synopsis

```
int tdrv014Write8
(
     TDRV014_HANDLE     FileDescriptor,
     int                PciResource,
     int                Offset,
     int                NumItems,
     uint32_t           *pData
);
```

Description

This function writes a number of 32bit values to a PCI Memory or PCI I/O area by using DWORD (32bit) accesses. Little-endian access is used.

Parameters

FileDescriptor

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

PciResource

This parameter specifies the desired PCI Memory or PCI I/O resource to be used for this access. The PLX PCI9056 target chip supports up to four base address registers. Following values are possible:

Value	Description
TDRV014_RES_MEM_1	First found PCI Memory area. *Reserved.*
TDRV014_RES_MEM_2	Second found PCI Memory area.
TDRV014_RES_MEM_3	Third found PCI Memory area.
TDRV014_RES_IO_1	First found PCI I/O area. *Reserved*.
TDRV014_RES_IO_2	Second found PCI I/O area.
TDRV014_RES_IO_3	Third found PCI I/O area.

The Base Address Register usage is programmable and can be changed by modifying the PLX PCI9056 EEPROM. Therefore the following table is just an example how the PCI Base Address Registers could be used.

PCI Base Address Register	PCI Address-Type	TDRV004_RESOURCE
0	MEM *(reserved)*	TDRV014_RES_MEM_1
1	I/O *(reserved)*	TDRV014_RES_IO_1
2	MEM *(used by VHDL Example)*	TDRV014_RES_MEM_2
3	MEM *(used by VHDL Example)*	TDRV014_RES_MEM_3

The PLX PCI9056 default configuration utilizes only BAR0 to BAR2.

Offset

Specifies the offset into the PCI Memory or PCI I/O area specified by *PciResource*.

NumItems

This value specifies the amount of data items to read.

pData

The 32bit values are copied from this buffer. It must be large enough to hold the specified amount of data.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;
uint32_t           Data[10];

/*
** write 10 32bit words to MemorySpace 2, offset 0x00
*/
Data[0] = 0x12345678;
Data[1] = 0x9abcdef0;
...
result = tdrv014Write32(     FileDescriptor,
                             TDRV014_RES_MEM_2,
                             0x00,
                             10,
                             &Data );
if (result < 0)
{
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_PARAMETER	The specified Offset+NumItems exceeds the available memory or I/O space.
ERROR_ACCESS_DENIED	The specified Resource is not available for access.
ERROR_BUSY	The device is already busy with XSVF, Reconfig or SPI action.

Other returned error codes are system error conditions.

4.3.18 tdrv014InterruptWait()

Name

tdrv014InterruptWait() – Wait for incoming Local Interrupt Source.

Synopsis

```
int tdrv014InterruptWait
(
     TDRV014_HANDLE     FileDescriptor,
     int                timeout
);
```

Description

This function enables the local interrupt source, and waits for Local Interrupt Source to arrive. After the interrupt has arrived, the local interrupt source is disabled inside the PLX PCI9056.

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

FileDescriptor

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

timeout

This value specifies the timeout in milliseconds the function will wait for the interrupt to arrive. To wait indefinitely, specify -1 as timeout parameter. The timeout granularity is in seconds.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;

/*
** Wait at least 5 seconds for incoming interrupt
*/
result = tdrv014InterruptWait( FileDescriptor, 5000 );
if (result >= 0)
{
     /* Interrupt arrived.                              */
     /* Now acknowledge interrupt source in FPGA logic   */
     /* to clear the PLX PCI9056 Local Interrupt Source     */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_BUSY	Another job already waiting for this interrupt. Only one job is allowed at the same time.
ERROR_SEM_TIMEOUT	The specified timeout occurred.

Other returned error codes are system error conditions.

4.3.19 tdrv014AllocPhysMem()

Name

tdrv014AllocPhysMem() – Allocate physically contiguous memory.

Synopsis

```
int tdrv014AllocPhysMem
(
     TDRV014_HANDLE              FileDescriptor,
     TDRV014_PHYSICAL_MEMORY     *pPhysMem
);
```

Description

This function allocates physically contiguous, non-cached memory, which can be used for DMA transfers. Because of the missing user-mode capability, the memory is allocated and managed by the device driver. Before accessing the memory from the user application, the memory must be mapped into the process context.

Parameters

FileDescriptor

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

pPhysMem

This value specifies a pointer to a TDRV014_PHYSICAL_MEMORY structure.

```
typedef struct
{
   U64 UserAddr;
   U64 PhysicalAddr;
   U64 CpuPhysical;
   U32 Size;
} TDRV014_PHYSICAL_MEMORY;
```

UserAddr

Virtual address used for accessing the memory from the user application. This value is valid after mapping the memory. **Do not change this value.**

PhysicalAddr

Physical address used for DMA transfer by the hardware. **Do not change this value.**

CpuPhysical

Physical address used for DMA transfer by the hardware. **Do not change this value.**

Size

This value describes the desired size of the memory section in bytes. The maximum size is also limited by system resources like mapping tables. The actual size of the allocated memory block is returned in this parameter.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE          FileDescriptor;
int                     result;
TDRV014_PHYSICAL_MEMORY PhysMem;

/*
** Allocate 1024 bytes of physically contiguous memory
*/
PhysMem.Size = 1024;

result = tdrv014AllocPhysMem( FileDescriptor, &PhysMem );
if (result >= 0)
{
     /* memory allocation successful */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_USER_BUFFER	The provided buffer is invalid.
ERROR_NO_SYSTEM_RESOURCES	Not enough memory resources available.

Other returned error codes are system error conditions.

4.3.20 tdrv014FreePhysMem()

Name

tdrv014FreePhysMem() – Free allocated physically contiguous memory.

Synopsis

```
int tdrv014FreePhysMem
(
     TDRV014_HANDLE              FileDescriptor,
     TDRV014_PHYSICAL_MEMORY     *pPhysMem
);
```

Description

This function frees previously allocated physically contiguous, non-cached memory.

Parameters

FileDescriptor

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

pPhysMem

This value specifies a pointer to a TDRV014_PHYSICAL_MEMORY structure, which has been populated by a previous call to tdrv014AllocPhysMem(). Do not change the structure content.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE          FileDescriptor;
int                     result;
TDRV014_PHYSICAL_MEMORY PhysMem;

/*
** Free previously allocated physically contiguous memory
*/
result = tdrv014FreePhysMem( FileDescriptor, &PhysMem );
if (result >= 0)
{
     /* freeing memory successful */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_USER_BUFFER	The provided buffer is invalid.
ERROR_INVALID_PARAMETER	The specified memory section was not found.

Other returned error codes are system error conditions.

4.3.21 tdrv014MapPhysMem()

Name

tdrv014MapPhysMem() – Map allocated physically contiguous memory into application

Synopsis

```
void* tdrv014MapPhysMem
(
     TDRV014_HANDLE              FileDescriptor,
     TDRV014_PHYSICAL_MEMORY     *pPhysMem
);
```

Description

This function maps previously allocated physically contiguous, non-cached memory, which can be used for DMA transfers, into the application's virtual memory space.

Parameters

FileDescriptor

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

pPhysMem

This value specifies a pointer to a TDRV014_PHYSICAL_MEMORY structure, which has been populated by a previous call to tdrv014AllocPhysMem(). Do not change the structure content, because it is used as a unique memory identifier.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE          FileDescriptor;
TDRV014_PHYSICAL_MEMORY PhysMem;
unsigned char           *pData;

/*
** Map the allocated physically contiguous memory for application access
*/
pData = tdrv014MapPhysMem( FileDescriptor, &PhysMem );
if (pData == NULL)
{
     /* handle error */
}
```

RETURNS

On success, a pointer to the virtual address is returned. In the case of an error, NULL is returned by the function. To get extended error information, call GetLastError.

ERROR CODES

ERROR_INVALID_USER_BUFFER	The provided buffer is invalid.
ERROR_INVALID_PARAMETER	The specified memory section was not found.

Other returned error codes are system error conditions.

4.3.22 tdrv014UnmapPhysMem()

Name

tdrv014UnmapPhysMem() – Unmap physically contiguous memory

Synopsis

```
int tdrv014UnmapPhysMem
(
     TDRV014_HANDLE              FileDescriptor,
     TDRV014_PHYSICAL_MEMORY     *pPhysMem
);
```

Description

This function maps previously allocated physically contiguous, non-cached memory, which can be used for DMA transfers, into the application's virtual memory space.

Parameters

FileDescriptor

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

pPhysMem

This value specifies a pointer to a TDRV014_PHYSICAL_MEMORY structure, which has been populated by a previous call to tdrv014MapPhysMem(). Do not change the structure content.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE          FileDescriptor;
int                     result;
TDRV014_PHYSICAL_MEMORY PhysMem;

/*
** Unmap the physically contiguous memory
*/
result = tdrv014UnmapPhysMem( FileDescriptor, &PhysMem );
if (result >= 0)
{
     /* unmapping memory successful  */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_USER_BUFFER	The provided buffer is invalid.
ERROR_INVALID_PARAMETER	The specified memory section was not found.

Other returned error codes are system error conditions.

4.3.23 tdrv014DmaConfig()

Name

tdrv014DmaConfig() – Configure specified DMA channel

Synopsis

```
int tdrv014DmaConfig
(
     TDRV014_HANDLE     FileDescriptor,
     int                DmaChannel,
     int                DataWidth,
     int                WaitStates,
     unsigned char      BurstMode,
     unsigned char      AddressingMode,
     int                UseReady
);
```

Description

This function configures the specified DMA channel.

Parameters

FileDescriptor

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

DmaChannel

This value specifies the DMA channel which shall be configured. Possible values are 0 or 1.

DataWidth

This value specifies the data width to be used on the local bus. Possible values are:

Value	Description
1	8bit local bus
2	16bit local bus
4	32bit local bus

WaitStates

This value specifies the number of wait states to be used on the local bus. Possible values are 0 to 15.

BurstMode

This value specifies whether or not burst transfers should be used on the local bus. Possible values are:

Value	Description
TDRV014_DMACONFIG_NOBURST	No burst transfers will be used on the local bus.
TDRV014_DMACONFIG_BURST4	4 DWORDs will be transferred in one burst access. The number of local accesses depends on the selected data width.
TDRV014_DMACONFIG_BURSTCONT	Continuous bursting will be used on the local bus.

AddressingMode

This value specifies if the local address is incremented or held constant during a DMA transfer. Possible values are:

Value	Description
TDRV014_DMACONFIG_ADDRINCR	The local address is incremented during a DMA transfer.
TDRV014_DMACONFIG_ADDRCONST	The local address is held constant during a DMA transfer.

UseReady

This value specifies if the Ready input line shall be used. Specify TRUE (1) to use the Ready signal during DMA transfers, or specify FALSE (0).

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;

/*
** Configure DMA channel 0:
**   - use 32bit local data width
**   - no waitstates, no bursting
**   - use incrementing local addresses
**   - use Ready signal
*/
result = tdrv014DmaConfig(   FileDescriptor,
                             0,
                             4,
                             0,
                             TDRV014_DMACONFIG_NOBURST,
                             TDRV014_DMACONFIG_ADDRINCR,
                             TRUE );
if (result >= 0)
{
     /* DMA channel configured properly */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_PARAMETER	At least one of the specified parameters is invalid.

Other returned error codes are system error conditions.

4.3.24 tdrv014DmaAbort()

Name

tdrv014DmaAbort() – Abort pending DMA transfer

Synopsis

```
int tdrv014DmaAbort
(
     TDRV014_HANDLE     FileDescriptor,
     int                DmaChannel
);
```

Description

This function aborts a pending DMA transfer of a specified DMA channel. Currently blocking functions waiting for termination of the affected DMA transfer will be unblocked.

Parameters

FileDescriptor

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

DmaChannel

This value specifies the DMA channel which shall be aborted. Possible values are 0 or 1.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;

/*
** Abort a pending DMA transfer on channel 0
*/
result = tdrv014DmaAbort( FileDescriptor, 0 );
if (result >= 0)
{
     /* aborting transfer successful */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_PARAMETER	Specified channel number is invalid.

Other returned error codes are system error conditions.

4.3.25 tdrv014DmaStatus()

Name

tdrv014DmaStatus() – Read Status of specified DMA channel

Synopsis

```
int tdrv014DmaStatus
(
     TDRV014_HANDLE     FileDescriptor,
     int                DmaChannel
);
```

Description

This function returns the current status of a specified DMA channel. This function can be used to check the status of a pending DMA transfer, e.g. to periodically check until it has finished.

Possible returned status values are:

Value	Description
TDRV014_DMASTATUS_DONE	The DMA channel has finished a previous transfer.
TDRV014_DMASTATUS_BUSY	The DMA channel is currently busy with a pending transfer.

Parameters

FileDescriptor

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

DmaChannel

This value specifies the DMA channel which shall be checked. Possible values are 0 or 1.

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE     FileDescriptor;
int                result;

/*
** Check the status of a pending DMA transfer on channel 0
*/
result = tdrv014DmaStatus( FileDescriptor, 0 );
if (result >= 0)
{
     if (result == TDRV014_DMASTATUS_DONE)
     {
          /* the DMA transfer has been finished */
     }
     if (result == TDRV014_DMASTATUS_BUSY)
     {
          /* the DMA transfer is still busy */
     }
} else {
     /* handle error */
}
```

RETURNS

On success, the DMA channel’s status is returned. In the case of an error, the appropriate negative error code is returned by the function.

ERROR CODES

ERROR_INVALID_PARAMETER	Specified channel number is invalid.

Other returned error codes are system error conditions.

4.3.26 tdrv014DmaWrite()

Name

tdrv014DmaWrite() – Start a DMA write transfer (Host RAM to Local FPGA Address)

Synopsis

```
int tdrv014DmaWrite
(
     TDRV014_HANDLE              FileDescriptor,
     int                         DmaChannel,
     unsigned long               BoardAddr,
     TDRV014_PHYSICAL_MEMORY     *pPhysMem,
     int                         Length,
     int                         Timeout,
     OVERLAPPED                  *pOverlapped
);
```

Description

This function starts a DMA transfer from the Host RAM to a local FPGA address. The function can either be used to return immediately with using an OVERLAPPED structure, or to block until the DMA transfer has finished or the specified timeout occurred.

Parameters

FileDescriptor

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

DmaChannel

This value specifies the DMA channel which shall be used. Possible values are 0 or 1.

BoardAddr

This value specifies the local FPGA address (destination).

pPhysMem

This value specifies the host RAM address (source). This address descriptor must be created by using tdrv014AllocPhysMem() and tdrv014MapPhysMem().

Length

This value specifies the size of the DMA transfer in bytes.

Timeout

This value specifies the allowed time for the transfer to finish in seconds. If the time elapses, the DMA transfer will be aborted.

pOverlapped

Specify NULL to cause the function block until the DMA transfer has finished, or a timeout occurred. If this value is a pointer to a Windows OVERLAPPED structure, the function will return immediately. The OVERLAPPED structure can be used to monitor the pending transfer, e.g. by using an event to wait (refer to the corresponding SDK documentation).

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE          FileDescriptor;
int                     result;
TDRV014_PHYSICAL_MEMORY PhysMem;

/*
** Using DMA channel 0, transfer 100 bytes from host RAM to
** local FPGA address 0x00000000, wait up to 10 seconds for completion.
** Block until eiter the transfer has finished or timeout.
*/
result = tdrv014DmaWrite( FileDescriptor,
                  0,
                  0x00000000,
                  &PhysMem,
                  100,
                  10,
                  NULL );
if (result >= 0)
{
     /* the DMA transfer has been finished */
} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_PARAMETER	Specified channel number is invalid.
ERROR_BUSY	Specified DMA channel is busy.
ERROR_SEM_TIMEOUT	The specified timeout occurred.

Other returned error codes are system error conditions.

4.3.27 tdrv014DmaRead()

Name

tdrv014DmaRead() – Start a DMA read transfer (Local FPGA Address to Host RAM)

Synopsis

```
int tdrv014DmaRead
(
     TDRV014_HANDLE              FileDescriptor,
     int                         DmaChannel,
     unsigned long               BoardAddr,
     TDRV014_PHYSICAL_MEMORY     *pPhysMem,
     int                         Length,
     int                         Timeout,
     OVERLAPPED                  *pOverlapped
);
```

Description

This function starts a DMA transfer from a local FPGA address to the Host RAM. The function can either be used to return immediately with using an OVERLAPPED structure, or to block until the DMA transfer has finished or the specified timeout occurred.

Parameters

FileDescriptor

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

DmaChannel

This value specifies the DMA channel which shall be used. Possible values are 0 or 1.

BoardAddr

This value specifies the local FPGA address (source).

pPhysMem

This value specifies the host RAM address (destination). This address descriptor must be created by using tdrv014AllocPhysMem() and tdrv014MapPhysMem().

Length

This value specifies the size of the DMA transfer in bytes.

Timeout

This value specifies the allowed time for the transfer to finish in seconds. If the time elapses, the DMA transfer will be aborted.

pOverlapped

Specify NULL to cause the function block until the DMA transfer has finished, or a timeout occurred. If this value is a pointer to a Windows OVERLAPPED structure, the function will return immediately. The OVERLAPPED structure can be used to monitor the pending transfer, e.g. by using an event to wait (refer to the corresponding SDK documentation).

Example

```
#include “tdrv014api.h”

TDRV014_HANDLE          FileDescriptor;
int                     result;
TDRV014_PHYSICAL_MEMORY PhysMem;
OVERLAPPED              Overlapped;
DWORD                   dwResult, timeout_ms;

/*
** Using DMA channel 1, transfer 100 bytes from
** local FPGA address 0x80000000 to Host RAM,
** wait up to 10 seconds for completion.
** Use Overlapped operation.
*/
Overlapped.hEvent  = CreateEvent( NULL, FALSE, FALSE, NULL );
timeout_ms         = 10000;

result = tdrv014DmaRead( FileDescriptor,
                  1,
                  0x80000000,
                  &PhysMem,
                  100,
                  10,
                  NULL );
...
```

```
if (result >= 0)
{
     /*
     ** The DMA transfer has been started properly.
     ** Now wait on the overlapped event for completion.
     */
     dwResult = WaitForSingleObject( Overlapped.hEvent, timeout_ms );
     if (dwResult == WAIT_TIMEOUT)
     {
          /*
          ** handle timeout of WaitForSingleObject,
          ** e.g. abort DMA transfer with tdrv014DmaAbort()
          */
     }
     if (dwResult == WAIT_OBJECT_0)
     {
          printf("Transfer finished.\n");
     }

} else {
     /* handle error */
}
```

RETURNS

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

ERROR CODES

ERROR_INVALID_PARAMETER	Specified channel number is invalid.
ERROR_BUSY	Specified DMA channel is busy.
ERROR_SEM_TIMEOUT	The specified timeout occurred.

Other returned error codes are system error conditions.