The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV021-SW-65

Windows Device Driver

Isolated Simultaneous Sampling (HV) AD/DA

Version 1.0.x

User Manual

Issue 1.0.0

December 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

TDRV021-SW-65

Windows Device Driver

Isolated Simultaneous Sampling (HV) AD/DA

Supported Modules:
TPMC520
TPMC530

Issue	Description	Date
1.0.0	First Issue	December 5, 2018

Table of Contents

1 Introduction

The TDRV021-SW-65 Windows device driver is a kernel mode driver which allows the operation of supported hardware modules on an Intel or Intel-compatible Windows operating system.

The TDRV021-SW-65 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
- Wait for Timer Interrupts
- Register Timer Callback functions
- Setting Correction Data for ADC and DAC temporary

The TDRV021-SW-65 device driver supports the modules listed below:

TPMC530	Isolated Simultaneous Sampling AD/DA	PMC
TPMC520	Isolated Simultaneous Sampling (HV) AD/DA	PMC

In this document all supported modules and devices will be called TDRV021. Specials for certain devices will be advised.

To get more information about the features and use of TDRV021 devices it is recommended to read the manuals listed below.

TPMC520 / TPMC530 User Manual
Installation-Guide on Distribution Media

2 Installation

Following files are located in the device driver directory TDRV021-SW-65 on the distribution media:

driver*	Directory containing driver files
example\tdrv021exa.c	Example application source
api\tdrv021api.h	Application Programming Interface header
TDRV021-SW-65-1.0.0.pdf	This document
Release.txt	Information about the Device Driver Release
ChangeLog.txt	Release history
tdrv021exa.exe	Example application (executable for quick start)

Copy all required Header and API files into your desired project directory.

For device driver installation execute the application “Setup” on the distribution media. For more information refer to the Installation-Guide, which is also part of the distribution media.

3 API Documentation

3.1 Usage of Function Interface

To use the standard API-function interface in non-managed applications, the tdrv021api.dll must be used.

3.2 General Functions

3.2.1 tdrv021Open

NAME

tdrv021Open – Opens a Device

SYNOPSIS

```
TDRV021_HANDLE tdrv021Open
(
      char          *DeviceName
)
```

DESCRIPTION

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

PARAMETERS

DeviceName

This parameter points to a null-terminated string that specifies the name of the device.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;

/*
** open file descriptor to device
*/
hdl = tdrv021Open(“\\\\.\\TDRV021_1”);
if (hdl == NULL)
{
     /* handle open error */
}
```

RETURNS

A device handle, or NULL if the function fails. To get extended error information, call ***GetLastError***.

ERROR CODES

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

3.2.2 tdrv021Close

NAME

tdrv021Close – Closes a Device

SYNOPSIS

```
TDRV021_STATUS tdrv021Close
(
     TDRV021_HANDLE       hdl
)
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE hdl;
TDRV021_STATUS result;

/*
** close file descriptor to device
*/
result = tdrv021Close(hdl);
if (result != TDRV021_OK)
{
     /* handle close error */
}
```

RETURNS

On success TDRV021_OK, or an appropriate error code.

ERROR CODES

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

3.2.3 tdrv021GetPciInfo

NAME

tdrv021GetPciInfo – get information of the module PCI header

SYNOPSIS

```
TDRV021_STATUS tdrv021GetPciInfo
(
    TDRV021_HANDLE          hdl,
    TDRV021_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 TDRV021_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;
} TDRV021_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 “tdrv021api.h”

TDRV021_HANDLE          hdl;
TDRV021_STATUS          result;
TDRV021_PCIINFO_BUF     pciInfoBuf

/*
** get module PCI information
*/
result = tdrv021GetPciInfo( hdl, &pciInfoBuf );

if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified device handle is invalid

3.2.4 tdrv021GetBoardInfo

NAME

tdrv021GetBoardInfo – get information about the module

SYNOPSIS

```
TDRV021_STATUS tdrv021GetBoardInfo
(
    TDRV021_HANDLE          hdl,
    TDRV021_BOARDINFO_BUF   *pBoardInfoBuf
)
```

DESCRIPTION

This function returns information about the module.

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 TDRV021_BOARDINFO_BUF that receives board information.

```
typedef struct
{
    unsigned int        firmwareVersion;
    unsigned int        numAdcs;
    unsigned int        numDacs;
    unsigned int        adcRange[TDRV021_MAX_ADC_RANGES];
    unsigned short      adcFlags[TDRV021_MAX_ADC_RANGES];
    unsigned int        serialNoLow;
    unsigned int        serialNoHigh;
} TDRV021_BOARDINFO_BUF;
```

firmwareVersion

This value will be filled with the content of the Firmware Version Register.

numAdcs

This value returns the available number of ADC channels.

numDacs

This value returns the available number of DAC channels.

adcRange[]

This array indicates the available Input Ranges. If the value is not zero, the Input Range is available. The returned value specifies the full-scale value, e.g. a value 5000 may indicate a maximum value of 5V at full scale (0x7FFF, if the corresponding adcFlags value specifies mV and bipolar input).
If the array value is zero, this Input Range is not supported.
The array index specifies the corresponding input range, index 0 for TDRV021_INPUTRANGE_0, index 1 for TDRV021_INPUTRANGE_1, and so on.
A maximum number of four Ranges are defined for this driver, TDRV021_MAX_ADC_RANGES is defined to 4.

adcFlags[]

This array contains additional information for the Input Ranges. If the values of the corresponding adcRange is not zero the following bits may be set.

Value	Description
TDRV021_BOARDINFO_ADCRANGE_MILLIVOLT	The unit of the adcRange value is specifies in mV.
TDRV021_BOARDINFO_ADCRANGE_MICROAMPERE	The unit of the adcRange value is specifies in µA.
TDRV021_BOARDINFO_ADCRANGE_UNIPOLAR	The Range is unipolar, returned values are between 0 and 65535.
TDRV021_BOARDINFO_ADCRANGE_BIPOLAR	The Range is unipolar, returned values are between -32768 and 32767.

serialNoLow

This value returns the lower part of the Boards Serial number.

serialNoHigh

This value returns the higher part of the Boards Serial number.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE          hdl;
TDRV021_STATUS          result;
TDRV021_BOARDINFO_BUF   boardInfo;
unsigned int            firmwareVersion;

…
```

…

```
/*
** get module information
*/
result = tdrv021GetBoardInfo(hdl, &boardInfo);
if (result == TDRV021_OK)
{
     printf("Firmware-Code Version: %04X\n", boardInfo.firmwareVersion);
}
else
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified device handle is invalid

3.2.5 tdrv021CorrectionEnable

NAME

tdrv021CorrectionEnable – Enable the internal data correction

SYNOPSIS

```
TDRV021_STATUS tdrv021CorrectionEnable
(
     TDRV021_HANDLE       hdl
)
```

DESCRIPTION

This function enables the internal data correction for both ADC and DAC channels, using the factory calibration data. 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.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;

/*
** Enable Data Correction
*/

result = tdrv021CorrectionEnable(hdl);
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.

3.2.6 tdrv021CorrectionDisable

NAME

tdrv021CorrectionDisable – Disable the internal data correction

SYNOPSIS

```
TDRV021_STATUS tdrv021CorrectionDisable
(
     TDRV021_HANDLE       hdl
)
```

DESCRIPTION

This function disables the internal data correction for both ADC and DAC channels.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;

/*
** Disable Data Correction
*/

result = tdrv021CorrectionDisable(hdl);
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.

3.2.7 tdrv021CorrectionUpdate

NAME

tdrv021CorrectionUpdate – Update and use modified correction data

SYNOPSIS

```
TDRV021_STATUS tdrv021CorrectionUpdate
(
     TDRV021_HANDLE        hdl
)
```

DESCRIPTION

This function updates the used correction data. Previous modified correction data will be used after this function has been called.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;

/*
** Update used Correction Data
*/

result = tdrv021CorrectionUpdate(hdl);
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.

3.2.8 tdrv021AdcCorrectionValueSet

NAME

tdrv021AdcCorrectionValueSet – Set correction data set of a specified ADC channel

SYNOPSIS

```
TDRV021_STATUS tdrv021AdcCorrectionValueSet
(
    TDRV021_HANDLE      hdl,
    int                 Channel,
    unsigned int        InputRange,
    signed short        Offset,
    signed short        Gain
)
```

DESCRIPTION

This function sets an application supplied correction data set for specified ADC channel. The data will be stored into the TDRV021 correction data area, but it will not be used instantly. The function *tdrv021CorrectionUpdate()* must be called to active the new correction data set.

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 to 16. The number of supported channels depends on the used TDRV021 module variant.

InputRange

Specifies the ADC Input Range. The Input Range depends on the used board, information about the Input Range are available via calling tdrv021GetBoardInfo() (see 3.2.4).
Following Input Range values are possible:

Value	Description
TDRV021_INPUTRANGE_0	Input Range 0 (e.g. TPMC530-10: +/- 5 Volt)
TDRV021_INPUTRANGE_1	Input Range 1 (e.g. TPMC530-10: +/- 10 Volt)
TDRV021_INPUTRANGE_2	Input Range 2 (e.g. TPMC530-10: not supported)
TDRV021_INPUTRANGE_3	Input Range 3 (e.g. TPMC530-10: not supported)

Offset

This argument specifies the new offset correction value. The value is specified in ¼ LSB.

Gain

This argument specifies the new gain correction value. The value is specified in ¼ LSB.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
signed short       OffsetCorr;
signed short       GainCorr;

/*
** Modify correction data Set of ADC #4 (for Input Range 1)
*/
OffsetCorr = 22;
GainCorr = -34;
result = tdrv021AdcCorrectionValueSet(hdl,
                                      4,
                                      TDRV021_INPUTRANGE_1,
                                      OffsetCorr,
                                      GainCorr);
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	At least one of the specified parameters is invalid.

3.2.9 tdrv021AdcCorrectionValueGet

NAME

tdrv021AdcCorrectionValueGet – Get correction data set of a specified ADC channel

SYNOPSIS

```
TDRV021_STATUS tdrv021AdcCorrectionValueGet
(
    TDRV021_HANDLE      hdl,
    int                 Channel,
    unsigned int        InputRange,
    signed short        *pOffset,
    signed short        *pGain
)
```

DESCRIPTION

This function reads the current correction data set for a specified ADC channel stored in the TDRV021 correction data area.

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 to 8. The number of supported channels depends on the used TDRV021 module variant.

InputRange

Specifies the ADC Input Range. The Input Range depends on the used board, information about the Input Range are available via calling tdrv021GetBoardInfo() (see 3.2.4).
Following Input Range values are possible:

Value	Description
TDRV021_INPUTRANGE_0	Input Range 0 (e.g. TPMC530-10: +/- 5 Volt)
TDRV021_INPUTRANGE_1	Input Range 1 (e.g. TPMC530-10: +/- 10 Volt)
TDRV021_INPUTRANGE_2	Input Range 2 (e.g. TPMC530-10: not supported)
TDRV021_INPUTRANGE_3	Input Range 3 (e.g. TPMC530-10: not supported)

pOffset

This parameter points to a short value which will be filled with the current offset correction value of the specified channel. The value is specified in ¼ LSB.

pGain

This parameter points to a short value which will be filled with the current gain correction value of the specified channel. The value is specified in ¼ LSB.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
signed short       OffsetCorr;
signed short       GainCorr;

/*
** Get correction data Set of ADC #7 (Input Range 0)
*/
result = tdrv021AdcCorrectionValueGet(     hdl,
                                           7,
                                           TDRV021_INPUTRANGE_0,
                                           &OffsetCorr,
                                           &GainCorr);
if (result != TDRV021_OK)
{
     /* handle error */
}
else
{
     printf(“Correction data: Offset: %d – Gain: %d\n”, OffsetCorr,
GainCorr);
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	At least one of the specified parameters is invalid.

3.2.10 tdrv021DacCorrectionValueSet

NAME

tdrv021DacCorrectionValueSet – Set correction data set of a specified DAC channel

SYNOPSIS

```
TDRV021_STATUS tdrv021DacCorrectionValueSet
(
    TDRV021_HANDLE      hdl,
    int                 Channel,
    unsigned int        OutputRange,
    signed short        Offset,
    signed short        Gain
)
```

DESCRIPTION

This function sets an application supplied correction data set for specified DAC channel. The data will be stored into the TDRV021 correction data area, but it will not be used instantly. The function *tdrv021CorrectionUpdate()* must be called to active the new correction data set.

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 8. The number of supported channels depends on the used TDRV021 module variant.

OutputRange

Specifies the DAC Output Range. Following values are possible:

Value	Description
TDRV021_OUTPUTRANGE_BIPOL5V	Output Range: +/- 5 Volt
TDRV021_OUTPUTRANGE_BIPOL10V	Output Range: +/- 10 Volt
TDRV021_OUTPUTRANGE_UNIPOL5V	Output Range: 0 - 5 Volt
TDRV021_OUTPUTRANGE_UNIPOL10V	Output Range: 0 - 10 Volt

Offset

This argument specifies the new offset correction value. The value is specified in ¼ LSB.

Gain

This argument specifies the new gain correction value. The value is specified in ¼ LSB.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
signed short       OffsetCorr;
signed short       GainCorr;

/*
** Modify correction data Set of DAC #2 (0V…5V)
*/
OffsetCorr = 12;
GainCorr = -24;
result = tdrv021DacCorrectionValueSet(hdl,
                                      2,
                                      TDRV021_OUTPUTRANGE_UNIPOL5V,
                                      OffsetCorr,
                                      GainCorr);
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	At least one of the specified parameters is invalid.

3.2.11 tdrv021DacCorrectionValueGet

NAME

tdrv021DacCorrectionValueGet – Get correction data set of a specified DAC channel

SYNOPSIS

```
TDRV021_STATUS tdrv021DacCorrectionValueGet
(
    TDRV021_HANDLE      hdl,
    int                 Channel,
    unsigned int        OutputRange,
    signed short        *pOffset,
    signed short        *pGain
)
```

DESCRIPTION

This function reads the current correction data set for specified DAC channel stored in the TDRV021 correction data area.

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 8. The number of supported channels depends on the used TDRV021 module variant.

OutputRange

Specifies the DAC Output Range. Following values are possible:

Value	Description
TDRV021_OUTPUTRANGE_BIPOL5V	Output Range: +/- 5 Volt
TDRV021_OUTPUTRANGE_BIPOL10V	Output Range: +/- 10 Volt
TDRV021_OUTPUTRANGE_UNIPOL5V	Output Range: 0 - 5 Volt
TDRV021_OUTPUTRANGE_UNIPOL10V	Output Range: 0 - 10 Volt

pOffset

This parameter points to a short value which will be filled with the current offset correction value of the specified channel. The value is specified in ¼ LSB.

pGain

This parameter points to a short value which will be filled with the current gain correction value of the specified channel. The value is specified in ¼ LSB.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
signed short       OffsetCorr;
signed short       GainCorr;

/*
** Get correction data set of DAC #2 (0V…5V)
*/
result = tdrv021DacCorrectionValueGet(     hdl,
                                           2,
                                           TDRV021_OUTPUTRANGE_UNIPOL5V,
                                           &OffsetCorr,
                                           &GainCorr);
if (result != TDRV021_OK)
{
     /* handle error */
}
else
{
     printf(“Correction data: Offset: %d – Gain: %d\n”, OffsetCorr,
GainCorr);
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	At least one of the specified parameters is invalid.

3.3 ADC Functions

3.3.1 tdrv021AdcConfigInput

NAME

tdrv021AdcConfigInput – Configure ADC Input

SYNOPSIS

```
TDRV021_STATUS tdrv021AdcConfigInput
(
    TDRV021_HANDLE      hdl,
    unsigned int        InputRange,
    unsigned int        SampleMode,
    unsigned int        OversamplingRatio
)
```

DESCRIPTION

This function configures the ADC Input Range, the Sample Mode and the Oversampling Ratio.

PARAMETERS

hdl

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

InputRange

Specifies the ADC Input Range. Refer to your boards User Manual - ADC Configuration Register to find information about the assigned ranges or determine the ranges by using the driver function tdrv021GetBoardInfo() (refer to 3.2.4). Following values are possible:

Value	Description
TDRV021_INPUTRANGE_0	Input Range 0 (e.g. TPMC530-10: +/- 5 Volt)
TDRV021_INPUTRANGE_1	Input Range 1 (e.g. TPMC530-10: +/- 10 Volt)
TDRV021_INPUTRANGE_2	Input Range 2 (e.g. TPMC530-10: not supported)
TDRV021_INPUTRANGE_3	Input Range 3 (e.g. TPMC530-10: not supported)

SampleMode

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

Value	Description
TDRV021_SAMPLEMODE_MANUAL	Sampling is started manually
TDRV021_SAMPLEMODE_SAMCLOCK	Sample Clock is used to start Conversion
TDRV021_SAMPLEMODE_EXTTRIGGER	External Trigger Input is used to start Conversion
TDRV021_SAMPLEMODE_INTSYNC	Internal Synchronization (use DAC trigger) ATTENTION: Do not configure the DAC for Internal Synchronization, if this mode is selected.

OversamplingRatio

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

Value	Description
TDRV021_OVERSAMPLING_NONE	No oversampling is used
TDRV021_OVERSAMPLING_X2	2-times oversampling
TDRV021_OVERSAMPLING_X4	4-times oversampling
TDRV021_OVERSAMPLING_X8	8-times oversampling
TDRV021_OVERSAMPLING_X16	16-times oversampling
TDRV021_OVERSAMPLING_X32	32-times oversampling
TDRV021_OVERSAMPLING_X64	64-times oversampling

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLEhdl;
TDRV021_STATUSresult;

/*
** Configure the ADC for +/-10V (for TPMC530-10),
** Sample with SampleClock, 16times Oversampling
*/

result = tdrv021AdcConfigInput(  hdl,
                                 TDRV021_INPUTRANGE_1,
                                 TDRV021_SAMPLEMODE_SAMCLOCK,
                                 TDRV021_OVERSAMPLING_X16);
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	At least one of the specified parameters is invalid.

3.3.2 tdrv021AdcConfigDma

NAME

tdrv021AdcConfigDma – Configure the DMA list for ADC

SYNOPSIS

```
TDRV021_STATUS tdrv021AdcConfigDma
(
    TDRV021_HANDLE      hdl,
    int                 BlockSize,
    int                 NumBlocks,
    int                 Channels
);
```

DESCRIPTION

This function configures the ADC channels for DMA usage. The BlockSize and the number of blocks define the size of the input FIFO, as well as the latency for incoming ADC samples.

PARAMETERS

hdl

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

BlockSize

This value specifies the size of one ADC sample block. This value affects the latency of the incoming ADC samples. The value is specified in bytes, and must be a multiple of a channel sample line (depends on the number of selected channels). The maximum value is 256KB. This value might be limited due to system restrictions.

NumBlocks

This value specifies the number of available ADC sample blocks. This value affects the total size of the ADC input FIFO.

Channels

This value specifies the channels enabled for DMA. Possible values are 4, 8, 12, or 16. The number of supported channels depends on the used TDRV021 module variant.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
```

```
TDRV021_STATUS     result;
int                AdcSamplesPerBlock;
int                BlockSize;
int                NumBlocks;
int                Channels;

/*
** Configure ADC DMA buffer list:
** Use 16 channels, 10 blocks, 100 samples per block
*/
AdcSamplesPerBlock = 100;
Channels           = 16;
BlockSize          = TDRV021_BLOCKSIZE( Channels, AdcSamplesPerBlock );
NumBlocks          = 10;

result = tdrv021AdcConfigDma( hdl, BlockSize, NumBlocks, Channels );
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.
TDRV021_ERR_NOMEM	Error allocating memory for DMA list.

3.3.3 tdrv021AdcConfigSampleClock

NAME

tdrv021AdcConfigSampleClock – Configure the Sample Clock

SYNOPSIS

```
TDRV021_STATUS tdrv021AdcConfigSampleClock
(
    TDRV021_HANDLE      hdl,
    unsigned int        TimeBase,
    unsigned int        TimerValue
)
```

DESCRIPTION

This function configures the Sample Clock frequency used for ADC sampling.

PARAMETERS

hdl

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

TimeBase

Specifies the Time Base for the Sample Clock timer. Following values are possible:

Value	Description
TDRV021_TIMEBASE_100NS	Time Base is 100ns
TDRV021_TIMEBASE_1US	Time Base is 1µs
TDRV021_TIMEBASE_1MS	Time Base is 1ms

TimerValue

This value specifies the number of TimeBase units, which must pass before a conversion is started.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
unsigned int       TimeBase;
unsigned int       TimerValue;


/*
** Configure Sample Clock to 10kHz (100us)
*/
TimeBase      = TDRV021_TIMEBASE_1US;
TimerValue    = 100;

result = tdrv021AdcConfigSampleClock( hdl, TimeBase, TimerValue );
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.

3.3.4 tdrv021AdcManualSingleValue

NAME

tdrv021AdcManualSingleValue – Read a single ADC channel

SYNOPSIS

```
TDRV021_STATUS tdrv021AdcManualSingleValue
(
    TDRV021_HANDLE      hdl,
    int                 Channel,
    unsigned 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 to 16. The number of supported channels depends on the used TDRV021 module variant.

Flags

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

Value	Description
TDRV021_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 “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
int                AdcValue;
unsigned int       Flags;

/*
** Read ADC Channel 1, wait for data
*/
Flags = TDRV021_ADCFLAG_WAITFORNEWDATA;

result = tdrv021AdcManualSingleValue( hdl, 1, Flags, &AdcValue );
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter
TDRV021_ERR_CHANNEL	The requested channel is not supported by the hardware module.

3.3.5 tdrv021AdcManualSingleSample

NAME

tdrv021AdcManualSingleSample – Read a single ADC Sample from all available channels

SYNOPSIS

```
TDRV021_STATUS tdrv021AdcManualSingleSample
(
    TDRV021_HANDLE      hdl,
    unsigned int        ChannelMask,
    unsigned 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 TDRV021 module variant.

Flags

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

Value	Description
TDRV021_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 16 ADC values.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
unsigned int       ChannelMask;
int                AdcBuf[16];
unsigned int       Flags;

/*
** Read ADC Channels 1, 2 and 16. Wait for data.
*/
ChannelMask = ((1 << 15) | (1 << 1) | (1 << 0));
Flags = TDRV021_ADCFLAG_WAITFORNEWDATA;

result = tdrv021AdcManualSingleSample( hdl, ChannelMask, Flags, AdcBuf );
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.
TDRV021_ERR_CHANNEL	At least one of the requested channels is not supported by the hardware module.

3.3.6 tdrv021AdcDmaSampleBlock

NAME

tdrv021AdcDmaSampleBlock – Read a sample block of ADC data

SYNOPSIS

```
TDRV021_STATUS tdrv021AdcDmaSampleBlock
(
    TDRV021_HANDLE      hdl,
    signed short        *pAdcBuf,
    size_t              numBytes,
    size_t              *validBytes
)
```

DESCRIPTION

This function reads a sample block of ADC data. The size of the block is configured using function tdrv021AdcConfigDma. In case of a FIFO overflow condition, the function return immediately without data. To enable the data sampling again after an overflow condition, it is required to use the function tdrv021AdcDmaFlush.

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	...

12 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#1	Ch#2	Ch#3	Ch#4
0x20	Ch#5	Ch#6	Ch#7	Ch#8	Ch#9	Ch#10	Ch#11	...

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 match the size previously configured using the function tdrv021AdcConfigDma.

validBytes

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

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
signed short       *pAdcBlock;
size_t             AdcBlockSize;
size_t             validBytes;

…
```

…

```
/*
** Read ADC Sample Block
*/

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

result = tdrv021AdcDmaSampleBlock(hdl, pAdcBlock, AdcBlockSize,
&validBytes);
if (result != TDRV021_OK)
{
     /* handle error */
}
free( pAdcBlock );
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.
TDRV021_ERR_BUSY	There is already a function waiting for a sample block.
TDRV021_ERR_TIMEOUT	Timeout occurred.
TDRV021_ERR_FIFOERROR	FIFO Overflow. Flush the DMA to receive new data.

3.3.7 tdrv021AdcDmaFlush

NAME

tdrv021AdcDmaFlush – Clears the ADC FIFO and reinitializes the DMA buffers

SYNOPSIS

```
TDRV021_STATUS tdrv021AdcDmaFlush
(
     TDRV021_HANDLE       hdl
)
```

DESCRIPTION

This function clears the ADC FIFO and reinitializes the DMA buffers. All previously received data is discarded. Using this function is required after a FIFO overflow, to unblock the data sampling.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;

/*
** Flush ADC Data
*/

result = tdrv021AdcDmaFlush(hdl);
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.

3.3.8 tdrv021AdcSetTimeout

NAME

tdrv021AdcSetTimeout – Configure the timeout used for ADC read jobs

SYNOPSIS

```
TDRV021_STATUS tdrv021AdcSetTimeout
(
      TDRV021_HANDLE      hdl,
      int                 DmaTimeout
)
```

DESCRIPTION

This function configures the ADC timeout used for transferring DMA blocks.

PARAMETERS

hdl

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

DmaTimeout

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

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;

/*
** Configure ADC DMA timeout to 5 seconds
*/
result = tdrv021AdcSetTimeout( hdl, 5000 );
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.

3.4 DAC Functions

3.4.1 tdrv021DacConfigOutput

NAME

tdrv021DacConfigOutput – Configure DAC Output

SYNOPSIS

```
TDRV021_STATUS tdrv021DacConfigOutput
(
    TDRV021_HANDLE      hdl,
    unsigned int        OutputRange,
    unsigned int        SampleMode
)
```

DESCRIPTION

This function configures the DAC Output Range and the Sample Mode.

PARAMETERS

hdl

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

OutputRange

Specifies the DAC Output Range. Following values are possible:

Value	Description
TDRV021_OUTPUTRANGE_BIPOL5V	Output Range: +/- 5 Volt
TDRV021_OUTPUTRANGE_BIPOL10V	Output Range: +/- 10 Volt
TDRV021_OUTPUTRANGE_UNIPOL5V	Output Range: 0 - 5 Volt
TDRV021_OUTPUTRANGE_UNIPOL10V	Output Range: 0 - 10 Volt

SampleMode

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

Value	Description
TDRV021_SAMPLEMODE_MANUAL	Sampling is started manually
TDRV021_SAMPLEMODE_SAMCLOCK	Sample Clock is used to start Conversion
TDRV021_SAMPLEMODE_EXTTRIGGER	External Trigger Input is used to start Conversion
TDRV021_SAMPLEMODE_INTSYNC	Internal Synchronization (use ADC trigger) ATTENTION: Do not configure the ADC for Internal Synchronization, if this mode is selected.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;

/*
** Configure the DAC for +/-5V, Sample with SampleClock
*/

result = tdrv021DacConfigOutput( hdl,
                                 TDRV021_OUTPUTRANGE_BIPOL5V,
                                 TDRV021_SAMPLEMODE_SAMCLOCK );
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	At least one of the specified parameters is invalid.

3.4.2 tdrv021DacConfigDma

NAME

tdrv021DacConfigDma – Configure DMA parameters for DAC

SYNOPSIS

```
TDRV021_STATUS tdrv021DacConfigDma
(
    TDRV021_HANDLE      hdl,
    int                 Channels,
    unsigned int        Flags
)
```

DESCRIPTION

This function configures DMA used for waveform output. Multiple DMA buffers might be concatenated to a descriptor list to output the whole waveform.

PARAMETERS

hdl

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

Channels

This value specifies the channels enabled for DMA. Possible values are 2, 4, 6, or 8. The number of supported channels depends on the used TDRV021 module variant.

Flags

This value specifies special behavior of the DAC. The following flags are defined and can be ORed:

Flags	Description
TDRV021_DACSUSPMODE	If this flag is set, there will be error announced if a DAC FIFO underrun occur, nor will the DAC update stop. The DAC will update with its current value until new data is supplied by DMA. If the flag is reset DAC underrun will be announced and the DAC update will stop. The DAC update must be re-enabled.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
int                Channels;
unsigned int       Flags;

/*
** Configure the DAC DMA mode
** use DAC Suspend on Frame End
*/
Channels           = 8;
Flags              = DACSUSPMODE;
result = tdrv021DacConfigDma( hdl, Channels, Flags );
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	At least one of the specified parameters is invalid.
TDRV021_ERR_CHANNEL	The requested number of channels is not supported by the hardware module.

3.4.3 tdrv021DacConfigSampleClock

NAME

tdrv021DacConfigSampleClock – Configure the Sample Clock

SYNOPSIS

```
TDRV021_STATUS tdrv021DacConfigSampleClock
(
    TDRV021_HANDLE      hdl,
    unsigned int        TimeBase,
    unsigned int        TimerValue
)
```

DESCRIPTION

This function configures the Sample Clock frequency used for DAC sampling.

PARAMETERS

hdl

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

TimeBase

Specifies the Time Base for the Sample Clock timer. Following values are possible:

Value	Description
TDRV021_TIMEBASE_100NS	Time Base is 100ns
TDRV021_TIMEBASE_1US	Time Base is 1µs
TDRV021_TIMEBASE_1MS	Time Base is 1ms

TimerValue

This value specifies the number of TimeBase units, which must pass before a conversion is started.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
unsigned int       TimeBase;
unsigned int       TimerValue;


/*
** Configure Sample Clock to 1kHz (1ms)
*/
TimeBase      = TDRV021_TIMEBASE_1MS;
TimerValue    = 1;

result = tdrv021DacConfigSampleClock( hdl, TimeBase, TimerValue );
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.

3.4.4 tdrv021DacManualSingleValue

NAME

tdrv021DacManualSingleValue – Write a single DAC channel

SYNOPSIS

```
TDRV021_STATUS tdrv021DacManualSingleValue
(
    TDRV021_HANDLE      hdl,
    int                 Channel,
    unsigned 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.

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 8. The number of supported channels depends on the used TDRV021 module variant.

Flags

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

Value	Description
TDRV021_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 “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
int                DacValue;
unsigned int       Flags;

/*
** Write to DAC Channel 4, initiate actual output of all DAC channels
*/
Flags     = TDRV021_DACFLAG_LOAD;
DacValue  = 0x7FFF;

result = tdrv021DacManualSingleValue( hdl, 4, Flags, DacValue );
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.
TDRV021_ERR_CHANNEL	The requested channel is not supported by the hardware module.

3.4.5 tdrv021DacManualSingleSample

NAME

tdrv021DacManualSingleSample – Writes a single DAC Sample to specified channels

SYNOPSIS

```
TDRV021_STATUS tdrv021DacManualSingleSample
(
    TDRV021_HANDLE      hdl,
    unsigned int        ChannelMask,
    unsigned 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.

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 TDRV021 module variant.

Flags

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

Value	Description
TDRV021_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 8 DAC values.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
unsigned int       ChannelMask;
int                DacBuf[8];
unsigned int       Flags;

/*
** Write DAC Channels 1, 2 and 4. Update the outputs.
*/
ChannelMask = ((1 << 3) | (1 << 1) | (1 << 0));
Flags = TDRV021_DACFLAG_LOAD;
DacBuf[0] = ...;
DacBuf[1] = ...;
DacBuf[3] = ...;

result = tdrv021DacManualSingleSample( hdl, ChannelMask, Flags, DacBuf );
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.
TDRV021_ERR_INVAL	At least one of the requested channels is not supported by the hardware module.

3.4.6 tdrv021DacDmaWaveform

NAME

tdrv021DacDmaWaveform – Write a DAC Waveform using DMA

SYNOPSIS

```
TDRV021_STATUS tdrv021DacDmaWaveform
(
    TDRV021_HANDLE      hdl,
    signed short        *pDacBuf,
    size_t              numBytes
)
```

DESCRIPTION

This function writes a waveform of DAC data to the configured outputs. The supplied data is transferred into separate DMA buffers.

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:

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	...

6 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#1	Ch#2
0x10	Ch#3	Ch#4	Ch#5	Ch#6	Ch#1	Ch#2	Ch#3	…

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 “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
unsigned int       ChannelMask;
signed short       *pDacBuf;
size_t             DacBlockSize;

/*
** Write DAC Waveform
*/

/* allocate memory */
DacBlockSize = ...;
pDacBuf = (signed short*)malloc( DacBlockSize );
/* fill data memory */
...
result = tdrv021DacDmaWaveform(hdl, pDacBuf, DacBlockSize);
if (result != TDRV021_OK)
{
     /* handle error */
}
free( pAdcBlock );
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.
TDRV021_ERR_NOMEM	Error allocating DMA memory

3.4.7 tdrv021DacDmaWaveformContinuous

NAME

tdrv021DacDmaWaveformContinuous – Continuously write a DAC Waveform using DMA

SYNOPSIS

```
TDRV021_STATUS tdrv021DacDmaWaveformContinuous
(
    TDRV021_HANDLE      hdl,
    signed short        *pDacBuf,
    size_t              numBytes
)
```

DESCRIPTION

This function writes a waveform of DAC data to the configured outputs. The supplied data is transferred into separate DMA buffers.

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.

numBytes

This value specifies the size of the data buffer in bytes. This value must match the size previously configured using the function tdrv021AdcConfigDma.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE      hdl;
TDRV021_STATUS      result;
unsigned int        ChannelMask;
signed short        *pDacBuf;
size_t              DacBlockSize;

…
```

```
…

/*
** Continuously write a DAC Waveform
*/

/* allocate memory */
DacBlockSize = ...;
pDacBuf = (signed short*)malloc( DacBlockSize );

result = tdrv021DacDmaWaveformContinuous(hdl, pDacBuf, DacBlockSize);
if (result != TDRV021_OK)
{
     /* handle error */
}
free( pDacBuf );
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.
TDRV021_ERR_NOMEM	Error allocating DMA memory

3.5 Timer Functions

3.5.1 tdrv021TimerStart

NAME

tdrv021TimerStart – Configure and start a timer

SYNOPSIS

```
TDRV021_STATUS tdrv021TimerStart
(
    TDRV021_HANDLE      hdl,
    int                 TimerNo,
    unsigned int        TimeBase,
    unsigned int        TimerValue
)
```

DESCRIPTION

This function configures and starts a timer.

PARAMETERS

hdl

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

TimerNo

This argument specifies the desired timer. Possible values are 1 and 2.

TimeBase

Specifies the Time Base for the timer. Following values are possible:

Value	Description
TDRV021_TIMEBASE_100NS	Time Base is 100ns
TDRV021_TIMEBASE_1US	Time Base is 1µs
TDRV021_TIMEBASE_1MS	Time Base is 1ms

TimerValue

This value specifies the number of TimeBase units, which must pass before the timer expires and raises an interrupt.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
int                TimerNo;
unsigned int       TimeBase;
unsigned int       TimerValue;


/*
** Configure Timer 1 for 1kHz (1ms)
*/
TimerNo       = 1;
TimeBase      = TDRV021_TIMEBASE_1MS;
TimerValue    = 1;

result = tdrv021TimerStart( hdl, TimerNo, TimeBase, TimerValue );
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.

3.5.2 tdrv021TimerStop

NAME

tdrv021TimerStop – Stop a previously started timer

SYNOPSIS

```
TDRV021_STATUS tdrv021TimerStop
(
    TDRV021_HANDLE      hdl,
    int                 TimerNo
)
```

DESCRIPTION

This function stops a previously started timer.

PARAMETERS

hdl

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

TimerNo

This argument specifies the desired timer. Possible values are 1 and 2.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
int                TimerNo;

…
```

```
…

/*
** Stop Timer 1
*/
TimerNo = 1;

result = tdrv021TimerStop( hdl, TimerNo );
if (result != TDRV021_OK)
{
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.

3.5.3 tdrv021TimerRead

NAME

tdrv021TimerRead – Read current counter value of a timer

SYNOPSIS

```
TDRV021_STATUS tdrv021TimerRead
(
    TDRV021_HANDLE      hdl,
    int                 TimerNo,
    unsigned int        *pTimerValue
)
```

DESCRIPTION

This function reads the current counter value of the specified timer.

PARAMETERS

hdl

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

TimerNo

This argument specifies the desired timer. Possible values are 1 and 2.

pTimerValue

This argument specifies a pointer to an unsigned int buffer, where the current counter value is stored.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE      hdl;
TDRV021_STATUS      result;
int                 TimerNo;
unsigned int        TimerValue;

…
```

…

```
/*
** Read current counter value of Timer 1
*/
TimerNo = 1;

result = tdrv021TimerRead( hdl, TimerNo, &TimerValue );
if (result == TDRV021_OK)
{
     printf("TimerValue = %d\n", TimerValue);
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.

3.5.4 tdrv021TimerWait

NAME

tdrv021TimerWait – Wait for an interrupt upon timer expiration

SYNOPSIS

```
TDRV021_STATUS tdrv021TimerWait
(
    TDRV021_HANDLE      hdl,
    int                 TimerNo,
    int                 timeout
)
```

DESCRIPTION

This function waits until the specified timer expires and generates an interrupt, or until the timeout occurs.

PARAMETERS

hdl

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

TimerNo

This argument specifies the desired timer. Possible values are 1 and 2.

timeout

This argument specifies the timeout in milliseconds.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE      hdl;
TDRV021_STATUS      result;
int                 TimerNo;

…
```

```
…

/*
** Wait for expiration of Timer 1, or timeout after 1 second
*/
TimerNo = 1;

result = tdrv021TimerWait( hdl, TimerNo, 1000 );
if (result == TDRV021_OK)
{
     printf(“Timer Interrupt\n”);
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.
TDRV021_ERR_TIMEOUT	The specified timeout occurred.

3.5.5 tdrv021TimerRegisterCallbackThread

NAME

tdrv021TimerRegisterCallbackThread – Register a User Callback Function for Timer Interrupt

SYNOPSIS

```
TDRV021_STATUS tdrv021TimerRegisterCallbackThread
(
    TDRV021_HANDLE      hdl,
    int                 threadPriority,
    int                 stackSize,
    int                 TimerNo,
    FUNCINTCALLBACK     callbackFunction,
    void                *funcparam,
    TDRV021_HANDLE      *pCallbackHandle
)
```

DESCRIPTION

This function registers a user callback function which is executed after the specified timer has expired and generated a corresponding interrupt. Multiple callback functions can be registered. The callback function is executed in a high-priority thread context, so using TDRV021 device driver functions is allowed. The callback function should be kept as short as possible. The specified callback function is executed with a status value passed to the callback function for proper error handling.

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
TDRV021_PRIORITY_NORMAL	Normal Thread Priority
TDRV021_PRIORITY_HIGH	High Thread Priority
TDRV021_PRIORITY_LOW	Low Thread Priority

Other values might be possible, depending on the used operating system.

threadSize

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

TimerNo

This argument specifies the desired timer. Possible values are 1 and 2.

callbackFunction

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

```
typedef void(*FUNCINTCALLBACK)(   TDRV021_HANDLE   hdl,
                                  unsigned int     InterruptOccurred,
                                  void             *param,
                                  TDRV021_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. Evaluating this value is not required for using timer interrupts.

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 timer interrupt has occurred properly, and no errors were detected, this parameter is TDRV021_OK. If this parameter differs from TDRV021_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 “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
int                TimerNo;
USER_DATA_AREA     userDataArea;
TDRV021_HANDLE     callbackHandle;

/* forward declaration of callback functions */
void callback_TIMER(    TDRV021_HANDLE     hdl,
                        unsigned int       interruptOccurred,
                        void               *param,
                        TDRV021_STATUS     status );

/*
**  Register callback function for Timer 1.
**  Use a “normal” priority, and 64KB stack.
*/
TimerNo = 1;
result = tdrv021TimerRegisterCallbackThread( hdl,
                                             TDRV021_PRIORITY_NORMAL,
                                             0x10000,
                                             TimerNo,
                                             callback_TIMER,
                                             &userDataArea,
                                             &callbackHandle );
...
...
if (result != TDRV021_OK)
{
     /* handle error */
}

…
```

```
…

/*
** Callback Function, using API Functions
*/
void callback_TIMER(    TDRV021_HANDLE     hdl,
                        unsigned int       interruptOccurred,
                        void          *param,
                        TDRV021_STATUS     status )
{
     TDRV021_STATUS     result;
     USER_DATA_AREA     *pUsrData = (USER_DATA_AREA*)param;
     int                Channel;
     int                Flags;
     static int         DacValue = 0;

     if (status != TDRV021_OK)
     {
          /* handle error status */
     }
     /* Write and update a specific DAC Output */
     Channel  = 1;
     Flags    = TDRV021_DACFLAG_LOAD;
     DacValue  += 100;
     if (DacValue > 0x3FFF) DacValue = 0;
     result = tdrv021DacManualSingleValue( hdl,
                                           Channel,
                                           Flags,
                                           DacValue );
     /* handle errors */
     return;
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVALID_PARAMETER	Function or callback handle pointer is NULL.

3.5.6 tdrv021TimerUnregisterCallback

NAME

tdrv021TimerUnregisterCallback – Unregister a User Callback Function

SYNOPSIS

```
TDRV021_STATUS tdrv021TimerUnregisterCallback
(
     TDRV021_HANDLE       cbHdl
)
```

DESCRIPTION

This function unregisters a previously registered user callback function.

PARAMETERS

cbHdl

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

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE          callbackHdl;
TDRV021_STATUS          result;

/*
**  Unregister a callback function
*/
result = tdrv021TimerUnregisterCallback( callbackHdl );
if (result == TDRV021_OK)
{
     / *OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified callback handle is invalid.

Other returned error codes are system error conditions.

3.6 Interrupt Functions

3.6.1 tdrv021InterruptWait

NAME

tdrv021InterruptWait – Wait for incoming Local Interrupt Source

SYNOPSIS

```
TDRV021_STATUS tdrv021InterruptWait
(
    TDRV021_HANDLE      hdl,
    uint32_t            interruptMask,
    uint32_t            *pInterruptOccurred,
    int                 timeout
)
```

DESCRIPTION

This function waits for the first occurrence of one of the specified Local Interrupt Sources. The function will return immediately, if there had been an occurrence of a specified interrupt since the last clear of the pending event e.g. by tdrv021ReadClearOccurredInterrupt().

The specified Local Interrupt Sources must be enabled using the function tdrv021InterruptEnable().

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” bits described in the hardware user manual. The function returns if at least one of the specified interrupt sources is detected. The following values are possible:

Value	Description
TDRV021_INTERRUPT_ADCCONV	Interrupt after ADC Conversion
TDRV021_INTERRUPT_ADCFIFOERR	ADC FIFO Error Interrupt
TDRV021_INTERRUPT_ADCTRIG	ADC Trigger Interrupt
TDRV021_INTERRUPT_DACSET	Interrupt after DAC Settle
TDRV021_INTERRUPT_DACAL	DAC Alert Interrupt
TDRV021_INTERRUPT_DACFIFOERR	DAC FIFO Error Interrupt
TDRV021_INTERRUPT_DACTRIG	DAC Trigger Interrupt
TDRV021_INTERRUPT_TIME1	Interval Timer 1 Interrupt
TDRV021_INTERRUPT_TIME2	Interval Timer 2 Interrupt
TDRV021_INTERRUPT_ADCDMADONE	ADC DMA Descriptor Complete Interrupt
TDRV021_INTERRUPT_ADCDMAABORT	ADC DMA Abort Received Interrupt
TDRV021_INTERRUPT_DACDMADONE	DAC DMA Descriptor Complete Interrupt
TDRV021_INTERRUPT_DACDMAABORT	DAC DMA Abort Received Interrupt
TDRV021_INTERRUPT_DACFRAME	DAC Frame Interrupt

pInterruptOccurred

If at least one of the specified interrupt sources occurs, the value is returned through this pointer.

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 “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
unsigned int       interruptMask;
unsigned int       interruptOccurred;

…
```

…

```
/*
** Wait at least 5 seconds for DAC Error Interrupt, indicating that the
last
** DAC value has been written.
*/
interruptMask = TDRV021_INTERRUPT_DACFIFOERR;
result = tdrv021InterruptWait(   hdl,
                                 interruptMask,
                                 &interruptOccurred,
                                 5000 );
if (result == TDRV021_OK)
{
     /* Interrupt arrived    */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_TIMEOUT	The specified timeout occurred.
TDRV021_ERR_INVAL	Invalid Parameter.

3.6.2 tdrv021ReadClearOccurredInterrupt

NAME

tdrv021ReadClearOccurredInterrupt – Read and clear mask of occurred Local Interrupt Source

SYNOPSIS

```
TDRV021_STATUS tdrv021ReadClearOccurredInterrupt
(
    TDRV021_HANDLE      hdl,
    uint32_t            interruptMask,
    uint32_t            *pInterruptOccurred
)
```

DESCRIPTION

This function reads a mask of Local Interrupt Sources that have occurred and clears the corresponding pending flags.

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 be read and cleared. The interrupt bits correspond to the “Interrupt Status Register” bits described in the hardware user manual. The following values are possible:

Value	Description
TDRV021_INTERRUPT_ADCCONV	Interrupt after ADC Conversion
TDRV021_INTERRUPT_ADCFIFOERR	ADC FIFO Error Interrupt
TDRV021_INTERRUPT_ADCTRIG	ADC Trigger Interrupt
TDRV021_INTERRUPT_DACSET	Interrupt after DAC Settle
TDRV021_INTERRUPT_DACAL	DAC Alert Interrupt
TDRV021_INTERRUPT_DACFIFOERR	DAC FIFO Error Interrupt
TDRV021_INTERRUPT_DACTRIG	DAC Trigger Interrupt
TDRV021_INTERRUPT_TIME1	Interval Timer 1 Interrupt
TDRV021_INTERRUPT_TIME2	Interval Timer 2 Interrupt
TDRV021_INTERRUPT_ADCDMADONE	ADC DMA Descriptor Complete Interrupt
TDRV021_INTERRUPT_ADCDMAABORT	ADC DMA Abort Received Interrupt
TDRV021_INTERRUPT_DACDMADONE	DAC DMA Descriptor Complete Interrupt
TDRV021_INTERRUPT_DACDMAABORT	DAC DMA Abort Received Interrupt
TDRV021_INTERRUPT_DACFRAME	DAC Frame Interrupt

pInterruptOccurred

If at least one of the specified interrupt sources has generated an event, the corresponding flag is returned through this pointer.

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
unsigned int       interruptMask;
unsigned int       interruptOccurred;

…
```

…

```
/*
** Read and Clear occurred DAC Error Interrupt and DAC ALERT Interrupt
event
*/
interruptMask = TDRV021_INTERRUPT_DACFIFOERR | TDRV021_INTERRUPT_DACAL;
result = tdrv021ReadClearOccurredInterrupt (    hdl,
                                                interruptMask,
                                                &interruptOccurred );
if (result == TDRV021_OK)
{
     /* pending interrupt events cleard    */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.

3.6.3 tdrv021InterruptEnable

NAME

tdrv021InterruptEnable – Enable Local Interrupt Source

SYNOPSIS

```
TDRV021_STATUS tdrv021InterruptEnable
(
    TDRV021_HANDLE      hdl,
    uint32_t            interruptMask
)
```

DESCRIPTION

This function enables the specified local interrupt sources.

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. The interrupt bits correspond to the “Interrupt Status Register” bits described in the hardware user manual. The following values are possible:

Value	Description
TDRV021_INTERRUPT_ADCCONV	Interrupt after ADC Conversion
TDRV021_INTERRUPT_ADCFIFOERR	ADC FIFO Error Interrupt
TDRV021_INTERRUPT_ADCTRIG	ADC Trigger Interrupt
TDRV021_INTERRUPT_DACSET	Interrupt after DAC Settle
TDRV021_INTERRUPT_DACAL	DAC Alert Interrupt
TDRV021_INTERRUPT_DACFIFOERR	DAC FIFO Error Interrupt
TDRV021_INTERRUPT_DACTRIG	DAC Trigger Interrupt
TDRV021_INTERRUPT_TIME1	Interval Timer 1 Interrupt
TDRV021_INTERRUPT_TIME2	Interval Timer 2 Interrupt
TDRV021_INTERRUPT_ADCDMADONE	ADC DMA Descriptor Complete Interrupt
TDRV021_INTERRUPT_ADCDMAABORT	ADC DMA Abort Received Interrupt
TDRV021_INTERRUPT_DACDMADONE	DAC DMA Descriptor Complete Interrupt
TDRV021_INTERRUPT_DACDMAABORT	DAC DMA Abort Received Interrupt
TDRV021_INTERRUPT_DACFRAME	DAC Frame Interrupt

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
unsigned int       interruptMask;

/*
** Enable the DAC Error Interrupt, indicating that the last
** DAC value has been written.
*/
interruptMask = TDRV021_INTERRUPT_DACFIFOERR;
result = tdrv021InterruptEnable( hdl,
                                 interruptMask );
if (result == TDRV021_OK)
{
     /* Interrupt(s) enables     */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.

3.6.4 tdrv021InterruptDisable

NAME

tdrv021InterruptDisable – Disable Local Interrupt Source

SYNOPSIS

```
TDRV021_STATUS tdrv021InterruptDisable
(
    TDRV021_HANDLE      hdl,
    uint32_t            interruptMask
)
```

DESCRIPTION

This function disables the specified local interrupt sources.

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. The interrupt bits correspond to the “Interrupt Status Register” bits described in the hardware user manual. The following values are possible:

Value	Description
TDRV021_INTERRUPT_ADCCONV	Interrupt after ADC Conversion
TDRV021_INTERRUPT_ADCFIFOERR	ADC FIFO Error Interrupt
TDRV021_INTERRUPT_ADCTRIG	ADC Trigger Interrupt
TDRV021_INTERRUPT_DACSET	Interrupt after DAC Settle
TDRV021_INTERRUPT_DACAL	DAC Alert Interrupt
TDRV021_INTERRUPT_DACFIFOERR	DAC FIFO Error Interrupt
TDRV021_INTERRUPT_DACTRIG	DAC Trigger Interrupt
TDRV021_INTERRUPT_TIME1	Interval Timer 1 Interrupt
TDRV021_INTERRUPT_TIME2	Interval Timer 2 Interrupt
TDRV021_INTERRUPT_ADCDMADONE	ADC DMA Descriptor Complete Interrupt
TDRV021_INTERRUPT_ADCDMAABORT	ADC DMA Abort Received Interrupt
TDRV021_INTERRUPT_DACDMADONE	DAC DMA Descriptor Complete Interrupt
TDRV021_INTERRUPT_DACDMAABORT	DAC DMA Abort Received Interrupt
TDRV021_INTERRUPT_DACFRAME	DAC Frame Interrupt

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
unsigned int       interruptMask;

/*
** Disable the DAC Error Interrupt
*/
interruptMask = TDRV021_INTERRUPT_DACFIFOERR;
result = tdrv021InterruptDisable (    hdl,
                                      interruptMask );
if (result == TDRV021_OK)
{
     /* Interrupt(s) disabled     */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVAL	Invalid Parameter.

3.6.5 tdrv021InterruptRegisterCallbackThread

NAME

tdrv021InterruptRegisterCallbackThread – Register a User Callback Function for Interrupt Handling

SYNOPSIS

```
TDRV021_STATUS tdrv021InterruptRegisterCallbackThread
(
    TDRV021_HANDLE      hdl,
    int                 threadPriority,
    int                 stackSize,
    unsigned int        interruptMask,
    FUNCINTCALLBACK     callbackFunction,
    void                *funcparam,
    TDRV021_HANDLE      *pCallbackHandle
)
```

DESCRIPTION

This function registers a user callback function which is executed after detection of the specified interrupt source. Multiple callback functions can be registered to each interrupt source. The callback function is executed in a high-priority thread context, so using TDRV021 device driver 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 for proper error handling.

This function can be useful to handle FIFO or DMA errors, or to wait for external trigger interrupts.

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
TDRV021_PRIORITY_NORMAL	Normal Thread Priority
TDRV021_PRIORITY_HIGH	High Thread Priority
TDRV021_PRIORITY_LOW	Low Thread Priority

Other values might be possible, depending on the used operating system.

threadSize

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 "Interrupt Status Register" bits described in the hardware user manual. The callback function is executed if at least one of the specified interrupt sources occurred. Following values are possible:

Value	Description
TDRV021_INTERRUPT_ADCCONV	Interrupt after ADC Conversion
TDRV021_INTERRUPT_ADCFIFOERR	ADC FIFO Error Interrupt
TDRV021_INTERRUPT_ADCTRIG	ADC Trigger Interrupt
TDRV021_INTERRUPT_DACSET	Interrupt after DAC Settle
TDRV021_INTERRUPT_DACAL	DAC Alert Interrupt
TDRV021_INTERRUPT_DACFIFOERR	DAC FIFO Error Interrupt
TDRV021_INTERRUPT_DACTRIG	DAC Trigger Interrupt
TDRV021_INTERRUPT_TIME1	Interval Timer 1 Interrupt
TDRV021_INTERRUPT_TIME2	Interval Timer 2 Interrupt
TDRV021_INTERRUPT_ADCDMADONE	ADC DMA Descriptor Complete Interrupt
TDRV021_INTERRUPT_ADCDMAABORT	ADC DMA Abort Received Interrupt
TDRV021_INTERRUPT_DACDMADONE	DAC DMA Descriptor Complete Interrupt
TDRV021_INTERRUPT_DACDMAABORT	DAC DMA Abort Received Interrupt

callbackFunction

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

```
typedef void(*FUNCINTCALLBACK)( TDRV021_HANDLE   hdl,
                                unsigned int     interruptOccurred,
                                void             *param,
                                TDRV021_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" bits described in the hardware 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 TDRV021_OK. If this parameter differs from TDRV021_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 “tdrv021api.h”

TDRV021_HANDLE     hdl;
TDRV021_STATUS     result;
unsigned int       interruptMask;
USER_DATA_AREA     userDataArea;
TDRV021_HANDLE     callbackHandle;

/* forward declaration of callback functions */
void callback_TRIGGER(  TDRV021_HANDLE     hdl,
                        unsigned int       interruptOccurred,
                        void               *param,
                        TDRV021_STATUS     status );

/*
**  Register callback function to handle ADC and DAC trigger inputs.
**  Use a “normal” priority, and 64KB stack.
*/
interruptMask = (TDRV021_INTERRUPT_DACTRIG | TDRV021_INTERRUPT_ADCTRIG);
result = tdrv021InterruptRegisterCallbackThread( hdl,
                                                 TDRV021_PRIORITY_NORMAL,
                                                 0x10000,
                                                 interruptMask,
                                                 callback_TRIGGER,
                                                 &userDataArea,
                                                 &callbackHandle );

…
```

```
…

if (result != TDRV021_OK)
{
     /* handle error */
}

...

/*
** Callback Function
*/
void callback_TRIGGER(  TDRV021_HANDLE     hdl,
                        unsigned int       interruptOccurred,
                        void          *param,
                        TDRV021_STATUS     status )
{
     TDRV021_STATUS     result;
     USER_DATA_AREA     *pUsrData = (USER_DATA_AREA*)param;


     if (status != TDRV021_OK)
     {
          /* handle error status */
     }

     /* Handle ADC Trigger Input */
     if (interruptOccurred & TDRV021_INTERRUPT_ADCTRIG)
     {
          /* use API functions, e.g. to read ADC data */
          ...
     }

     /* Handle DAC Trigger Input */
     if (interruptOccurred & TDRV021_INTERRUPT_DACTRIG)
     {
          /* uase API functions, e.g. to write DAC data */
          ...
     }

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

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified TDRV021_HANDLE is invalid.
TDRV021_ERR_INVALID_PARAMETER	Function or callback handle pointer is NULL.

3.6.6 tdrv021InterruptUnregisterCallback

NAME

tdrv021InterruptUnregisterCallback – Unregister a User Callback Function

SYNOPSIS

```
TDRV021_STATUS tdrv021InterruptUnregisterCallback
(
     TDRV021_HANDLE       hdl
)
```

DESCRIPTION

This function unregisters a previously registered user callback function.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tdrv021api.h”

TDRV021_HANDLE          callbackHdl;
TDRV021_STATUS          result;

/*
**  Unregister a callback function
*/
result = tdrv021InterruptUnregisterCallback( callbackHdl );
if (result == TDRV021_OK)
{
     / *OK */
} else {
     /* handle error */
}
```

RETURNS

On success, TDRV021_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV021_ERR_INVALID_HANDLE	The specified callback handle is invalid.

Other returned error codes are system error conditions.

4 Appendix

4.1 Hibernate Mode

The driver supports the “Hibernate”-mode of Windows.

The driver will remember the basic ADC and DAC configuration and will restore this configuration immediately after leaving the hibernation mode. DMA configuration will not be reconfigured and must be updated after hibernation mode before starting a DMA transfer again.

If there are pending requests and the system wants to enter hibernation mode, the requests will be aborted with an appropriate error code. The application may now decide how to handle this situation.

4.2 Data Correction

Because of special hardware configurations on some of the supported boards (e.g. TPMC520-10) the returned ADC input values are incorrect (by factor x) if data correction is disabled. This factor is automatically corrected by hardware if data correction is enabled (default). Therefore we recommend keeping the data correction enabled.