The Embedded I/O Company

TDRV018-SW-65

Windows Device Driver

Reconfigurable FPGA

Version 1.0.x

User Manual

Issue 1.0.0

December 2022

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

TDRV018-SW-65

Windows Device Driver

Reconfigurable FPGA

Supported Modules:
TPMC634
TXMC633
TXMC635
TXMC637
TXMC638
TPCE636

Issue	Description	Date
1.0.0	First Issue	December 6, 2022

Table of Contents

1 Introduction

The TDRV018-SW-65 Windows device driver is a kernel mode driver which allows the operation of the supported hardware module on an Intel or Intel-compatible Windows 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 TDRV018-SW-65 device driver supports the following features:

- Program onboard SPI Flash, initiate FPGA reconfiguration
- Program onboard FPGA directly
- Read/write FPGA registers (32bit / 16bit / 8bit)
- Read/write specific PCI Configuration EEPROM registers
- Wait for interrupts
- Register Callback functions for interrupt handling
- Driver functions (except for SPI/FPGA programming) are thread-safe, as long as unique handles are used.

The TDRV018-SW-65 device driver supports the modules listed below:

TPMC634	Reconfigurable FPGA	PMC
TXMC633	Reconfigurable FPGA with 64 TTL I/O / 32 Differential I/O Lines	XMC
TXMC635	Reconfigurable FPGA with 48 x TTL IO / 32 x 16 Bit Analog In / 8 x 16 Bit Analog Out	XMC
TXMC637	Reconfigurable FPGA with 16 x 16 bit Analog Input 8 x 16 bit Analog Output and 32 digital I/O	XMC
TXMC638	Reconfigurable FPGA with 24 x 16 Bit Analog Input	XMC
TPCE636	Reconfigurable FPGA with 16 x 16 bit Analog Input and 16 x 16 bit Analog Output	PCIe

In this document all supported modules and devices will be called TDRV018. 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.

TPMC634 (or compatible) User Manual
TXMC633 (or compatible) User Manual
TXMC635 (or compatible) User Manual
TXMC637 (or compatible) User Manual
TXMC638 (or compatible) User Manual
TPCE636 (or compatible) User Manual

2 Installation

Following files are located in directory TDRV018-SW-65 on the distribution media:

driver\	Directory containing driver files
tdrv018.h	Header file with IOCTL codes and structure definitions
example\tdrv018exa.c	Example application
installer_32bit.exe	Installation tool for 32bit systems
installer_64bit.exe	Installation tool for 64bit systems
dpinst.xml	Installation XML file
TDRV018-SW-65-1.0.0.pdf	This document
Release.txt	Information about the Device Driver Release
ChangeLog.txt	Release history

2.1 Software Installation

2.1.1 Windows 10

This section describes how to install the TDRV018-SW-65 Device Driver on a Windows 10 (32bit or 64bit) operating system.

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

Copy needed files (tdrv018.h) to desired target directory.

After successful installation a device is created for each module found (TDRV018_1, TDRV018_2 ...).

2.2 Confirming Windows Driver Installation

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

1. Open the Windows Device Manager:
    a. For Windows XP, open the "***Control Panel***" from "***My Computer***" and click the "***System***" icon and choose the "***Hardware***" tab, and then click the "***Device Manager***" button.
    b. For Windows 7, open the "***Control Panel***" from "***My Computer***" and then click the "***Device Manager***" entry.
2. Click the "**+**" in front of "***Embedded I/O***".
   The driver (e.g. "***TPMC634 FPGA/BCC Device (TDRV018)***") should appear for each installed device.

3 API Documentation

3.1 General Functions

3.1.1 tdrv018Open

NAME

tdrv018Open – open a device.

SYNOPSIS

```
TDRV018_HANDLE tdrv018Open
(
    char        *DeviceName
)
```

DESCRIPTION

Before I/O can be performed to a device, a device 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. The first TDRV018 device is named “\\\\.\\TDRV018_1” the second device is named “\\\\.\\TDRV018_2” and so on.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE     hdl;

/*
** open the specified device
*/
hdl = tdrv018Open(“\\\\.\\TDRV018_1”);
if (hdl == NULL)
{
     /* handle open error */
}
```

RETURNS

A device handle, or NULL 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.

3.1.2 tdrv018Close

NAME

tdrv018Close – close a device.

SYNOPSIS

```
TDRV018_STATUS tdrv018Close
(
    TDRV018_HANDLE      hdl
)
```

DESCRIPTION

This function closes a previously opened device.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;

/*
** close the device
*/
result = tdrv018Close(hdl);
if (result != TDRV018_OK)
{
     /* handle close error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.3 tdrv018GetPciInfo

NAME

tdrv018GetPciInfo – get information of the module PCI header

SYNOPSIS

```
TDRV018_STATUS tdrv018GetPciInfo
(
        TDRV018_HANDLE              hdl,
        TDRV018_PCIINFO_BUF         *pPciInfoBuf
)
```

DESCRIPTION

This function returns information of the module PCI header in the provided data buffer.

PARAMETERS

hdl

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

pPciInfoBuf

This argument is a pointer to the structure TDRV018_PCIINFO_BUF that receives information of the module PCI header.

```
typedef struct
{
        unsigned short      vendorId;
        unsigned short      deviceId;
        unsigned short      subSystemId;
        unsigned short      subSystemVendorId;
        int                 pciBusNo;
        int                 pciDevNo;
        int                 pciFuncNo;
} TDRV018_PCIINFO_BUF;
```

vendorId

PCI module vendor ID.

deviceId

PCI module device ID

subSystemId

PCI module sub system ID

subSystemVendorId

PCI module sub system vendor ID

pciBusNo

Number of the PCI bus, where the module resides.

pciDevNo

PCI device number

pciFuncNo

PCI function number

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE           hdl;
TDRV018_STATUS           result;
TDRV018_PCIINFO_BUF      pciInfoBuf

/*
** get module PCI information
*/
result = tdrv018GetPciInfo(hdl, &pciInfoBuf);

if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.4 tdrv018GetBoardInfo

NAME

tdrv018GetBoardInfo – get information of the board

SYNOPSIS

```
TDRV018_STATUS tdrv018GetBoardInfo
(
        TDRV018_HANDLE              hdl,
        unsigned int                *pFpgaStatus,
        unsigned int                *pDipSwitch,
        unsigned int                *pFirmwareVersion
)
```

DESCRIPTION

This function returns information about the board status. This function is only supported for TPMC634 (or compatible) devices.

PARAMETERS

hdl

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

pFpgaStatus

This argument returns the current FPGA Status. The following values are possible:

Value	Description
TDRV018_FPGASTAT_DONE	DONE Signal State
TDRV018_FPGASTAT_INIT	INIT Signal State

pDipSwitch

This argument returns the value of the onboard 4-bit DIP switch. Bit 0 corresponds to DIP switch position 1, bit 1 corresponds to DIP switch position 2 and so on.

pFirmwareVersion

This argument returns the firmware version of the PCI target device.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE          hdl;
TDRV018_STATUS          result;
unsigned int            FpgaStatus;
unsigned int            DipSwitch;
unsigned int            FWVersion;

/*
** get board information
*/
result = tdrv018GetBoardInfo( hdl, &FpgaStatus, &DipSwitch, &FWVersion );

if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.1.5 tdrv018GetBoardName

NAME

tdrv018GetBoardName – get Name of the board

SYNOPSIS

```
TDRV018_STATUS tdrv018GetBoardName
(
    TDRV018_HANDLE          hdl,
    unsigned char           *pBoardName,
    int                     len
)
```

DESCRIPTION

This function returns the name of the board to distinguish between the supported hardware modules and devices.

PARAMETERS

hdl

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

pBoardName

This argument returns a null-terminated ASCII string describing the specific hardware module. The following boarding naming scheme is implemented:

Value	**Description**
“TPMC634”	TPMC634 Device
“TXMC633-BCC”	TXMC633 BCC Device
“TXMC633-FPGA”	TXMC633 User-FPGA Device

len

This argument specifies the maximum length available for storing the board name.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE          hdl;
TDRV018_STATUS          result;
char                    BoardName[40];

/*
** get board name
*/
result = tdrv018GetBoardName( hdl, &BoardName, 40 );

if (result == TDRV018_OK)
{
     printf(“Board Name: %s\n”, BoardName);
} else {
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.6 tdrv018GetBoardSerial

NAME

tdrv018GetBoardSerial – get Serial Number of the board

SYNOPSIS

```
TDRV018_STATUS tdrv018GetBoardSerial
(
    TDRV018_HANDLE              hdl,
    unsigned int                *pSerialNumber
)
```

DESCRIPTION

This function returns the serial number of the board. This function is only supported by BCC devices.

PARAMETERS

hdl

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

pSerialNumber

This argument returns the board serial number as a decimal number.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE          hdl;
TDRV018_STATUS          result;
unsigned int            SerialNumber;

/*
** get board serial number
*/
result = tdrv018GetBoardSerial( hdl, &SerialNumber );

if (result == TDRV018_OK)
{
     printf(“Board Serial Number: %d\n”, SerialNumber);
} else {
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.1.7 tdrv018GetBoardFWVersion

NAME

tdrv018GetBoardFWVersion – get Firmware Version of the board

SYNOPSIS

```
TDRV018_STATUS tdrv018GetBoardFWVersion
(
      TDRV018_HANDLE              hdl,
      unsigned int                *pFWVersion
)
```

DESCRIPTION

This function returns the Firmware Version of the board. This function is not supported by User-FPGA devices.

PARAMETERS

hdl

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

pFWVersion

This argument returns the Firmware Version of the device. For details, please refer to the corresponding hardware user manual.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE         hdl;
TDRV018_STATUS         result;
unsigned int           FWVersion;

/*
** get board serial number
*/
result = tdrv018GetBoardFWVersion( hdl, &FWVersion );

if (result != TDRV018_OK)
{
    /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.1.8 tdrv018GetFpgaStatus

NAME

tdrv018GetFpgaStatus – get status of the User-FPGA

SYNOPSIS

```
TDRV018_STATUS tdrv018GetFpgaStatus
(
        TDRV018_HANDLE                  hdl,
        unsigned int                    *pFpgaStatus
)
```

DESCRIPTION

This function returns the configuration status of the User-FPGA. This function is not supported by User-FPGA devices.

PARAMETERS

hdl

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

pFpgaStatus

This argument returns the current FPGA Status. The following values are possible:

Value	**Description**
TDRV018_FPGASTAT_DONE	DONE Signal State
TDRV018_FPGASTAT_INIT	INIT Signal State

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE          hdl;
TDRV018_STATUS          result;
unsigned int            FpgaStatus;

/*
** get FPGA Status
*/
result = tdrv018GetFpgaStatus( hdl, &FpgaStatus );

if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.2 Local Bus Control Functions

The Local Bus Control Functions are only supported for TPMC634 (or compatible) devices.

3.2.1 tdrv018LocalBusControl

NAME

tdrv018LocalBusControl – Configure and control the Local Bus

SYNOPSIS

```
TDRV018_STATUS tdrv018LocalBusControl
(
        TDRV018_HANDLE          hdl,
        unsigned int            ControlFlags
)
```

DESCRIPTION

This function controls the Local Bus Interface.

PARAMETERS

hdl

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

ControlFlags

This value specifies a 32bit unsigned int value, which contains the combined Local Bus Control Flags. The following OR'ed values are possible:

Value	**Description**
TDRV018_LOCALBUS_RESET	If set, the Local Bus Reset signal is asserted.
TDRV018_LOCALBUS_TOUT_DIS	If set, the Local Bus Timeout is disabled.
TDRV018_LOCALBUS_PLLRESET	If set, the Local Bus PLL is held in reset mode.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;

/*
** Perform a Local Bus Reset
*/
/* assert Local Bus Reset */
result = tdrv018LocalBusControl( hdl, TDRV018_LOCALBUS_RESET );
if (result != TDRV018_OK)
{
     /* handle error */
}

... wait some time ...

/* de-assert Local Bus Reset */
result = tdrv018LocalBusControl( hdl, 0 );
if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_INVAL	The specified flags are invalid.
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.2.2 tdrv018LocalBusStatus

NAME

tdrv018LocalBusStatus – Read Status of the Local Bus

SYNOPSIS

```
TDRV018_STATUS tdrv018LocalBusStatus
(
        TDRV018_HANDLE          hdl,
        unsigned int            *pStatus
)
```

DESCRIPTION

This function reads the status of the Local Bus Interface. The Event flags are cleared after execution of this function.

PARAMETERS

hdl

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

pStatus

This argument returns the Local Bus Status and Event Flags. The following OR'ed values are possible:

Value	**Description**
TDRV018_LOCALBUS_PLLLOCKED	The PLL is currently locked.
TDRV018_LOCALBUS_RESETACTIVE	The Local Bus Reset is currently active.
TDRV018_LOCALBUS_MASTERABORT	A Master Abort has been detected.
TDRV018_LOCALBUS_TARGETERROR	A Target Error has been detected.
TDRV018_LOCALBUS_PLLLOSSOFLOCK	A Loss-of-Lock has been detected.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;
Unsigned int      Status;

/*
** Check the Local Bus Status
*/
result = tdrv018LocalBusStatus( hdl, &Status );
if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.3 EEPROM Access Functions

The EEPROM Access Functions are only supported for TPMC634 (or compatible) devices.

3.3.1 tdrv018EepromWrite

NAME

tdrv018EepromWrite – Write 16bit value to PCI Configuration EEPROM

SYNOPSIS

```
TDRV018_STATUS tdrv018EepromWrite
(
      TDRV018_HANDLE        hdl,
      unsigned int          Offset,
      unsigned short        Value
)
```

DESCRIPTION

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

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

PARAMETERS

hdl

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

Offset

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

Offset	Access
00h – 01h	R
02h – 7Fh	R / W

Refer to the TPMC634 User Manual for detailed information on these registers.

Value

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

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;
unsigned int      Offset;
unsigned short    Value;

/*
** Change the Subsystem Vendor ID to TEWS TECHNOLOGIES (0x1498)
*/
Offset = 0x05;
Value  = 0x1498;

result = tdrv018EepromWrite( hdl, Offset, Value );
if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_INVAL	The specified offset is invalid, or read-only.
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.3.2 tdrv018EepromRead

NAME

tdrv018EepromRead – Read 16bit value from PCI Configuration EEPROM

SYNOPSIS

```
TDRV018_STATUS tdrv018EepromRead
(
        TDRV018_HANDLE          hdl,
        unsigned int            Offset,
        unsigned short          *pValue
)
```

DESCRIPTION

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

PARAMETERS

hdl

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

Offset

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

Offset	Access
00h – 01h	R
02h – 7Fh	R / W

Refer to the TPMC634 User Manual for detailed information on these registers.

pValue

This value is a pointer to an *unsigned short* value, which receives the 16bit word.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;
unsigned int      Offset;
unsigned short    Value;


/*
** Read Subsystem ID
*/
Offset = 0x04;

result = tdrv018EepromRead( hdl, Offset, &Value );
if (result == TDRV018_OK)
{
     printf( “Value = 0x%04X\n”, Value );
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_INVAL	The specified offset is invalid.
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.4 Pull-Resistor Configuration Functions

The Pull-Resistor Configuration Functions are only supported for TXMC633 and TXMC635 (or compatible) BCC devices.

3.4.1 tdrv018PullConfigSet

NAME

tdrv018PullConfigSet – Configure the pull-resistor settings

SYNOPSIS

```
TDRV018_STATUS tdrv018PullConfigSet
(
        TDRV018_HANDLE      hdl,
        int                 PullControl,
        unsigned int        PullConfig
)
```

DESCRIPTION

This function writes the configuration value into the Pull Resistor Configuration Register of the selected device.

PARAMETERS

hdl

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

PullControl

This parameter specifies if the Pull-Resistor Configuration shall be controlled by the User-FPGA or by the PullConfig value. The following values are possible:

Value	Description
TDRV018_PULLCNT_USERFPGA	Pull-Configuration is controlled by the User-FPGA
TDRV018_PULLCNT_BCC	Pull-Configuration is controlled by the BCC and the PullConfig value.

PullConfig

This value specifies a 32bit unsigned int value, which directly corresponds to the Pull Resistor Configuration Register of the supported hardware module. For details, please refer to the corresponding hardware user-manual.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;

/*
** Set the pull-resistor configuration
*/
result = tdrv018PullConfigSet( hdl, 0x000000AA );
if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.4.2 tdrv018PullConfigGet

NAME

tdrv018PullConfigGet – Read the current pull-resistor configuration

SYNOPSIS

```
TDRV018_STATUS tdrv018PullConfigGet
(
        TDRV018_HANDLE          hdl,
        int                     *pPullControl,
        unsigned int            *pPullConfig
)
```

DESCRIPTION

This function reads the configuration value from the Pull Resistor Configuration Register of the selected device.

PARAMETERS

hdl

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

pPullControl

This parameter specifies if the Pull-Resistor Configuration is controlled by the User-FPGA or by the PullConfig value. The following values are possible:

Value	**Description**
TDRV018_PULLCNT_USERFPGA	Pull-Configuration is controlled by the User-FPGA
TDRV018_PULLCNT_BCC	Pull-Configuration is controlled by the BCC and the PullConfig value.

pPullConfig

This value specifies a pointer to a 32bit unsigned int value, which receives the content of the Pull Resistor Configuration Register of the supported hardware module. For details, please refer to the corresponding hardware user-manual.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;
int               PullControl;
unsigned int      PullConfig;

/*
** Read the pull-resistor configuration
*/
result = tdrv018PullConfigGet( hdl, &PullControl, &PullConfig );
if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.5 FPGA Programming Functions

3.5.1 tdrv018ProgSpiFlashFromFile

NAME

tdrv018ProgSpiFlashFromFile – Program the SPI Flash from a supplied bitstream file

SYNOPSIS

```
TDRV018_STATUS tdrv018ProgSpiFlashFromFile
(
        TDRV018_HANDLE          hdl,
        char                    *pFilename
)
```

DESCRIPTION

This function writes the content of a supplied binary configuration bitstream file (.bin) into the onboard SPI configuration flash. The FPGA is not affected by this function. To initiate the FPGA configuration, use the function tdrv018ConfigFromSpiFlash().

Only one programming function may be executed at once.

PARAMETERS

hdl

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

pFilename

This value is a pointer to a null-terminated character string, specifying the binary configuration bitstream file name (.bin).

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;

/*
** Program the SPI Configuration Flash from a file
*/
result = tdrv018ProgSpiFlashFromFile( hdl, “tpmc634fpga.bin” );
if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_IO	Error during SPI transfer

3.5.2 tdrv018ProgSpiFlash

NAME

tdrv018ProgSpiFlash – Program the SPI Flash

SYNOPSIS

```
TDRV018_STATUS tdrv018ProgSpiFlash
(
        TDRV018_HANDLE       hdl,
        unsigned char        *pData,
        unsigned int         numBytes
)
```

DESCRIPTION

This function writes supplied configuration data into the onboard SPI configuration flash. The FPGA is not affected by this function. To initiate the FPGA configuration, use the function tdrv018ConfigFromSpiFlash().

Only one programming function may be executed at once.

PARAMETERS

hdl

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

pData

This value is a pointer to the configuration data, residing in memory.

numBytes

This value specifies the number of bytes to be written into the SPI flash.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;
unsigned char     *pData;
unsigned int      numBytes;

/*
** Program the SPI Configuration Flash from a memory location
*/

/* allocate and fill the memory area */
numBytes = ...;
pData = (unsigned char*)malloc( numBytes );
...

result = tdrv018ProgSpiFlash( hdl, pData, numBytes );
if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_IO	Error during SPI transfer

3.5.3 tdrv018ProgFpgaFromFile

NAME

tdrv018ProgFpgaFromFile – Program the FPGA from a supplied bitstream file using Select Map

SYNOPSIS

```
TDRV018_STATUS tdrv018ProgFpgaFromFile
(
        TDRV018_HANDLE        hdl,
        char                  *pFilename
        int                   timeout
)
```

DESCRIPTION

This function writes the content of a supplied configuration bitstream file directly into the FPGA using Select Map. The FPGA is not functional during the programming process.

The PCI Setup of a directly attached User-FPGA is lost after reconfiguration, so the device is not accessible anymore. Assuming no changes regarding the PCI setup (number and size of base address registers), the PCI header can be restored using API function tdrv018RestorePciHeader.

Only one programming function may be executed at once.

PARAMETERS

hdl

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

pFilename

This value is a pointer to a null-terminated character string, specifying the configuration bitstream file name.

timeout

This value specifies the timeout in milliseconds the function will wait for the FPGA loading to finish, i.e. the DONE signal switches to HIGH. The granularity depends on the operating system. To wait indefinitely, specify -1 as timeout parameter.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;

/*
** Directly program the FPGA from a file, wait up to 1 second for DONE
*/
result = tdrv018ProgFpgaFromFile( hdl, “tpmc634fpga.bin”, 1000 );
if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_IO	Error during FPGA Configuration. DONE remained LOW.

3.5.4 tdrv018ProgFpga

NAME

tdrv018ProgFpga – Program the FPGA using Select Map

SYNOPSIS

```
TDRV018_STATUS tdrv018ProgFpga
(
      TDRV018_HANDLE        hdl,
      unsigned char         *pData,
      unsigned int          numBytes,
      int                   timeout
)
```

DESCRIPTION

This function writes the content of a supplied configuration bitstream file directly into the FPGA using Select Map. The FPGA is not functional during the programming process.

The PCI Setup of a directly attached User-FPGA is lost after reconfiguration, so the device is not accessible anymore. Assuming no changes regarding the PCI setup (number and size of base address registers), the PCI header can be restored using API function tdrv018RestorePciHeader.

Only one programming function may be executed at once.

PARAMETERS

hdl

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

pFilename

This value is a pointer to a null-terminated character string, specifying the configuration bitstream file name.

pData

This value is a pointer to the configuration data, residing in memory.

numBytes

This value specifies the number of bytes to be written into the FPGA.

timeout

This value specifies the timeout in milliseconds the function will wait for the FPGA loading to finish, i.e. the DONE signal switches to HIGH. The granularity depends on the operating system. To wait indefinitely, specify -1 as timeout parameter.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;
unsigned char     *pData;
unsigned int      numBytes;

/*
** Directly program the FPGA, wait up to 1 second for DONE
*/
/* allocate and fill the memory area */
numBytes = ...;
pData = (unsigned char*)malloc( numBytes );
...

result = tdrv018ProgFpga( hdl, pData, numBytes, 1000 );
if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_IO	Error during FPGA Configuration. DONE remained LOW.

3.5.5 tdrv018ConfigFromSpiFlash

NAME

tdrv018ConfigFromSpiFlash – Start loading the FPGA from the SPI Flash

SYNOPSIS

```
TDRV018_STATUS tdrv018ConfigFromSpiFlash
(
    TDRV018_HANDLE      hdl,
    int                 timeout
)
```

DESCRIPTION

This function starts loading the FPGA bitstream from the SPI Flash into the FPGA.

The PCI Setup of a directly attached User-FPGA is lost after reconfiguration, so the device is not accessible anymore. Assuming no changes regarding the PCI setup (number and size of base address registers), the PCI header can be restored using API function tdrv018RestorePciHeader.

PARAMETERS

hdl

This argument specifies the device handle 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 FPGA loading to finish, i.e. the DONE signal switches to HIGH. The granularity depends on the operating system. To wait indefinitely, specify -1 as timeout parameter.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;

/*
** Load the FPGA from the SPI Flash, wait up to 1 second for DONE
*/
result = tdrv018ConfigFromSpiFlash( hdl, 1000 );
if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_IO	Error during FPGA Configuration. DONE remained LOW.

3.5.6 tdrv018PcieLinkEnable

NAME

tdrv018PcieLinkEnable – Enable PCIe Link for the User-FPGA

SYNOPSIS

```
TDRV018_STATUS tdrv018PcieLinkEnable
(
    TDRV018_HANDLE      hdl
)
```

DESCRIPTION

This function gracefully enables the PCIe Link for the User-FPGA. This function is only supported by BCC devices. Reconfiguring a User-FPGA might result in PCIe errors, which might cause unpredictable behavior of the overall system.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;

/*
** Enable the PCIe Link of the User-FPGA
*/
result = tdrv018PcieLinkEnable( hdl );
if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.5.7 tdrv018PcieLinkDisable

NAME

tdrv018PcieLinkDisable – Disable PCIe Link for the User-FPGA

SYNOPSIS

```
TDRV018_STATUS tdrv018PcieLinkDisable
(
    TDRV018_HANDLE      hdl
)
```

DESCRIPTION

This function gracefully disables the PCIe Link for the User-FPGA. This function is only supported by BCC devices. Reconfiguring a User-FPGA might result in PCIe errors, which might cause unpredictable behavior of the overall system. So before starting a configuration which might affect the PCIe Link State of the User-FPGA, disabling the PCIe link is suggested.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;

/*
** Disable the PCIe Link of the User-FPGA
*/
result = tdrv018PcieLinkDisable( hdl );
if (result != TDRV018_OK)
{
    /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.5.8 tdrv018RestorePciHeader

NAME

tdrv018RestorePciHeader – Restore the PCI header of the User-FPGA

SYNOPSIS

```
TDRV018_STATUS tdrv018RestorePciHeader
(
    TDRV018_HANDLE        hdl
)
```

DESCRIPTION

This function restores the PCI header of the User-FPGA using values stored upon driver start. After reconfiguration of a User-FPGA, the PCI header configuration is lost. To allow further accesses to the User-FPGA, the PCI header must be restored using this function. This function is only supported by User-FPGA devices.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE    hdl;
TDRV018_STATUS    result;

/*
** Restore the PCI header of the User-FPGA
*/
result = tdrv018RestorePciHeader( hdl );
if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	**Description**
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.6 Register Access Functions

3.6.1 tdrv018Read8

NAME

tdrv018Read8 – read 8-bit values from PCI BAR space

SYNOPSIS

```
TDRV018_STATUS tdrv018Read8
(
        TDRV018_HANDLE        hdl,
        int                   pciResource,
        int                   offset,
        int                   numItems,
        unsigned char         *pData
)
```

DESCRIPTION

This function reads the specified number of items from the PCI BAR space by using single byte (8-bit) accesses.

PARAMETERS

hdl

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

pciResource

This parameter specifies the desired PCI Memory resource to be used for this access. In general, a PCI target (PCIe bridge) supports up to six base address registers. Following values are possible:

Value	**Description**
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

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

PCI Base Address Register	PCI Address-Type	TDRV018 Resource
0	MEM	TDRV018_RES_MEM_1
1	MEM *(not used)*	TDRV018_RES_MEM_2
2	MEM *(not used)*	TDRV018_RES_MEM_3

offset

This argument specifies the start offset within the PCI BAR space.

numItems

This argument specifies the number of items (8-bit) to read.

pData

This argument is a pointer to an unsigned char buffer which will be filled with the specified number of items from the PCI BAR space. The allocated space must be large enough to hold the specified amount of data.

EXAMPLE

```
#include “tdrv018api.h”

#define NUM   256

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
int                offset;
unsigned char      dataBuf[NUM];


offset = 0x0000;
/*
** read 256 Bytes from the User FPGA Register Area
*/
result = tdrv018Read8(hdl, TDRV018_RES_MEM_1, offset, NUM, dataBuf);

if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_INVAL	The specified access range exceeds PCI BAR limits
TDRV018_ERR_ACCESS	The specified PCI resource is not available

3.6.2 tdrv018ReadBE16

NAME

tdrv018ReadBE16 – read 16-bit values from PCI BAR space in big-endian order

SYNOPSIS

```
TDRV018_STATUS tdrv018ReadBE16
(
        TDRV018_HANDLE          hdl,
        int                     pciResource,
        int                     offset,
        int                     numItems,
        unsigned short          *pData
)
```

DESCRIPTION

This function reads the specified number of items from the PCI BAR space by using 16-bit accesses. The values are returned as big-endian values that mean on Intel x86 architectures the multi-byte data will be byte-swapped.

PARAMETERS

hdl

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

pciResource

This parameter specifies the desired PCI Memory resource to be used for this access. In general, a PCI target (PCIe bridge) supports up to six base address registers. Following values are possible:

Value	**Description**
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

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

PCI Base Address Register	PCI Address-Type	TDRV018 Resource
0	MEM	TDRV018_RES_MEM_1
1	MEM *(not used)*	TDRV018_RES_MEM_2
2	MEM *(not used)*	TDRV018_RES_MEM_3

offset

This argument specifies the start offset within the PCI BAR space.

numItems

This argument specifies the number of items (16-bit) to read.

pData

This argument is a pointer to an unsigned short buffer which will be filled with the specified number of items from the PCI BAR space. The allocated space must be large enough to hold the specified amount of data.

EXAMPLE

```
#include “tdrv018api.h”

#define NUM   128

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
int                offset;
unsigned short     dataBuf[NUM];

offset = 0x0000;
/*
** read 256 Bytes from the User FPGA Register Area
*/
result = tdrv018ReadBE16(hdl, TDRV018_RES_MEM_1, offset, NUM, dataBuf);

if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_INVAL	The specified access range exceeds PCI BAR limits
TDRV018_ERR_ACCESS	The specified PCI resource is not available

3.6.3 tdrv018ReadLE16

NAME

tdrv018ReadLE16 – read 16-bit values from PCI BAR space in little-endian order

SYNOPSIS

```
TDRV018_STATUS tdrv018ReadLE16
(
        TDRV018_HANDLE          hdl,
        int                     pciResource,
        int                     offset,
        int                     numItems,
        unsigned short          *pData
)
```

DESCRIPTION

This function reads the specified number of items from the PCI BAR space by using 16-bit accesses. The values are returned as little-endian values that means on Intel x86 architectures the multi-byte data will not be byte-swapped.

PARAMETERS

hdl

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

pciResource

This parameter specifies the desired PCI Memory resource to be used for this access. In general, a PCI target (PCIe bridge) supports up to six base address registers. Following values are possible:

Value	**Description**
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

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

PCI Base Address Register	PCI Address-Type	TDRV018 Resource
0	MEM	TDRV018_RES_MEM_1
1	MEM *(not used)*	TDRV018_RES_MEM_2
2	MEM *(not used)*	TDRV018_RES_MEM_3

offset

This argument specifies the start offset within the PCI BAR space.

numItems

This argument specifies the number of items (16-bit) to read.

pData

This argument is a pointer to an unsigned short buffer which will be filled with the specified number of items from the PCI BAR space. The allocated space must be large enough to hold the specified amount of data.

EXAMPLE

```
#include “tdrv018api.h”

#define NUM   128

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
int                offset;
unsigned short     dataBuf[NUM];

offset = 0x0000;
/*
** read 256 Bytes from the User FPGA Register Area
*/
result = tdrv018ReadLE16(hdl, TDRV018_RES_MEM_1, offset, NUM, dataBuf);

if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_INVAL	The specified access range exceeds PCI BAR limits
TDRV018_ERR_ACCESS	The specified PCI resource is not available

3.6.4 tdrv018ReadBE32

NAME

tdrv018ReadBE32 – read 32-bit values from PCI BAR space in big-endian order

SYNOPSIS

```
TDRV018_STATUS tdrv018ReadBE32
(
        TDRV018_HANDLE          hdl,
        int                     pciResource,
        int                     offset,
        int                     numItems,
        unsigned int            *pData
)
```

DESCRIPTION

This function reads the specified number of items from the PCI BAR space by using 32-bit accesses. The values are returned as big-endian values that means on Intel x86 architectures the multi-byte data will be byte-swapped.

PARAMETERS

hdl

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

pciResource

This parameter specifies the desired PCI Memory resource to be used for this access. In general, a PCI target (PCIe bridge) supports up to six base address registers. Following values are possible:

Value	**Description**
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

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

PCI Base Address Register	**PCI Address-Type**	**TDRV018 Resource**
0	MEM	TDRV018_RES_MEM_1
1	MEM *(not used)*	TDRV018_RES_MEM_2
2	MEM *(not used)*	TDRV018_RES_MEM_3

offset

This argument specifies the start offset within the PCI BAR space.

numItems

This argument specifies the number of items (32-bit) to read.

pData

This argument is a pointer to an unsigned int buffer which will be filled with the specified number of items from the PCI BAR space. The allocated space must be large enough to hold the specified amount of data.

EXAMPLE

```
#include “tdrv018api.h”

#define NUM   1

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
int                offset;
unsigned int       dataBuf[NUM];

offset = 0;
/*
** read Digital Input Register of FPGA Example Design
*/
result = tdrv018ReadBE32(hdl, TDRV018_RES_MEM_1, offset, NUM, dataBuf);

if (result != TDRV018_OK)
{
    /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_INVAL	The specified access range exceeds PCI BAR limits
TDRV018_ERR_ACCESS	The specified PCI resource is not available

3.6.5 tdrv018ReadLE32

NAME

tdrv018ReadLE32 – read 32-bit values from PCI BAR space in little-endian order

SYNOPSIS

```
TDRV018_STATUS tdrv018ReadLE32
(
        TDRV018_HANDLE         hdl,
        int                    pciResource,
        int                    offset,
        int                    numItems,
        unsigned int           *pData
)
```

DESCRIPTION

This function reads the specified number of items from the PCI BAR space by using 32-bit accesses. The values are returned as little-endian values that means on Intel x86 architectures the multi-byte data will not be byte-swapped.

PARAMETERS

hdl

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

pciResource

This parameter specifies the desired PCI Memory resource to be used for this access. In general, a PCI target (PCIe bridge) supports up to six base address registers. Following values are possible:

Value	**Description**
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

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

PCI Base Address Register	PCI Address-Type	TDRV018 Resource
0	MEM	TDRV018_RES_MEM_1
1	MEM *(not used)*	TDRV018_RES_MEM_2
2	MEM *(not used)*	TDRV018_RES_MEM_3

offset

This argument specifies the start offset within the PCI BAR space.

numItems

This argument specifies the number of items (32-bit) to read.

pData

This argument is a pointer to an unsigned int buffer which will be filled with the specified number of items from the PCI BAR space. The allocated space must be large enough to hold the specified amount of data.

EXAMPLE

```
#include “tdrv018api.h”

#define NUM   1

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
int                offset;
unsigned int       dataBuf[NUM];

offset = 0;
/*
** read Digital Input Register of FPGA Example Design
*/
result = tdrv018ReadLE32(hdl, TDRV018_RES_MEM_1, offset, NUM, dataBuf);

if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_INVAL	The specified access range exceeds PCI BAR limits
TDRV018_ERR_ACCESS	The specified PCI resource is not available

3.6.6 tdrv018Write8

NAME

tdrv018Write8 – write 8-bit values to the PCI BAR space

SYNOPSIS

```
TDRV018_STATUS tdrv018Write8
(
        TDRV018_HANDLE          hdl,
        int                     pciResource,
        int                     offset,
        int                     numItems,
        unsigned char           *pData
)
```

DESCRIPTION

This function writes the specified number of items to the PCI BAR space by using single byte (8-bit) accesses.

PARAMETERS

hdl

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

pciResource

This parameter specifies the desired PCI Memory resource to be used for this access. In general, a PCI target (PCIe bridge) supports up to six base address registers. Following values are possible:

Value	**Description**
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

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

PCI Base Address Register	PCI Address-Type	TDRV018 Resource
0	MEM	TDRV018_RES_MEM_1
1	MEM *(not used)*	TDRV018_RES_MEM_2
2	MEM *(not used)*	TDRV018_RES_MEM_3

offset

This argument specifies the start offset within the PCI BAR space.

numItems

This argument specifies the number of items (8-bit) to write.

pData

This argument is a pointer to an unsigned char buffer with the data items to write. The allocated space must be large enough to hold the specified amount of data.

EXAMPLE

```
#include “tdrv018api.h”

#define NUM   4

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
int                offset;
unsigned char      dataBuf[NUM];

dataBuf[0] = 0xAA;
dataBuf[1] = 0x55;
…

offset = 0x00;

/*
** write 4 bytes to a 32bit Scratchpad Register
*/
result = tdrv018Write8(hdl, TDRV018_RES_MEM_1, offset, NUM, dataBuf);

if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_INVAL	The specified access range exceeds PCI BAR limits
TDRV018_ERR_ACCESS	The specified PCI resource is not available

3.6.7 tdrv018WriteBE16

NAME

tdrv018WriteBE16 – write 16-bit values to the PCI BAR space big-endian order

SYNOPSIS

```
TDRV018_STATUS tdrv018WriteBE16
(
        TDRV018_HANDLE          hdl,
        int                     pciResource,
        int                     offset,
        int                     numItems,
        unsigned short          *pData
)
```

DESCRIPTION

This function writes the specified number of items to the PCI BAR space by using 16-bit accesses.

The values are written in big-endian order that means on Intel x86 architectures the multi-byte data will be byte-swapped.

The register sets of FPGA on-chip bus slave devices and DRAM memory areas can be accessed via addressable data regions in PCI BAR space.

PARAMETERS

hdl

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

pciResource

This parameter specifies the desired PCI Memory resource to be used for this access. In general, a PCI target (PCIe bridge) supports up to six base address registers. Following values are possible:

Value	Description
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

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

PCI Base Address Register	PCI Address-Type	TDRV018 Resource
0	MEM	TDRV018_RES_MEM_1
1	MEM *(not used)*	TDRV018_RES_MEM_2
2	MEM *(not used)*	TDRV018_RES_MEM_3

offset

This argument specifies the start offset within the PCI BAR space.

numItems

This argument specifies the number of items (16-bit) to write.

pData

This argument is a pointer to an unsigned short buffer with the data items to write. The allocated space must be large enough to hold the specified amount of data.

EXAMPLE

```
#include “tdrv018api.h”

#define NUM   0x8000

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
int                offset;
unsigned short     dataBuf[NUM];

dataBuf[0] = 0xAA55;
dataBuf[1] = 0x55AA;
…

offset = 0x30000;
/*
** write 64KB to the DDRA memory page
*/
result = tdrv018WriteBE16(hdl, TDRV018_RES_MEM_1, offset, NUM, dataBuf);

if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_INVAL	The specified access range exceeds PCI BAR limits
TDRV018_ERR_ACCESS	The specified PCI resource is not available

3.6.8 tdrv018WriteLE16

NAME

tdrv018WriteLE16 – write 16-bit values to the PCI BAR space in little-endian order

SYNOPSIS

```
TDRV018_STATUS tdrv018WriteLE16
(
        TDRV018_HANDLE          hdl,
        int                     pciResource,
        int                     offset,
        int                     numItems,
        unsigned short          *pData
)
```

DESCRIPTION

This function writes the specified number of items to the PCI BAR space by using 16-bit accesses.

The values are written in little-endian order that means on Intel x86 architectures the multi-byte data will not be byte-swapped.

The register sets of FPGA on-chip bus slave devices and DRAM memory areas can be accessed via addressable data regions in PCI BAR space.

PARAMETERS

hdl

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

pciResource

This parameter specifies the desired PCI Memory resource to be used for this access. In general, a PCI target (PCIe bridge) supports up to six base address registers. Following values are possible:

Value	Description
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

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

PCI Base Address Register	PCI Address-Type	TDRV018 Resource
0	MEM	TDRV018_RES_MEM_1
1	MEM *(not used)*	TDRV018_RES_MEM_2
2	MEM *(not used)*	TDRV018_RES_MEM_3

offset

This argument specifies the start offset within the PCI BAR space.

numItems

This argument specifies the number of items (16-bit) to write.

pData

This argument is a pointer to an unsigned short buffer with the data items to write. The allocated space must be large enough to hold the specified amount of data.

EXAMPLE

```
#include “tdrv018api.h”

#define NUM   0x8000

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
int                offset;
unsigned short     dataBuf[NUM];

dataBuf[0] = 0xAA55;
dataBuf[1] = 0x55AA;
…

offset = 0x30000;
/*
** write 64KB to the DDRA memory page
*/
result = tdrv018WriteLE16(hdl, TDRV018_RES_MEM_1, offset, NUM, dataBuf);

if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_INVAL	The specified access range exceeds PCI BAR limits
TDRV018_ERR_ACCESS	The specified PCI resource is not available

3.6.9 tdrv018WriteBE32

NAME

tdrv018WriteBE32 – write 32-bit values to the PCI BAR space big-endian order

SYNOPSIS

```
TDRV018_STATUS tdrv018WriteBE32
(
        TDRV018_HANDLE          hdl,
        int                     pciResource,
        int                     offset,
        int                     numItems,
        unsigned int            *pData
)
```

DESCRIPTION

This function writes the specified number of items to the PCI BAR space by using 32-bit accesses.

The values are written in big-endian order that means on Intel x86 architectures the multi-byte data will be byte-swapped.

The register sets of FPGA on-chip bus slave devices and DRAM memory areas can be accessed via addressable data regions in PCI BAR space.

PARAMETERS

hdl

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

pciResource

This parameter specifies the desired PCI Memory resource to be used for this access. In general, a PCI target (PCIe bridge) supports up to six base address registers. Following values are possible:

Value	Description
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

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

PCI Base Address Register	PCI Address-Type	TDRV018 Resource
0	MEM	TDRV018_RES_MEM_1
1	MEM *(not used)*	TDRV018_RES_MEM_2
2	MEM *(not used)*	TDRV018_RES_MEM_3

offset

This argument specifies the start offset within the PCI BAR space.

numItems

This argument specifies the number of items (32-bit) to write.

pData

This argument is a pointer to an unsigned int buffer with the data items to write. The allocated space must be large enough to hold the specified amount of data.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
int                offset;
unsigned int       data;

data      = 0x10020000;      /* PLB DDRA address space */
offset    = 0x20004;         /* memory controller device register */
/*
** adjust the selected memory page from DDRA to 0x10020000 by
** setting the DDRA memory address register in the memory controller
** device
*/
result = tdrv018WriteBE32(hdl, TDRV018_RES_MEM_1, offset, 1, &data);

if (result != TDRV018_OK)
{
    /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_INVAL	The specified access range exceeds PCI BAR limits
TDRV018_ERR_ACCESS	The specified PCI resource is not available

3.6.10 tdrv018WriteLE32

NAME

tdrv018WriteLE32 – write 32-bit values to the PCI BAR space in little-endian order

SYNOPSIS

```
TDRV018_STATUS tdrv018WriteLE32
(
        TDRV018_HANDLE          hdl,
        int                     pciResource,
        int                     offset,
        int                     numItems,
        unsigned int            *pData
)
```

DESCRIPTION

This function writes the specified number of items to the PCI BAR space by using 32-bit accesses.

The values are written in little-endian order that means on Intel x86 architectures the multi-byte data will not be byte-swapped.

The register sets of FPGA on-chip bus slave devices and DRAM memory areas can be accessed via addressable data regions in PCI BAR space.

PARAMETERS

hdl

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

pciResource

This parameter specifies the desired PCI Memory resource to be used for this access. In general, a PCI target (PCIe bridge) supports up to six base address registers. Following values are possible:

Value	**Description**
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

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

PCI Base Address Register	PCI Address-Type	TDRV018 Resource
0	MEM	TDRV018_RES_MEM_1
1	MEM *(not used)*	TDRV018_RES_MEM_2
2	MEM *(not used)*	TDRV018_RES_MEM_3

offset

This argument specifies the start offset within the PCI BAR space.

numItems

This argument specifies the number of items (32-bit) to write.

pData

This argument is a pointer to an unsigned int buffer with the data items to write. The allocated space must be large enough to hold the specified amount of data.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
int                offset;
unsigned int       data;

data      = 0x10020000;      /* PLB DDRA address space */
offset    = 0x20004;         /* memory controller device register */
/*
** adjust the selected memory page from DDRA to 0x10020000 by
** setting the DDRA memory address register in the memory controller
** device
*/
result = tdrv018WriteLE32(hdl, TDRV018_RES_MEM_1, offset, 1, &data);

if (result != TDRV018_OK)
{
    /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	**Description**
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_INVAL	The specified access range exceeds PCI BAR limits
TDRV018_ERR_ACCESS	The specified PCI resource is not available

3.7 Resource Mapping Functions

3.7.1 tdrv018PciResourceMap

NAME

tdrv018PciResourceMap – map a PCI resource directly into the process context

SYNOPSIS

```
TDRV018_STATUS tdrv018PciResourceMap
(
        TDRV018_HANDLE          hdl,
        int                     pciResource,
        unsigned char           **pPtr,
        unsigned int            *pSize
)
```

DESCRIPTION

This function maps the specified PCI resource of the hardware module directly into the process context. The retrieved pointer can be used for direct non-cached register access.

PARAMETERS

hdl

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

pciResource

This parameter specifies the desired PCI Memory resource to be used for this access. In general, a PCI target (PCIe bridge) supports up to six base address registers. Following values are possible:

Value	Description
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

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

PCI Base Address Register	PCI Address-Type	TDRV018 Resource
0	MEM	TDRV018_RES_MEM_1
1	MEM *(not used)*	TDRV018_RES_MEM_2
2	MEM *(not used)*	TDRV018_RES_MEM_3

pPtr

This argument is a pointer to an unsigned char pointer that receives the start address of the mapped PCI resource.

pSize

This argument returns the size of the mapped PCI resource in bytes.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
unsigned char      *pReg;
unsigned int       size;

/*
** map first memory PCI resource
*/
result = tdrv018PciResourceMap(hdl, TDRV018_RES_MEM_1, &pReg, &size);

if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_ACCESS	Specified PCI resource not available
TDRV018_ERR_NOMEM	Unable to allocate memory

3.7.2 tdrv018PciResourceUnmap

NAME

tdrv018PciResourceUnmap – unmap a previously mapped PCI resource

SYNOPSIS

```
TDRV018_STATUS tdrv018PciResourceUnmap
(
        TDRV018_HANDLE          hdl,
        unsigned char           *pPtr
)
```

DESCRIPTION

This function unmaps a previously mapped PCI resource, freeing the system resources used for this mapping.

PARAMETERS

hdl

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

pPtr

This argument is a pointer to an unsigned char pointer that represents the start address of the previously mapped PCI resource. This pointer must have been received from the corresponding mapping function.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
unsigned char      *pReg;

/*
** unmap a previously mapped PCI resource
*/
result = tdrv018PciResourceUnmap(hdl, pReg);

if (result != TDRV018_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV018_ERR_INVAL	Invalid pointer specified

3.8 Interrupt Functions

3.8.1 tdrv018InterruptConfig

NAME

tdrv018InterruptConfig – Configure the Interrupt handling method for User-FPGA implementations

SYNOPSIS

```
TDRV018_STATUS tdrv018InterruptConfig
(
        TDRV018_HANDLE          hdl,
        int                     ControlType,
        int                     ReadPciResource,
        int                     ReadAccessWidth,
        unsigned int            ReadOffset,
        unsigned int            ReadMask,
        int                     WritePciResource,
        int                     WriteAccessWidth,
        unsigned int            WriteOffset,
        unsigned int            WriteMask,
        unsigned int            WriteValue
);
```

DESCRIPTION

This function configures the interrupt handling method for User-FPGA specific implementations. Interrupt handling must be implemented on driver-level, so the driver must be configured properly to acknowledge an interrupt of the user-specific FPGA implementation.

Make sure to configure the interrupt handling before the User-FPGA implementation raises interrupts.

PARAMETERS

hdl

This value specifies the callback handle retrieved by a call to the corresponding register-function.

IntAckMethod

This value specifies the interrupt acknowledgement method. Following values are possible:

Value	Description
TDRV018_INTACK_READ	Interrupt is cleared upon reading a status register.
TDRV018_INTACK_READCLEAR	Interrupt is cleared by writing the read register bits into the same register, using the same access width as for the read access.
TDRV018_INTACK_READWRITE	Interrupt is cleared upon writing a static value to a specific register.
TDRV018_INTACK_READWRITEMASK	Interrupt is cleared upon writing a static value to a specific register using a bit-mask, leaving specified original register bits unchanged.

ReadPciResource

This parameter specifies the desired PCI Memory resource to be used for reading the interrupt status. In general, a PCI target supports up to six base address registers. Following values are possible:

Value	Description
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

ReadAccessWidth

This parameter specifies the desired access width for reading the interrupt status. The driver always uses little-endian accesses. Following values are possible:

Value	Description
TDRV018_ACCESSWIDTH_8	A BYTE (8bit) register access is used.
TDRV018_ACCESSWIDTH_16	A WORD (16bit) register access is used.
TDRV018_ACCESSWIDTH_32	A DWORD (32bit) register access is used.

ReadOffset

This argument specifies the register offset within the PCI BAR space used for reading the interrupt status.

ReadMask

This argument specifies the bit-mask used for interrupt detection. This argument can be used to mask-out static register bits for proper support of PCI interrupt sharing.

WritePciResource

This parameter specifies the desired PCI Memory resource to be used for writing a static value. In general, a PCI target supports up to six base address registers. This parameter is not used for IntAckMethod READ and READCLEAR. Following values are possible:

Value	**Description**
TDRV018_RES_MEM_1	First found PCI Memory area.
TDRV018_RES_MEM_2	Second found PCI Memory area.
TDRV018_RES_MEM_3	Third found PCI Memory area.
TDRV018_RES_MEM_4	Fourth found PCI Memory area.
TDRV018_RES_MEM_5	Fifth found PCI Memory area.
TDRV018_RES_MEM_6	Sixth found PCI Memory area.

WriteAccessWidth

This parameter specifies the desired access width for writing the static value. The driver always uses little-endian accesses. This parameter is not used for IntAckMethod READ and READCLEAR. Following values are possible:

Value	**Description**
TDRV018_ACCESSWIDTH_8	A BYTE (8bit) register access is used.
TDRV018_ACCESSWIDTH_16	A WORD (16bit) register access is used.
TDRV018_ACCESSWIDTH_32	A DWORD (32bit) register access is used.

WriteOffset

This argument specifies the register offset within the PCI BAR space used for writing the static value. This parameter is not used for IntAckMethod READ and READCLEAR.

WriteMask

This argument specifies the bit-mask used for write access. This argument can be used to mask-out static register bits, changing only the desired ones. Specifying 0x00000000 is not valid. This parameter is not used for IntAckMethod READ and READCLEAR.

WriteValue

This argument specifies the static value to be used for writing. This parameter is not used for IntAckMethod READ and READCLEAR.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE          hdl;
TDRV018_STATUS          result;

/*
**  Example 1:
**  Configure the Interrupt Handler to use READCLEAR method
**  - InterruptStatus Register in first PCI MemRes, at Offset 0x20
**  - Use 32bit accesses
*/
result = tdrv018InterruptConfig( hdl,
              TDRV018_INTACK_READCLEAR,
              TDRV018_RES_MEM_1,            // ReadPciResource
              TDRV018_ACCESSWIDTH_32,       // ReadAccessWidth
              0x20,                         // ReadOffset
              0xFFFFFFFF,                   // check all register-bits
              0, 0, 0, 0, 0                 // do not use write-
                                            // parameters
          );
if (result == TDRV018_OK)
{
     / *OK */
} else {
     /* handle error */
}


…
```

```
/*
**  Example 2:
**  Configure the Interrupt Handler to use READWRITE method
**  - InterruptStatus Register at PCI Memory Resource 1, Offset 0x20
**  - Use 32bit accesses, check all register-bits
**  - disable the Interrupt by writing 0x00000000 to PCI Memory Resource 1,
**        Offset 0x24
*/
result = tdrv018InterruptConfig( hdl,
              TDRV018_INTACK_READWRITE,
              TDRV018_RES_MEM_1,           // ReadPciResource
              TDRV018_ACCESSWIDTH_32,      // ReadAccessWidth
              0x20,                        // ReadOffset
              0xFFFFFFFF,                  // ReadMask
              TDRV018_RES_MEM_1,           // WritePciResource
              TDRV018_ACCESSWIDTH_32,      // WriteAccessWidth
              0x24,                        // WriteOffset
              0xFFFFFFFF,                  // WriteMask
              0x00000000                   // WriteValue
         );
if (result == TDRV018_OK)
{
     / *OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified handle is invalid.
TDRV018_ERR_INVAL	The specified flags are invalid.
TDRV018_ERR_NOSYS	This function is not supported by the device.

3.8.2 tdrv018InterruptWait

NAME

tdrv018InterruptWait – Wait for incoming Local Interrupt Source

SYNOPSIS

```
TDRV018_STATUS tdrv018InterruptWait
(
    TDRV018_HANDLE      hdl,
    unsigned int        interruptMask,
    unsigned int        *pInterruptOccurred,
    int                 timeout
);
```

DESCRIPTION

This function enables the specified local interrupt sources, and waits for interrupts on the specified local interrupt sources. After an interrupt has arrived, the corresponding occurred local interrupt source is disabled inside the Infrastructure Module (IM). Multiple functions may wait for the same interrupt source to occur.

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

PARAMETERS

hdl

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

interruptMask

This parameter specifies specific interrupt bits to wait for. The interrupt bits correspond to the Infrastructure Module's "Interrupt Pending Register" bits described in the FDK user manual. Please refer to the hardware user manual for further information on the possible interrupt bits. The function returns if at least one of the specified interrupt sources is detected.

pInterruptOccurred

If at least one of the specified interrupt sources occurs, the value is returned through this pointer. The interrupt bits correspond to the Infrastructure Module's "Interrupt Pending Register" bits described in the FDK user manual. Please refer to the hardware user manual for further information on the possible interrupt bits.

timeout

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

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE     hdl;
TDRV018_STATUS     result;
unsigned int       interruptMask;
unsigned int       interruptOccurred;

/*
** Wait at least 5 seconds for incoming interrupts on UINTP0 and/or UINTP1
*/
interruptMask = (1 << 17) | (1 << 16);
result = tdrv018InterruptWait(   hdl,
                                 interruptMask,
                                 &interruptOccurred,
                                 5000 );
if (result == TDRV018_OK)
{
     /* Interrupt arrived.                           */
     /* Now acknowledge interrupt source in FPGA logic   */
     /* to clear the Local Interrupt Source.             */
     /* Use tdrv018Read and tdrv018Write functions for   */
     /* register access.                                 */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_TIMEOUT	The specified timeout occurred.

3.8.3 tdrv018InterruptRegisterCallbackThread

NAME

tdrv018InterruptRegisterCallbackThread – Register a User Callback Function for Interrupt Handling

SYNOPSIS

```
TDRV018_STATUS tdrv018InterruptRegisterCallbackThread
(
        TDRV018_HANDLE      hdl,
        int                 threadPriority,
        int                 stackSize,
        unsigned int        interruptMask,
        FUNCINTCALLBACK     callbackFunction,
        void                *funcparam,
        TDRV018_HANDLE      *pCallbackHandle
)
```

DESCRIPTION

This function registers a user callback function which is executed after detection of the specified interrupt source. It is possible to register multiple callback functions to one or a set (bit mask) of interrupt sources.

The callback function is executed in a thread context, so using TDRV018 device driver functions and system functions is allowed. The callback function should be kept as short as possible. The specified callback function is executed with the occurred interrupt bits and the specified function parameter as function arguments. Additionally, a status value is passed to the callback function, which reflects the result of the involved API functions.

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

PARAMETERS

hdl

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

threadPriority

This parameter specifies the priority to be used for the callback thread. Possible values are:

Value	**Description**
TDRV018_PRIORITY_NORMAL	Normal Priority (THREAD_PRIORITY_NORMAL)
TDRV018_PRIORITY_HIGH	High Priority (THREAD_PRIORITY_HIGHEST)
TDRV018_PRIORITY_LOW	Low Priority (THREAD_PRIORITY_LOWEST)

Other values might be possible (see also Windows SDK Documentation, *SetThreadPriority*).

stackSize

This parameter specifies the stack size to be used for the callback tread. The value is specified in bytes.

interruptMask

This parameter specifies specific interrupt bits to wait for. The interrupt bits correspond to the Infrastructure Module's "Interrupt Pending Register" bits described in the FDK user manual. Please refer to the hardware user manual for further information on the possible interrupt bits. The callback function is executed if at least one of the specified interrupt sources occurred.

callbackFunction

This parameter is a function pointer to the user callback function. The callback function pointer is defined as follows:

```
typedef void(*FUNCINTCALLBACK)( TDRV018_HANDLE   hdl,
                                unsigned int     interruptOccurred,
                                void             *param,
                                TDRV018_STATUS   status );
```

hdl

This parameter specifies a device handle which can be used for hardware access or other API functions by the callback function.

interruptOccurred

This parameter is a 32bit value reflecting the occurred interrupts. It is useful if the callback function handles multiple interrupt sources. The interrupt bits correspond to the Infrastructure Module's "Interrupt Pending Register" bits described in the FDK user manual. Please refer to the hardware user manual for further information on the possible interrupt bits.

param

This parameter is the user-specified *funcparam* value (see below) which has been specified on callback registration. This value can be used to pass a pointer to a specific control structure, to supply the callback function with specific information.

status

This parameter hands over interrupt callback status information. The callback function needs to check this parameter. If the specified interrupt source has occurred properly, and no errors were detected, this parameter is TDRV018_OK. If this parameter differs from TDRV018_OK, an internal error has been detected and the callback handling is stopped. The callback function must implement an appropriate error handling.

funcparam

This value specifies a user parameter, which will be handed over to the callback function on execution. This parameter can be used to pass a pointer to a specific control structure used by the callback function.

pCallbackHandle

This value specifies a pointer to a handle, where the callback handle will be returned. This callback handle must be used to unregister a callback function.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE      hdl;
TDRV018_STATUS      result;
unsigned int        interruptMask;
USER_DATA_AREA      userDataArea;
TDRV018_HANDLE      callbackHandle;

/* forward declaration of callback functions */
void callback_TIMER0(   TDRV018_HANDLE      hdl,
                        unsigned int        interruptOccurred,
                        void                *param,
                        TDRV018_STATUS      status);
/*
**  Register callback function for TIMER0 (UINTP0)
**  Use a “normal” priority, and 64KB stack.
*/
interruptMask = (1 << 16);
result = tdrv018InterruptRegisterCallbackThread(hdl,
                                          TDRV018_PRIORITY_NORMAL,
                                          0x10000,
                                          interruptMask,
                                          callback_TIMER0,
                                          &userDataArea,
                                          &callbackHandle);
...
```

```
...
if (result != TDRV018_OK)
{
     /* handle error */
}

/*
** Initialize and start the Timer function, using register accesses.
** Refer to the FDK documentation for register description.
*/
...
/*
** Callback Function, using API Functions for Register Access
*/
void callback_TIMER0(   TDRV018_HANDLE     hdl,
                        unsigned int       interruptOccurred,
                        void               *param,
                        TDRV018_STATUS     status)
{
     TDRV018_STATUS     result;
     USER_DATA_AREA     *pUsrData = (USER_DATA_AREA*)param;
     unsigned int       u32value;

     if (status != TDRV018_OK)
     {
          /* handle error status */
     }

     printf("[Timer 0 Interrupt]\n);

     /* Acknowledge TIMER0 interrupt source by writing to
     ** "Timer Based Interrupt Status Register" (offset may differ). */
     u32value = (1 << 0);
     result = tdrv018WriteBE32(   hdl,
                                  TDRV018_RES_MEM_1,
                                  0x1002C,
                                  1,
                                  &u32value );
     /* handle errors */
     return;
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified TDRV018_HANDLE is invalid.
TDRV018_ERR_INVAL	Function or callback handle pointer is NULL.
TDRV018_ERR_TASK_CREATE	Creation of the callback thread (task) failed.

3.8.4 tdrv018InterruptUnregisterCallback

NAME

tdrv018InterruptUnregisterCallback – Unregister a User Callback Function

SYNOPSIS

```
TDRV018_STATUS tdrv018InterruptUnregisterCallback
(
    TDRV018_HANDLE        hdl
)
```

DESCRIPTION

This function unregisters a previously registered user callback thread or ISR function.

PARAMETERS

hdl

This value specifies the callback handle retrieved by a call to the corresponding register-function.

EXAMPLE

```
#include “tdrv018api.h”

TDRV018_HANDLE     callbackHdl;
TDRV018_STATUS     result;

/*
** Unregister a callback function
*/
result = tdrv018InterruptUnregisterCallback(callbackHdl);

if (result == TDRV018_OK)
{
     / *OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV018_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV018_ERR_INVALID_HANDLE	The specified callback handle is invalid.

3.9 Endian Conversion Functions

The following conversion functions can be used to develop endian-neutral software, especially for direct access to mapped PCI resources.

3.9.1 endian_be16

NAME

endian_be16 – big-endian conversion function

SYNOPSIS

```
unsigned short endian_be16
(
    unsigned short          u16value
)
```

DESCRIPTION

This function converts a short integer value (16-bit) from the native CPU endian order to big-endian order. That means on Intel x86 architectures the value will be byte-swapped, as opposed to PowerPC architectures.

PARAMETERS

u16value

This argument specifies the data to convert

EXAMPLE

```
#include "tdrv018api.h"

unsigned short *pRawData, bigEndianData;

/* setup pRawData pointer to the correct location first */

bigEndianData = endian_be16(*pRawData);
```

RETURN VALUE

This function returns the passed value in the big-endian order.

3.9.2 endian_le16

NAME

endian_le16 – little-endian conversion function

SYNOPSIS

```
unsigned short endian_le16
(
        unsigned short              u16value
)
```

DESCRIPTION

This function converts a short integer value (16-bit) from the native CPU endian order to little-endian order. That means on PowerPC architectures the value will be byte-swapped, as opposed to Intel x86 architectures.

PARAMETERS

u16value

This argument specifies the data to convert

EXAMPLE

```
#include “tdrv018api.h”

unsigned short *pRawData, littleEndianData;

/* setup pRawData pointer to the correct location first */

littleEndianData = endian_le16(*pRawData);
```

RETURN VALUE

This function returns the passed value in the little-endian order.

3.9.3 endian_be32

NAME

endian_be32 – big-endian conversion function

SYNOPSIS

```
unsigned int endian_be32
(
        unsigned short                u32value
)
```

DESCRIPTION

This function converts an integer value (32-bit) from the native CPU endian order to big-endian order. That means on Intel x86 architectures the value will be byte-swapped, as opposed to PowerPC architectures.

PARAMETERS

u32value

This argument specifies the data to convert

EXAMPLE

```
#include “tdrv018api.h”

unsigned short *pRawData, bigEndianData;

/* setup pRawData pointer to the correct location first */

bigEndianData = endian_be32(*pRawData);
```

RETURN VALUE

This function returns the passed value in the big-endian order.

3.9.4 endian_le32

NAME

endian_le32 – little-endian conversion function

SYNOPSIS

```
unsigned short endian_le32
(
    unsigned short          u32value
)
```

DESCRIPTION

This function converts an integer value (32-bit) from the native CPU endian order to little-endian order. That means on PowerPC architectures the value will be byte-swapped, as opposed to Intel x86 architectures.

PARAMETERS

u32value

This argument specifies the data to convert

EXAMPLE

```
#include “tdrv018api.h”

unsigned short *pRawData, littleEndianData;

/* setup pRawData pointer to the correct location first */

littleEndianData = endian_le32(*pRawData);
```

RETURN VALUE

This function returns the passed value in the little-endian order.