

TQMC500-SW-82

Linux Device Driver

20 Isolated Analog Inputs, simultaneous sampling

Version 1.0.x

User Manual

Issue 1.0.0

September 2026

TQMC500-SW-82

Linux Device Driver

20 Isolated Analog Inputs, simultaneous sampling

Supported Modules:

TQMC500

This document contains information, which is proprietary to TEWS Technologies GmbH. Any reproduction without written permission is forbidden.

TEWS Technologies GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS Technologies GmbH reserves the right to change the product described in this document at any time without notice.

TEWS Technologies GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2026 by TEWS Technologies GmbH

Issue	Description	Date
1.0.0	First Issue	11.09.2026

Table of Contents

1	INTRODUCTION.....	4
2	INSTALLATION.....	5
2.1	Build and Install the Device Driver.....	5
2.2	Uninstall the Device Driver.....	6
2.3	Load the Device Driver Into the Running Kernel.....	6
2.4	Remove Device Driver From the Running Kernel.....	7
2.5	Change Major Device Number.....	7
2.6	Change the Transfer Mode.....	7
3	FUNCTIONAL DESCRIPTION	8
3.1	Transfer Mode.....	8
3.2	Transfer Sizes.....	8
3.3	Evaluation of read access methods.....	9
4	API DOCUMENTATION	10
4.1	General Functions.....	10
4.1.1	tqmc500Open.....	10
4.1.2	tqmc500Close	12
4.1.3	tqmc500GetModuleInfo.....	14
4.1.4	tqmc500GetBoardTemperature	17
4.2	Configuration Functions	19
4.2.1	tqmc500SetRange.....	19
4.2.2	tqmc500SetSampleClock.....	22
4.2.3	tqmc500SetOversampling.....	24
4.2.4	tqmc500SetTransferSize.....	26
4.2.5	tqmc500ResetAdc	28
4.3	Input Functions	30
4.3.1	tqmc500ReadSample.....	30
4.3.2	tqmc500ReadVoltage.....	32
4.3.3	tqmc500ReadSequence.....	34
4.3.4	tqmc500ReadVoltageSequence	37
4.3.5	tqmc500ReadContinuous.....	40
4.3.6	tqmc500ReadVoltageContinuous	43
4.3.7	tqmc500StopContinuous.....	47
4.3.8	tqmc500ReadFrame	49
4.3.9	tqmc500ReadFrameVoltage	51
4.4	Data Correction	54
4.4.1	tqmc500SetTemporaryCorrectionValues.....	54
4.4.2	tqmc500GetFactoryCorrectionValues	56
5	DIAGNOSTIC.....	58
5.1	Debugging Logs.....	58
5.2	Read Information From Kernel	59
5.2.1	Check if your device is recognized at the PCI bus.....	59
5.2.2	Check if the device driver is loaded and in use.....	59
5.2.3	Check if your device is accessible in the filesystem	59

1 Introduction

The TQMC500-SW Device Driver provides support for the TEWS Technologies TQMC500 module.

In addition to the low-level device driver, an Application Programming Interface (API) is provided for hardware access.

The purpose of this document is to describe the device driver functionality, especially the Application Programming Interface.

The TQMC500-SW device driver supports the following features:

- configuration of sample rate, input range and polarity
- reading single analog inputs
- simultaneously readout all analog inputs
- capturing a coherent sample block of a selection of inputs
- continuously reading out all inputs
- configurable DMA functionality
- configurable FIFO functionality in case DMA is unfavorable

The TQMC500-SW Device Driver supports the modules listed below:

TQMC500	20 Isolated Analog Inputs	QMC
---------	---------------------------	-----

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

TQMC500 Hardware User Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TQMC500-SW-82':

TQMC500-SW-82-1.0.0.pdf	This manual in PDF format
TQMC500-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

The GZIP compressed archive TQMC500-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tqmc500/':

tqmc500.c	Driver source code
tqmc500def.h	Driver include file
tqmc500.h	Driver include file for application program
dmaset.c	DMA buffer handling library
Makefile	Device driver make file
makenode	Script for device node creation
COPYING	Copy of the GNU Public License (GPL)
api/tqmc500api.h	API include file
api/tqmc500api.c	API source file
include/config.h	Include of system dependent config.h
include/tpxxxhwdep.c	Low level hardware access functions source file
include/tpxxxhwdep.h	Access functions header file
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
example/tqmc500exa.c	Example application
example/Makefile	Example application makefile

In order to perform an installation, extract all files of the archive TQMC500-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzf TQMC500-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tqmc500.h and api/tqmc500api.h to */usr/include*

2.1 Build and Install the Device Driver

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

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

 # depmod -a

2.2 Uninstall the Device Driver

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

 # make uninstall

2.3 Load the Device Driver Into the Running Kernel

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

 # modprobe tqmc500drv
- Skip this step if your kernel has enabled a dynamic device file system (devfs or sysfs with udev). After the first build or if you are using dynamic major device allocation it is necessary to create new device nodes on the file system. Please execute the script file *makenode* to do so. If a dynamic device file system is active, the driver will take care of the creation and destruction of device nodes.

 # sh makenode

On success the device driver will create a minor device for each TQMC500 module found. The first module of the first TQMC500 module can be accessed with device node */dev/tqmc500_0*, the second module with device node */dev/tqmc500_1*, the third TQMC500 module with device node */dev/tqmc500_2* and so on. The assignment of device nodes to physical TQMC500 modules depends on the search order of the PCI bus driver.

2.4 Remove Device Driver From the Running Kernel

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

```
# modprobe -r tqmc500drv
```

If your kernel has enabled devfs or sysfs (udev), all /dev/tqmc500_x nodes will be automatically removed from your file system after this.

Be sure that the driver isn't opened by any application program. If opened you will get the response "*tqmc500drv: Device or resource busy*" and the driver will remain loaded in the system until you close all opened files and execute *modprobe -r* again.

2.5 Change Major Device Number

This paragraph is only for Linux kernels without dynamic device management. The TQMC500 driver uses dynamic allocation of major device numbers per default. If this isn't suitable for the application it's possible to define a major number for the driver.

To change the major number, edit the file tqmc500def.h, change the following symbol to appropriate value and enter make install to create a new driver.

TQMC500_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.
---------------	---

Example:

```
#define TQMC500_MAJOR 122
```

Be sure that the desired major number is not used by other drivers. Please check /proc/devices to see which numbers are free.

Keep in mind that it is necessary to create new device nodes if the major number for the TQMC500 driver has changed and the makenode script is not used.

2.6 Change the Transfer Mode

The device driver uses DMA transfers for fast and efficient sample acquisition by default. Optionally, a FIFO and I/O register-driven access mode may be chosen. The information in chapter 3.1 Transfer Mode provide a guideline for maximum achievable sample rates and should be considered before switching the transfer mode to FIFO.

In order to compile the device driver with FIFO mode enabled, change the following line in the source file *tqmc500def.h*:

```
#define TQMC500_TRANSFER_MODE TRANSFER_MODE_DMA
```

FIFO mode is enabled by changing the definition to TRANSFER_MODE_FIFO.

3 Functional Description

3.1 Transfer Mode

Users may choose between two data transfer modes, which are used to transfer samples from the input unit to the user application: DMA and FIFO mode.

DMA mode uses the onboard DMA controller to transfer data directly into host RAM without CPU intervention. In this mode, a software buffer will be allocated and all generated samples will be transferred into the buffer as soon as data acquisition is started. This mode generally promises the best results in terms of transfer rates, CPU usage and maximum possible sampling frequency. DMA mode is used by default and should always be chosen in systems that have no relevant constraint regarding the additional host memory usage and PCI bus master that is implied in this mode.

FIFO mode offers an alternative to DMA mode in systems, where usage of the onboard DMA controller is unfavorable or the allocation of the needed buffers in host RAM for DMA is impracticable. Unlike in DMA mode, in FIFO mode only the samples of the requested channels are read from the input unit and are transferred directly to the user space buffer.

Sampled values are read from the respective data registers, which takes significantly longer than DMA transfer. Sample rates and the number of observed channels should be chosen with care and in accordance with the used hardware and system load. Even under best circumstances, operation near the maximum sample rate cannot be expected to produce satisfactory results when operating in FIFO mode. As a rule of thumb, 1 μ s per channel should be calculated with when estimating the sampling time. Use channels that reside on the same data register to optimize throughput, e.g. channels 0 & 1, channels 2 & 3 and so on. For example, when observing 8 channels, choose a sample rate lower than 100 kSPS and use channels 0 – 7. If a higher number of channels or sample rate is needed, lower the sample rate or consider changing to DMA mode. In any case, system benchmarking is required before using a high sample rate with FIFO mode in a productive environment.

3.2 Transfer Sizes

The transfer size is a software-modifiable parameter for the buffered access modes. It defines the number of samples transferred before an interrupt is generated at the host system. Modifying the transfer size also changes the latency between the acquisition of a sample and the interrupt, that is generated to signal the readiness of the buffer, which contains it. Consequently, transfer sizes may be chosen in accordance with the sample rate to produce a suitable tradeoff between sample latency and system load. A higher transfer size results in lower system load, but also increases the latency. If buffered accesses are terminated with errors indicating buffer overflows, and lowering the sample rate is not an option, transfer sizes should be increased to buy more time for the host system to process buffers.

Furthermore, transfer sizes must also match the selected transfer mode as not all sizes are supported by both modes. A maximum transfer size of 512 samples is supported, when operating in FIFO mode. In DMA mode, all defined transfer sizes are available, which range from 128 to 8192 samples.

Note, that the selection of the transfer size has no influence on the latency when acquiring samples by single access or frame access.

3.3 Evaluation of read access methods

The device driver offers various API functions for the purpose of reading analog voltage data from the device input. The right function may be chosen depending on the use case to achieve best tradeoff between performance and handling simplicity.

In the most simple case, a user may want to read the current voltage level from one of the board inputs. For this purpose, `tqmc500ReadSample()` or `tqmc500ReadVoltage()` may be used. Both functions read the current analog level from one single selected input. Additional to `tqmc500ReadSample()`, `tqmc500ReadVoltage()` uses information about the currently selected input range to calculate the corresponding voltage with the data retrieved from the ADC channel. Multiple consecutive calls of `tqmc500ReadSample()` and `tqmc500ReadVoltage()` always return the newest sample from the device, but the samples are not guaranteed to be coherent.

For the purpose of acquiring a sequence of coherent samples, `tqmc500ReadSequence()` or `tqmc500ReadVoltageSequence()` may be called. A sequence is always limited to a defined number of samples, and the acquisition does not return the buffer before all requested samples have been collected.

If continuous data sampling is needed without a predetermined number of samples, `tqmc500ReadContinuous()` and `tqmc500ReadVoltageContinuous()` offer the respective functionality for that. Both functions need to be called periodically to collect data from the buffer, which is steadily filled until `tqmc500StopContinuous()` is called or a buffer overflow occurs.

In cases, where the exact moment of the data conversion is not relevant, but the samples of different channels all need to be sampled simultaneously, `tqmc500ReadFrame()` and `tqmc500ReadFrameVoltage()` can be used. The samples returned by these functions all belong to the same conversion pulse and were thus sampled at the same time.

All the sequential and continuous read functions use one of the buffer types described in chapter 3.1 Transfer Mode. The functions for single channel and frame data acquisition return the data directly without buffering.

4 API Documentation

4.1 General Functions

4.1.1 tqmc500Open

NAME

tqmc500Open – open a device

SYNOPSIS

```
TQMC500_HANDLE tqmc500Open  
(  
    char      *deviceName  
)
```

DESCRIPTION

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

PARAMETERS

deviceName

This parameter points to a null-terminated string that specifies the name of the device. The first device is named “/dev/tqmc500_0” the second device is named “/dev/tqmc500_1” and so on.

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE  hdl;

/*
** open the specified device
*/
hdl = tqmc500Open("/dev/tqmc500_0");
if (hdl == NULL)
{
    /* handle open error */
}
```

RETURNS

A device handle, or NULL if the function fails.

ERROR CODES

The error codes are stored in *errno*. The error code is a standard error code set by the I/O system.

4.1.2 tqmc500Close

NAME

tqmc500Close – close a device

SYNOPSIS

```
TQMC500_STATUS tqmc500Close  
(  
    TQMC500_HANDLE    hdl  
)
```

DESCRIPTION

This function closes a previously opened device. It will succeed only if there is no data transfer in progress.

PARAMETERS

hdl

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

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE    hdl;
TQMC500_STATUS    result;

/*
** close the device
*/

result = tqmc500Close(hdl);
if (result != TQMC500_OK)
{
    /* handle close error */
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.

4.1.3 tqmc500GetModuleInfo

NAME

tqmc500GetModuleInfo – get information about the hardware module

SYNOPSIS

```
TQMC500_STATUS tqmc500GetModuleInfo
(
    TQMC500_HANDLE          hdl,
    TQMC500_PCIINFO_BUF     *pPciInfoBuf,
    TQMC500_BOARDINFO_BUF   *pBoardInfoBuf,
    int                     *transferMode,
    unsigned int             *transferSize
)
```

DESCRIPTION

This function returns information about the hardware module, including number of channels and PCI header as well as the PCI localization.

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 TQMC500_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;
} TQMC500_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

pBoardInfoBuf

This argument is a pointer to the structure TQMC500_BOARDINFO_BUF that receives information about the module hardware.

typedef struct

```
{
    unsigned int          firmwareVersion;
    unsigned int          numAdcChannels;
    TQMC500_SAMPLECLOCK_BUF clockInfo;
    unsigned int          oversamplingRate;
} TQMC500_BOARDINFO_BUF;
```

firmwareVersion

This value returns the version of the firmware used on the device.

numAdcChannels

This value returns the number of analog input lines available on the specified device.

clockInfo

This struct holds information about the currently configured sample clock.

oversamplingRate

This value returns the configured oversampling rate.

transferMode

This variable will receive a value representing the selected transfer mode. A value of 1 represents DMA mode, whereas a value of 2 represents FIFO mode.

transferSize

This variable will receive the number of samples that is currently used as buffer transfer size. Possible values are listed below the description of `tqmc500SetTransferSize()`.

EXAMPLE

```
#include "tqmc500api.h"
```

```
TQMC500_HANDLE    hdl;
TQMC500_STATUS    result;
TQMC500_PCIINFO_BUF  pciInfoBuf;
TQMC500_BOARDINFO_BUF boardInfoBuf;
```

```
/*
** get module PCI information
*/
```

```
result = tqmc500GetModuleInfo( hdl, &pciInfoBuf, &boardInfoBuf );
if (result != TQMC500_OK)
{
    /* handle error */
}
```

```
printf ("Input-Lines:   %d", boardInfoBuf.numAdcChannels);
printf ("Firmware:     %0X", boardInfoBuf.firmwareVersion);
printf ("PCI ID:         %04X:%04X", pciInfoBuf.vendorId, pciInfoBuf.deviceId);
```

RETURN VALUE

On success, `TQMC500_OK` is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
<code>TQMC500_ERR_INVALID_HANDLE</code>	The specified device handle is invalid.

4.1.4 tqmc500GetBoardTemperature

NAME

tqmc500GetBoardTemperature – read the current onboard FPGA temperature

SYNOPSIS

```
TQMC500_STATUS tqmc500SetRange
(
    TQMC500_HANDLE    hdl,
    double             *temperature
);
```

DESCRIPTION

Read the on-chip FPGA temperature from the device and store the value to a variable of type double.

PARAMETERS

hdl

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

temperature

The variable pointed to by this pointer will receive the value of the currently measured temperature.

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE    hdl;
TQMC500_STATUS    result;
double            temperature;

/*
** Read the current temperature from the device.
*/

result = tqmc500GetBoardTemperature( hdl, &temperature );
if (result != TQMC500_OK)
{
    /* handle error */
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.

4.2 Configuration Functions

4.2.1 tqmc500SetRange

NAME

tqmc500SetRange – configure the input range of given ADC channels

SYNOPSIS

```
TQMC500_STATUS tqmc500SetRange
(
    TQMC500_HANDLE    hdl,
    unsigned int       channelMask,
    TQMC500_RANGE     range
);
```

DESCRIPTION

This function may be used to set the input range of multiple selected ADC channels. Ranges and polarity may be set for each channel individually.

Please note, that any function, which configures an ADC will halt the data acquisition for a short period of time. Thus, configuration should generally take place before samples are requested.

PARAMETERS

hdl

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

channelMask

Use the channel mask to select the channels of interest. Each bit of the mask corresponds to one channel of the device. The first bit corresponds to the first channel of the device, the second bit to the second channel, and so on. A set bit, which has no corresponding channel on the device will cause the function to fail.

range

The range to be configured on the selected inputs. See the following table for available input ranges:

Name	Range	SE / DF	Polarity
RANGE_BIPOLAR_20V_DF	±20 V	Differential	Bipolar
RANGE_BIPOLAR_12V5_DF	±12.5 V	Differential	Bipolar
RANGE_BIPOLAR_10V_DF	±10 V	Differential	Bipolar
RANGE_BIPOLAR_5V_DF	±5 V	Differential	Bipolar
RANGE_BIPOLAR_12V5_SE	±12.5 V	Single Ended	Bipolar
RANGE_BIPOLAR_10V_SE	±10 V	Single Ended	Bipolar
RANGE_BIPOLAR_6V25_SE	±6.25 V	Single Ended	Bipolar
RANGE_BIPOLAR_5V_SE	±5 V	Single Ended	Bipolar
RANGE_BIPOLAR_2V5_SE	±2.5 V	Single Ended	Bipolar
RANGE_UNIPOLAR_12V5_SE	0 V to 12.5 V	Single Ended	Unipolar
RANGE_UNIPOLAR_10V_SE	0 V to 10 V	Single Ended	Unipolar
RANGE_UNIPOLAR_5V_SE	0 V to 5 V	Single Ended	Unipolar

EXAMPLE

```
#include "tqmc500api.h"
```

```
TQMC500_HANDLE    hdl;
TQMC500_STATUS    result;
TQMC500_RANGE     range;
unsigned int       channelMask;
```

```
/*
** Set the range of the lowest 8 channels to unipolar, single-ended range from
** 0 V to 10 V.
*/
range = RANGE_UNIPOLAR_10V_SE;
channelMask = 0x000000ff;

result = tqmc500SetRange( hdl, channelMask, range );
if (result != TQMC500_OK)
{
    /* handle error */
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_INVALID	The channel mask selected at least one invalid channel for the operation.
TQMC500_ERR_NOT_SUPPORTED	The provided range does not exist or is not applicable.

4.2.2 tqmc500SetSampleClock

NAME

tqmc500SetSampleClock – configure the clock signal which triggers analog-to-digital conversion

SYNOPSIS

```
TQMC500_STATUS tqmc500SetSampleClock
(
    TQMC500_HANDLE          hdl,
    TQMC500_CLOCKSOURCE     clockBase,
    unsigned int             divider
);
```

DESCRIPTION

Configures the clock signal which is used by all ADCs as sample clock. The signal is generated from a clock base frequency and a mandatory divider. Note that, independent of the clock base and divider, the maximum sampling frequency of the ADCs is 800 kSPS. Choosing a combination of base and divider, which results in a higher frequency will cause this function to fail.

PARAMETERS

hdl

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

clockBase

The base frequency of the clock source is set by this parameter. Users may choose one of the following base frequencies:

Name	Clock Frequency	Minimum Divider
CLOCK_20MHZ	20 MHz	25
CLOCK_22M05HZ	22.05 MHz	28
CLOCK_60MHZ	60 MHz	75

divider

The clock source's frequency is divided by this value before being used as conversion clock. Set the value high enough in order not to overstep the maximum sampling frequency. The maximum value for divider is 134,217,728 (28 bit register).

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE    hdl;
TQMC500_STATUS    result;
TQMC500_CLOCKSOURCE    clkSrc;
unsigned int       divider;

/*
** Set the sampling rate to 44.1 kSPS
*/
divider = 500;
clkSrc = CLOCKSOURCE_22M05HZ;

result = tqmc500SetSampleClock( hdl, clkSrc, divider );
if (result != TQMC500_OK)
{
    /* handle error */
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_INVAL	The parameter defining the clock source was assigned with an invalid value.
TQMC500_ERR_NOT_SUPPORTED	The provided combination of clock source and divider exceeds the maximum sampling frequency or the divider exceeds its bit width. Increase or decrease the divider respectively, or choose a clock source with a lower frequency.
TQMC500_ERR_NOT_AVAILABLE	The provided clock rate is incompatible with the configured oversampling rate because the resulting maximum conversion ratio would be exceeded.

4.2.3 tqmc500SetOversampling

NAME

tqmc500SetOversampling – apply an oversampling ratio to all channels to decrease signal noise

SYNOPSIS

```
TQMC500_STATUS tqmc500SetOversampling
(
    TQMC500_HANDLE    hdl,
    unsigned int       oversamplingRatio
);
```

DESCRIPTION

The signal-to-noise ratio on all channels may be improved at the cost of the maximum sample rate by applying oversampling. All channels share the same oversampling ratio, which implies that the maximum oversampling ratio is limited by the channel which needs the highest sampling rate. The ADC bundles a power of two values to gain the enhanced sample, so valid values of *oversamplingRatio* are also a power of two. The maximum oversampling ratio is 256.

PARAMETERS

hdl

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

oversamplingRatio

The oversampling ratio defines the number of samples that are bundled together to gain one enhanced sample. Valid values are: 1, 2, 4, 8, 16, 32, 64, 128 and 256. A value of 1 effectively disables oversampling. The maximum sample rate is reduced by the selected oversampling rate in the following manner:

Oversampling Rate	Maximum Sample Rate
1	800 kSPS
2	400 kSPS
4	200 kSPS
8	100 kSPS
... and so on	

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE    hdl;
TQMC500_STATUS    result;
unsigned int       oversamplingRatio;

/*
** Set the oversampling ratio of all channels to 8
*/

oversamplingRatio = 8;
result = tqmc500SetOversampling( hdl, oversamplingRatio );
if (result != TQMC500_OK)
{
    /* handle error */
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_INVAL	The provided oversampling ratio is not valid. Valid values are in the range of zero to 256 and are a power of two.
TQMC500_ERR_NOT_AVAILABLE	The provided oversampling ratio is incompatible with the selected clock rate because the resulting maximum conversion ratio is exceeded.

4.2.4 tqmc500SetTransferSize

NAME

tqmc500SetTransferSize – configure the latency of data acquisition while using a buffered acquisition mode

SYNOPSIS

```
TQMC500_STATUS tqmc500SetTransferSize
(
    TQMC500_HANDLE          hdl,
    TQMC500_BUFFER_TRANSFER_SIZE size
);
```

DESCRIPTION

The transfer size defines the number of samples transferred from the device before an interrupt, which triggers data transfer to the user application, is generated. Consequently, changing the transfer size also changes the minimum latency between data acquisition from hardware and usability in the application at a given sample rate. A lower transfer size also results in a lower minimum latency, but implies a higher system load. Transfer sizes are thus to be selected as a tradeoff between latency and system load.

PARAMETERS

hdl

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

size

Number of samples to transfer in one buffer acquisition step from the device. Sizes must be chosen in accordance with the selected transfer mode. In FIFO mode, the maximum transfer size is limited to 512. The following values are accepted:

Name	Value	FIFO Mode	Latency at 800 kSPS
BUFFER_TRANSFER_SIZE_128	128	yes	160 μ s
BUFFER_TRANSFER_SIZE_256	256	yes	320 μ s
BUFFER_TRANSFER_SIZE_512	512	yes	640 μ s
BUFFER_TRANSFER_SIZE_1K	1024	no	1.28 ms
BUFFER_TRANSFER_SIZE_2K	2048	no	2.56 ms
BUFFER_TRANSFER_SIZE_4K	4096	no	5.12 ms
BUFFER_TRANSFER_SIZE_8K	8192	no	10.2 ms

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE      hdl;
TQMC500_STATUS      result;
TQMC500_BUFFER_TRANSFER_SIZE      transferSize = BUFFER_TRANSFER_SIZE_4K;

/*
** Set the buffer transfer size to 4096 samples
*/

result = tqmc500SetTransferSize( hdl, transferSize );
if (result != TQMC500_OK)
{
    /* handle error */
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_NOT_AVAILABLE	Buffer transfer is currently in progress. Terminate it to allow reconfiguration of the transfer size.
TQMC500_ERR_INVAL	The specified transfer size is not supported for this mode.
TQMC500_ERR_NOT_SUPPORTED	Insufficient memory for the requested DMA buffer size.

4.2.5 tqmc500ResetAdc

NAME

tqmc500ResetAdc – reset the analog input frontend

SYNOPSIS

```
TQMC500_STATUS tqmc500ResetAdc  
(  
    TQMC500_HANDLE    hdl  
);
```

DESCRIPTION

On occurrence of an error, sampling will be halted in order not to produce faulty data. In this case, the device has to be reset before new samples may be acquired. Consider choosing a more suitable configuration before performing the reset in order to avoid further errors. A device reset does not alter the range, oversampling or sample clock configuration.

PARAMETERS

hdl

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

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE    hdl;
TQMC500_STATUS    result;

/*
** Reset the device
*/

result = tqmc500ResetAdc( hdl );
if (result != TQMC500_OK)
{
    /* handle error */
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.

4.3 Input Functions

4.3.1 tqmc500ReadSample

NAME

tqmc500ReadSample – read the current sample of a single channel

SYNOPSIS

```
TQMC500_STATUS tqmc500ReadSample
(
    TQMC500_HANDLE    hdl,
    unsigned int       channel,
    int                *sample
);
```

DESCRIPTION

Reads the latest sample, which was captured by the given ADC channel. This function may be called at any time independent of the other, buffered reading functions to retrieve the most current value from an input.

PARAMETERS

hdl

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

channel

Number of the channel. Channel counting starts at zero.

sample

Pointer to a variable of type int. The value read from the ADC will be stored into that variable. Note that the data coding depends on the configuration of the input as described in the hardware user manual.

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE    hdl;
TQMC500_STATUS    result;
unsigned int       channel = 8;
int               sample;

/*
** Read the most current sample from channel 8
*/

result = tqmc500ReadSample( hdl, channel, &sample );
if (result != TQMC500_OK)
{
    /* handle error */
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_INVAL	Channel index is out of bounds.
TQMC500_ERR_ADC	An error related to the ADC occurred. This could mean that the sample rate is too high or that an internal error of the ADC occurred. Reset the ADC and check its configuration before continuing to sample.

4.3.2 tqmc500ReadVoltage

NAME

tqmc500ReadVoltage – read the current input sample from a single ADC channel and convert it into a voltage

SYNOPSIS

```
TQMC500_STATUS tqmc500ReadVoltage
(
    TQMC500_HANDLE    hdl,
    unsigned int       channel,
    double              *voltage
);
```

DESCRIPTION

Reads the latest sample, which was captured by the given ADC channel and converts it into a voltage. This function may be called at any time independent of the other, buffered reading functions to retrieve the most current value from an input.

PARAMETERS

hdl

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

channel

Number of the channel. Channel counting starts at zero.

voltage

Pointer to a variable of type double. The calculated analog voltage will be stored into that variable. The value returned is given in V.

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE hdl;
TQMC500_STATUS result;
unsigned int channel = 19;
double voltage;

/*
** Read the current voltage level from the last channel (i.e. channel 19)
*/

result = tqmc500ReadVoltage( hdl, channel, &voltage );
if (result != TQMC500_OK)
{
    /* handle error */
} else
{
    printf("The voltage on channel %d is %.4f V.\n", channel, voltage);
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_INVAL	Channel index is out of bounds.
TQMC500_ERR_ADC	An error related to the ADC occurred. This could mean that the sample rate is too high or that an internal error of the ADC occurred. Reset the ADC and check its configuration before continuing to sample.

4.3.3 tqmc500ReadSequence

NAME

tqmc500ReadSequence – read a sequence of consecutive samples from selected channels

SYNOPSIS

```
TQMC500_STATUS tqmc500ReadSequence
(
    TQMC500_HANDLE    hdl,
    unsigned int       channelMask,
    unsigned int       numSamples,
    int                *samples
);
```

DESCRIPTION

Reads a sequence of consecutive samples from one or multiple channels into a provided sample buffer. All samples read are guaranteed to be coherent, meaning the samples of a given index were all sampled simultaneously. The channels to be read are selected by the channel mask. The sample buffer passed to the function by a pointer must be large enough to hold at least the product of the number of samples and the number of channels. The size of the array is not checked against in the function, so users must assure sufficient buffer size in order not to produce false behavior. Note that subsequent calls of this function will not capture samples in a continuous manner. See tqmc500ReadContinuous() for that use case.

PARAMETERS

hdl

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

channelMask

Use the channel mask to select the channels of interest. Each bit of the mask corresponds to one channel of the device. Bit 0 corresponds to the first channel of the device, bit 1 to the second channel, and so on. A set bit, which has no corresponding channel on the device will cause the function to fail.

numSamples

The number of samples to be read from every selected channel.

samples

Pointer to the buffer, where acquired samples will be stored to. The buffer pointed at must be large enough to hold at least the requested amount of samples for all selected channels. The first sample of the first selected input will appear at index zero in the buffer. The first sample of the second selected channel will be at index one, and so on. Note that the data coding depends on the configuration of the input as described in the hardware user manual.

The following table illustrates a sample buffer after acquisition from channels 0 and 2:

Buffer Index	Description
0	Channel 0 Sample 0
1	Channel 2 Sample 0
2	Channel 0 Sample 1
3	Channel 2 Sample 1
... and so on	

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE    hdl;
TQMC500_STATUS    result;
unsigned int       channelMask;
unsigned int       numSamples = 256;
unsigned int       numChannels = 4;
int               samples[numSamples * numChannels];

/*
** Read a sequence of 256 samples from channels 0, 1, 2 and 7
*/

channelMask = (1 << 7) | (1 << 2) | (1 << 1) | (1 << 0);
result = tqmc500ReadSequence( hdl, channelMask, numSamples, samples );
if (result != TQMC500_OK)
{
    /* handle error */
} else
{
    printf("Channel 0 first sample: %d\n", samples[0]);
    printf("Channel 0 second sample: %d\n", samples[numChannels * 1]);
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_INVALID	The bitmask selected at least one invalid channel for the operation.
TQMC500_ERR_NOT_SUPPORTED	The requested number of samples is out of bounds for this operation. Use values smaller than the buffer size and do not use zero to avoid this error.
TQMC500_ERR_ADC	An error related to the ADC occurred. This could mean that the sample rate is too high or that an internal error of the ADC occurred. Reset the ADC and check its configuration before continuing to sample.
TQMC500_ERR_FIFO	A FIFO overflow error occurred and the FIFO is no longer usable. Adjust your settings and reset the device to be able to acquire ADC data again.
TQMC500_ERR_DMA	An error related to DMA transfer occurred. This either means, that the DMA controller encountered a functional PCI error or that the data could not be transferred to the host fast enough. Check your system load and bus configuration to handle this error. A device reset is required to restart acquiring samples.
TQMC500_ERR_BUFFER	The driver ran out of DMA memory and sampling was halted. A device reset is required.
TQMC500_ERR_NOT_AVAILABLE	Another instance is using data acquisition. Wait for the other instance to finish acquiring data or reset the device and try again.
TQMC500_ERR_TRY_AGAIN	The device was reset while waiting for samples. Try again to restart acquiring samples.
TQMC500_ERR_ABORTED	The operation was cancelled by a signal.

4.3.4 tqmc500ReadVoltageSequence

NAME

tqmc500ReadVoltageSequence – read a sequence of consecutive samples from selected channels and convert all values into voltages

SYNOPSIS

```
TQMC500_STATUS tqmc500ReadVoltageSequence
(
    TQMC500_HANDLE    hdl,
    unsigned int       channelMask,
    unsigned int       numSamples,
    double             *voltages
);
```

DESCRIPTION

Reads a sequence of consecutive samples from one or multiple channels and converts the values into the corresponding voltage levels. The voltages are then written into a user-provided buffer. All samples read are guaranteed to be coherent, meaning the samples of a given index were all sampled simultaneously. The channels to be read are selected by the channel mask. The sample buffer passed to the function by a pointer must be large enough to hold at least the product of the number of samples and the number of channels. The size of the array is not checked against in the function, so users must assure sufficient buffer size in order not to produce false behavior. Note that subsequent calls of this function will not capture samples in a continuous manner. See tqmc500ReadContinuous() for that use case.

PARAMETERS

hdl

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

channelMask

Use the channel mask to select the channels of interest. Each bit of the mask corresponds to one channel of the device. Bit 0 corresponds to the first channel of the device, bit 1 to the second channel, and so on. A set bit, which has no corresponding channel on the device will cause the function to fail.

numSamples

The number of samples to be read from every selected channel.

voltages

Pointer to the buffer, where calculated voltages will be stored to. The buffer pointed at must be large enough to hold at least the requested amount of values for all selected channels. The first

value of the first selected input will appear at index zero in the buffer. The first value of the second selected channel will be at index one, and so on. Values in the buffer are returned in V.

The following table illustrates a voltage buffer after acquisition from channels 0 and 2:

Buffer Index	Description
0	Channel 0 Value 0
1	Channel 2 Value 0
2	Channel 0 Value 1
3	Channel 2 Value 1
... and so on	

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE hdl;
TQMC500_STATUS result;
unsigned int channelMask;
unsigned int numSamples = 256;
double voltages[numSamples * 4];

/*
** Read a sequence of 256 samples from channels 0, 1, 2 and 7
*/

channelMask = (1 << 7) | (1 << 2) | (1 << 1) | (1 << 0);
result = tqmc500ReadVoltageSequence( hdl, channelMask, numSamples, voltages
);
if (result != TQMC500_OK)
{
    /* handle error */
} else
{
    printf("Channel 0 first voltage: %.4f\n", voltages[0]);
    printf("Channel 0 second voltage: %.4f\n", voltages[4 * 1]);
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_INVALID	The bitmask selected at least one invalid channel for the operation.
TQMC500_ERR_NOT_SUPPORTED	The requested number of samples is out of bounds for this operation. Use values smaller than the buffer size and not zero to avoid this error.
TQMC500_ERR_ADC	An error related to the ADC occurred. This could mean that the sample rate is too high or that an internal error of the ADC occurred. Reset the ADC and check its configuration before continuing to sample.
TQMC500_ERR_FIFO	A FIFO overflow error occurred and the FIFO is no longer usable. Adjust your settings and reset the device to be able to acquire ADC data again.
TQMC500_ERR_DMA	An error related to DMA transfer occurred. This either means, that the DMA controller encountered a functional PCI error or that the data could not be transferred to the host fast enough. Check your system load and bus configuration to handle this error. A device reset is required to restart acquiring samples.
TQMC500_ERR_BUFFER	The driver ran out of DMA memory and sampling was halted. A device reset is required.
TQMC500_ERR_NOT_AVAILABLE	Another instance is using data acquisition. Wait for the other instance to finish acquiring data or reset the device and try again.
TQMC500_ERR_TRY_AGAIN	The device was reset while waiting for samples. Try again to restart acquiring samples.
TQMC500_ERR_ABORTED	The operation was cancelled by a signal.

4.3.5 tqmc500ReadContinuous

NAME

tqmc500ReadContinuous – read a selection of analog inputs continuously while guaranteeing coherence

SYNOPSIS

```
TQMC500_STATUS tqmc500ReadContinuous
(
    TQMC500_HANDLE    hdl,
    unsigned int       channelMask,
    unsigned int       numSamples,
    int                *samples
);
```

DESCRIPTION

Needs to be called repeatedly in order to capture all samples of the selected channels in a continuous manner. All samples read are guaranteed to be coherent, meaning the samples of a given index were all sampled simultaneously. The channels to be read are selected by the channel mask. The sample buffer passed to the function by a pointer must be large enough to hold at least the product of the number of samples and the number of channels. The size of the array is not checked against in the function, so users must assure sufficient buffer size in order not to produce unwanted behavior.

Sampling starts as soon as this function is called the first time. The function will sleep until the requested number of samples have been acquired. Subsequent calls to this function must happen before the used buffer (i.e. FIFO or DMA buffer) is filled completely or else the function will return an error indicating an overflow.

As soon as an overflow occurs, sampling is stopped in order to disallow acquisition of incoherent samples. Call tqmc500ResetAdc() after an overflow to prepare sampling again. However, note that data acquisition starts only just after calling tqmc500ReadContinuous().

PARAMETERS

hdl

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

channelMask

Use the channel mask to select the channels of interest. Each bit of the mask corresponds to one channel of the device. Bit 0 corresponds to the first channel of the device, bit 1 to the second channel, and so on. A set bit, which has no corresponding channel on the device will cause the function to fail.

numSamples

The number of samples to be read from every selected channel. A value of zero will cause the function to return immediately.

samples

Pointer to the buffer, where acquired samples will be stored to. The buffer pointed at must be large enough to hold at least the requested amount of samples for all selected channels. The first sample of the first selected input will appear at index zero in the buffer. The first sample of the second selected channel will be at index one, and so on. Note that the data coding depends on the configuration of the input as described in the hardware user manual.

The following table illustrates a sample buffer after acquisition from channels 0 and 2:

Buffer Index	Description
0	Channel 0 Sample 0
1	Channel 2 Sample 0
2	Channel 0 Sample 1
3	Channel 2 Sample 1
... and so on	

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE    hdl;
TQMC500_STATUS    result = TQMC500_OK;
unsigned int      channelMask = 0b10101010;
unsigned int      numChannels = 4;
unsigned int      numSamples = 4000;
int               samples[numChannels * numSamples];
int               sampleNum;

/*
** Continuously read blocks of 4000 samples from channels 1,3,5 and 7
**/
while(result == TQMC500_OK){
    result = tqmc500ReadContinuous( hdl, channelMask, numSamples,
    samples );
    if (result != TQMC500_OK)
    {
        /* handle error */
    }
}
```

```

for(sampleNum = 0; sampleNum < numSamples; sampleNum++){
    int ch1Data = samples[numChannels * sampleNum];
    int ch3Data = samples[numChannels * sampleNum + 1];

    printf("Channel 1 Sample %d: %d\n", sampleNum, ch1Data);
    printf("Channel 3 Sample %d: %d\n", sampleNum, ch3Data);
    /* handle other channels accordingly */
}
}

```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_INVALID	The bitmask selected at least one invalid channel for the operation.
TQMC500_ERR_NOT_SUPPORTED	The requested number of samples is out of bounds for this operation. Use values smaller than the buffer size and not zero to avoid this error.
TQMC500_ERR_ADC	An error related to the ADC occurred. This could mean that the sample rate is too high or that an internal error of the ADC occurred. Reset the ADC and check its configuration before continuing to sample.
TQMC500_ERR_FIFO	A FIFO overflow error occurred and the FIFO is no longer usable. Adjust your settings and reset the device to be able to acquire ADC data again.
TQMC500_ERR_DMA	An error related to DMA transfer occurred. This either means, that the DMA controller encountered a functional PCI error or that the data could not be transferred to the host fast enough. Check your system load and bus configuration to handle this error. A device reset is required to restart acquiring samples.
TQMC500_ERR_BUFFER	The driver ran out of DMA memory and sampling was halted. Make sure to collect data frequently enough to avoid this error. A device reset is required.
TQMC500_ERR_NOT_AVAILABLE	Another thread is currently waiting in sequential mode. Wait for the other thread to release the device to avoid this error.
TQMC500_ERR_TRY_AGAIN	The device was reset while waiting for samples. Try again to restart acquiring samples.
TQMC500_ERR_ABORTED	The operation was cancelled by a signal.

4.3.6 tqmc500ReadVoltageContinuous

NAME

tqmc500ReadVoltageContinuous – read a selection of analog inputs continuously while guaranteeing coherence and convert the sample values into voltages

SYNOPSIS

```
TQMC500_STATUS tqmc500ReadVoltageContinuous
(
    TQMC500_HANDLE    hdl,
    unsigned int       channelMask,
    unsigned int       numSamples,
    double             *voltages
);
```

DESCRIPTION

Needs to be called repeatedly in order to capture all samples of the selected channels in a continuous manner and convert the values into the corresponding voltage levels. All samples read are guaranteed to be coherent, meaning the samples of a given index were all sampled simultaneously. The channels to be read are selected by the channel mask. The sample buffer passed to the function by a pointer must be large enough to hold at least the product of the number of samples and the number of channels. The size of the array is not checked against in the function, so users must assure sufficient buffer size in order not to produce false behavior.

Sampling starts as soon as this function is called the first time. The function will sleep until the requested number of samples have been acquired. Subsequent calls to this function must happen before the used buffer (i.e. FIFO or DMA buffer) is filled completely or else the function will return an error indicating an overflow.

As soon as an overflow occurs, sampling is stopped in order to disallow acquisition of incoherent samples. Call tqmc500ResetAdc() after an overflow to prepare sampling again. However, note that data acquisition starts only just after calling tqmc500ReadContinuous().

PARAMETERS

hdl

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

channelMask

Use the channel mask to select the channels of interest. Each bit of the mask corresponds to one channel of the device. Bit 0 corresponds to the first channel of the device, bit 1 to the second channel, and so on. A set bit, which has no corresponding channel on the device will cause the function to fail.

numSamples

The number of samples to be read from every selected channel.

voltages

Pointer to the buffer, where calculated voltages will be stored to. The buffer pointed at must be large enough to hold at least the requested amount of values for all selected channels. The first value of the first selected input will appear at index zero in the buffer. The first value of the second selected channel will be at index one, and so on. Values in the buffer are returned in V.

The following table illustrates a voltage buffer after acquisition from channels 0 and 2:

Buffer Index	Description
0	Channel 0 Value 0
1	Channel 2 Value 0
2	Channel 0 Value 1
3	Channel 2 Value 1
... and so on	

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE    hdl;
TQMC500_STATUS    result = TQMC500_OK;
unsigned int       channelMask = 0;
unsigned int       numSamples = 4096;
double            voltages[TQMC500_10_NUM_CHANNELS * numSamples];
unsigned int       sampleCount = 0;

/*
** Continuously read blocks of 4096 voltage values from all channels
*/

for(int channel = 0; channel < TQMC500_10_NUM_CHANNELS; channel++){
    channelMask |= (1 << channel);
}

while(result == TQMC500_OK){
    result = tqmc500ReadVoltageContinuous( hdl, sampleMask, numSamples,
    voltages );
    if (result != TQMC500_OK)
    {
        /* handle error */
    }

    for(int bufferIdx = 0; bufferIdx < numSamples; bufferIdx++){

        double ch0Volt = voltages[bufferIdx * TQMC500_10_NUM_CHANNELS];
        double ch1Volt = voltages[bufferIdx * TQMC500_10_NUM_CHANNELS + 1];

        printf("Sample %d Voltages - ", sampleCount);
        printf("Channel 0: %.4f  ", ch0Voltage);
        printf("Channel 1: %.4f\n", ch1Voltage);

        /* handle other channels accordingly */
        sampleCount++;
    }

    /* do something with the voltage data */
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_INVALID	The bitmask selected at least one invalid channel for the operation.
TQMC500_ERR_NOT_SUPPORTED	The requested number of samples is out of bounds for this operation. Use values smaller than the buffer size and not zero to avoid this error.
TQMC500_ERR_ADC	An error related to the ADC occurred. This could mean that the sample rate is too high or that an internal error of the ADC occurred. Reset the ADC and check its configuration before continuing to sample.
TQMC500_ERR_FIFO	The FIFO buffer overflowed and is now in error state. No new samples will be filled into the FIFO until the device is reset.
TQMC500_ERR_DMA	An error related to DMA transfer occurred. This either means, that the DMA controller encountered a functional PCI error or that the data could not be transferred to the host fast enough. Check your system load and bus configuration to handle this error. A device reset is required to restart acquiring samples.
TQMC500_ERR_BUFFER	The driver ran out of DMA memory and sampling was halted. Make sure to collect data frequently enough to avoid this error. A device reset is required.
TQMC500_ERR_NOT_AVAILABLE	Another thread is currently waiting in sequential mode. Wait for the other thread to release the device to avoid this error.
TQMC500_ERR_TRY_AGAIN	The device was reset while waiting for samples. Try again to restart acquiring samples.
TQMC500_ERR_ABORTED	The operation was cancelled by a signal.

4.3.7 tqmc500StopContinuous

NAME

tqmc500StopContinuous – stop continuous transfer of sample data from device

SYNOPSIS

```
TQMC500_STATUS tqmc500StopContinuous  
(  
    TQMC500_HANDLE    hdl  
);
```

DESCRIPTION

Use this function after tqmc500ReadContinuous() or tqmc500ReadVoltageContinuous() to stop the continuous transfer of sample data from the device. If data transfer is not stopped and data is not collected in time using one of these functions, the device will run into an overflow error and stop sampling. Use this function for a graceful shutdown.

PARAMETERS

hdl

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

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE hdl;
TQMC500_STATUS result;
const int blockSize = 512;
int samples[2 * blockSize];
const unsigned int totalNumberOfSamples = blockSize * 1024;
unsigned int sampleCount = 0;

/*
** Continuously read blocks of 512 samples 1024 times and terminate
gracefully ** afterwards
*/
while(result == TQMC500_OK && sampleCount < totalNumberOfSamples){
    result = tqmc500ReadContinuous( hdl, 0x3, blockSize, samples );

    /* ... use samples ... */
    sampleCount += blockSize;
}

result = tqmc500StopContinuous( hdl );
if (result != TQMC500_OK)
{
    /* handle error */
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_NOT_AVAILABLE	Continuous mode is not in use, so there is no need to stop it.

4.3.8 tqmc500ReadFrame

NAME

tqmc500ReadFrame – read the most current sample from the selected channels

SYNOPSIS

```
TQMC500_STATUS tqmc500ReadFrame
(
    TQMC500_HANDLE    hdl,
    unsigned int       channelMask,
    int                *samples
);
```

DESCRIPTION

This function may be used to acquire the most current data from all selected input channels in one call. The samples of all channels belong to the same frame, meaning they were all sampled at the same time.

PARAMETERS

hdl

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

channelMask

Use the channel mask to select the channels of interest. Each bit of the mask corresponds to one channel of the device. Bit 0 corresponds to the first channel of the device, bit 1 to the second channel, and so on. A set bit, which has no corresponding channel on the device will cause the function to fail.

samples

Pointer to the buffer, where acquired samples will be stored to. The buffer pointed at must be large enough to hold at least one sample for all selected channels. Note that the data coding depends on the configuration of the input as described in the hardware user manual.

The following table illustrates a sample buffer after acquisition from channels 0 and 3:

Buffer Index	Description
0	Channel 0
1	Channel 3

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE hdl;
TQMC500_STATUS result;
int samples[TQMC500_10_NUM_CHANNELS];
unsigned int channelMask = 0;

/*
** Read the most current sample from all channels
*/
for(int channel = 0; channel < TQMC500_10_NUM_CHANNELS; channel++){
    channelMask |= (1 << channel);
}

result = tqmc500ReadFrame( hdl, channelMask, samples );
if (result != TQMC500_OK)
{
    /* handle error */
}

for(int channel = 0; channel < TQMC500_10_NUM_CHANNELS; channel++){
    printf("Channel %d raw data: %d\n", channel, samples[channel]);
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_ADC	An error related to the ADC occurred. This could mean that the sample rate is too high or that an internal error of the ADC occurred. Reset the ADC and check its configuration before continuing to sample.

4.3.9 tqmc500ReadFrameVoltage

NAME

tqmc500ReadFrameVoltage – read the most current sample from the selected channels and convert the samples into analog voltage levels

SYNOPSIS

```
TQMC500_STATUS tqmc500ReadFrameVoltage
(
    TQMC500_HANDLE    hdl,
    unsigned int       channelMask,
    double             *voltages
);
```

DESCRIPTION

This function may be used to acquire the most current data from all selected input channels in one call and convert the values into the corresponding voltage levels. The samples of all channels belong to the same frame, meaning they were all sampled at the same time.

PARAMETERS

hdl

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

channelMask

Use the channel mask to select the channels of interest. Each bit of the mask corresponds to one channel of the device. Bit 0 corresponds to the first channel of the device, bit 1 to the second channel, and so on. A set bit, which has no corresponding channel on the device will cause the function to fail.

voltages

Pointer to the buffer, where calculates voltages will be stored to. The buffer pointed at must be large enough to hold at least the requested amount of values for all selected channels. The value of the first selected input will appear at index zero in the buffer. The value of the second selected channel will be at index one, and so on. Values in the buffer are returned in V.

The following table illustrates a voltage buffer after acquisition from channels 0 and 3:

Buffer Index	Description
0	Channel 0
3	Channel 3

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE    hdl;
TQMC500_STATUS    result;
double            voltages[TQMC500_NUM_CHANNELS];
int               numChannels = 8;

/*
** Read the most current voltage level from the first 8 channels
*/
for(int channel = 0; channel < numChannels; channel++){
    channelMask |= (1 << channel);
}

result = tqmc500ReadFrameVoltage( hdl, channelMask, voltages );
if (result != TQMC500_OK)
{
    /* handle error */
}

for(int channel = 0; channel < numChannels; channel++){
    printf("Channel %d current voltage: %.4f\n", channel, voltages[channel]);
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_ADC	An error related to the ADC occurred. This could mean that the sample rate is too high or that an internal error of the ADC occurred. Reset the ADC and check its configuration before continuing to sample.

4.4 Data Correction

4.4.1 tqmc500SetTemporaryCorrectionValues

NAME

tqmc500SetTemporaryCorrectionValues – set the correction data for a single ADC channel

SYNOPSIS

```
TQMC500_STATUS tqmc500SetTemporaryCorrectionValues
(
    TQMC500_HANDLE    hdl,
    unsigned int       channel,
    short              gain,
    short              offset
);
```

DESCRIPTION

Temporarily overwrite the factory-determined correction values for the given channel. By default, all ADC channels are corrected by a linear model, which uses correction data stored in the onboard EEPROM. Users may overwrite the data by a call to this function. Note that the values passed to this function are lost as soon as the input range of the given channel is changed. In order to deactivate data correction, users may call this function with a gain and offset of zero on every channel after configuring their input ranges. Refer to the hardware manual for a detailed explanation of the values used for data correction.

The data correction unit applies the following equation for correcting the captured samples:

$$\text{Data}_{\text{Corrected}} = \text{Data} \cdot \left(1 - \frac{\text{Gain}_{\text{CORR}}}{4 \cdot 2^{16}}\right) - \frac{\text{Offset}_{\text{CORR}}}{4}$$

- Data is the raw digital value captured by the ADC
- $\text{Data}_{\text{Corrected}}$ is the corrected digital value which is provided to the host system
- $\text{Gain}_{\text{CORR}}$ and $\text{Offset}_{\text{CORR}}$ are the correction values provided by parameters to this function

PARAMETERS

hdl

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

channel

Number of the channel. Channel counting starts at zero.

gain

The gain correction value, which is used for input data correction.

offset

The offset correction value, which is used for input data correction.

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE    hdl;
TQMC500_STATUS    result;
short             gain;
short             offset;

/*
** Apply a gain and offset correction on the first channel until the input
** range is changed
*/

gain = 100;
offset = 50;

result = tqmc500SetTemporaryCorrectionValues( hdl, 0, gain, offset );
if (result != TQMC500_OK)
{
    /* handle error */
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_INVAL	The channel number is out of bounds.

4.4.2 tqmc500GetFactoryCorrectionValues

NAME

tqmc500GetFactoryCorrectionValues – read the correction data for a single ADC channel

SYNOPSIS

TQMC500_STATUS tqmc500GetFactoryCorrectionValues

```
(  
    TQMC500_HANDLE    hdl,  
    unsigned int      channel,  
    TQMC500_RANGE     range,  
    short              *gain,  
    short              *offset  
);
```

DESCRIPTION

Read the correction data values for the given channel and input range from the onboard EEPROM. By default, all ADC channels are corrected by a linear model, which uses correction data stored in the onboard EEPROM. This function may be used to read out the values for a single channel.

PARAMETERS

hdl

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

channel

Number of the channel. Channel counting starts at zero.

range

Correction values are determined for each input range separately. Specify the input range for which the correction data will be retrieved by this argument. See tqmc500SetRange() for available definitions of ranges.

gain

Pointer to a variable of type int, where the gain correction value will be stored to. Refer to the calculation formula in the hardware user manual for information about the data correction model.

offset

Pointer to a variable of type int, where the offset correction value will be stored to. It is specified in $\frac{1}{4}$ LSB. Refer to the calculation formula in the hardware user manual for information about the data correction model.

EXAMPLE

```
#include "tqmc500api.h"

TQMC500_HANDLE  hdl;
TQMC500_STATUS  result;
TQMC500_RANGE   range;
short           gain, offset;

/*
** Read the correction values for the 0 - 5 V unipolar input range
*/

range = RANGE_UNIPOLAR_5V_SE;

result = tqmc500GetFactoryCorrectionValues( hdl, range, &gain, &offset );
if (result != TQMC500_OK)
{
    /* handle error */
}
```

RETURNS

On success, TQMC500_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TQMC500_ERR_INVALID_HANDLE	The specified device handle is invalid.
TQMC500_ERR_INVAL	The channel number is out of bounds.
TQMC500_ERR_NOT_SUPPORTED	The specified input range does not exist.

5 Diagnostic

5.1 Debugging Logs

If any difficulties during the usage of the TQMC500 arise, it might be helpful to follow some status information from the driver. In order to enable debug output, compile the driver using the `debug` target.

```
# make debug
```

The debug target activates all available layers of debug outputs. Outputs may be filtered using one of the following masks, to reduce the number of logs:

Debug Topic	Binary Value	Value	Description
Driver	0b00000001	1	Shows abnormalities during driver execution
Device	0b00000010	2	Shows abnormalities during device operation
ISR	0b00000100	4	Shows interrupt related information
Buffer	0b00001000	8	Debug info on DMA/FIFO buffer handling
Data Correction	0b00100000	32	Debug info on data correction
Timing	0b01000000	64	Activates time measurements and logs of critical procedures
Tracing	0b10000000	128	Traces normal driver functionality

Debug level values may be added to enable multiple layers of debugging. The resulting value has to be provided in decimal for the make command. Specify a debug mask when compiling the driver by the following scheme:

```
# make debug DMASK=129      # compile the driver with trace and driver logs
                             # enabled
```

Logs output by the driver may then be read from the kernel log in one of the following ways:

```
# dmesg -Hw                  # use dmesg to read the kernel log
# tail -f /var/log/syslog    # read the log from the syslog file
# journalctl -f              # use journalctl to read the kernel log
```

Please note, that debug logs can have a significant influence on the driver performance and should only be enabled in a development system. Use the `install` target in a productive environment.

5.2 Read Information From Kernel

5.2.1 Check if your device is recognized at the PCI bus

Even without a loaded or installed driver, your device will be shown in the output of the program *lspci*, which you can call from the command line:

```
# lspci | grep TEWS
```

A correctly recognized device is shown by an output similar to

```
# 07:00.0 Signal processing controller: TEWS Technologies GmbH Device c1f4
```

showing that the device is located at PCI bus 07.

If your device is not shown, make sure it is safely mounted into your carrier board and / or the PCI connectors are safely connected.

5.2.2 Check if the device driver is loaded and in use

To check, if the driver was correctly loaded and is currently used, run the following command:

```
# lsmod | grep tqmc500drv
```

A correctly loaded driver is indicated by an output line similar to

```
# tqmc500drv                36864    2
```

showing that two processes or modules currently use the driver. If no output is shown, load the driver (see section 2.3).

5.2.3 Check if your device is accessible in the filesystem

A correctly loaded driver with dynamic device file system enabled will create device nodes in the */dev* directory, which are needed by applications to perform I/O-operations with the hardware. Check for correctly created nodes by a call to

```
# ls -l /dev | grep tqmc500
```

If the nodes have been created correctly, the output will look similar like this:

```
# crw----- 1 root root  511,    0 Aug 28 13:32 tqmc500_0
```

This output shows, that one TQMC500 device node was created for major number 511 and is accessible for read/write-operations only to *root*. If no output is shown, make sure the driver is loaded and the device is recognized at the PCI bus.