The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV014-SW-82

Linux Device Driver

Reconfigurable FPGA

Version 1.0.x

User Manual

Issue 1.0.1

May 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-82

Linux Device Driver

Reconfigurable FPGA

Supported Modules:
TCP631

Issue	Description	Date
1.0.0	First Issue	April 24, 2009
1.0.1	Address TEWS LLC removed	May 28, 2010

Table of Contents

1 Introduction

The TDRV014-SW-82 Linux device driver allows the operation of the TDRV014 compatible devices conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close()*,and *ioctl()* functions.

The TDRV014-SW-82 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

The TDRV014-SW-82 supports the modules listed below:

TCP631	Reconfigurable FPGA	CompactPCI

In this document all supported modules and devices will be called TDRV014. Specials for a certain device will be advised.

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

TCP631 (or compatible) User manual
TCP631 (or compatible) Engineering Manual
PLX PCI9056 User Manual

2 Installation

The directory TDRV014-SW-82 on the distribution media contains the following files:

TDRV014-SW-82-1.0.0.pdf	This manual in PDF format
TDRV014-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
fpgaexa.tar.gz	FPGA example SVF
ChangeLog.txt	Release history
Release.txt	Information about the Device Driver Release

The GZIP compressed archive TDRV014-SW-82-SRC.tar.gz contains the following files and directories:

Directory path 'tdrv014':

tdrv014.c	Driver source code
tdrv014def.h	Driver include file
tdrv014.h	Driver include file for application program
Makefile	Device driver make file
makenode	Script for device node creation
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/Makefile	Example application makefile
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
include/tpxxxhwdep.h	HAL library header file
include/tpxxxhwdep.c	HAL library source file

In order to perform an installation, extract all files of the archive TDRV014-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TDRV014-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tdrv014.h to */usr/include*

2.1 Build and install the device driver

- Login as *root*
- Change to the target directory
- To create and install the driver in the module directory */lib/modules/<version>/misc* enter:

  # make install

- To update the device driver's module dependencies, enter:

  # **depmod -aq**

2.2 Uninstall the device driver

- Login as *root*
- Change to the target directory
- To remove the driver from the module directory */lib/modules/<version>/misc* enter:

  # make uninstall

2.3 Install device driver into the running kernel

- To load the device driver into the running kernel, login as root and execute the following commands:

  # modprobe tdrv014drv

- After the first build or if you are using dynamic major device allocation it is necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled a device file system (devfs or sysfs with udev) then you have to skip running the *makenode* script. Instead of creating device nodes from the script the driver itself takes creating and destroying of device nodes in its responsibility.

  # sh makenode

On success the device driver will create a minor device for each TDRV014 device found. The first TDRV014 device can be accessed with device node /dev/tdrv014_0, the second module with device node /dev/tdrv014_1 and so on.

The assignment of device nodes to physical TDRV014 modules depends on the search order of the PCI bus driver.

2.4 Remove device driver from the running kernel

- To remove the device driver from the running kernel login as root and execute the following command:

  # modprobe –r tdrv014drv

If your kernel has enabled devfs or sysfs (udev), all /dev/tdrv014_x nodes will be automatically removed from your file system after this.

Be sure that the driver isn't opened by any application program. If opened you will get the response "*tdrv014drv: Device or resource busy*" and the driver will still remain in the system until you close all opened files and execute *modprobe –r* again.

2.5 Change Major Device Number

This paragraph is only for Linux kernels without dynamic device file system installed. The TDRV014 driver uses dynamic allocation of major device numbers per default. If this isn't suitable for the application it is possible to define a major number for the driver.

To change the major number, edit the file tdrv014def.h, change the following symbol to appropriate value, and enter `make install` to create a new driver.

TDRV014_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TDRV014_MAJOR   122
```

Be sure that the desired major number isn't used by other drivers. Please check *./proc/devices* to see which numbers are free.

3 Device Input/Output functions

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

3.1 open()

NAME

open() opens a file descriptor.

SYNOPSIS

#include <fcntl.h>

int open (*const char *filename, int flags*)

DESCRIPTION

The **open** function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file is to be opened. This is a bit mask. Create the value by the bitwise OR of the appropriate parameters (using the | operator in C). See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int fd;

fd = open("/dev/tdrv014_0", O_RDWR);
if (fd == -1) {
   /* handle error condition */
}
```

RETURNS

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

ERRORS

ENODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during open. For more information about open error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.2 close()

NAME

close() closes a file descriptor.

SYNOPSIS

#include <unistd.h>

int **close** (int *filedes*)

DESCRIPTION

The **close** function closes the file descriptor *filedes*.

EXAMPLE

```
int fd;

if (close(fd) != 0) {
  /* handle close error conditions */
}
```

RETURNS

The normal return value from **close** is 0. In case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

ENODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during close. For more information about close error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 ioctl()

NAME

ioctl() device control functions

SYNOPSIS

#include <sys/ioctl.h>

int **ioctl**(*int filedes, int request [, void *argp])*

DESCRIPTION

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

The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *tdrv014.h*:

Value	Meaning
TDRV014_IOC_JTAGENABLE	Enable JTAG controller CPLD
TDRV014_IOC_JTAGDISABLE	Disable JTAG controller CPLD
TDRV014_IOCX_SVFCMD	Execute SVF command
TDRV014_IOCQ_GETDONESTATUS	Read DONE Status of FPGA
TDRV014_IOCQ_GETPOWERSTATUS	Read current status of onboard power supplies
TDRV014_IOCT_JTAGCHAINCONFIGSET	Configure onboard JTAG chain
TDRV014_IOCQ_JTAGCHAINCONFIGGET	Read current JTAG chain configuration
TDRV014_IOCT_SETFPGAMODE	Configure FPGA configuration source
TDRV014_IOCS_PLXWRITE	Write 16bit value to PLX PCI9056 EEPROM
TDRV014_IOCG_PLXREAD	Read 16bit value from PLX PCI9056 EEPROM
TDRV014_IOCG_READ_U8	Read 8bit values from FPGA resource
TDRV014_IOCG_READ_U16	Read 16bit values from FPGA resource
TDRV014_IOCG_READ_U32	Read 32bit values from FPGA resource
TDRV014_IOCS_WRITE_U8	Write 8bit values to FPGA resource
TDRV014_IOCS_WRITE_U16	Write 16bit values to FPGA resource
TDRV014_IOCS_WRITE_U32	Write 32bit values to FPGA resource
TDRV014_IOCT_WAIT_FOR_INT	Wait for incoming Local Interrupt Source

See below for more detailed information on each control code.

To use these TDRV014 specific control codes the header file *tdrv014.h* must be included in the application.

RETURNS

On success, zero or a value greater than zero is returned. In case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code. Special ioctl functions may return a specific result.

ERRORS

EINVAL	Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument *request*.

Other function dependent error codes will be described for each ioctl code separately. Note, the TDRV014 device driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TDRV014_IOC_JTAGENABLE

NAME

TDRV014_IOC_JTAGENABLE – Enable JTAG controller CPLD

DESCRIPTION

This function enables access to the onboard JTAG controller CPLD. The FPGA is put into reset to prevent access violations on the local bus. No additional parameter is used for this function, so the optional argument can be omitted.

EXAMPLE

```
#include <tdrv014.h>

int fd;
int result;

result = ioctl(fd, TDRV014_IOC_JTAGENABLE);

if (result < 0) {
  /* handle ioctl error */
}
```

ERRORS

EBUSY	The device is busy with an SVF action.

Other returned error codes are system error conditions.

3.3.2 TDRV014_IOC_JTAGDISABLE

NAME

TDRV014_IOC_JTAGDISABLE – Disable JTAG controller CPLD

DESCRIPTION

This function disables access to the onboard JTAG controller CPLD. Depending on the configuration, the FPGA will start fetching its new configuration either from the Platform Flash or via the JTAG interface. No additional parameter is used for this function, so the optional argument can be omitted.

EXAMPLE

```
#include <tdrv014.h>

int fd;
int result;

result = ioctl(fd, TDRV014_IOC_JTAGDISABLE);

if (result < 0) {
  /* handle ioctl error */
}
```

ERRORS

EBUSY	The device is busy with an SVF action.

Other returned error codes are system error conditions.

3.3.3 TDRV014_IOCS_SVFCMD

NAME

TDRV014_IOCS_SVCMD – Execute SVF command

DESCRIPTION

This ioctl function executes a single SVF command and returns the result in the supplied buffer. A pointer to the caller's data buffer (*TDRV014_SVFCMD*) is passed by the parameter *argp* to the driver.

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

The data structure *TDRV014_SVFCMD* is generated and filled with valid content by the SVF parser.

```
typedef struct {
   int CmdId;
   int structsize;

   union {
      struct {
         int run_state;
         int end_state;
         int run_count;
      } runtest;

      struct {
         int NumStates;
         int State[1];
      } state;

      struct {
         int trst_mode;
      } trst;
      struct {
         int StableState;
      } enddr;

      struct {
         int StableState;
      } endir;

      struct {
         int Frequency;
      } frequency;
```

```
        struct {
            int         length;
            int         idx_tdi;
            int         num_tdi;
            int         idx_tdo;
            int         num_tdo;
            int         idx_mask;
            int         num_mask;
            int         idx_smask;
            int         num_smask;
            uint32_t    data[1];        /* variable length, so this must be the last item in this structure */
        } tditdo;
    } u;
} TDRV014_SVFCMD;
```

CmdId

Specifies the SVF command ID. Supported values are:

Value	**Description**
TDRV014_CMDID_ENDDR	Set End State for Data Register Scan use structure *u.enddr* for passing parameters
TDRV014_CMDID_ENDIR	Set End State for Instruction Register Scan use *u.endir* for passing parameters
TDRV014_CMDID_FREQUENCY	Configure JTAG frequency use structure *u.frequency* for passing parameters
TDRV014_CMDID_HDR	Set Header for Data Register Scan use structure *u.tditdo* for passing parameters
TDRV014_CMDID_HIR	Set Header for Instruction Register Scan use structure *u.tditdo* for passing parameters
TDRV014_CMDID_RUNTEST	Execute RUNTEST use structure *u.runtest* for passing parameters
TDRV014_CMDID_SDR	Scan Data Register use structure *u.tditdo* for passing parameters
TDRV014_CMDID_SIR	Scan Instruction Register use structure *u.tditdo* for passing parameters
TDRV014_CMDID_TDR	Set Trailer for Data Register Scan use structure *u.tditdo* for passing parameters
TDRV014_CMDID_TIR	Set Trailer for Instruction Register Scan use structure *u.tditdo* for passing parameters
TDRV014_CMDID_TRST	Execute Target Reset use structure *u.trst* for passing parameters
TDRV014_CMDID_STATE	Move into specified TAP state use structure *u.state* for passing parameters

structsize

This parameter specifies the complete size of this dynamic structure. This is required for proper memory mapping or copying of the user data.

u.runtest.run_state

This parameter specifies the JTAG state used for the RUNTEST command. Besides -1 (which corresponds to the IDLE TAP-State), supported values are:

Value	Description
TDRV014_TAPSTATE_IDLE	IDLE / Run-Test run state

u.runtest.end_state

This parameter specifies the end state used when the RUNTEST command has finished. Supported values are:

Value	Description
TDRV014_TAPSTATE_RESET	Test-Logic-Reset end state
TDRV014_TAPSTATE_IDLE	IDLE end state
TDRV014_TAPSTATE_DRPAUSE	DRPAUSE end state
TDRV014_TAPSTATE_IRPAUSE	IRPAUSE end state

u.runtest.run_count

This parameter specifies the number of JTAG clocks.

u.state.NumStates

This parameter specifies the number of TAP states specified in the following parameter.

u.state.State

This parameter array specifies the TAP states which have to be entered one after the other. Supported values are:

Value	Description
TDRV014_TAPSTATE_RESET	Test-Logic-Reset end state
TDRV014_TAPSTATE_IDLE	IDLE end state
TDRV014_TAPSTATE_DRPAUSE	DRPAUSE end state
TDRV014_TAPSTATE_IRPAUSE	IRPAUSE end state

u.trst.trst_mode

This parameter specifies the Target Reset mode. Supported values are:

Value	Description
TDRV014_TRSTMODE_ON	Set Reset to ON
TDRV014_TRSTMODE_OFF	Set Reset to OFF

u.enddr.StableState, u.endir.StableState

This parameter specifies the stable end state for JTAG Data Register Scan respective Instruction Register Scan. Supported values are:

Value	Description
TDRV014_TAPSTATE_RESET	TRST end state
TDRV014_TAPSTATE_IDLE	IDLE end state
TDRV014_TAPSTATE_DRPAUSE	DRPAUSE end state
TDRV014_TAPSTATE_IRPAUSE	IRPAUSE end state

u.frequency.Frequency

This parameter specifies the JTAG frequency to be used (in Hz). The closest achievable frequency will be used, based upon the available system clock frequency.

u.tditdo.length

This parameter specifies the number of valid JTAG bits for this command. The specified number of bits is shifted through the onboard JTAG chain.

u.tditdo.idx_tdi

This parameter specifies the start of TDI data in the structure member *data* (see below). If this value is -1, no TDI data has been specified.

u.tditdo.num_tdi

This parameter specifies the number of valid 32bit values for TDI data in the structure member *data* (see below).

u.tditdo.idx_tdo

This parameter specifies the start of TDO data in the structure member *data* (see below). If this value is -1, no TDO data has been specified.

u.tditdo.num_tdo

This parameter specifies the number of valid 32bit values for TDO data in the structure member *data* (see below).

u.tditdo.idx_mask

This parameter specifies the start of MASK data in the structure member *data* (see below). If this value is -1, no MASK data has been specified.

u.tditdo.num_mask

This parameter specifies the number of valid 32bit values for MASK data in the structure member *data* (see below).

u.tditdo.idx_smask

This parameter specifies the start of SMASK data in the structure member *data* (see below). If this value is -1, no SMASK data has been specified.

u.tditdo.num_smask

This parameter specifies the number of valid 32bit values for SMASK data in the structure member *data* (see below).

u.tditdo.data

This parameter is an array of 32bit values, which contains data for TDI, TDO, MASK and SMASK values. This data array is of a dynamic size and can be enlarged by allocated the required amount of memory for the complete structure.

Depending on the SVF file, programming an FPGA or platform flash might take a long time. The used Platform Flash device may require minutes to be erased, so please be patient during the programming process. The progress can be monitored by evaluating the current SVF file position. It is not possible to monitor the progress of one single SVF command, though.

EXAMPLE

```
#include <tdrv014.h>

int              fd;
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) {
     /*
     ** Execute SVF command
     */
     result = ioctl(fd, TDRV014_IOCS_SVFCMD, pSvfCmd);

     if (result < 0) {
       /* handle ioctl error */
     }

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

ERRORS

EACCES	Access to JTAG controller CPLD is not enabled.
EINVAL	There was an error during SVF processing.
EINTR	The function was cancelled.
EFAULT	Error while copying data to or from user space.
EBUSY	The device is already busy with SVF action.
ENOMEM	Error getting enough internal memory for SVF data.

Other returned error codes are system error conditions.

3.3.4 TDRV014_IOCQ_GETDONESTATUS

NAME

TDRV014_IOCQ_GETDONESTATUS – Read DONE status of FPGA

DESCRIPTION

This ioctl function returns the status of the FPGA's DONE signal to check if the configuration process was successful. No additional parameter is used for this function, so the optional argument can be omitted.

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)

EXAMPLE

```
#include <tdrv014.h>

int            fd;
int            result;


result = ioctl(fd, TDRV014_IOCQ_GETDONESTATUS);

if (result < 0) {
   /* handle ioctl error */
} else {
   printf("DONE Status: %s\n",
               (result == TDRV014_DONESTATUS_HIGH) ? "HIGH" : "LOW");
}
```

3.3.5 TDRV014_IOCQ_GETPOWERSTATUS

NAME

TDRV014_IOCQ_GETPOWERSTATUS – Read current status of onboard power supplies

DESCRIPTION

This ioctl function returns the current status of the onboard power supplies. No additional parameter is used for this function, so the optional argument can be omitted.

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 ioctl function, the JTAG controller CPLD must be enabled.

EXAMPLE

```
#include <tdrv014.h>

int           fd;
int           result;

result = ioctl(fd, TDRV014_IOCQ_GETPOWERSTATUS);

if (result >= 0) {
  /* handle ioctl error */
} else {
  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");
}
```

ERRORS

EACCES	Access to JTAG controller CPLD is not enabled.

Other returned error codes are system error conditions.

3.3.6 TDRV014_IOCT_JTAGCHAINCONFIGSET

NAME

TDRV014_IOCT_JTAGCHAINCONFIGSET – Configure onboard JTAG chain

DESCRIPTION

This ioctl function configures the onboard JTAG chain. Specific JTAG chain segments can be disabled. An *unsigned char* value must be passed to the driver by the parameter *argp*.

Following values (binary OR'ed) are possible:

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

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

EXAMPLE

```
#include <tdrv014.h>

int           fd;
int           result;
unsigned char ChainCfg;

/* 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 = ioctl(fd, TDRV014_IOCT_JTAGCHAINCONFIGSET, ChainCfg);

if (result < 0) {
  /* handle ioctl error */
}
```

ERRORS

EACCES	Access to JTAG controller CPLD is not enabled.
EINVAL	Invalid parameter specified.

Other returned error codes are system error conditions.

3.3.7 TDRV014_IOCT_JTAGCHAINCONFIGGET

NAME

TDRV014_IOCT_JTAGCHAINCONFIGGET – Read current configuration of onboard JTAG chain

DESCRIPTION

This ioctl function returns the current configuration of the onboard JTAG chain. Specific JTAG chain segments can be disabled. No additional parameter is used for this function, so the optional argument can be omitted.

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 ioctl function, the JTAG controller CPLD must be enabled.

EXAMPLE

```
#include <tdrv014.h>

int            fd;
int            result;

result = ioctl(fd, TDRV014_IOCT_JTAGCHAINCONFIGGET);

if (result < 0) {
   /* handle ioctl error */
} else {
   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");
}
```

ERRORS

EACCES	Access to JTAG controller CPLD is not enabled.

Other returned error codes are system error conditions.

3.3.8 TDRV014_IOCT_SETFPGAMODE

NAME

TDRV014_IOCT_SETFPGAMODE – Configure FPGA configuration source

DESCRIPTION

This ioctl function configures the configuration source of the FPGA. An *unsigned char* value must be passed to the driver by the parameter *argp*.

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

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

EXAMPLE

```
#include <tdrv014.h>

int           fd;
int           result;
unsigned char FpgaMode;

/* configure FPGA to load from Platform Flash */
FpgaMode =  TDRV014_FPGAMODE_FLASH;
result = ioctl(fd, TDRV014_IOCT_SETFPGAMODE, FpgaMode);

if (result < 0) {
  /* handle ioctl error */
}
```

ERRORS

EACCES	Access to JTAG controller CPLD is not enabled.
EINVAL	Invalid parameter specified.

Other returned error codes are system error conditions.

3.3.9 TDRV014_IOCS_PLXWRITE

NAME

TDRV014_IOCS_PLXWRITE – Write 16bit value to PLX PCI9056 EEPROM.

DESCRIPTION

This ioctl function writes a 16bit value to a specific PLX PCI9056 EEPROM memory offset. A pointer to the caller's data buffer (*TDRV014_PLX_BUF*) is passed by the parameter *argp* to the driver.

```
typedef struct {
   unsigned long   Offset;
   unsigned short  Value;
} TDRV014_PLX_BUF;
```

Offset

Specifies the offset into the PLX PCI9056 EEPROM, where the supplied data word should be written. 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.

Value

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

Note that the PLX PCI9056 reloads the new configuration from the EEPROM after a PCI reset, i.e. the system must be rebooted to make PLX PCI9056 dependent changes take effect.

EXAMPLE

```
#include <tdrv014.h>

int              fd;
int              result;
int              BufferSize;
TDRV014_PLX_BUF  PlxBuf;

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

result = ioctl(fd, TDRV014_IOCS_PLXWRITE, &PlxBuf);

if (result < 0) {
  /* handle ioctl error */
}
```

ERRORS

EINVAL	The specified offset is invalid, or read-only
EBUSY	The device is busy with SVF action.
EFAULT	Error while copying data from user space.

Other returned error codes are system error conditions.

3.3.10 TDRV014_IOCG_PLXREAD

NAME

TDRV014_IOCG_PLXREAD – Read 16bit value from PLX PCI9056 EEPROM.

DESCRIPTION

This ioctl function reads a 16bit value from a specific PLX PCI9056 EEPROM memory offset. A pointer to the caller's data buffer (*TDRV014_PLX_BUF*) is passed by the parameter *argp* to the driver.

```
typedef struct {
   unsigned long   Offset;
   unsigned short  Value;
} TDRV014_PLX_BUF;
```

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.

Value

This value holds the retrieved 16bit word.

EXAMPLE

```
#include <tdrv014.h>

int              fd;
int              result;
int              BufferSize;
TDRV014_PLX_BUF  PlxBuf;

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

result = ioctl(fd, TDRV014_IOCG_PLXREAD, &PlxBuf);

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

ERRORS

EINVAL	The specified offset is invalid.
EBUSY	The device is busy with SVF action.
EFAULT	Error while copying data to user space.

Other returned error codes are system error conditions.

3.3.11 TDRV014_IOCG_READ_U8

NAME

TDRV014_IOCG_READ_U8 – Read 8bit values from FPGA resource.

DESCRIPTION

This ioctl function reads a number of 8bit values from a PCI Memory or PCI I/O area by using BYTE accesses. A pointer to the caller's data buffer (*TDRV014_MEMIO_BUF*) is passed by the parameter *argp* to the driver. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct {
   int            Resource;
   int            Offset;
   int            NumItems;
   unsigned char  pData[1];          /* dynamically expandable */
} TDRV014_MEMIO_BUF;
```

Resource

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	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 *Resource*.

NumItems

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 <tdrv014.h>

int                 fd;
int                 result;
unsigned long       BufferSize;
TDRV014_MEMIO_BUF   *pMemIoBuf;
unsigned char       *pValues;

/*
** read 50 bytes from MemorySpace 2, offset 0x00 (allocate enough memory)
*/
BufferSize       = ( sizeof(TDRV014_MEMIO_BUF) + 50*sizeof(unsigned char) );
pMemIoBuf        = (TDRV014_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->NumItems   = 50;
pMemIoBuf->Resource   = TDRV014_RES_MEM_2;
pMemIoBuf->Offset     = 0;

result = ioctl(fd, TDRV014_IOCG_READ_U8, pMemIoBuf);
if (result < 0) {
  /* handle ioctl error */
} else {
  pValues = (unsigned char*)pMemIoBuf->pData;
}
free( pMemIoBuf );
```

ERRORS

EINVAL	Specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with SVF action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

3.3.12 TDRV014_IOCG_READ_U16

NAME

TDRV014_IOCG_READ_U16 – Read 16bit values from FPGA resource.

DESCRIPTION

This ioctl function reads a number of 16bit values from a PCI Memory or PCI I/O area by using WORD accesses (big endian). A pointer to the caller's data buffer (*TDRV014_MEMIO_BUF*) is passed by the parameter *argp* to the driver. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct {
   int            Resource;
   int            Offset;
   int            NumItems;
   unsigned char  pData[1];          /* dynamically expandable */
} TDRV014_MEMIO_BUF;
```

Resource

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	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 *Resource*.

NumItems

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 <tdrv014.h>

int                 fd;
int                 result;
unsigned long       BufferSize;
TDRV014_MEMIO_BUF   *pMemIoBuf;
unsigned short      *pValues;

/*
** read 50 bytes from MemorySpace 2, offset 0x00 (allocate enough memory)
*/
BufferSize       = ( sizeof(TDRV014_MEMIO_BUF) + 50*sizeof(unsigned char) );
pMemIoBuf        = (TDRV014_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->NumItems   = 50;
pMemIoBuf->Resource   = TDRV014_RES_MEM_2;
pMemIoBuf->Offset     = 0;

result = ioctl(fd, TDRV014_IOCG_READ_U16, pMemIoBuf);
if (result < 0) {
  /* handle ioctl error */
} else {
  pValues = (unsigned short*)pMemIoBuf->pData;
}
free( pMemIoBuf );
```

ERRORS

EINVAL	Specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with SVF action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

3.3.13 TDRV014_IOCG_READ_U32

NAME

TDRV014_IOCG_READ_U32 – Read 32bit values from FPGA resource.

DESCRIPTION

This ioctl function reads a number of 32bit values from a PCI Memory or PCI I/O area by using DWORD accesses (big endian). A pointer to the caller's data buffer (*TDRV014_MEMIO_BUF*) is passed by the parameter *argp* to the driver. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct {
   int            Resource;
   int            Offset;
   int            NumItems;
   unsigned char  pData[1];          /* dynamically expandable */
} TDRV014_MEMIO_BUF;
```

Resource

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	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 *Resource*.

NumItems

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 <tdrv014.h>

int                 fd;
int                 result;
unsigned long       BufferSize;
TDRV014_MEMIO_BUF   *pMemIoBuf;
uint32_t            *pValues;

/*
** read 50 bytes from MemorySpace 2, offset 0x00 (allocate enough memory)
*/
BufferSize       = ( sizeof(TDRV014_MEMIO_BUF) + 50*sizeof(unsigned char) );
pMemIoBuf        = (TDRV014_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->NumItems   = 50;
pMemIoBuf->Resource   = TDRV014_RES_MEM_2;
pMemIoBuf->Offset     = 0;

result = ioctl(fd, TDRV014_IOCG_READ_U32, pMemIoBuf);
if (result < 0) {
  /* handle ioctl error */
} else {
  pValues = (uint32_t*)pMemIoBuf->pData;
}
free( pMemIoBuf );
```

ERRORS

EINVAL	Specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with SVF action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

3.3.14 TDRV014_IOCS_WRITE_U8

NAME

TDRV014_IOCS_WRITE_U8 – Read 8bit values from FPGA resource

DESCRIPTION

This ioctl function writes a number of *16bit* values to a PCI-Memory or PCI-I/O area by using BYTE accesses. A pointer to the caller's data buffer (*TDRV014_MEMIO_BUF*) is passed by the parameter *argp* to the driver. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct {
   int              Resource;
   unsigned long    Offset;
   unsigned long    NumItems;
   unsigned char    pData[1];              /* dynamically expandable */
} TDRV014_MEMIO_BUF;
```

Resource

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	TDRV014_RES_MEM_3

The PLX PCI9056 default configuration utilizes only BAR0 to BAR2.

Offset

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

NumItems

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 <tdrv014.h>

int                 fd;
int                 result;
unsigned long       BufferSize;
TDRV014_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(TDRV014_MEMIO_BUF) + 10*sizeof(unsigned char) );
pMemIoBuf        = (TDRV014_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->NumItems   = 10;
pMemIoBuf->Resource   = TDRV014_RES_MEM_2;
pMemIoBuf->Offset     = 0;
pValues               = (unsigned char*)pMemIoBuf->pData;
pValues[0]            = 0x01;
pValues[1]            = 0x02;
…

result = ioctl(fd, TDRV014_IOCS_WRITE_U8, pMemIoBuf);
if (result < 0) {
  /* handle ioctl error */
}
free( pMemIoBuf );
```

ERRORS

EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

3.3.15 TDRV014_IOCS_WRITE_U16

NAME

TDRV014_IOCS_WRITE_U16 – Write 16bit values to FPGA resource.

DESCRIPTION

This ioctl function writes a number of *16bit* values to a PCI-Memory or PCI-I/O area by using WORD accesses (big endian). A pointer to the caller's data buffer (*TDRV014_MEMIO_BUF*) is passed by the parameter *argp* to the driver. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct {
   int             Resource;
   unsigned long   Offset;
   unsigned long   NumItems;
   unsigned char   pData[1];          /* dynamically expandable */
} TDRV014_MEMIO_BUF;
```

Resource

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	TDRV014_RES_MEM_3

The PLX PCI9056 default configuration utilizes only BAR0 to BAR2.

Offset

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

NumItems

This value specifies the amount of data items to write.

pData

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

EXAMPLE

```
#include <tdrv014.h>

int                 fd;
int                 result;
unsigned long       BufferSize;
TDRV014_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(TDRV014_MEMIO_BUF) + 10*sizeof(unsigned short) );
pMemIoBuf      = (TDRV014_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->NumItems   = 10;
pMemIoBuf->Resource   = TDRV014_RES_MEM_2;
pMemIoBuf->Offset     = 0;
pValues               = (unsigned short*)pMemIoBuf->pData;
pValues[0]            = 0x0001;
pValues[1]            = 0x0002;
…

result = ioctl(fd, TDRV014_IOCS_WRITE_U16, pMemIoBuf);
if (result < 0) {
   /* handle ioctl error */
}
free( pMemIoBuf );
```

ERRORS

EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

3.3.16 TDRV014_IOCS_WRITE_U32

NAME

TDRV014_IOCS_WRITE_U32 – Write 32bit values to FPGA resource.

DESCRIPTION

This ioctl function writes a number of *32bit* values to a PCI-Memory or PCI-I/O area by using DWORD accesses (big endian). A pointer to the caller's data buffer (*TDRV014_MEMIO_BUF*) is passed by the parameter *argp* to the driver. This data buffer can be enlarged to the desired needs. The data section must be included inside this structure.

```
typedef struct {
   int              Resource;
   unsigned long    Offset;
   unsigned long    NumItems;
   unsigned char    pData[1];              /* dynamically expandable */
} TDRV014_MEMIO_BUF;
```

Resource

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	TDRV014_RES_MEM_3

The PLX PCI9056 default configuration utilizes only BAR0 to BAR2.

Offset

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

Size

This value specifies the amount of data items to write.

pData

The values are copied from this buffer. It must be large enough to hold the specified amount of data. The data pointer is typecasted into an *uint32_t* pointer.

EXAMPLE

```
#include <tdrv014.h>

int                 fd;
int                 result;
unsigned long       BufferSize;
TDRV014_MEMIO_BUF   *pMemIoBuf;
uint32_t            *pValues;

/*
** write 10 32bit dwords to MemorySpace 2, offset 0x00
** allocate enough memory to hold the data structure + write data
*/
BufferSize       = ( sizeof(TDRV014_MEMIO_BUF) + 10*sizeof(unsigned long) );
pMemIoBuf        = (TDRV014_MEMIO_BUF*)malloc( BufferSize );
pMemIoBuf->NumItems   = 10;
pMemIoBuf->Resource   = TDRV014_RES_MEM_2;
pMemIoBuf->Offset     = 0;
pValues               = (uint32_t*)pMemIoBuf->pData;
pValues[0]            = 0x00000001;
pValues[1]            = 0x00000002;
…

result = ioctl(fd, TDRV014_IOCS_WRITE_U32, pMemIoBuf);
if (result < 0) {
   /* handle ioctl error */
}
free( pMemIoBuf );
```

ERRORS

EINVAL	The specified Offset+Size exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

3.3.17 TDRV014_IOCT_WAIT_FOR_INT

NAME

TDRV014_IOCT_WAIT_FOR_INT – Wait for incoming Local Interrupt Source

DESCRIPTION

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

An *int* value is passed by the parameter *argp* to the driver. This value contains the timeout in milliseconds. 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 <tdrv014.h>

int    fd;
int    result;
int    Timeout;

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

result = ioctl(fd, TDRV014_IOCT_WAIT_FOR_INT, Timeout);

if (resul < 0) {
   /* acknowledge interrupt source in FPGA logic       */
   /* to clear the PLX PCI9056 Local Interrupt Source  */
} else {
   /* handle the error */
}
```

ERRORS

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

Other returned error codes are system error conditions.

4 API Documentation

4.1 General Functions

4.1.1 tdrv014open()

Name

tdrv014open() – opens a device.

Synopsis

```
int 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”
int FileDescriptor;

/*
** open file descriptor to device
*/
FileDescriptor = tdrv014open( “/dev/tdrv014_0” );
if (FileDescriptor < 0) {
  /* handle open error */
}
```

RETURNS

A device descriptor number, or -1 if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno.*

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
(
  int  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”
int FileDescriptor;
int result;

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

RETURNS

Zero, or -1 if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno*.

The 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           FileDescriptor;
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”

int           FileDescriptor;
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”

int           FileDescriptor;
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_fetchcommand_withinfo()

Name

svfparser_fetchcommand_withinfo() – Fetches SVF command from SVF file with additional information

Synopsis

```
SVFCMD* svfparser_fetchcommand_withinfo
(
  SVFHANDLE* handle
);
```

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

handle

This argument is a pointer to an SVFHANDLE structure.

```
typedef struct
{
   FILE*   file;
   int     cmdno;
   int     lineno;
} SVFHANDLE;
```

file

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

cmdno

This value returns the number of the current SVF command, relative to the beginning of the SVF file. This value may be useful for debugging programming problems or invalid SVF files.

lineno

This value returns the line number of the current SVF command in the SVF file. This value may be useful for debugging programming problems or invalid SVF files.

Example

```
#include “svf_parser.h”

SVFHANDLE*    handle;
SVFCMD*       pSvfCmd;

handle.file = svfparser_openfile(...);

/*
** fetch an SVF command from an SVF file
*/
pSvfCmd = svfparser_fetchcommand_withinfo( &handle );
if (!pSvfCmd) {
   /* handle error */
}
printf(“SVF Command #%d on line %d\n”, handle.cmdno, handle.lineno);
```

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.

4.2.5 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
(
  int           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”

int           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
(
  int           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”

int           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

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

4.3.3 tdrv014PlaySvf()

Name

tdrv014PlaySvf() – Play a complete SVF file

Synopsis

```
int tdrv014PlaySvf
(
  int           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”

int            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
(
  int              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”

int              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

EACCES	Access to JTAG controller CPLD is not enabled.
EINVAL	There was an error during SVF processing.
EINTR	The function was cancelled.
EFAULT	Error while copying data to or from user space.
EBUSY	The device is already busy with SVF action.
ENOMEM	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
(
  int           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”

int            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
(
  int           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”

int            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

EACCES	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
(
  int           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”

int            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

EACCES	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
(
  int           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”

int           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

EACCES	Access to JTAG controller CPLD is not enabled.
EINVAL	Invalid parameter specified.

Other returned error codes are system error conditions.

4.3.9 tdrv014SetFpgaMode()

Name

tdrv014SetFpgaMode() – Configure FPGA configuration source.

Synopsis

```
int tdrv014SetFpgaMode
(
  int           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”

int            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

EACCES	Access to JTAG controller CPLD is not enabled.
EINVAL	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
(
  int              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”

int              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

EINVAL	The specified offset is invalid.
EBUSY	The device is busy with SVF action.
EFAULT	Error while copying data to user space.

Other returned error codes are system error conditions.

4.3.11 tdrv014PlxEepromWrite()

Name

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

Synopsis

```
int tdrv014PlxEepromRead
(
  int              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”

int              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

EINVAL	The specified offset is invalid, or read-only
EBUSY	The device is busy with SVF action.
EFAULT	Error while copying data from user space.

Other returned error codes are system error conditions.

4.3.12 tdrv014Read8()

Name

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

Synopsis

```
int tdrv014Read8
(
  int              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	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*. This value is a byte offset.

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”

int              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

EINVAL	Specified Offset+NumItems exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with SVF action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

4.3.13 tdrv014Read16()

Name

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

Synopsis

```
int tdrv014Read8
(
  int              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.

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	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*. This value is a byte offset.

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”

int              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

EINVAL	Specified Offset+NumItems exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with SVF action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

4.3.14 tdrv014Read32()

Name

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

Synopsis

```
int tdrv014Read8
(
  int         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.

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	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*. This value is a byte offset.

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”

int         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

EINVAL	Specified Offset+NumItems exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with SVF action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

4.3.15 tdrv014Write8()

Name

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

Synopsis

```
int tdrv014Write8
(
  int              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	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*. This value is a byte offset.

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”

int              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

EINVAL	The specified Offset+NumItems exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

4.3.16 tdrv014Write16()

Name

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

Synopsis

```
int tdrv014Write8
(
  int              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.

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	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*. This value is a byte offset.

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”

int              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

EINVAL	The specified Offset+NumItems exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

4.3.17 tdrv014Write32()

Name

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

Synopsis

```
int tdrv014Write8
(
  int         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.

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	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*. This value is a byte offset.

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”

int         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

EINVAL	The specified Offset+NumItems exceeds the available memory or I/O space.
EACCES	The specified Resource is not available for access.
EBUSY	The device is already busy with XSVF, Reconfig or SPI action.
EFAULT	Error while copying data to user space.
ENOMEM	Error getting enough internal memory for data.

Other returned error codes are system error conditions.

4.3.18 tdrv014InterruptWait()

Name

tdrv014InterruptWait() – Wait for incoming Local Interrupt Source.

Synopsis

```
int tdrv014InterruptWait
(
  int    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.

Example

```
#include “tdrv014api.h”

int            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

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

Other returned error codes are system error conditions.

5 Diagnostic

If the TDRV014 does not work properly it is helpful to get some status information from the driver respective kernel.

The Linux */proc* file system provides information about kernel, resources, driver, devices, and so on. The following screen dumps displays information of a correct running TDRV014 driver (see also the proc man pages).

```
# lspci -v
  …
04:02.0 Signal processing controller: TEWS Technologies GmbH Device 2277
(rev ba)
        Subsystem: TEWS Technologies GmbH Device 2014
        Flags: bus master, 66MHz, medium devsel, latency 64, IRQ 17
        Memory at febffc00 (32-bit, non-prefetchable) [size=512]
        I/O ports at e800 [size=256]
        Memory at fea00000 (32-bit, non-prefetchable) [size=1M]
        Memory at fe900000 (32-bit, non-prefetchable) [size=1M]
        Capabilities: [40] Power Management version 2
        Capabilities: [48] CompactPCI hot-swap <?>
        Capabilities: [4c] Vital Product Data <?>
        Kernel driver in use: TEWS TECHNOLOGIES - TDRV014 Device Driver
        Kernel modules: tdrv014drv
  …
```

```
# cat /proc/devices
Character devices:
  1 mem
  2 pty
  …
248 tdrv014drv
  …
```

```
# cat /proc/ioports
  …
e000-efff : PCI Bus 0000:04
  e800-e8ff : 0000:04:02.0
    e800-e8ff : TDRV014
  …
```

```
# cat /proc/iomem
00000000-0009f7ff : System RAM
  …
fe900000-febfffff : PCI Bus 0000:04
  fe900000-fe9fffff : 0000:04:02.0
    fe900000-fe9fffff : TDRV014
  fea00000-feaffffff : 0000:04:02.0
    fea00000-feaffffff : TDRV014
  febffc00-febffdff : 0000:04:02.0
    febffc00-febffdff : TDRV014
  …
```