The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC501-SW-65

Windows Device Driver

Optically Isolated 32 Channel 16 Bit ADC

Version 2.0.x

User Manual

Issue 2.0.0

March 2011

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

TPMC501-SW-65

Windows Device Driver

Optically Isolated 32 Channel 16 Bit ADC

Supported Modules:
TPMC501

Issue	Description	Date
1.0	First Issue	August 4, 2004
1.0.1	General revision, new address TEWS LLC	May 20, 2009
2.0.0	Windows 7 support, API functions added	March 4, 2011

Table of Contents

1 INTRODUCTION 4

2 INSTALLATION 5

2.1 Software Installation 5

2.1.1 Windows 2000 / XP 5

2.1.2 Windows 7 6

2.1.3 Confirming Driver Installation 6

3 API DOCUMENTATION 7

3.1 General Functions 7

3.1.1 tpmc501Open 7

3.1.2 tpmc501Close 9

3.2 Device Access Functions 11

3.2.1 tpmc501Read 11

3.2.2 tpmc501StartSequencer 14

3.2.3 tpmc501GetDataBuffer 17

3.2.4 tpmc501StopSequencer 20

3.2.5 tpmc501SetModelType 22

3.2.6 tpmc501GetModuleInfo 24

4 DEVICE DRIVER PROGRAMMING 26

4.1 TPMC501 Files and I/O Functions 26

4.1.1 Opening a TPMC501 Device 26

4.1.2 Closing a TPMC501 Device 28

4.1.3 TPMC501 Device I/O Control Functions 29

4.1.3.1 IOCTL_TPMC501_READ 31

4.1.3.2 IOCTL_TPMC501_START_SEQ 34

4.1.3.3 IOCTL_TPMC501_STOP_SEQ 39

4.1.3.4 IOCTL_TPMC501_CONF_MOD_TYPE 40

4.1.3.5 IOCTL_TPMC501_MOD_INFO 42

1 Introduction

The TPMC501-SW-65 Windows device driver is a kernel mode driver which allows the operation of the supported hardware module on an Intel or Intel-compatible Windows operating system. Supported Windows versions are:

- Windows 2000
- Windows XP
- Windows XP Embedded
- Windows 7 (32bit and 64bit)

The standard file and device (I/O) functions (CreateFile, CloseHandle, and DeviceIoControl) provide the basic interface for opening and closing a resource handle and for performing device I/O control operations.

The TPMC501-SW-65 device driver supports the following features:

- reading converted AD values from a specified channel
- configuring the sequencer for a free running measurement
- direct transfer of converted AD values to a dynamic ring buffer in the user space of the application task (Direct I/O)
- AD data correction with factory calibration data stored in the onboard EEPROM

The TPMC501-SW-65 device driver supports the modules listed below:

TPMC501	Optically Isolated 32 Channel 16 Bit ADC	(PMC)

To get more information about the features and use of TPMC501 devices it is recommended to read the manuals listed below.

TPMC501 User manual
TPMC501 Engineering Manual

2 Installation

Following files are located in directory TPMC501-SW-65 on the distribution media:

i386\	Directory containing driver files for 32bit Windows versions
amd64\	Directory containing driver files for 64bit Windows versions
installer_32bit.exe	Installation tool for 32bit systems (Windows XP or later)
installer_64bit.exe	Installation tool for 64bit systems (Windows XP or later)
tpmc501.inf	Windows installation script
tpmc501.h	Header file with IOCTL codes and structure definitions
example\tpmc501exa.c	Example application
api\tpmc501api.c	Application Programming Interface source
api\tpmc501api.h	Application Programming Interface header
TPMC501-SW-65-2.0.0.pdf	This document
Release.txt	Information about the Device Driver Release
ChangeLog.txt	Release history

2.1 Software Installation

2.1.1 Windows 2000 / XP

This section describes how to install the TPMC501 Device Driver on a Windows 2000 / XP operating system.

After installing the TPMC501 card(s) and boot-up your system, Windows 2000 / XP setup will show a "***New hardware found***" dialog box.

1. The "***Upgrade Device Driver Wizard***" dialog box will appear on your screen.
   Click "***Next***" button to continue.
2. In the following dialog box, choose "***Search for a suitable driver for my device***".
   Click "***Next***" button to continue.
3. In Drive A, insert the TPMC501 driver disk; select "***Disk Drive***" in the dialog box.
   Click "***Next***" button to continue.
4. Now the driver wizard should find a suitable device driver on the diskette.
   Click "***Next***" button to continue.
5. Complete the upgrade device driver and click "***Finish***" to take all the changes effect.
6. Now copy all needed files (tpmc501.h and API files) to the desired target directories.

After successful installation the TPMC501 device driver will start immediately and creates devices (TPMC501_1, TPMC501_2 ...) for all recognized TPMC501 modules.

2.1.2 Windows 7

This section describes how to install the TPMC501-SW-65 Device Driver on a Windows 7 (32bit or 64bit) operating system.

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

Copy needed files (tpm501.h and API files) to desired target directory.

After successful installation a device is created for each module found (TPMC501_1, TPMC501_2 ...).

2.1.3 Confirming Driver Installation

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

1. Open the Windows Device Manager:

   a. For Windows 2000 / XP, open the "***Control Panel***" from "***My Computer***" and click the "***System***" icon and choose the "***Hardware***" tab, and then click the "***Device Manager***" button.

   b. For Windows 7, open the "***Control Panel***" from "***My Computer***" and then click the "***Device Manager***" entry.

2. Click the "**+**" in front of "***Embedded I/O***".
   The driver ***"TEWS TECHNOLOGIES - TPMC501 32(16) Channel 16-Bit ADC (TPMC501)***" should appear for each installed device.

3 API Documentation

3.1 General Functions

3.1.1 tpmc501Open

NAME

tpmc501Open – Opens a Device

SYNOPSIS

```
TPMC501_HANDLE tpmc501Open
(
     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 “tpmc501api.h”

TPMC501_HANDLE hdl;

/*
** open file descriptor to device
*/
hdl = tpmc501Open(“\\\\.\\TPMC501_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.

3.1.2 tpmc501Close

NAME

tpmc501Close – Closes a Device

SYNOPSIS

```
TPMC501_STATUS tpmc501Close
(
     TPMC501_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 “tpmc501api.h”

TPMC501_HANDLE hdl;
TPMC501_STATUS result;

/*
** close file descriptor to device
*/
result = tpmc501Close( hdl );

if (result != TPMC501_OK)
{
     /* handle close error */
}
```

RETURNS

On success TPMC501_OK, or an appropriate error code.

ERROR CODES

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

3.2 Device Access Functions

3.2.1 tpmc501Read

NAME

tpmc501Read – Read converted AD value

SYNOPSIS

```
TPMC501_STATUS tpmc501Read
(
     TPMC501_HANDLE  hdl,
     int             channel,
     int             gain,
     int             flags,
     int             *pAdcVal
);
```

DESCRIPTION

This function starts an AD conversion on the specified channel and returns the converted value.

PARAMETERS

hdl

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

channel

This argument specifies the input channel number. Valid channels for single-ended mode are 1…32, for differential mode 1...16.

gain

This argument specifies the input gain that shall be used. Allowed values are 1, 2, 5, 10 or 1, 2, 4, 8 depending on the module type.

flags

Set of bit flags that control the AD conversion. The following flags could be OR'ed:

Flag	Meaning
TPMC501_DIFF	If this flag is set the ADC input works in differential mode otherwise in single-ended (default).
TPMC501_CORR	Perform an offset and gain correction with factory calibration data stored in the TPMC501 EEPROM.
TPMC501_FAST	If this flag is set the fast (polled) mode will be used. The driver will not use interrupts, instead it will wait in a busy loop until the settling time (if necessary) and the conversion is finished. Conversions using this mode will be handled faster, but the processor executes a busy loop and other tasks will not be handled during the loops.

pAdcVal

This argument points to an integer variable where the AD value will be returned.

EXAMPLE

```
#include "tpmc501api.h"

TPMC501_HANDLE   hdl;
TPMC501_STATUS   result;
int              AdcData;
int              channel, gain, flags;

channel   = 32;
gain      = 2;
flags     = TPMC501_CORR | TPMC501_FAST;

result = tpmc501Read(hdl, channel, gain, flags, &AdcData);

if (result != TPMC501_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC501_ERR_ACCESS	The module type has not been configured.
TPMC501_ERR_INVALID_HANDLE	The specified TPMC501_HANDLE is invalid.
TPMC501_ERR_INVAL	At least one of the parameters is invalid.
TPMC501_ERR_TIMEOUT	ADC conversion timed out.
TPMC501_ERR_RANGE	Invalid channel number.
TPMC501_ERR_BUSY	This error occurs if the sequencer is still running. Please stop the sequencer before executing this function.

3.2.2 tpmc501StartSequencer

NAME

tpmc501StartSequencer – Start sequencer operation

SYNOPSIS

```
TPMC501_STATUS tpmc501StartSequencer
(
    TPMC501_HANDLE     hdl,
    unsigned int       CycleTime,
    unsigned int       NumOfBufferPages,
    unsigned int       NumOfChannels,
    TPMC501_CHAN_CONF  *ChanConf
);
```

DESCRIPTION

This function starts an AD conversion on the specified channel and returns the converted value.

PARAMETERS

hdl

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

CycleTime

This argument specifies the repeat frequency of the sequencer in 100 µs steps. Each time the sequencer timer reaches the programmed cycle time a new AD conversion of all active channels is started. Valid values are in the range from 100 µs to 6.5535 seconds.

NumOfBufferPages

This argument specifies the number of sample blocks in the ring buffer. A sample block contains the samples of all channels (NumOfChannels) per sequencer cycle.

NumOfChannels

This argument specifies the number of active channels for this job. The maximum number is 32.

ChanConf

This array of channel configuration structures specifies the configuration of the active channels. The channel configuration defines the channel number, the gain and some flags. The ordering of channels in a ring buffer page is the same as defined in this array.

```
typedef struct
{
   UINT32 ChanToUse;
   UINT32 gain;
   UINT32 flags;
} TPMC501_CHAN_CONF, *PTPMC501_CHAN_CONF;
```

ChanToUse

This parameter specifies the input channel number. Valid channels for single-ended mode are 1…32, for differential mode 1...16.

gain

This Parameter specifies the gain for this channel. Valid gains are 1, 2, 5, 10 for *TPMC501-10/-12/-20/-22* and 1, 2, 4, 8 for *TPMC501-11/-13/-21/-23.*

flags

Set of bit flags that control the AD conversion. The following flags could be OR'ed:

Flag	Meaning
TPMC501_DIFF	If this flag is set the ADC input works in differential mode otherwise in single-ended (default).
TPMC501_CORR	Perform an offset and gain correction with factory calibration data stored in the TPMC501 EEPROM.

EXAMPLE

```
#include "tpmc501api.h"

TPMC501_HANDLE     hdl;
TPMC501_STATUS     result;
unsigned int       CycleTime;
unsigned int       NumOfBufferPages;
unsigned int       NumOfChannels;
TPMC501_CHAN_CONF  ChanConf[TPMC501_MAX_CHAN];

CycleTime = 5000;
NumOfBufferPages = 100;
int NumOfChannels = 2;

ChanConf[0].ChanToUse   = 1;
ChanConf[0].gain        = 1;
ChanConf[0].flags       = TPMC501_CORR;

ChanConf[1].ChanToUse   = 20;
ChanConf[1].gain        = 5;
ChanConf[1].flags       = TPMC501_CORR;
…
```

```
// start the sequencer
result = tpmc501StartSequencer(hdl, CycleTime, NumOfBufferPages,
                                NumOfChannels, ChanConf);

if (result != TPMC501_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC501_ERR_ACCESS	The module type has not been configured.
TPMC501_ERR_INVALID_HANDLE	The specified TPMC501_HANDLE is invalid.
TPMC501_ERR_INVAL	At least one of the parameters is invalid.
TPMC501_ERR_RANGE	Invalid channel number.
TPMC501_ERR_BUSY	This error occurs if the sequencer is still running. Please stop the sequencer before executing this function.

3.2.3 tpmc501GetDataBuffer

NAME

tpmc501GetDataBuffer – Get next data block of sequencer samples

SYNOPSIS

```
TPMC501_STATUS tpmc501GetDataBuffer
(
     TPMC501_HANDLE  hdl,
     int             **pData,
     unsigned int    *pStatus
);
```

DESCRIPTION

This function returns a pointer to the next available data block in the ring buffer. If no data block is available the functions returns TPMC501_ERR_NODATA. In this case it must be called again until new data are available.

PARAMETERS

hdl

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

pData

This argument is a pointer to an array of integer items that contains the converted data of all configured channels of a sequencer cycle. The number of channels and the channel configuration was setup with the tpmc501StartSequencer function. The first array item [0] belongs to the channel configured by ChanConfig[0], the second array item [1] belongs to the channel configured by ChanConfig[1] and so forth. Please refer to the example application for details.

pStatus

This argument is a pointer to a variable which returns the actual sequencer error status. Keep in mind to check this status before each reading. If status is 0 no error is pending. A set of bits specifies the error condition.

Value	Description
TPMC501_BUF_OVERRUN	This bit indicates a ring buffer overrun. The error occurred if there is no space in ring buffer to write the new AD data. In this case the new AD values are dismissed. The sequencer was not stopped.
TPMC501_DATA_OVERFLOW	This indicates an overrun in the sequencer data RAM. The error occurred if the driver is too slow to read the data in time. The sequencer was stopped after this error occurred.
TPMC501_TIMER_ERR	Sequencer timer error (see also TPMC501 hardware manual). The sequencer was stopped after this error occurred.
TPMC501_INST_RAM_ERR	Sequencer instruction RAM error (see also TPMC501 hardware manual). The sequencer was stopped after this error occurred.

EXAMPLE

```
#include “tpmc501api.h”

TPMC501_HANDLE   hdl;
TPMC501_STATUS   result;
unsigned int     seqStatus;
int              *pData;


result = tpmc501GetDataBuffer(hdl, &pData, &seqStatus);

if (result != TPMC501_OK)
{
     if (result == TPMC501_ERR_NODATA)
     {
          /* try again reading data */
     }
     else
     {
          /* handle error */
     }
}
```

RETURNS

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC501_ERR_INVALID_HANDLE	The specified TPMC501_HANDLE is invalid.
TPMC501_ERR_NODATA	No new data available in the ring buffer
TPMC501_ERR_NOT_READY	The sequencer is stopped.

3.2.4 tpmc501StopSequencer

NAME

tpmc501StopSequencer – Stop the sequencer

SYNOPSIS

```
TPMC501_STATUS tpmc501StopSequencer
(
     TPMC501_HANDLE hdl
);
```

DESCRIPTION

This function stops execution of the sequencer mode on the specified device.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tpmc501api.h”

TPMC501_HANDLE   hdl;
TPMC501_STATUS   result;

result = tpmc501StopSequencer(hdl);

if (result != TPMC501_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC501_ERR_INVALID_HANDLE	The specified TPMC501_HANDLE is invalid.

3.2.5 tpmc501SetModelType

NAME

tpmc501SetModelType – Set the module type of the TPMC501

SYNOPSIS

```
TPMC501_STATUS tpmc501SetModelType
(
     TPMC501_HANDLE  hdl,
     int             ModuleType
);
```

DESCRIPTION

This TPMC501 function configures the model type of the TPMC501.

This function must be called before the first AD conversion can be started.

PARAMETERS

hdl

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

ModuleType

This argument specifies the model type of the TPMC501. The following model types are supported.

Value	Description
TPMC501_TYPE_10	TPMC501-10 (Gain 1/2/5/10, +/-10V, Front I/O)
TPMC501_TYPE_11	TPMC501-11 (Gain 1/2/4/8, +/-10V, Front I/O)
TPMC501_TYPE_12	TPMC501-12 (Gain 1/2/5/10, 0-10V, Front I/O)
TPMC501_TYPE_13	TPMC501-13 (Gain 1/2/4/8, 0-10V, Front I/O)
TPMC501_TYPE_20	TPMC501-20 (Gain 1/2/5/10, +/-10V, Back I/O)
TPMC501_TYPE_21	TPMC501-21 (Gain 1/2/4/8, +/-10V, Back I/O)
TPMC501_TYPE_22	TPMC501-22 (Gain 1/2/5/10, 0-10V, Back I/O)
TPMC501_TYPE_23	TPMC501-23 (Gain 1/2/4/8, 0-10V, Back I/O)

EXAMPLE

```
#include “tpmc501api.h”

TPMC501_HANDLE   hdl;
TPMC501_STATUS   result;

result = tpmc501SetModelType(hdl, TPMC501_TYPE_11);

if (result != TPMC501_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC501_ERR_INVALID_HANDLE	The specified TPMC501_HANDLE is invalid.
TPMC501_ERR_RANGE	Invalid channel number.

3.2.6 tpmc501GetModuleInfo

NAME

tpmc501GetModuleInfo – Get module information data

SYNOPSIS

```
TPMC501_STATUS tpmc501GetModuleInfo
(
     TPMC501_HANDLE         hdl,
     TPMC501_INFO_BUFFER    *pModuleInfo
);
```

DESCRIPTION

This function reads module information data such as configured module type, location on the PCI bus and factory programmed correction data.

PARAMETERS

hdl

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

pModuleInfo

This argument specifies a pointer to the module information buffer.

```
typedef struct
{
   UINT32     Variant;
   UINT32     PciBusNo;
   UINT32     PciDevNo;
   UINT32     ADCOffsetCal[4];
   UINT32     ADCGainCal[4];
} TPMC501_INFO_BUFFER, *PTPMC501_INFO_BUFFER;
```

Variant

This parameter returns the configured module variant (e.g. 10 for a TPMC501-10).

PciBusNo, PciDevNo

These parameters specifies the PCI location of this module

ADCOffsetCal[4]

This array returns the factory programmed offset correction value for the different gains. Array index 0 contains the value for gain 1, index 1 contains the value for gain 2 and so forth.

ADCGainCal[4]

This array returns the factory programmed gain correction for the different gains. Array index 0 contains the value for gain 1, index 1 contains the value for gain 2 and so forth.

EXAMPLE

```
#include “tpmc501api.h”

TPMC501_HANDLE        hdl;
TPMC501_STATUS        result;
TPMC501_INFO_BUFFER   ModuleInfo

result = tpmc501GetModuleInfo(hdl, &ModuleInfo);

if (result != TPMC501_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC501_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

TPMC501_ERR_INVALID_HANDLE	The specified TPMC501_HANDLE is invalid.

4 Device Driver Programming

The TPMC501-SW-65 Windows device driver is a kernel mode device driver.

The standard file and device (I/O) functions (CreateFile, CloseHandle, and DeviceIoControl) provide

the basic interface for opening and closing a resource handle and for performing device I/O control operations.

All of these standard Win32 functions are described in detail in the Windows Platform SDK Documentation (Windows base services / Hardware / Device Input and Output).

For details refer to the Win32 Programmers Reference of your used programming tools (C++, Visual Basic etc.)

4.1 TPMC501 Files and I/O Functions

The following section does not contain a full description of the Win32 functions for interaction with the TPMC501 device driver. Only the required parameters are described in detail.

4.1.1 Opening a TPMC501 Device

Before you can perform any I/O the *TPMC501* device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the *TPMC501* device.

```
HANDLE CreateFile(
      LPCTSTR   lpFileName,
      DWORD     dwDesiredAccess,
      DWORD     dwShareMode,
      LPSECURITY_ATTRIBUTES lpSecurityAttributes,
      DWORD     dwCreationDistribution,
      DWORD     dwFlagsAndAttributes,
      HANDLE    hTemplateFile
);
```

Parameters

lpFileName

This parameter points to a null-terminated string, which specifies the name of the TPMC501 to open. The *lpFileName* string should be of the form **\\.\TPMC501_*x*** to open the device *x*. The ending x is a one-based number. The first device found by the driver is \\.\TPMC501_1, the second \\.\TPMC501_2 and so on.

dwDesiredAccess

This parameter specifies the type of access to the TPMC501.
For the TPMC501 this parameter must be set to read-write access (GENERIC_READ | GENERIC_WRITE)

dwShareMode

Set of bit flags that specify how the object can be shared. Set to 0.

lpSecurityAttributes

This argument is a pointer to a security structure. Set to NULL for TPMC501 devices.

dwCreationDistribution

Specifies the action to take on existing files, and which action to take when files do not exist. TPMC501 devices must be always opened **OPEN_EXISTING**.

dwFlagsAndAttributes

Specifies the file attributes and flags for the file. This value must be set to FILE_FLAG_OVERLAPPED for TPMC501 devices (see also the device I/O function IOCTL_TPMC501_START_SEQ).

hTemplateFile

This value must be NULL for TPMC501 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TPMC501 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call ***GetLastError***.

Example

```
HANDLE    hDevice;

hDevice = CreateFile(
     “\\\\.\\TPMC501_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,                    // no security attrs
     OPEN_EXISTING,           // TPMC501 device always open existing
     FILE_FLAG_OVERLAPPED,    // overlapped I/O
     NULL
);

if (hDevice == INVALID_HANDLE_VALUE) {
     ErrorHandler( "Could not open device" ); // process error
}
```

See Also

CloseHandle(), Win32 documentation CreateFile()

4.1.2 Closing a TPMC501 Device

The **CloseHandle** function closes an open TPMC501 handle.

```
BOOL CloseHandle(
      HANDLE hDevice;
);
```

Parameters

hDevice

Identifies an open TPMC501 handle.

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call ***GetLastError*.**

Example

```
HANDLE  hDevice;

if( !CloseHandle( hDevice ) ) {
     ErrorHandler( "Could not close device" ); // process error
}
```

See Also

CreateFile (), Win32 documentation CloseHandle ()

4.1.3 TPMC501 Device I/O Control Functions

The **DeviceIoControl** function sends a control code directly to a specified device driver, causing the corresponding device to perform the specified operation.

```
BOOL DeviceIoControl(
    HANDLE          hDevice,
    DWORD           dwIoControlCode,
    LPVOID          lpInBuffer,
    DWORD           nInBufferSize,
    LPVOID          lpOutBuffer,
    DWORD           nOutBufferSize,
    LPDWORD         lpBytesReturned,
    LPOVERLAPPED    lpOverlapped
);
```

Parameters

hDevice

Handle to the TPMC501 that is to perform the operation.

dwIoControlCode

Specifies the control code for the operation. This value identifies the specific operation to be performed. The following values are defined in *tpmc501.h* :

Value	Meaning
IOCTL_TPMC501_READ	Read a converted AD value
IOCTL_TPMC501_START_SEQ	Setup and start the sequencer
IOCTL_TPMC501_STOP_SEQ	Stop the sequencer
IOCTL_TPMC501_CONF_MOD_TYPE	Configure which model type is mounted
IOCTL_TPMC501_MOD_INFO	Get module information

See below for more detailed information on each control code.

To use these TPMC501 specific control codes the header file tpmc501.h must be included in the application

lpInBuffer

Pointer to a buffer that contains the data required to perform the operation.

nInBufferSize

Specifies the size of the buffer pointed to by *lpInBuffer*.

lpOutBuffer

Pointer to a buffer that receives the operation's output data.

nOutBufferSize

Specifies the size of the buffer in bytes pointed to by *lpOutBuffer*.

lpBytesReturned

Pointer to a variable that receives the size, in bytes, of the data stored into the buffer pointed to by *lpOutBuffer*. A valid pointer is required.

lpOverlapped

Pointer to an *Overlapped* structure. This parameter is required because the TPMC501 device driver uses overlapped I/O.

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call ***GetLastError***.

See Also

Win32 documentation DeviceIoControl()

4.1.3.1 IOCTL_TPMC501_READ

This TPMC501 control function starts an AD conversion on the specified channel and returns the converted value (16 bit sign extended) in a long word buffer to the caller. The Parameter *lpOutBuffer* passes a pointer to this buffer to the device driver.

The *lpInBuffer* parameter passes a pointer to a channel configuration structure (TPMC501_CHAN_CONF) to the driver which contains parameter required to perform the operation.

```
typedef struct {
   UINT32 ChanToUse;
   UINT32 gain;
   UINT32 flags;
} TPMC501_CHAN_CONF, *PTPMC501_CHAN_CONF;
```

ChanToUse

This parameter specifies the input channel number. Valid channels for single-ended mode are 1…32, for differential mode 1...16.

gain

Specifies the gain for this AD conversion. Valid gains are 1, 2, 5, 10 for *TPMC501-10/-12/-20/-22* and 1, 2, 4, 8 for *TPMC501-11/-13/-21/-23.*

flags

Set of bit flags that controls the AD conversion. The following flags could be OR'ed:

Flag	Meaning
TPMC501_DIFF	If this flag is set the ADC input works in differential mode otherwise in single-ended (default).
TPMC501_CORR	Perform an offset and gain correction with factory calibration data stored in the TPMC501 EEPROM.
TPMC501_FAST	If this flag is set the fast (polled) mode will be used. The driver will not use interrupts, instead it will wait in a busy loop until the settling time (if necessary) and the conversion is finished. Conversions using this mode will be handled faster, but the processor executes a busy loop and other tasks will not be handled during the loops.

Example

```
#include “tpmc501.h”

HANDLE             hDevice;
BOOLEAN            success;
ULONG              NumWritten;
int                ReadData;
TPMC501_CHAN_CONF  ChanConf;

//
// Start conversion at channel 1, set gain to 1 and correct the
// reading with the factory calibration data
//
ChanConf.ChanToUse = 1;
ChanConf.gain      = 1;
ChanConf.flags     = TPMC501_CORR;

success = DeviceIoControl (
     hDevice,
     IOCTL_TPMC501_READ,
     &ChanConf,
     sizeof(ChanConf),
     &ReadData,
     sizeof(int),
     &NumWritten,
     NULL
);

if (!success)
{
     ErrorHandler ( "Device I/O control error” );
}
```

Error Codes

ERROR_ACCESS_DENIED	The module type has not been set.
ERROR_INVALID_PARAMETER	This error will be returned if the size of the read/write buffer is too small or at least one of the parameters is invalid.
ERROR_IO_TIMEOUT	ADC conversion timed out.
ERROR_MEMBER_NOT_IN_GROUP	Invalid channel number.
ERROR_DEVICE_BUSY	This error occurs if the sequencer is still running. Please stop the sequencer before executing this function.

All other returned error codes are system error conditions.

4.1.3.2 IOCTL_TPMC501_START_SEQ

This overlapped TPMC501 control function starts the internal sequencer to perform a continuous AD conversion of the specified channels. After each conversion cycle the device driver stores the AD value directly into a user supplied ring buffer. A list of active channels, the sequencer cycle time and other parameter which controls the conversion must be passed with the following job description structure to the device driver.

```
typedef struct {
    UINT32 CycleTime;
    UINT32 NumOfBufferPages;
    UINT32 NumOfChannels;
    TPMC501_CHAN_CONF ChanConf[TPMC501_MAX_CHAN];
} TPMC501_JOB_DESC, *PTPMC501_JOB_DESC;
```

Members

CycleTime

Specifies the repeat frequency of the sequencer in 100 µs steps. Each time the sequencer timer reaches the programmed cycle time a new AD conversion of all active channels is started. Valid values are in the range from 100 µs to 6.5535 seconds.

NumOfBufferPages

Specifies the maximum number of “pages” in the ring buffer. A page contains the AD values of all active channels from a sequencer cycle. The ring buffer looks like a two-dimensional array: buffer[NumOfBufferPages][NumOfChannels]

NumOfChannels

Specifies the number of active channels for this job. The maximum number is 32.

ChanConf[TPMC501_MAX_CHAN]

This array of channel configuration structures specifies the configuration of the active channels. The channel configuration defines the channel number, the gain and some flags. Please refer to IOCTL_TPMC501_READ for detailed description of this structure. The ordering of channels in a ring buffer page is the same as defined in this array.

Ring Buffer Layout

The user supplied ring buffer contains the converted AD values of each sequencer cycle. This buffer is directly mapped into the system virtual space of the device driver (Direct I/O) and filled after each sequencer interrupt with the new AD values. That is the reason why this function was performed as an overlapped (asynchronous) operation (see also Win32 documentation). As long as the device I/O control function is pending the device driver is able to lock the user buffer in memory and access these pages from the interrupt service routine. To stop the sequencer and finish device I/O control function execute the *IOCTL_TPMC501_STOP_SEQ* control function.

```
typedef struct {
    UINT32 status;
    UINT32 PutIndex;
    UINT32 GetIndex;
    INT32 buffer[1];
} TPMC501_RING_BUFFER, *PTPMC501_RING_BUFFER;
```

Members

status

This field contains the actual sequencer error status. Keep in mind to check this status before each reading. If status is 0 no error is pending. A set of bits specifies the error condition.

Value	Description
TPMC501_BUF_OVERRUN	This bit indicates a ring buffer overrun. The error occurred if there is no space in ring buffer to write the new AD data. In this case the new AD values are dismissed. The sequencer was not stopped.
TPMC501_DATA_OVERFLOW	This indicates an overrun in the sequencer data RAM. The error occurred if the driver is too slow to read the data in time. The sequencer was stopped after this error occurred.
TPMC501_TIMER_ERR	Sequencer timer error (see also TPMC501 hardware manual). The sequencer was stopped after this error occurred.
TPMC501_INST_RAM_ERR	Sequencer instruction RAM error (see also TPMC501 hardware manual). The sequencer was stopped after this error occurred.

PutIndex

Index of the next ring buffer page to write by the device driver. The index is incremented by 1 (device driver) after each write. At the ring buffer limit it is set to 0 again. The user application only read this index.

GetIndex

Index of the next ring buffer page to read by the application task. The index is incremented by 1 (application) after each read. At the ring buffer limit it is set to 0 again. The ring buffer is empty if *PutIndex* is equal to *GetIndex*.

buffer[1]

This is a dynamic expandable array which holds the converted AD values. The real dimension of this buffer is given by **NumOfBufferPages * NumOfChannles**. Therefore don't use this type in a sizeof() function to determine the size of this array.
See also the code example to understand the structure of the ring buffer and the access methods.

Example

```
#include “tpmc501.h”

#define RING_BUFFER_SPACE   10000

HANDLE                hDevice;
BOOLEAN               success;
ULONG                 NumWritten;
OVERLAPPED            SeqOverlapped;
TPMC501_CHAN_CONF     ChanConf;
TPMC501_JOB_DESC      job;
PTPMC501_RING_BUFFER  pRing;

//
//  Allocate enough memory for the ring buffer and initialze the
//  buffer control header
//
pRing = (PTPMC501_RING_BUFFER)malloc( RING_BUFFER_SPACE );
pRing->status          = 0;
pRing->PutIndex        = 0;
pRing->GetIndex        = 0;

SeqOverlapped.Offset   = 0;
SeqOverlapped.hEvent   = 0;

job.CycleTime          = 1;          //  0.0001 second
job.NumOfChannels      = 3;          //  active channels

job.ChanConf[0].ChanToUse   = 1;
job.ChanConf[0].gain        = 1;
job.ChanConf[0].flags       = TPMC501_CORR;

job.ChanConf[1].ChanToUse   = 20;
job.ChanConf[1].gain        = 5;
job.ChanConf[1].flags       = TPMC501_CORR;

job.ChanConf[2].ChanToUse   = 5;
job.ChanConf[2].gain        = 2;
job.ChanConf[2].flags       = TPMC501_CORR;
```

```
//
//  Calculate the maximum number of ring buffer pages to use
//  A pages contains the ADC values from all desired channels
//  (job.NumOfChannels) of a single sequencer cycle.
//  The ring buffer looks like a two-dimensional array
//
//      buffer[NumOfBufferPages][NumOfChannels]
//

//  NOTE
//  Not all of the space in the ring buffer is available for data.
//        Subtract the offset of the buffer[] array

job.NumOfBufferPages = ( RING_BUFFER_SPACE –
          FIELD_OFFSET( TPMC501_RING_BUFFER, buffer ) ) /
          ( job.NumOfChannels * sizeof( pRing->buffer[0] ) );

success = DeviceIoControl (
  hDevice,
  IOCTL_TPMC501_START_SEQ,
  &job,
  sizeof(job),
  pRing,
  RING_BUFFER_SPACE,
  &NumWritten,
  &SeqOverlapped
);

if( success ) {
  printf( "\nThis should never happen.\n" );
}
else {
  if( GetLastError() == ERROR_IO_PENDING ) {
     printf( "\nSequencer successful started...\n" );
  }
  else {
     printf( "\nStart sequencer failed -->  Error = %d\n",
     GetLastError() );
     PrintErrorMessage();
  }
}

//
//  Process converted AD values. See the example program for details....
//
```

Error Codes

ERROR_ACCESS_DENIED	The module type has not been configured.
ERROR_INVALID_PARAMETER	This error will be returned if the size of the read/write buffer is too small or at least one of the parameters is invalid.
ERROR_MEMBER_NOT_IN_GROUP	Invalid channel number.
ERROR_DEVICE_BUSY	This error occurs if the sequencer is still running. Please stop the sequencer before executing this function.

All other returned error codes are system error conditions.

4.1.3.3 IOCTL_TPMC501_STOP_SEQ

This TPMC501 control function stops the running sequencer and finishes the outstanding (overlapped) *IOCTL_TPMC501_START_SEQ* device control function.

Example

```
#include “tpmc501.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;

//
// stop the running sequencer
//

success = DeviceIoControl (
     hDevice,
     IOCTL_TPMC501_STOP_SEQ,
     NULL,
     0,
     NULL,
     0,
     &NumBytes,
     NULL
);

if (!success)
{
     // process error
     ErrorHandler ( "Device I/O control error” );
}
```

Error Codes

ERROR_ACCESS_DENIED	The module type has not been configured.

All other returned error codes are system error conditions.

4.1.3.4 IOCTL_TPMC501_CONF_MOD_TYPE

This TPMC501 control function specifies the modeltype of the TPMC501. The *lpInBuffer* parameter passes a pointer to an unsigned long value to the driver which contains parameters required to perform the operation. The *lpOutBuffer* parameter will not be used. This function can not be called if the sequencer is started. The unsigned long value specifies the model type, the following values are valid:

Value	Description
TPMC501_TYPE_10	TPMC501-10 (Gain 1/2/5/10, +/-10V, Front I/O)
TPMC501_TYPE_11	TPMC501-11 (Gain 1/2/4/8, +/-10V, Front I/O)
TPMC501_TYPE_12	TPMC501-12 (Gain 1/2/5/10, 0-10V, Front I/O)
TPMC501_TYPE_13	TPMC501-13 (Gain 1/2/4/8, 0-10V, Front I/O)
TPMC501_TYPE_20	TPMC501-20 (Gain 1/2/5/10, +/-10V, Back I/O)
TPMC501_TYPE_21	TPMC501-21 (Gain 1/2/4/8, +/-10V, Back I/O)
TPMC501_TYPE_22	TPMC501-22 (Gain 1/2/5/10, 0-10V, Back I/O)
TPMC501_TYPE_23	TPMC501-23 (Gain 1/2/4/8, 0-10V, Back I/O)

This function must be called before the first AD conversion can be started.

Example

```
#include “tpm501.h”

HANDLE        hDevice;
BOOLEAN       success;
ULONG         NumBytes;
ULONG         modelType;

//
// Tell the driver we are using a TPMC501-10
//
modelType = TPMC501_TYPE_10;
success = DeviceIoControl (
     hDevice,
     IOCTL_TPMC501_CONF_MOD_TYPE,
     &modelType,
     sizeof(modelType),
     NULL,
     0,
     &NumBytes,
     NULL
);
…
```

```
if (!success)
{
     ErrorHandler ( "Device I/O control error” );
}
```

Error Codes

ERROR_INVALID_PARAMETER	This error will be returned if the size of the read/write buffer is too small or at least one of the parameters is invalid.

All other returned error codes are system error conditions.

4.1.3.5 IOCTL_TPMC501_MOD_INFO

This control function returns module information data such as configured module type, location on the PCI bus and factory programmed correction data.

The information data is returned in the data structure TPMC501_INFO_BUFFER pointed by *lpOutBuffer*. The argument *nOutBufferSize* specifies the size of the buffer.

```
typedef struct
{
   UINT32     Variant;
   UINT32     PciBusNo;
   UINT32     PciDevNo;
   UINT32     ADCOffsetCal[4];
   UINT32     ADCGainCal[4];
} TPMC501_INFO_BUFFER, *PTPMC501_INFO_BUFFER;
```

Variant

This parameter returns the configured module variant (e.g. 10 for a TPMC501-10).

PciBusNo, PciDevNo

These parameters specifies the PCI location of this module

ADCOffsetCal[4]

This array returns the factory programmed offset correction value for the different gains. Array index 0 contains the value for gain 1, index 1 contains the value for gain 2 and so forth.

ADCGainCal[4]

This array returns the factory programmed gain correction for the different gains. Array index 0 contains the value for gain 1, index 1 contains the value for gain 2 and so forth.

Example

```
#include “tpmc501.h”

HANDLE                  hDevice;
BOOLEAN                 success;
ULONG                   NumBytes;
TPMC501_INFO_BUFFER     ModuleInfo


success = DeviceIoControl (
     hDevice,
     IOCTL_TPMC501_MOD_INFO,
     NULL,
     0,
     &ModuleInfo,
     sizeof(TPMC501_INFO_BUFFER),
     &NumBytes,
     NULL
);

if (!success )
{
     ErrorHandler ( "Device I/O control error” );
}
```

Error Codes

ERROR_INVALID_PARAMETER	This error will be returned if the size of the read/write buffer is too small or at least one of the parameters is invalid.

All other returned error codes are system error conditions.