The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV019-SW-95

QNX - Neutrino Device Driver

ADC, DAC and Digital I/O

Version 1.0.x

User Manual

Issue 1.0.0

October 2018

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

TDRV019-SW-95

QNX - Neutrino Device Driver

ADC, DAC and Digital I/O

Supported Modules:
TPMC532
TPMC533
TPMC542

Issue	Description	Date
1.0.0	First Issue	October 1, 2018

Table of Contents

1 Introduction

The TDRV019-SW-95 QNX-Neutrino device driver allows the operation of the supported devices on QNX-Neutrino operating systems.

The TDRV019 device driver is basically implemented as a user installable Resource Manager. The standard file (I/O) functions (open, close and devctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

The TDRV019-SW-95 device driver supports the following features:

- Configuration of ADC and DAC
- Read Single ADC Samples
- Write Single DAC Samples
- Read ADC Samples using DMA
- Write DAC Samples using DMA
- Configure Frame and Data Synchronization Signals
- Configure and use digital I/O signals
- Driver functions are thread-safe as long as unique handles are used.

The TDRV019-SW-95 device driver supports the modules listed below:

TPMC532	16x/8x ADC, 8x/4x DAC and 14x Digital I/O	PMC
TPMC533	32x ADC, 16x/0x DAC and 8x Digital I/O	PMC
TPMC542	16x/8x DAC and 20x Digital I/O	PMC

In this document all supported modules and devices will be called TDRV019. Specials for certain devices will be advised.

To get more information about the features and use of TDRV019 devices it is recommended to read the manuals listed below.

Corresponding Hardware User Manual

2 Installation

Following files are located in the directory TDRV019-SW-95 on the distribution media:

TDRV019-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TDRV019-SW-95-1.0.0.pdf	This manual in PDF format
ChangeLog.txt	Release history
Release.txt	Information about the Device Driver Release

The GZIP compressed archive TDRV019-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tdrv019':

/driver/tdrv019.c	Driver source code
/driver/tdrv019.h	Definitions and data structures for driver and application
/driver/tdrv019def.h	Device driver include
/driver/node.c	Queue management source code
/driver/node.h	Queue management definitions
/driver/nto/*	Driver Build path
/api/example.c	API Library
/api/nto/*	API Library Build path
/example/tdrv019exa.c	Example application
/example/nto/*	Example application Build path

2.1 Building Executables on Native Systems

For installation copy the tar-archive into the /usr/src directory and unpack it (e.g. `tar –xzvf TDRV019-SW-95-SRC.tar.gz`). After that the necessary directory structure for the automatic build and the source files are available underneath the new directory called *tdrv019*.

It is absolutely important to extract the TDRV019 tar archive in the /usr/src directory. Otherwise the automatic build with make will fail.

2.1.1 Build the Device Driver

Change to the /usr/src/tdrv019/driver directory

Execute the Makefile:

```
# make install
```

After successful completion the driver binary (tdrv019) will be installed in the /bin directory.

2.1.2 Build the API Library

Change to the /usr/src/tdrv019/api directory

Execute the Makefile:

```
# make install
```

After successful completion the API Library will be installed and is available for later usage.

2.1.3 Build the Example Application

Change to the /usr/src/tdrv019/example directory

Execute the Makefile:

```
# make install
```

After successful completion the example binary (*tdrv019exa*) will be installed in the /bin directory.

2.2 Building Executables with Momentics IDE (5.0)

This chapter gives just a simple description how to build the drivers with the Momentics IDE (5.0), for more detailed information please refer to the appropriate documentation.

For installation unpack the tar-archive into the desired working directory.

After that the necessary directory structure for the automatic build and the source files are available beneath the new directory called *tdrv019*.

2.2.1 Build the Device Driver

Create a new project (“Makefile Project with Existing Code”) in your workspace:
- Select a “Project Name” (e.g. TDRV019)
- Select the path “tdrv019\driver” in the working directory as “Existing Code Location”
- Select the “Toolchain for Indexer Settings” (e.g. “QNX Multi-toolchain”)

Now the device driver can be built by “Building the Project”.

After successful completion the IDE shows a “Binaries”-path containing the built binary of tdrv019 device driver. (e.g. “tdrv019 – [x86/le]”)

2.2.2 Build the API Library

Create a new project (“Makefile Project with Existing Code”) in your workspace:
- Select a “Project Name” (e.g. TDRV019-API)
- Select the path “tdrv019\api” in the working directory as “Existing Code Location”
- Select the “Toolchain for Indexer Settings” (e.g. “QNX Multi-toolchain”)

Now the API Library can be built by “Building the Project”.

2.2.3 Build the Example Application

Create a new project (“Makefile Project with Existing Code”) in your workspace:
- Select a “Project Name” (e.g. TDRV019-Example)
- Select the path “tdrv019\example” in the working directory as “Existing Code Location”
- Select the “Toolchain for Indexer Settings” (e.g. “QNX Multi-toolchain”)
- Copy the TDRV019 API Library binary file (libtdrv019api.a) into the local QNX library path

Now the example can be built by “Building the Project”.

After successful completion the IDE shows a “Binaries”-path containing the built binary of tdrv019 example application. (e.g. “tdrv019exa – [x86/le]”)

2.2.4 Integrate the Device Driver Files to a QNX-Image

To add the device driver file and the example application file to a QNX-Image, just a few steps are necessary.

Copy the desired binary files of the device driver and example project into “sbin” beneath the “install”-path of the target project using the Momentics-IDE.

Add the filenames of the added files into the build-file (e.g. “x86-generic.build”) in “images”. For example the filenames (e.g. tdrv019, tdrv019exa) can be inserted behind the serial driver names (insert each filename in a separate line).

After a rebuild of the QNX-Image, the driver files will be available on the disk and can be used after booting.

2.3 Building Executables with Momentics IDE (7.0)

This chapter gives just a simple description how to build the drivers with the Momentics IDE (7.0), for more detailed information please refer to the appropriate documentation.

For installation unpack the tar-archive into the desired working directory.

After that the necessary directory structure for the automatic build and the source files are available beneath the new directory called *tdrv019*.

2.3.1 Build the Device Driver

Create a new project (“Makefile Project with Existing Code”) in your workspace:
- Select a “Project Name” (e.g. TDRV019)
- Select the path “tdrv019\driver” in the working directory as “Existing Code Location”
- Select the “Toolchain for Indexer Settings” (e.g. “QNX Multi-toolchain”)

Now we have to specify the name of the driver executable and additional libraries needed for the driver. Open the projects properties (Alt+Enter), select C/C++ Build→Environment and add the following environment variables and values to the necessary configurations:
- NAME = tdrv019
- LIBS = pci

Now the device driver can be built by “Building the Project”.

After successful completion the IDE shows a “Binaries”-path containing the built binaries of tdrv019 device driver of the enabled configurations (e.g. “tdrv019 – [x86/le]” and “tdrv019 – [x86_64/le]”).

2.3.2 Build the API Library

Create a new project (“Makefile Project with Existing Code”) in your workspace:
- Select a “Project Name” (e.g. TDRV019-API)
- Select the path “tdrv019\api” in the working directory as “Existing Code Location”
- Select the “Toolchain for Indexer Settings” (e.g. “QNX Multi-toolchain”)

Now we have to specify the name of the driver API library. Open the projects properties (Alt+Enter), select C/C++ Build→Environment and add the following environment variables and values to the necessary configurations:
- NAME = tdrv019api

Now the API Library can be built by “Building the Project”.

2.3.3 Build the Example Application

Create a new project (“Makefile Project with Existing Code”) in your workspace:
- Select a “Project Name” (e.g. TDRV019-Example)
- Select the path “tdrv019\example” in the working directory as “Existing Code Location”
- Select the “Toolchain for Indexer Settings” (e.g. “QNX Multi-toolchain”)
- Copy the TDRV019 API Library binary file (libtdrv019api.a) into the local QNX library path

Now we have to specify the name of the driver example executable. Open the projects properties (Alt+Enter), select C/C++ Build→Environment and add the following environment variables and values to the necessary configurations:
- NAME = tdrv019exa
- LIBS = tdrv019api

Now the example can be built by “Building the Project”.

After successful completion the IDE shows a “Binaries”-path containing the built binaries of tdrv019 example application of the enabled configurations. (e.g. “tdrv019exa – [x86/le]” and “tdrv019exa – [x86_64/le]”)

2.3.4 Integrate the Device Driver Files to a QNX-Image

To add the device driver file and the example application file to a QNX-Image, just a few steps are necessary.

Copy the desired binary files of the device driver and example project into “sbin” beneath the “install”-path of the target project using the Momentics-IDE.

Add the filenames of the added files into the build-file (e.g. “x86-generic.build”) in “images”. For example the filenames (e.g. tdrv019, tdrv019exa) can be inserted behind the serial driver names (insert each filename in a separate line).

After a rebuild of the QNX-Image, the driver files will be available on the disk and can be used after booting.

2.4 Start the Driver Process

To start the TDRV019 device driver, you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tdrv019 [-v] &
```

The TDRV019 Resource Manager creates one device for each supported module, and registers the created devices in the Neutrino's pathname space under following names.

```
/dev/tdrv019_0
/dev/tdrv019_1
…
/dev/tdrv019_x
```

The pathname must be used in the application program to open a path to the desired TDRV019 device.

For debugging, you can start the TDRV019 Resource Manager with the –v option. Now the Resource Manager will print versatile information about TDRV019 configuration and command execution on the terminal window.

Make sure that only one instance of the device driver process is started.

3 API Documentation

3.1 General Functions

3.1.1 tdrv019Open

NAME

tdrv019Open – open a device.

SYNOPSIS

```
TDRV019_HANDLE tdrv019Open
(
        char            *DeviceName
)
```

DESCRIPTION

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

The tdrv019Open function can be called multiple times (e.g. in different tasks).

PARAMETERS

DeviceName

This parameter points to a null-terminated string that specifies the name of the device. The first TDRV019 device is named "/dev/tdrv019_0" the second device is named "/dev/tdrv019_1" and so on.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;

/*
** open the specified device
*/
hdl = tdrv019Open(“/dev/tdrv019_0”);
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 tdrv019Close

NAME

tdrv019Close – close a device.

SYNOPSIS

```
TDRV019_STATUS tdrv019Close
(
    TDRV019_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 <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** close the device
*/
result = tdrv019Close(hdl);
if (result != TDRV019_OK)
{
    /* handle close error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.3 tdrv019GetPciInfo

NAME

tdrv019GetPciInfo – get information of the module PCI header

SYNOPSIS

```
TDRV019_STATUS tdrv019GetPciInfo
(
        TDRV019_HANDLE              hdl,
        TDRV019_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 TDRV019_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;
} TDRV019_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 <tdrv019api.h>

TDRV019_HANDLE          hdl;
TDRV019_STATUS          result;
TDRV019_PCIINFO_BUF     pciInfoBuf

/*
** get module PCI information
*/
result = tdrv019GetPciInfo(hdl, &pciInfoBuf);
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.4 tdrv019GetBoardInfo

NAME

tdrv019GetBoardInfo – get information about the module

SYNOPSIS

```
TDRV019_STATUS tdrv019GetBoardInfo
(
        TDRV019_HANDLE                  hdl,
        TDRV019_BOARDINFO_BUF   *pBoardInfoBuf
)
```

DESCRIPTION

This function returns information about the current module 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.

pBoardInfoBuf

This argument is a pointer to the structure TDRV019_BOARDINFO_BUF that receives information of the module PCI header.

```
typedef struct
{
        char                    ModuleName[20];
        unsigned int            ModuleVariant;
        unsigned int            NumAdcChannels;
        unsigned int            NumDacChannels;
        unsigned int            NumDioLines;
} TDRV019_BOARDINFO_BUF;
```

ModuleName

This value returns the module name as a null-terminated string. For TPMC532-10, the returned value is “TPMC532-10”.

ModuleVariant

This value returns the module variant. For TPMC532-10R, the returned value is 10.

NumAdcChannels

This value returns the number of available ADC channels.

NumDacChannels

This value returns the number of available DAC channels.

NumDioLines

This value returns the number of available Digital I/O Lines.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE           hdl;
TDRV019_STATUS           result;
TDRV019_BOARDINFO_BUF    boardInfoBuf

/*
** get module information
*/
result = tdrv019GetBoardInfo( hdl, &boardInfoBuf );
if (result == TDRV019_OK)
{
     printf(“Module Name: %s\n”, boardInfoBuf.ModuleName);
} else {
     /* handle error */
}
```

RETURN VALUE

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.5 tdrv019GetBoardTemperature

NAME

tdrv019GetBoardTemperature – Read a value from the onboard temperature sensor

SYNOPSIS

```
TDRV019_STATUS tdrv019GetBoardTemperature
(
    TDRV019_HANDLE        hdl,
    double                *pTemperatureValue
)
```

DESCRIPTION

This function reads the current temperature form the onboard temperature sensor.

PARAMETERS

hdl

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

pTemperatureValue

This parameter specifies a pointer to a double floating point buffer, where the temperature value is returned.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;
double             TemperatureValue;

/*
** Read current board temperature
*/
result = tdrv019GetBoardTemperature( hdl, &TemperatureValue );
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter

3.2 ADC Functions

3.2.1 tdrv019AdcSetCorrectionValues

NAME

tdrv019AdcSetCorrectionValues – Set the correction data for a specific ADC channel

SYNOPSIS

```
TDRV019_STATUS tdrv019AdcSetCorrectionValues
(
    TDRV019_HANDLE          hdl,
    int                     Channel,
    int                     GainCorr,
    int                     OffsetCorr
)
```

DESCRIPTION

This function enables the internal data correction for a specific ADC channel, using the specified correction values. Factory calibration data are used automatically after configuring the input range. The correction is enabled by default.

PARAMETERS

hdl

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

Channel

This argument specifies the channel. Valid values are 1 to the number of available channels.

GainCorr

This argument specifies the gain correction value which shall be used for the internal data correction. The value is specified in ¼ LSB (refer to the calculation formula in the corresponding hardware user manual).

OffsetCorr

This argument specifies the offset correction value which shall be used for the internal data correction. The value is specified in ¼ LSB (refer to the calculation formula in the corresponding hardware user manual).

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE      hdl;
TDRV019_STATUS      result;
int                 GainCorr;
int                 OffsetCorr;


/*
** Set Correction values for ADC channel 1
*/
GainCorr      = 12;
OffsetCorr    = 34;

result = tdrv019AdcSetCorrectionValues(hdl, 1, GainCorr, OffsetCorr);
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_NOTSUPPORTED	The function is not supported by the hardware.
TDRV019_ERR_CHANNEL	The requested channel is not supported by the hardware module.

3.2.2 tdrv019AdcGetCorrectionValues

NAME

tdrv019AdcGetCorrectionValues – Read the Factory Calibration data for a specific ADC channel

SYNOPSIS

```
TDRV019_STATUS tdrv019AdcGetCorrectionValues
(
        TDRV019_HANDLE          hdl,
        int                     Channel,
        unsigned int            Range,
        int                     *pGainCorr,
        int                     *pOffsetCorr
)
```

DESCRIPTION

This function reads the Factory Calibration data of the specified ADC channel for the specified input range.

PARAMETERS

hdl

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

Channel

This argument specifies the channel. Valid values are 1 to the number of available channels.

Range

This argument specifies the input range. Valid values are:

Value	Description
TDRV019_INPUTRANGE_CURRENT	Currently selected input range
TDRV019_INPUTRANGE_BIPOL5V	Input Range: +/- 5 Volt
TDRV019_INPUTRANGE_BIPOL10V	Input Range: +/- 10 Volt

pGainCorr

This argument specifies an int pointer where the Factory Calibration value for Gain Correction will be returned.

pOffsetCorr

This argument specifies an int pointer where the Factory Calibration value for Offset Correction will be returned.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;
int                GainCorr, OffsetCorr;

/*
** Read Factory Calibration data for ADC channel 1, input range +/- 5V
*/
result = tdrv019AdcGetCorrectionValues(hdl,
                            1,
                            TDRV019_INPUTRANGE_BIPOL5V,
                            &GainCorr,
                            &OffsetCorr);
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_NOTSUPPORTED	The function is not supported by the hardware.
TDRV019_ERR_CHANNEL	The requested channel is not supported by the hardware module.

3.2.3 tdrv019AdcConfigInput

NAME

tdrv019AdcConfigInput – Configure ADC Input

SYNOPSIS

```
TDRV019_STATUS tdrv019AdcConfigInput
(
        TDRV019_HANDLE          hdl,
        unsigned int            AdcChannels,
        int                     InputRange,
        int                     SampleMode,
        int                     OversamplingRatio
)
```

DESCRIPTION

This function configures the ADC Input Range, the Sample Mode and the Oversampling Ratio for all channels of an ADC chip.

PARAMETERS

hdl

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

AdcChannels

This argument specifies a bitmask of the affected ADC channels. All 8 channels of one ADC chip must be configured identically. Following values are possible and may be OR'ed:

Value	**Description**
TDRV019_ADCCHANNELS_1_8	ADC channels 1 to 8 (ADC Chip #1)
TDRV019_ADCCHANNELS_9_16	ADC channels 9 to 16 (ADC Chip #2)
TDRV019_ADCCHANNELS_17_24	ADC channels 17 to 24 (ADC Chip #3)
TDRV019_ADCCHANNELS_25_32	ADC channels 25 to 32 (ADC Chip #4)

InputRange

Specifies the ADC Input Range. Following values are possible:

Value	**Description**
TDRV019_INPUTRANGE_BIPOL5V	Input Range: +/- 5 Volt
TDRV019_INPUTRANGE_BIPOL10V	Input Range: +/- 10 Volt

SampleMode

This value specifies the ADC Sample Mode. Possible values are:

Value	Description
TDRV019_SAMPLEMODE_MANUAL	Sampling is started manually
TDRV019_SAMPLEMODE_SEQUENCER	Sampling is done using the Sequencer Mode.

OversamplingRatio

This value specifies the ADC Oversampling Ratio. Possible values are :

Value	Description
TDRV019_OVERSAMPLING_NONE	No oversampling is used
TDRV019_OVERSAMPLING_X2	2-times oversampling
TDRV019_OVERSAMPLING_X4	4-times oversampling
TDRV019_OVERSAMPLING_X8	8-times oversampling
TDRV019_OVERSAMPLING_X16	16-times oversampling
TDRV019_OVERSAMPLING_X32	32-times oversampling
TDRV019_OVERSAMPLING_X64	64-times oversampling

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE  hdl;
TDRV019_STATUS  result;

/*
** Configure ADC 1-16 for +/-10V, Sample with Sequencer, 16times
** Oversampling
*/
result = tdrv019AdcConfigInput(  hdl,
                        TDRV019_ADCCHANNELS_1_8 | TDRV019_ADCCHANNELS_9_16,
                        TDRV019_INPUTRANGE_BIPOL10V,
                        TDRV019_SAMPLEMODE_SEQUENCER,
                        TDRV019_OVERSAMPLING_X16);
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_CHANNEL	The requested channels are not supported by the hardware module.
TDRV019_ERR_INVAL	At least one of the specified parameters is invalid.

3.2.4 tdrv019AdcConfigSequencer

NAME

tdrv019AdcConfigSequencer – Configure the ADC Sequencer

SYNOPSIS

```
TDRV019_STATUS tdrv019AdcConfigSequencer
(
        TDRV019_HANDLE          hdl,
        int                     SequencerMode,
        int                     ConversionClockSource,
        int                     NumConversions
)
```

DESCRIPTION

This function configures the ADC Sequencer.

PARAMETERS

hdl

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

SequencerMode

This value specifies the Input Mode of the sequencer. Valid values are:

Value	Description
TDRV019_SEQMODE_NORMAL	Sampling is started with the next Convert signal
TDRV019_SEQMODE_FRAME	Sampling is started with the next Frame signal

ConversionClockSource

This value specifies the clock source used for conversion. The corresponding clock configuration must be done separately. Valid values are:

Value	Description
TDRV019_CONVCLKSRC_CLOCK1	Convert signal based on Conversion Clock 1
TDRV019_CONVCLKSRC_CLOCK2	Convert signal based on Conversion Clock 2

NumConversions

This value specifies the number of conversions per data set.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Configure ADC Sequencer:
** Use Frame Mode, 50 samples per data set, Conversion Clock 1
*/
result = tdrv019AdcConfigSequencer( hdl,
                   TDRV019_SEQMODE_FRAME,
                   TDRV019_CONVCLKSRC_CLOCK1,
                   50 );
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter.

3.2.5 tdrv019AdcManualSingleValue

NAME

tdrv019AdcManualSingleValue – Read a single ADC channel

SYNOPSIS

```
TDRV019_STATUS tdrv019AdcManualSingleValue
(
      TDRV019_HANDLE        hdl,
      int                   Channel,
      int                   Flags,
      int                   *pAdcValue
)
```

DESCRIPTION

This function reads an ADC value of a specific channel. Depending on the specified flags, the function initiates the conversion and waits for it to finish.

PARAMETERS

hdl

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

Channel

This value specifies the desired ADC channel. Valid values are 1 up to the number of available channels. The number of supported channels depends on the used module and variant.

Flags

This value specifies flags regarding the data acquisition. The following flags are possible:

Value	**Description**
TDRV019_ADCFLAG_WAITFORNEWDATA	Wait for new ADC data. If the ADCs are configured for Manual Sampling Mode, a conversion is started. If this flag is not set, the current ADC data register value is returned immediately.

pAdcValue

This parameter specifies a pointer to an int buffer, where the ADC value is returned.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;
int                AdcValue;
int                Flags;

/*
** Read ADC Channel 1, wait for data
*/
Flags = TDRV019_ADCFLAG_WAITFORNEWDATA;

result = tdrv019AdcManualSingleValue( hdl, 1, Flags, &AdcValue );
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter
TDRV019_ERR_CHANNEL	The requested channel is not supported by the hardware module.

3.2.6 tdrv019AdcManualSingleSample

NAME

tdrv019AdcManualSingleSample – Read a single ADC Sample from all available channels

SYNOPSIS

```
TDRV019_STATUS tdrv019AdcManualSingleSample
(
        TDRV019_HANDLE          hdl,
        unsigned int            ChannelMask,
        int                     Flags,
        int                     *pAdcBuf
)
```

DESCRIPTION

This function reads ADC values of specified channels. Depending on the specified flags, the function initiates the conversion and waits for it to finish.

PARAMETERS

hdl

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

ChannelMask

This value specifies the desired ADC channels using a bitmask. Bit 0 refers to channel 1, bit 1 refers to channel 2 and so on. The number of supported channels depends on the used module and variant.

Flags

This value specifies flags regarding the data acquisition. The following flags are possible:

Value	Description
TDRV019_ADCFLAG_WAITFORNEWDATA	Wait for new ADC data. If the ADCs are configured for Manual Sampling Mode, a conversion is started. Otherwise the current ADC data register values are returned immediately.

pAdcBuf

This parameter specifies a pointer to an int data array, where the ADC values are returned. The buffer must be large enough to receive up to 32 ADC values.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;
unsigned int       ChannelMask;
int                AdcBuf[32];
int                Flags;

/*
** Read ADC Channels 1, 2 and 16. Wait for data.
*/
ChannelMask = ((1 << 15) | (1 << 1) | (1 << 0));
Flags = TDRV019_ADCFLAG_WAITFORNEWDATA;

result = tdrv019AdcManualSingleSample( hdl, ChannelMask, Flags, AdcBuf );
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter.
TDRV019_ERR_CHANNEL	At least one of the requested channels is not supported by the hardware module.

3.2.7 tdrv019AdcSequencerSampleblock

NAME

tdrv019AdcSequencerSampleblock – Read a sample block of ADC data

SYNOPSIS

```
TDRV019_STATUS tdrv019AdcSequencerSampleblock
(
        TDRV019_HANDLE          hdl,
        signed short            *pAdcBuf,
        size_t                  numBytes,
        size_t                  *validBytes,
        unsigned int            *pStatus
)
```

DESCRIPTION

This function reads a number of ADC samples. It returns either a complete data set, or fills the buffer with valid data. In case of a FIFO overflow condition, the function returns with the remaining data. To enable the data sampling again after an overflow condition, it is sufficient to execute this function again.

PARAMETERS

hdl

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

pAdcBuf

This value specifies a pointer to a signed short data buffer, where the sample block will be stored.
Depending on the number of selected channels, the data will be stored as follows:

16 Channels:

Index / Offset	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	Ch#8
0x10	Ch#9	Ch#10	Ch#11	Ch#12	Ch#13	Ch#14	Ch#15	Ch#16
0x20	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	Ch#8
0x30	Ch#9	Ch#10	Ch#11	Ch#12	Ch#13	Ch#14	Ch#15	...

8 Channels:

Index / Offset	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	Ch#8
0x10	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	...

numBytes

This value specifies the size of the data buffer in bytes. This value should be at least the size of a data set (if used), and must be a multiple of samples (Number of Sequencer Channels x 16bit).

Timeout

This value specifies the timeout used to wait for transferring a data block. The value is specified in milliseconds. Specify -1 to wait indefinitely.

validBytes

This parameter specifies a pointer where the number of valid data bytes is returned. If this value does not match numBytes, a data set has been stored or a FIFO error has occurred.

pStatus

This parameter specifies a pointer for the status of the corresponding data buffer. Possible values are:

Value	Description
TDRV019_DMASTATUS_DATASETCOMPLETE	The data buffer contains the end of a data set.
TDRV019_DMASTATUS_FIFOERROR	A FIFO overflow has occurred. The data buffer contains the remaining valid ADC data.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;
signed short       *pAdcBlock;
size_t             AdcBlockSize;
size_t             validBytes;
unsigned int       Status;

…
```

```
…

/*
** Read ADC Sample Block, wait up to 1 second
*/

/* allocate memory */
AdcBlockSize = ...;
pAdcBlock = (signed short*)malloc( AdcBlockSize );

result = tdrv019AdcSequencerSampleblock(hdl,
                             pAdcBlock,
                             AdcBlockSize,
                             1000,
                             &validBytes,
                             &Status);
if (result != TDRV019_OK)
{
     /* handle error */
}
free( pAdcBlock );
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified TPMC530_HANDLE is invalid.
TDRV019_ERR_INVAL	Invalid Parameter.
TDRV019_ERR_BUSY	There is already a function waiting for a sample block.
TDRV019_ERR_TIMEOUT	Timeout occurred.
TDRV019_ERR_FIFOERROR	FIFO Overflow. Data acquisition has been stopped.

3.2.8 tdrv019AdcFifoFlush

NAME

tdrv019AdcFifoFlush – Clears the ADC FIFO

SYNOPSIS

```
TDRV019_STATUS tdrv019AdcDmaFlush
(
    TDRV019_HANDLE        hdl
)
```

DESCRIPTION

This function clears the ADC FIFO. All previously received data is discarded. Using this function should be used after a FIFO overflow.

PARAMETERS

hdl

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

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Flush ADC Data
*/
result = tdrv019AdcFifoFlush(hdl);
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.

3.3 DAC Functions

3.3.1 tdrv019DacSetCorrectionValues

NAME

tdrv019DacSetCorrectionValues – Set the correction data for a specific DAC channel

SYNOPSIS

```
TDRV019_STATUS tdrv019DacSetCorrectionValues
(
    TDRV019_HANDLE      hdl,
    int                 channel,
    int                 GainCorr,
    int                 OffsetCorr
)
```

DESCRIPTION

This function enables the internal data correction for a specific DAC channel, using the specified correction values. Factory calibration data are used automatically after configuring the output range. The correction is enabled by default.

PARAMETERS

hdl

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

channel

This argument specifies the channel. Valid values are 1 to the number of available channels.

GainCorr

This argument specifies the gain correction value which shall be used for the internal data correction. The value is specified in ¼ LSB (refer to the calculation formula in the corresponding hardware user manual).

OffsetCorr

This argument specifies the offset correction value which shall be used for the internal data correction. The value is specified in ¼ LSB (refer to the calculation formula in the corresponding hardware user manual).

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;
int                GainCorr;
int                OffsetCorr;

/*
** Set Correction values for DAC channel 1
*/
GainCorr      = 12;
OffsetCorr    = 34;

result = tdrv019DacSetCorrectionValues(hdl, 1, GainCorr, OffsetCorr);
if (result != TDRV019_OK)
{
    /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_NOTSUPPORTED	The function is not supported by the hardware.
TDRV019_ERR_CHANNEL	The requested channel is not supported by the hardware module.

3.3.2 tdrv019DacGetCorrectionValues

NAME

tdrv019DacGetCorrectionValues – Read the Factory Calibration data for a specific DAC channel

SYNOPSIS

```
TDRV019_STATUS tdrv019DacCorrectionDisable
(
    TDRV019_HANDLE      hdl,
    int                 channel,
    unsigned int        range,
    int                 *pGainCorr,
    int                 *pOffsetCorr
)
```

DESCRIPTION

This function reads the Factory Calibration data of the specified DAC channel for the specified output range. The supported output ranges depend on the hardware module.

PARAMETERS

hdl

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

channel

This argument specifies the channel. Valid values are 1 to the number of available channels.

range

This argument specifies the output range. Valid values are:

Value	**Description**
TDRV019_OUTPUTRANGE_CURRENT	Currently selected output range
TDRV019_OUTPUTRANGE_UNIPOL5V	Output Range: 0-5 Volt
TDRV019_OUTPUTRANGE_UNIPOL6V	Output Range: 0-6 Volt
TDRV019_OUTPUTRANGE_UNIPOL10V	Output Range: 0-10 Volt
TDRV019_OUTPUTRANGE_UNIPOL10P8V	Output Range: 0-10.8 Volt
TDRV019_OUTPUTRANGE_UNIPOL12V	Output Range: 0-12 Volt
TDRV019_OUTPUTRANGE_BIPOL5V	Output Range: +/- 5 Volt
TDRV019_OUTPUTRANGE_BIPOL6V	Output Range: +/- 6 Volt
TDRV019_OUTPUTRANGE_BIPOL10V	Output Range: +/- 10 Volt
TDRV019_OUTPUTRANGE_BIPOL10P8V	Output Range: +/- 10.8 Volt
TDRV019_OUTPUTRANGE_BIPOL12V	Output Range: +/- 12 Volt

pGainCorr

This argument specifies an int pointer where the Factory Calibration value for Gain Correction will be returned.

pOffsetCorr

This argument specifies an int pointer where the Factory Calibration value for Offset Correction will be returned.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;
int                GainCorr, OffsetCorr;

/*
** Read Factory Calibration data for DAC channel 1, output range +/- 5V
*/
result = tdrv019DacGetCorrectionValues(hdl,
                        1,
                        TDRV019_OUTPUTRANGE_BIPOL5V,
                        &GainCorr,
                        &OffsetCorr);
if (result != TDRV019_OK)
{
    /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_NOTSUPPORTED	The function is not supported by the hardware.
TDRV019_ERR_CHANNEL	The requested channel is not supported by the hardware module.

3.3.3 tdrv019DacConfigOutput

NAME

tdrv019DacConfigOutput – Configure DAC Output

SYNOPSIS

```
TDRV019_STATUS tdrv019DacConfigOutput
(
      TPMC530_HANDLE        hdl,
      unsigned int          DacChannels,
      int                   OutputRange,
      int                   SampleMode
)
```

DESCRIPTION

This function configures the DAC Output Range and the Sample Mode for all channels of a DAC chip. The number of DAC channels and the supported output ranges depend on the hardware module.

PARAMETERS

hdl

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

DacChannels

This argument specifies a bitmask of the affected DAC channels. All 4 DAC channels of one DAC chip must be configured identically. Following values are possible and may be OR'ed:

Value	**Description**
TDRV019_DACCHANNELS_1_4	DAC channels 1 to 4 (DAC Chip #1)
TDRV019_DACCHANNELS_5_8	ADC channels 5 to 8 (DAC Chip #2)
TDRV019_DACCHANNELS_9_12	ADC channels 9 to 12 (DAC Chip #3)
TDRV019_DACCHANNELS_13_16	DAC channels 13 to 16 (DAC Chip #4)
TDRV019_DACCHANNELS_17_20	DAC channels 17 to 20 (ADC Chip #5)
TDRV019_DACCHANNELS_21_24	DAC channels 21 to 24 (ADC Chip #6)
TDRV019_DACCHANNELS_25_28	DAC channels 25 to 28 (ADC Chip #7)
TDRV019_DACCHANNELS_29_32	DAC channels 29 to 32 (ADC Chip #8)

OutputRange

Specifies the DAC Output Range. Following values are possible:

Value	Description
TDRV019_OUTPUTRANGE_UNIPOL5V	Output Range: 0-5 Volt
TDRV019_OUTPUTRANGE_UNIPOL6V	Output Range: 0-6 Volt
TDRV019_OUTPUTRANGE_UNIPOL10V	Output Range: 0-10 Volt
TDRV019_OUTPUTRANGE_UNIPOL10P8V	Output Range: 0-10.8 Volt
TDRV019_OUTPUTRANGE_UNIPOL12V	Output Range: 0-12 Volt
TDRV019_OUTPUTRANGE_BIPOL5V	Output Range: +/- 5 Volt
TDRV019_OUTPUTRANGE_BIPOL6V	Output Range: +/- 6 Volt
TDRV019_OUTPUTRANGE_BIPOL10V	Output Range: +/- 10 Volt
TDRV019_OUTPUTRANGE_BIPOL10P8V	Output Range: +/- 10.8 Volt
TDRV019_OUTPUTRANGE_BIPOL12V	Output Range: +/- 12 Volt
TDRV019_OUTPUTRANGE_0MA_20MA	Output Range: 0mA to 20mA Current
TDRV019_OUTPUTRANGE_0MA_24MA	Output Range: 0mA to 24mA Current
TDRV019_OUTPUTRANGE_4MA_20MA	Output Range: 4mA to 20mA Current

SampleMode

This value specifies the DAC Sample Mode. Possible values are:

Value	Description
TDRV019_SAMPLEMODE_MANUAL	Sampling is started manually
TDRV019_SAMPLEMODE_SEQUENCER	Sampling is done using the Sequencer Mode

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Configure the DAC for +/-5V, Sample with Sequencer
*/
result = tdrv019DacConfigOutput( hdl,
                                 TDRV019_OUTPUTRANGE_BIPOL5V,
                                 TDRV019_SAMPLEMODE_SEQUENCER );
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	At least one of the specified parameters is invalid.
TDRV019_ERR_NOTSUP	Channel or output range not supported.

3.3.4 tdrv019DacConfigOutputRange

NAME

tdrv019DacConfigOutputRange – Configure DAC Output Range

SYNOPSIS

```
TDRV019_STATUS tdrv019DacConfigOutputRange
(
        TPMC530_HANDLE        hdl,
        unsigned int          DacChannel,
        int                   OutputRange
)
```

DESCRIPTION

This function configures the DAC Output Range of an individual DAC channel. The number of DAC channels and the supported output ranges depend on the hardware module.

PARAMETERS

hdl

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

DacChannel

This argument specifies the desired DAC channel. Possible values are 1 to the available number of channels.

OutputRange

Specifies the DAC Output Range. Following values are possible:

Value	Description
TDRV019_OUTPUTRANGE_UNIPOL5V	Output Range: 0-5 Volt
TDRV019_OUTPUTRANGE_UNIPOL6V	Output Range: 0-6 Volt
TDRV019_OUTPUTRANGE_UNIPOL10V	Output Range: 0-10 Volt
TDRV019_OUTPUTRANGE_UNIPOL10P8V	Output Range: 0-10.8 Volt
TDRV019_OUTPUTRANGE_UNIPOL12V	Output Range: 0-12 Volt
TDRV019_OUTPUTRANGE_BIPOL5V	Output Range: +/- 5 Volt
TDRV019_OUTPUTRANGE_BIPOL6V	Output Range: +/- 6 Volt
TDRV019_OUTPUTRANGE_BIPOL10V	Output Range: +/- 10 Volt
TDRV019_OUTPUTRANGE_BIPOL10P8V	Output Range: +/- 10.8 Volt
TDRV019_OUTPUTRANGE_BIPOL12V	Output Range: +/- 12 Volt
TDRV019_OUTPUTRANGE_0MA_20MA	Output Range: 0mA to 20mA Current
TDRV019_OUTPUTRANGE_0MA_24MA	Output Range: 0mA to 24mA Current
TDRV019_OUTPUTRANGE_4MA_20MA	Output Range: 4mA to 20mA Current

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Configure the output range of DAC channel 10 to 0-10V
*/
result = tdrv019DacConfigOutputRange( hdl,
                                 10,
                                 TDRV019_OUTPUTRANGE_UNIPOL10V );
if (result != TPDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	At least one of the specified parameters is invalid.
TDRV019_ERR_NOTSUP	Channel or output range not supported.

3.3.5 tdrv019DacConfigSequencer

NAME

tdrv019DacConfigSequencer – Configure the DAC Sequencer

SYNOPSIS

```
TDRV019_STATUS tdrv019DacConfigSequencer
(
      TDRV019_HANDLE      hdl,
      int                 SequencerMode,
      int                 ConversionClockSource,
      int                 NumConversions
)
```

DESCRIPTION

This function configures the DAC Sequencer.

PARAMETERS

hdl

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

SequencerMode

This value specifies the Input Mode of the sequencer. Valid values are:

Value	**Description**
TDRV019_SEQMODE_NORMAL	Sampling is started with the next Convert signal
TDRV019_SEQMODE_FRAME	Sampling is started with the next Frame signal

ConversionClockSource

This value specifies the clock source used for conversion. The corresponding clock configuration must be done separately. Valid values are:

Value	**Description**
TDRV019_CONVCLKSRC_CLOCK1	Convert signal based on Conversion Clock 1
TDRV019_CONVCLKSRC_CLOCK2	Convert signal based on Conversion Clock 2

NumConversions

This value specifies the number of conversions per data set.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Configure DAC Sequencer:
** Use Frame Mode, 50 samples per data set, Conversion Clock 2
*/
result = tdrv019DacConfigSequencer( hdl,
                   TDRV019_SEQMODE_FRAME,
                   TDRV019_CONVCLKSRC_CLOCK2,
                   50 );
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter.

3.3.6 tdrv019DacManualSingleValue

NAME

tdrv019DacManualSingleValue – Write a single DAC channel

SYNOPSIS

```
TDRV019_STATUS tdrv019DacManualSingleValue
(
        TDRV019_HANDLE         hdl,
        int                    Channel,
        int                    Flags,
        int                    DacValue
)
```

DESCRIPTION

This function writes a DAC output value to a specific channel. Depending on the specified flags, the function initiates loading of the DAC output, resulting in the actual output of all DAC channels.

This function is only available if the DAC channel is configured for Manual Sampling mode.

PARAMETERS

hdl

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

Channel

This value specifies the desired DAC channel. Valid values are 1 to the number of supported channels.

Flags

This value specifies flags regarding the data acquisition. The following flags are possible:

Value	Description
TDRV019_DACFLAG_LOAD	If the DACs are configured for Manual Sampling Mode, output loading is initiated. If flag is not set, the function updates the data register value without affecting the actual output.

DacValue

This parameter specifies the new DAC value which shall be written to the specified channel.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE      hdl;
TDRV019_STATUS      result;
int                 DacValue;
int                 Flags;

/*
** Write to DAC Channel 4, initiate actual output of all DAC channels
*/
Flags     = TDRV019_DACFLAG_LOAD;
DacValue  = 0x7FFF;

result = tdrv019DacManualSingleValue( hdl, 4, Flags, DacValue );
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter.
TDRV019_ERR_CHANNEL	The requested channel is not supported by the hardware module.

3.3.7 tdrv019DacManualSingleSample

NAME

tdrv019DacManualSingleSample – Writes a single DAC Sample to specified channels

SYNOPSIS

```
TDRV019_STATUS tdrv019DacManualSingleSample
(
        TDRV019_HANDLE          hdl,
        unsigned int            ChannelMask,
        int                     Flags,
        int                     *pDacBuf
)
```

DESCRIPTION

This function writes DAC values to the specified channels. Depending on the specified flags, the function initiates the output loading.

This function is only available if the desired DAC channels are configured for Manual Sampling mode.

PARAMETERS

hdl

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

ChannelMask

This value specifies the desired DAC channels using a bitmask. Bit 0 refers to channel 1, bit 1 refers to channel 2 and so on. The number of supported channels depends on the used module and variant.

Flags

This value specifies flags regarding the data acquisition. The following flags are possible:

Value	Description
TDRV019_DACFLAG_LOAD	If the DACs are configured for Manual Sampling Mode, output loading is initiated. If flag is not set, the function updates the data register values without affecting the actual output.

pDacBuf

This parameter specifies a pointer to an int data array, where the DAC values are stored. The buffer must be large enough to hold up to 32 DAC values.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;
unsigned int       ChannelMask;
int                DacBuf[32];
int                Flags;

/*
** Write DAC Channels 1, 2 and 4. Update the outputs.
*/
ChannelMask = ((1 << 3) | (1 << 1) | (1 << 0));
Flags = TDRV019_DACFLAG_LOAD;
DacBuf[0] = ...;
DacBuf[1] = ...;
DacBuf[3] = ...;

result = tdrv019DacManualSingleSample( hdl, ChannelMask, Flags, DacBuf );
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter.
TDRV019_ERR_CHANNEL	At least one of the requested channels is not supported by the hardware module.

3.3.8 tdrv019DacSequencerSampleblock

NAME

tdrv019DacSequencerSampleblock – Write a sample block of DAC data

SYNOPSIS

```
TDRV019_STATUS tdrv019DacSequencerSampleblock
(
    TDRV019_HANDLE      hdl,
    signed short        *pDacBuf,
    size_t              numBytes
)
```

DESCRIPTION

This function writes a sample block of DAC data to the configured sequencer outputs.

PARAMETERS

hdl

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

pDacBuf

This value specifies a pointer to a signed short data buffer containing the DAC data. Depending on the number of selected channels, the data must be stored as follows:

16 Channels:

Offset \ Index	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	Ch#8
0x10	Ch#9	Ch#10	Ch#11	Ch#12	Ch#13	Ch#14	Ch#15	Ch#16
0x20	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	…

12 Channels:

Offset \ Index	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	Ch#8
0x10	Ch#9	Ch#10	Ch#11	Ch#12	Ch#1	Ch#2	Ch#3	...

8 Channels:

Index / Offset	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	Ch#8
0x10	Ch#1	Ch#2	Ch#3	Ch#4	Ch#5	Ch#6	Ch#7	...

4 Channels:

Index / Offset	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#2	Ch#3	Ch#4	Ch#1	Ch#2	Ch#3	Ch#4
0x10	Ch#1	Ch#2	Ch#3	Ch#4	Ch#1	Ch#2	Ch#3	...

numBytes

This value specifies the size of the data buffer in bytes. This value must be a multiple of complete channel sets (samples).

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;
signed short       *pDacBuf;
size_t             DacBlockSize;

/*
** Write DAC sample block
*/

/* allocate memory */
DacBlockSize = ...;
pDacBuf = (signed short*)malloc( DacBlockSize );
/* fill data memory */
...
result = tdrv019DacSequencerSampleblock(hdl, pDacBuf, DacBlockSize);
if (result != TDRV019_OK)
{
    /* handle error */
}
free( pDacBuf );
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter.
TDRV019_ERR_NOMEM	Error allocating memory

3.3.9 tdrv019DacFifoFlush

NAME

tdrv019DacFifoFlush – Clears the DAC FIFO

SYNOPSIS

```
TDRV019_STATUS tdrv019DacDmaFlush
(
    TDRV019_HANDLE      hdl
);
```

DESCRIPTION

This function clears the DAC FIFO. All previously loaded data is discarded. Using this function should be used after a FIFO underflow.

PARAMETERS

hdl

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

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Flush DAC Data
*/
result = tdrv019DacFifoFlush(hdl);
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.

3.4 Clock/Frame Generator Functions

3.4.1 tdrv019ClockGenConfig

NAME

tdrv019ClockGenConfig – Configure the Clock Generators

SYNOPSIS

```
TDRV019_STATUS tdrv019ClockGenConfig
(
        TDRV019_HANDLE          hdl,
        int                     ClockGen,
        int                     ClockSource,
        unsigned int            Divider,
        int                     Output
)
```

DESCRIPTION

This function configures the Clock frequency of the specified clock generator. After configuration, the clock generator remains inactive until it is enabled.

PARAMETERS

hdl

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

ClockGen

Specifies the clock generator which shall be configured. Following values are possible:

Value	Description
TDRV019_CLOCKGEN_CLOCK1	Clock Generator 1
TDRV019_CLOCKGEN_CLOCK2	Clock Generator 2

ClockSource

Specifies the clock source used for the specified clock generator. Following values are possible:

Value	Description
TDRV019_CLKSRC_20MHZ	Internal Clock Source 20MHz
TDRV019_CLKSRC_22P05MHZ	Internal Clock Source 22.05MHz
TDRV019_CLKSRC_60MHZ	Internal Clock Source 60MHz

Divider

This value specifies the number of ClockSource clocks, which must pass before a Clock Generator clock is generated.
The resulting Clock Generator Frequency is calculated as follows:

$$f_{Gen} = \frac{f_{ClkSrc}}{(Divider + 1)}$$

Output

Specifies the I/O line used for the clock generator output. Following values are possible:

Value	**Description**
TDRV019_OUTPUT_NONE	Generator Signal is not driven on any I/O line
TDRV019_OUTPUT_REARIO	Generator Signal is driven on P14 Rear I/O line
TDRV019_OUTPUT_FRONTIO	Generator Signal is driven on Front I/O line

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Configure Clock Generator 1 to 10kHz, use P14 Rear I/O
*/
result = tdrv019ClockGenConfig( hdl,
                        TDRV019_CLOCKGEN_CLOCK1,
                        TDRV019_CLKSRC_20MHZ,
                        1999,
                        TDRV019_OUTPUT_REARIO);
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter.

3.4.2 tdrv019GeneratorEnable

NAME

tdrv019GeneratorEnable – Enable the specified Clock/Frame Trigger Generators

SYNOPSIS

```
TDRV019_STATUS tdrv019GeneratorEnable
(
        TDRV019_HANDLE        hdl,
        int                   Generator
)
```

DESCRIPTION

This function enables the specified clock/frame trigger generator. This function can be used to start the generators simultaneously.

Before enabling a specific generator, make sure that it is configured properly.

PARAMETERS

hdl

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

Generator

Specifies the clock/frame trigger generator which shall be enabled. Following values are possible and can be OR'ed:

Value	Description
TDRV019_GENERATOR_CLOCK1	Clock Generator 1
TDRV019_GENERATOR_CLOCK2	Clock Generator 2
TDRV019_GENERATOR_FRAME	Frame Trigger Generator

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;


/*
** Enable Clock Generator 1 and 2 simultaneously
*/
result = tdrv019GeneratorEnable( hdl,
              TDRV019_GENERATOR_CLOCK1 | TDRV019_GENERATOR_CLOCK2);
if (result != TDRV019_OK)
{
    /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter.

3.4.3 tdrv019GeneratorDisable

NAME

tdrv019GeneratorDisable – Disable the specified Clock/Frame Trigger Generators

SYNOPSIS

```
TDRV019_STATUS tdrv019GeneratorDisable
(
        TDRV019_HANDLE        hdl,
        int                   Generator
)
```

DESCRIPTION

This function disables the specified clock/frame trigger generator. This function can be used to stop the generators simultaneously.

PARAMETERS

hdl

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

Generator

Specifies the clock generator which shall be disabled. Following values are possible and can be OR'ed:

Value	**Description**
TDRV019_GENERATOR_CLOCK1	Clock Generator 1
TDRV019_GENERATOR_CLOCK2	Clock Generator 2
TDRV019_GENERATOR_FRAME	Frame Trigger Generator

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Disable Clock Generator 1 and 2 simultaneously
*/
result = tdrv019GeneratorDisable( hdl,
              TDRV019_GENERATOR_CLOCK1 | TDRV019_GENERATOR_CLOCK2);
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter.

3.4.4 tdrv019FrameGenConfig

NAME

tdrv019FrameGenConfig – Configure the Frame Trigger Generator

SYNOPSIS

```
TDRV019_STATUS tdrv019FrameGenConfig
(
      TDRV019_HANDLE      hdl,
      int                 ClockGenSource,
      unsigned int        Divider,
      int                 Output,
      int                 NumFrames
)
```

DESCRIPTION

This function configures the Clock frequency of the specified clock generator. After configuration, the clock generator remains inactive until it is enabled.

PARAMETERS

hdl

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

ClockGenSource

Specifies the clock generator which shall be used as source. Following values are possible:

Value	Description
TDRV019_CLOCKGEN_CLOCK1	Clock Generator 1
TDRV019_CLOCKGEN_CLOCK2	Clock Generator 2

Divider

This value specifies the number of ClockGen clocks, which must pass before a Frame Trigger is generated.
The resulting Frame Trigger Frequency is calculated as follows:

$$f_{Frame} = \frac{f_{ClkGenSrc}}{(Divider + 1)}$$

Output

Specifies the I/O line used for the frame trigger output. Following values are possible:

Value	**Description**
TDRV019_OUTPUT_NONE	Generator Signal is not driven on any I/O line
TDRV019_OUTPUT_REARIO	Generator Signal is driven on P14 Rear I/O line
TDRV019_OUTPUT_FRONTIO	Generator Signal is driven on Front I/O line

NumFrames

This value specifies the number of Frame Tigger signals which shall be generated. For continuous operation, specify 0.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Configure Frame Trigger Generator for a pulse every 100 clocks of
** Clock Generator 1. Continuous operation.
*/
result = tdrv019FrameGenConfig( hdl, TDRV019_CLOCKGEN_CLOCK1, 99, 0);
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	**Description**
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter.

3.4.5 tdrv019ConversionSignalSelect

NAME

tdrv019ConversionSignalSelect – Assign a source signal to a conversion signal

SYNOPSIS

```
TDRV019_STATUS tdrv019ConversionSignalSelect
(
        TDRV019_HANDLE          hdl,
        int                     ConversionSignal,
        unsigned int            SourceSignal
)
```

DESCRIPTION

This function assigns a source signal to a conversion signal (Clock and Frame Trigger).

PARAMETERS

hdl

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

ConversionSignal

Specifies the conversion signal. Following values are possible:

Value	**Description**
TDRV019_CONVSIG_CLOCK1	Clock 1
TDRV019_CONVSIG_CLOCK2	Clock 2
TDRV019_CONVSIG_FRAME	Frame Trigger

SourceSignal

Specifies the signal which shall be used as source. Following values are possible:

Value	**Description**
TDRV019_SIGSRC_GENERATOR	Use corresponding Generator as source
TDRV019_SIGSRC_REARIO	Use corresponding Rear I/O signal
TDRV019_SIGSRC_FRONTIO	Use corresponding Front I/O signal

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Configure Conversion Signal Clock 1 to use P14 Rear I/O as source
*/
result = tdrv019ConversionSignalSelect( hdl,
                                        TDRV019_CONVSIG_CLOCK1,
                                        TDRV019_SIGSRC_REARIO);
if (result != TDRV019_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV019_ERR_INVAL	Invalid Parameter.

3.5 Digital I/O Functions

3.5.1 tdrv019DioRead

NAME

tdrv019DioRead – read current input value of the I/O lines

SYNOPSIS

```
TDRV019_STATUS tdrv019DioRead
(
        TDRV019_HANDLE      hdl,
        unsigned int        *input
)
```

DESCRIPTION

This function reads the current input value of the I/O lines.

PARAMETER

hdl

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

input

This argument points to a buffer where the current value of the I/O lines will be returned. Bit 0 returns the value of the first I/O line, bit 1 the value of the second I/O line, and so on. Only available I/O lines will return valid values. The number of available I/O lines depends on the used module type and variant.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;
unsigned int       input;

/*
** read current state of I/O lines
*/
result = tdrv019Read(hdl, &input);
if (result != TDRV019_OK)
{
     /* handle error */
}
else
{
     printf(“INPUT: 0x%08X\n”, input);
}
```

RETURN VALUE

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The device handle is invalid
TDRV019_ERR_INVAL	A NULL pointer is referenced for an input value

3.5.2 tdrv019DioWrite

NAME

tdrv019DioWrite – set output value

SYNOPSIS

```
TDRV019_STATUS tdrv019DioWrite
(
        TDRV019_HANDLE      hdl,
        unsigned int        output
)
```

DESCRIPTION

This function sets the output value.

The specified value will only appear on the I/O lines which are configured for output.

PARAMETER

hdl

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

output

This argument specifies the output value for I/O. Bit 0 specifies the value of the first I/O line, bit 1 the value of the second I/O line, and so on. The number of available I/O lines depends on the used module type and variant.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Set output value (set first eigth I/O lines), clear all others
*/
result = tdrv019DioWrite(hdl, 0x000000FF);
if (result != TDRV019_OK)
{
     /* error handling */
}
```

RETURN VALUE

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The device handle is invalid

3.5.3 tdrv019DioWriteMasked

NAME

tdrv019DioWriteMasked – set output value for specified I/O lines

SYNOPSIS

```
TDRV019_STATUS tdrv019DioWriteMasked
(
        TDRV019_HANDLE      hdl,
        unsigned int        output,
        unsigned int        mask
)
```

DESCRIPTION

This function sets the output value for specified I/O lines. The mask specifies which I/O bits shall be set to the specified output value and which shall keep the current value.

This specified value will only appear on the I/O lines which are configured for output.

PARAMETER

hdl

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

output

This argument specifies the output value for I/O lines. Bit 0 specifies the value of the first I/O line, bit 1 the value of second I/O line, and so on.

mask

This argument specifies the output mask for output lines. Bit 0 specifies the mask for the first I/O line, bit 1 the value for the second I/O line, and so on.
A set bit (1) means the bit shall be set to the value specified by *output*.
A reset bit (0) means that the old output value will not be changed.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Set a part of the output value (set/reset first four I/O lines)
*/
result = tdrv019DioWriteMasked(hdl, 0x12345678, 0x0000000F);
if (result != TDRV019_OK)
{
     /* error handling */
}
```

RETURN VALUE

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The device handle is invalid

3.5.4 tdrv019DioSetOutputLine

NAME

tdrv019DioSetOutputLine – set a specified output line

SYNOPSIS

```
TDRV019_STATUS tdrv019DioSetOutputLine
(
        TDRV019_HANDLE      hdl,
        int                 outputLine
)
```

DESCRIPTION

This function sets a single bit of the output value.

This specified value will only appear if the corresponding I/O line is configured for output.

PARAMETER

hdl

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

outputLine

This argument specifies a data bit that shall be set. Allowed values are 1 up to the number of available I/O lines.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Set I/O line 8
*/
result = tdrv019DioSetOutputLine(hdl, 8);
if (result != TDRV019_OK)
{
     /* error handling */
}
```

RETURN VALUE

On success, TDRV006_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The device handle is invalid
TDRV019_ERR_INVAL	An invalid line number is specified

3.5.5 tdrv019DioClearOutputLine

NAME

tdrv019DioClearOutputLine – reset a specified I/O line

SYNOPSIS

```
TDRV019_STATUS tdrv019DioClearOutputLine
(
        TDRV019_HANDLE      hdl,
        int                 outputLine
)
```

DESCRIPTION

This function resets a single bit of the output value.

This specified value will only appear if the corresponding I/O line is configured for output.

PARAMETER

hdl

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

outputLine

This argument specifies data bit that shall be reset. Allowed values are 1 up to the number of available I/O lines.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** Clear I/O line 4
*/
result = tdrv019DioClearOutputLine(hdl, 4);
if (result != TDRV019_OK)
{
     /* error handling */
}
```

RETURN VALUE

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The device handle is invalid
TDRV019_ERR_INVAL	An invalid line number is specified

3.5.6 tdrv019DioConfigureDirection

NAME

tdrv019DioConfigureDirection – set the I/O line direction

SYNOPSIS

```
TDRV019_STATUS tdrv019DioConfigureDirection
(
        TDRV019_HANDLE      hdl,
        unsigned int        DirectionValue,
        unsigned int        DirectionMask
)
```

DESCRIPTION

This function sets the I/O line direction. The value specifies which I/O lines shall be configured for output and which I/O lines should be used for input.

PARAMETER

hdl

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

DirectionValue

This value specifies the direction of the corresponding I/O lines. An active (1) bit will configure the corresponding I/O line to OUTPUT, an unset (0) bit will configure the corresponding I/O line to INPUT. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

DirectionMask

This parameter specifies the bitmask. Only active bits (1) will have an effect on the I/O direction, the direction of all other I/O lines will be left unchanged. Bit 0 of this value corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*
** configure new I/O direction:
** set lowest 4 I/O lines to OUTPUT, and higher 4 I/O lines to INPUT.
** leave all other I/O lines unchanged.
*/
result = tdrv012ConfigureDirection(hdl,
                        (0xf0 << 0) | (0x0f << 0),
                        (0x00 << 0) | (0x0f << 0));
if (result != TDRV019_OK)
{
     /* error handling */
}
```

RETURN VALUE

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The device handle is invalid

3.5.7 tdrv019DioDebounceConfig

NAME

tdrv019DioDebounceConfig – Configure digital I/O (input) debouncer

SYNOPSIS

```
TDRV019_STATUS tdrv019DioDebounceConfig
(
        TDRV019_HANDLE          hdl,
        unsigned short          DebounceTime
)
```

DESCRIPTION

This function configures the digital I/O input debouncing mechanism to avoid detection of invalid signal changes in noisy environments.

PARAMETERS

hdl

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

DebounceTime

Specifies the debounce time. For the filter calculation, refer to the corresponding hardware user manual.

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*-------------------------------------------------------
  Enable Debouncer
  -------------------------------------------------------*/
result = tdrv019DioDebounceConfig(hdl, 10000 );     /* DebounceTime */
if (result == TDRV019_OK)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid

3.5.8 tdrv019DioPullResistorConfig

NAME

tdrv019DioPullResistorConfig – Configure Pull Resistors for DIO and synchronization signals

SYNOPSIS

```
TDRV019_STATUS tdrv019DioPullResistorConfig
(
        TDRV019_HANDLE         hdl,
        unsigned int           PullFront,
        unsigned int           PullRear
)
```

DESCRIPTION

This function configures the pull resistors of front I/O and rear I/O.

PARAMETERS

hdl

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

PullFront

Specifies the pull resistor used for front I/O. The following values are possible:

Value	Description
TDRV019_DIOPULL_FLOATING	I/O lines are left floating
TDRV019_DIOPULL_5V	I/O lines are pulled to 5V
TDRV019_DIOPULL_3P3V	I/O lines are pulled to 3.3V
TDRV019_DIOPULL_GND	I/O lines are pulled to GND

PullRear

Specifies the pull resistor used for rear I/O. The following values are possible:

Value	Description
TDRV019_DIOPULL_FLOATING	I/O lines are left floating
TDRV019_DIOPULL_5V	I/O lines are pulled to 5V
TDRV019_DIOPULL_3P3V	I/O lines are pulled to 3.3V
TDRV019_DIOPULL_GND	I/O lines are pulled to GND

EXAMPLE

```
#include <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;

/*------------------------------------------------------------
  Configure Front I/O DIO to 5V Pull-Up, Rear I/O Sync to 3.3V
  ------------------------------------------------------------*/
result = tdrv019DioPullResistorConfig(hdl,
                       TDRV019_DIOPULL_5V,
                       TDRV019_DIOPULL_3P3V );
if (result == TDRV019_OK)
{
    /* function succeeded */
} else {
    /* handle error */
}
```

RETURN VALUE

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	**Description**
TDRV019_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV019_ERR_INVAL	At least one of the specified parameters is invalid

3.6 Interrupt Functions

3.6.1 tdrv019InterruptWait

NAME

tdrv019InterruptWait – Wait for incoming Local Interrupt Source

SYNOPSIS

```
TDRV019_STATUS tdrv019InterruptWait
(
    TDRV019_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 interrupt sources. 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 Interrupt Status Register. 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 Interrupt Status Register. 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 <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;
unsigned int       interruptMask;
unsigned int       interruptOccurred;

/*
** Wait at least 5 seconds for incoming interrupts on Frame Trigger
*/
interruptMask = (1 << 28);
result = tdrv019InterruptWait(   hdl,
                                 interruptMask,
                                 &interruptOccurred,
                                 5000 );
if (result == TDRV019_OK)
{
     /* Interrupt arrived. */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified TDRV019_HANDLE is invalid.
TDRV019_ERR_TIMEOUT	The specified timeout occurred.

3.6.2 tdrv019InterruptRegisterCallbackThread

NAME

tdrv019InterruptRegisterCallbackThread – Register a User Callback Function for Interrupt Handling

SYNOPSIS

```
TDRV019_STATUS tdrv019InterruptRegisterCallbackThread
(
    TDRV019_HANDLE      hdl,
    int                 threadPriority,
    int                 stackSize,
    unsigned int        interruptMask,
    FUNCINTCALLBACK     callbackFunction,
    void                *funcparam,
    TDRV019_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 TDRV019 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
TDRV019_PRIORITY_NORMAL	Normal Priority (THREAD_PRIORITY_NORMAL)
TDRV019_PRIORITY_HIGH	High Priority (THREAD_PRIORITY_HIGHEST)
TDRV019_PRIORITY_LOW	Low Priority (THREAD_PRIORITY_LOWEST)

Other values might be possible.

stackSize

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

interruptMask

This parameter specifies specific interrupt bits to wait for. The interrupt bits correspond to the Interrupt Status Register bits described in the hardware 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)( TDRV019_HANDLE  hdl,
                                unsigned int    interruptOccurred,
                                void            *param,
                                TDRV019_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 Interrupt Status Register. 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 TDRV019_OK. If this parameter differs from TDRV019_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 <tdrv019api.h>

TDRV019_HANDLE     hdl;
TDRV019_STATUS     result;
unsigned int       interruptMask;
USER_DATA_AREA     userDataArea;
TDRV019_HANDLE     callbackHandle;

/* forward declaration of callback functions */
void callback_FRAME(    TDRV019_HANDLE     hdl,
                        unsigned int       interruptOccurred,
                        void               *param,
                        TDRV019_STATUS     status);
/*
**  Register callback function for FRAME
**  Use a “normal” priority, and 64KB stack.
*/
interruptMask = (1 << 16);
result = tdrv019InterruptRegisterCallbackThread(hdl,
                                          TDRV019_PRIORITY_NORMAL,
                                          0x10000,
                                          interruptMask,
                                          callback_FRAME,
                                          &userDataArea,
                                          &callbackHandle);
...
```

```
...
if (result != TDRV019_OK)
{
     /* handle error */
}

...
/*
** Callback Function
*/
void callback_FRAME(    TDRV019_HANDLE     hdl,
                        unsigned int       interruptOccurred,
                        void               *param,
                        TDRV019_STATUS     status)
{
     TDRV019_STATUS     result;
     USER_DATA_AREA     *pUsrData = (USER_DATA_AREA*)param;
     unsigned int       u32value;

     if (status != TDRV019_OK)
     {
          /* handle error status */
     }

     printf("[FRAME Interrupt]\n");

     /* Do something useful */

     /* or handle errors */
     return;
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified TDRV019_HANDLE is invalid.
TDRV019_ERR_INVAL	Function or callback handle pointer is NULL.
TDRV019_ERR_TASK_CREATE	Creation of the callback thread (task) failed.

3.6.3 tdrv019InterruptUnregisterCallback

NAME

tdrv019InterruptUnregisterCallback – Unregister a User Callback Function

SYNOPSIS

```
TDRV019_STATUS tdrv019InterruptUnregisterCallback
(
    TDRV019_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 <tdrv019api.h>

TDRV019_HANDLE     callbackHdl;
TDRV019_STATUS     result;

/*
** Unregister a callback function
*/
result = tdrv019InterruptUnregisterCallback(callbackHdl);
if (result == TDRV019_OK)
{
     / *OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV019_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV019_ERR_INVALID_HANDLE	The specified callback handle is invalid.