

TPMC501-SW-65

Windows 2000/XP Device Driver

Optically Isolated 32 Channel 16 Bit ADC

Version 1.0.x

User Manual

Issue 1.0.1

May 2009

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7
25469 Halstenbek, Germany
www.tews.com

Phone: +49 (0) 4101 4058 0
Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com

TEWS TECHNOLOGIES LLC

9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com

Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

TPMC501-SW-65

Windows 2000/XP Device Driver

Optically Isolated 32 Channel 16 Bit ADC

Supported Modules:
TPMC501

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.

©2004-2009 by TEWS TECHNOLOGIES GmbH

| Issue | Description | Date |
|-------|--|----------------|
| 1.0 | First Issue | August 4, 2004 |
| 1.0.1 | General revision, new address TEWS LLC | May 20, 2009 |

Table of Contents

| | | |
|----------|--|----------|
| 1 | INTRODUCTION..... | 4 |
| 2 | INSTALLATION..... | 5 |
| | 2.1 Software Installation..... | 5 |
| | 2.1.1 Windows 2000 / XP..... | 5 |
| | 2.1.2 Confirming Windows 2000 / XP Installation..... | 5 |
| 3 | TPMC501 DEVICE DRIVER PROGRAMMING..... | 6 |
| | 3.1 TPMC501 Files and I/O Functions..... | 6 |
| | 3.1.1 Opening a TPMC501 Device..... | 6 |
| | 3.1.2 Closing a TPMC501 Device..... | 8 |
| | 3.1.3 TPMC501 Device I/O Control Functions..... | 9 |
| | 3.1.3.1 IOCTL_TP501_READ..... | 11 |
| | 3.1.3.2 IOCTL_TP501_START_SEQ..... | 14 |
| | 3.1.3.3 IOCTL_TP501_STOP_SEQ..... | 19 |
| | 3.1.3.4 IOCTL_TP501_CONF_MOD_TYPE..... | 20 |

1 Introduction

The TPMC501-SW-65 Windows WDM (Windows Driver Model) device driver is a kernel mode driver which allows the operation of the TPMC501 on an Intel or Intel-compatible x86 Windows 2000 or Windows XP operating system.

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

The 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 on the distribution media:

Directory path 'TPMC501-SW-65':

| | |
|-------------------------|---|
| tpmc501.sys | Windows driver binary |
| tpmc501.h | TPMC501 include file for driver and application |
| tpmc501.inf | Windows installation script |
| example/tpmc501exa.c | Microsoft Visual C example application |
| TPMC501-SW-65-1.0.1.pdf | PDF copy of this manual |
| ChangeLog.txt | Release history |
| Release.txt | Release information |

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, TPMC501-SW-65.pdf) 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 Confirming Windows 2000 / XP Installation

To confirm that the driver has been properly loaded in Windows 2000 / XP, perform the following steps:

1. From Windows 2000 / XP, open the "**Control Panel**" from "**My Computer**".
2. Click the "**System**" icon and choose the "**Hardware**" tab, and then click the "**Device Manager**" button.
3. Click the "+" in front of "**Other Devices**".
The driver "**TPMC501 32(16) Channel 16 bit ADC**" should appear.

3 TPMC501 Device Driver Programming

The TPMC501-SW-65 Windows WDM 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.)

3.1 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.

3.1.1 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 dwCreationDisposition,
    DWORD dwFlagsAndAttributes,
    HANDLE hTemplateFile
);
```

Parameters

LPCTSTR 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.

DWORD 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)

DWORD dwShareMode

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

LPSECURITY_ATTRIBUTES*lpSecurityAttributes*

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

DWORD *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**.

DWORD *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_TP501_START_SEQ).

HANDLE *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()

3.1.2 Closing a TPMC501 Device

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

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

Parameters

HANDLE *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 ()

3.1.3 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,          // handle to device of interest
    DWORD           dwIoControlCode, // control code of operation to perform
    LPVOID          lpInBuffer,       // pointer to buffer to supply input data
    DWORD           nInBufferSize,    // size of input buffer
    LPVOID          lpOutBuffer,      // pointer to buffer to receive output data
    DWORD           nOutBufferSize,   // size of output buffer
    LPDWORD         lpBytesReturned,  // pointer to variable to receive output byte count
    LPOVERLAPPED    lpOverlapped     // pointer to overlapped structure for asynchronous
                                     // operation
);

```

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_TP501_READ | Read a converted AD value |
| IOCTL_TP501_START_SEQ | Setup and start the sequencer |
| IOCTL_TP501_STOP_SEQ | Stop the sequencer |
| IOCTL_TP501_CONF_MOD_TYPE | Configure which model type is mounted |

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()`

3.1.3.1 IOCTL_TP501_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 (TP501_CHAN_CONF) to the driver which contains parameter required to perform the operation.

```
typedef struct {
    ULONG ChanToUse;        // channel number to use 1..32
    ULONG gain;             // gain to use for this channel
    ULONG flags;            // flags to control the operation
} TP501_CHAN_CONF, *PTP501_CHAN_CONF;
```

ChanToUse

Specifies the channel number at which to start the AD conversion. Valid channels for single-ended mode are 1..32. For differential mode only the channels from 1...16 are possible.

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 |
|------------|---|
| TP501_DIFF | If this bit is set the ADC input works in differential mode otherwise in single-ended (default). |
| TP501_CORR | Perform an offset and gain correction with factory calibration data stored in the TPMC501 EEPROM. |

Example

```
#include "tpmc501.h"

HANDLE          hDevice;
BOOLEAN         success;
ULONG           NumWritten;
long            ReadData;
TP501_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     = TP501_CORR;

success = DeviceIoControl (
    hDevice,                // TPMC501 handle
    IOCTL_TP501_READ,       // read AD value
    &ChanConf,               // channel configuration
    sizeof( ChanConf ),     // size of ChanConf structure
    &ReadData,               // contains the converted data
    sizeof(long),           // size of output buffer
    &NumWritten,             // unused but required
    NULL                    // unused here
);

if( success ) {
    printf( "ADC Value = %d\n", ReadData );
}
else {    // process error
    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.

3.1.3.2 IOCTL_TP501_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 {
    ULONG CycleTime;           // sequencer cycle time in 100 µs steps
    ULONG NumOfBufferPages;    // number of AD data pages
    ULONG NumOfChannels;       // number of active channels
    TP501_CHAN_CONF ChanConf[TP501_MAX_CHAN];
} TP501_JOB_DESC, *PTP501_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[TP501_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_TP501_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_TP501_STOP_SEQ` control function.

```
typedef struct {
    ULONG status;           // sequencer error status
    ULONG PutIndex;         // index of the next page to write by the driver
    ULONG GetIndex;         // index of the next page to read by the user task
    long buffer[1];         // dynamic expandable array of AD values
} TP501_RING_BUFFER, *PTP501_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 |
|---------------------|---|
| TP501_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. |
| TP501_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. |
| TP501_TIMER_ERR | Sequencer timer error (see also TPMC501 hardware manual). The sequencer was stopped after this error occurred. |
| TP501_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;
TP501_CHAN_CONF        ChanConf;
TP501_JOB_DESC         job;
PTP501_RING_BUFFER     pRing;

//
//  Allocate enough memory for the ring buffer and initialize the
//  buffer control header
//
pRing = (PTP501_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     = TP501_CORR;

job.ChanConf[1].ChanToUse = 20;
job.ChanConf[1].gain      = 5;
job.ChanConf[1].flags     = TP501_CORR;

job.ChanConf[2].ChanToUse = 5;
job.ChanConf[2].gain      = 2;
job.ChanConf[2].flags     = TP501_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( TP501_RING_BUFFER, buffer ) ) /
                        ( job.NumOfChannels * sizeof( pRing->buffer[0] ) );

success = DeviceIoControl (
    hDevice,                      // TPMC501 handle
    IOCTL_TP501_START_SEQ,       // start the sequencer
    &job,                          // pointer to the job structure
    sizeof( job ),               // size of job structure
    pRing,                       // pointer to the ring buffer
    RING_BUFFER_SPACE,          // size of memory block
    &NumWritten,                 // unused but required
    &SeqOverlapped               // real overlapped I/O
);

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.

3.1.3.3 IOCTL_TP501_STOP_SEQ

This TPMC501 control function stops the running sequencer and finishes the outstanding (overlapped) *IOCTL_TP501_START_SEQ* device control function.

Example

```
#include "tpmc501.h"

HANDLE          hDevice;
BOOLEAN         success;
ULONG           NumWritten;
OVERLAPPED      DummyOverlapped;
TP501_CHAN_CONF ChanConf;

DummyOverlapped.Offset = 0;
DummyOverlapped.hEvent = 0;

//
// stop the running sequencer
//

success = DeviceIoControl (
    hDevice,                // TPMC501 handle
    IOCTL_TP501_STOP_SEQ,   // stop the sequencer
    NULL,                   // not used
    0,                      // not used
    NULL,                   // not used
    0,                      // not used
    &NumWritten,             // unused but required
    &DummyOverlapped        // unused but required
);

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.

3.1.3.4 IOCTL_TP501_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 |
|---------------|---|
| TP501_TYPE_10 | TPMC501-10 (Gain 1/2/5/10, +/-10V, Front I/O) |
| TP501_TYPE_11 | TPMC501-11 (Gain 1/2/4/8, +/-10V, Front I/O) |
| TP501_TYPE_12 | TPMC501-12 (Gain 1/2/5/10, 0-10V, Front I/O) |
| TP501_TYPE_13 | TPMC501-13 (Gain 1/2/4/8, 0-10V, Front I/O) |
| TP501_TYPE_20 | TPMC501-20 (Gain 1/2/5/10, +/-10V, Back I/O) |
| TP501_TYPE_21 | TPMC501-21 (Gain 1/2/4/8, +/-10V, Back I/O) |
| TP501_TYPE_22 | TPMC501-22 (Gain 1/2/5/10, 0-10V, Back I/O) |
| TP501_TYPE_23 | TPMC501-23 (Gain 1/2/4/8, 0-10V, Back I/O) |

This function must be called before any other I/O control function is called. This function must be used to tell the driver what kind of TPMC501 is used.

Example

```
#include "tpm501.h"

HANDLE          hDevice;
BOOLEAN         success;
ULONG           NumWritten;
ULONG           modelType;

//
// Tell the driver we are using a TPMC501-10
//
modelType = TP501_TYPE_10;
success = DeviceIoControl (
    hDevice,                                // TPMC501 handle
    IOCTL_TP501_CONF_MOD_TYPE,             // configure module type
    &modelType,                             // module type
    sizeof(modelType),                     // size of module type
    NULL,                                   // no output buffer
    0,                                     // no output buffer
    &NumWritten,                           // unused but required
    NULL                                    // not used here
);
```

```
if( success ) {  
    printf( "Success\n" );  
}  
else {    // process error  
    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.