The Embedded I/O Company

TEWS TECHNOLOGIES

TIP501-SW-65

Windows 2000/XP Device Driver

16 Channel 16-Bit ADC

Version 1.0.x

User Manual

Issue 1.0.2

June 2008

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

TIP501-SW-65

Windows 2000/XP Device Driver

16 Channel 16-Bit ADC

Supported Modules:
TIP501

Issue	Description	Date
1.0	First Issue	May 26, 2004
1.0.1	New Address TEWS LLC, file list modified	August 22, 2007
1.0.2	Files moved to subdirectory, Carrier Driver description added	June 20, 2008

Table of Contents

1 Introduction

1.1 Device Driver

The TIP501-SW-65 Windows 2000/XP (WDM – Windows Driver Model) device driver allows the operation of the TIP501 IndustryPack Module conforming to the Windows I/O system specification. This includes a device-independent basic I/O interface (*CreateFile()*, *CloseHandle()*, and *DeviceIoControl()* functions).

Because the TIP501-SW-65 device driver is stacked on the TEWS TECHNOLOGIES IPAC carrier driver, it's necessary to install also the appropriate IPAC carrier driver. Please refer to the IPAC carrier driver user manual for further information.

The TIP501-SW-65 device driver supports the following features:

- reading analog input values from specified channels
- standard and pipeline mode
- single-ended and differential input mode
- automatic offset and gain correction with factory calibration data

The TIP501-SW-65 device driver supports the modules listed below:

TIP501	16 Channel 16-Bit ADC	IndustryPack

To get more information about the features and use of TIP501 devices it is recommended to read the manuals listed below.

TIP501 User Manual

TIP501 Engineering Manual

CARRIER-SW-65 User Manual

1.2 IPAC Carrier Driver

IndustryPack (IPAC) carrier boards have different implementations of the system to IndustryPack bus bridge logic, different implementations of interrupt and error handling and so on. Also the different byte ordering (big-endian versus little-endian) of CPU boards will cause problems on accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which work with any supported carrier board, TEWS TECHNOLOGIES has designed a so called Carrier Driver that hides all differences of different carrier boards under a well defined interface.

The TEWS TECHNOLOGIES IPAC Carrier Driver CARRIER-SW-65 is part of this TIP501-SW-65 distribution. It is located in directory CARRIER-SW-65 on the corresponding distribution media.

This IPAC Device Driver requires a properly installed IPAC Carrier Driver. Due to the design of the Carrier Driver, it is sufficient to install the IPAC Carrier Driver once, even if multiple IPAC Device Drivers are used.

Please refer to the CARRIER-SW-65 User Manual for a detailed description how to install and setup the CARRIER-SW-65 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

Following files are located in directory TIP501-SW-65 on the distribution media:

tip501.sys	Device driver binary
tip501.h	Header file with IOCTL code definitions and driver specific data types
tip501.inf	Installation script
TIP501-SW-65-1.0.2.pdf	This document
\example\tip501exa.c	Microsoft Visual C example application
ChangeLog.txt	Release history
Release.txt	Release information

For installation the files have to be copied to the desired target directory.

2.1 Software Installation

The TIP501 Device Driver software assumes a correctly installed and active IPAC carrier driver.

2.1.1 Windows 2000/XP

This section describes how to install the TIP501 Device Driver on a Windows 2000/XP operating system.

After installing the TIP501 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) Insert the TIP501 driver media; and select "***Disk Drive***" and/or "***CD-ROM***" in the dialog box.
Click "***Next***" button to continue.

(4) Now the driver wizard should find a suitable device driver on the media.
Click "***Next***" button to continue.

(5) Completing the upgrade device driver and click "***Finish***" to take all the changes effect.

(6) Now copy all needed files (tip501.h) to the desired target directories.

After successful installation the TIP501 device driver will start immediately and creates devices (TIP501_1, TIP501_2, ...) for all recognized TIP501 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 "***TIP501***" should appear.

(4) TIP501 Device Driver Programming

The TIP501-SW-65 Windows 2000/XP 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 device 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 TIP501 Device Driver Programming

The TIP501-SW-65 Windows 2000/XP 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 device 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 I/O Functions

The following section doesn't contain a full description of the Win32 functions for interaction with the TIP501 device driver. Only the required parameters are described in detail.

4.1.1 Opening a TIP501 Device

Before you can perform any I/O the *TIP501* device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the *TIP501* device.

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

Parameters

lpFileName

Points to a null-terminated string that specifies the name of the TIP501 to open.
The *lpFileName* string should be of the form **\\.\tip501_*x*** to open the device *x.* The ending x is a one-based number. The first device found by the driver is \\.\tip501_1, the second \\.\tip501_2 and so on.

dwDesiredAccess

Specifies the type of access to the TIP501. For the TIP501 this parameter must be set to read-write access (GENERIC_READ | GENERIC_WRITE).

dwShareMode

A set of bit flags that specifies how the object can be shared for read and write. Unimportant for TIP501, set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for TIP501 devices.

dwCreationDistribution

Specifies which action to take on files that exist and which action to take when files that do not exist. TIP501 devices must be always opened *OPEN_EXISTING*.

dwFlagsAndAttributes

Specifies the file attributes and flags for the file. This value must be set to 0 (no overlapped I/O).

hTemplateFile

This value must be 0 for TIP501 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TIP501 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call **GetLastError**.

Example

```
HANDLE    hDevice;

hDevice = CreateFile(
     “\\\\.\\tip501_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // TIP501 device always open existing
     0,                 // no 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 TIP501 Device

The **CloseHandle** function closes an open TIP501 handle.

BOOL CloseHandle(
HANDLE hDevice)

Parameters

hDevice

Identifies an already opened TIP501 handle.

Return Value

If the function succeeds, the return value is nonzero (TRUE).

If the function fails, the return value is zero (FALSE). 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 TIP501 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 TIP501 that is to perform the operation.

dwIoControlCode

Specifies the control code for an operation. This value identifies the specific operation to be performed. The following values are defined in *TIP501.h*:

Value	Meaning
IOCTL_TIP501_READ	Read analog input value
IOCTL_TIP501_INFO	Read device information data

See behind for more detailed information on each control code.

To use these TIP501 specific control codes the header file tip501.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, in bytes, of the buffer pointed to by *lpInBuffer*.

lpOutBuffer

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

nOutBufferSize

Specifies the size, in bytes, of the buffer 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 value must be set to NULL (no 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***.

The driver returns always standard Win32 error codes on failure, please refer to the Windows Platform SDK Documentation for a detailed description of returned error codes.

See Also

Win32 documentation DeviceIoControl()

4.1.3.1 IOCTL_TIP501_READ

The read function starts an AD conversion at the specified input channel of the TIP501 and returns the converted analog value to the caller. A pointer to the read buffer structure (*T501_READ_BUFFER*) must be passed by the arguments *lpInBuffer* and *lpOutBuffer* to the driver. The arguments *nOutBufferSize* and *nInBufferSize* specifies the length of this buffer.

Before calling the read function some elements of the read buffer must be set to appropriate values (see below for a detailed description of each element). After successful execution the element *data* returns the converted analog input value as a two's complement integer value.

The read function will always use the fastest possible operating mode.

```
typedef struct {
    int     chan;
    int     gain;
    int     flags;
    int     data;
} T501_READ_BUFFER;
```

chan

Specifies the channel number at which the conversion will be started. Valid channel numbers are 1...16 if single-ended is selected or 1...8 for differential input.

gain

Specifies the gain for the input voltage amplifier. Valid gain constants are listed below:

Definition	Gain	Valid for TIP501 Variant
T501_GAIN_1	1	TIP501-10/-11/-20/-21
T501_GAIN_2	2	TIP501-10/-11/-20/-21
T501_GAIN_4	4	TIP501-11/-21
T501_GAIN_5	5	TIP501-10/-20
T501_GAIN_8	8	TIP501-11/-21
T501_GAIN_10	10	TIP501-10/-20

flags

This bit mask controls the read operation; you create the value by the bitwise OR of the appropriate parameters (using the | operator in C).

Flag	**Description**
T501_DIFF	Use differential analog inputs. If this flag is omitted single-ended inputs will be selected
T501_PIPELINE	Use data pipeline mode. In this mode converted data from the previous conversion will be read and a new conversion is started. If this flag is omitted, data from the new started conversion will be read (see also the TIP501 User Manual for more information)
T501_CORRECTION	Perform an automatic offset and gain correction with factory calibration data stored in the TIP501 ID-PROM. If this flag is omitted the converted data will be read directly

data

Converted analog input value (two's complement).

Example

```
#include “tip501.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
T501_READ_BUFFER rdBuf;


rdBuf.chan    = 1;
rdBuf.gain    = T501_GAIN_1;
rdBuf.flags   = T501_DIFF | T501_CORRECTION;

success = DeviceIoControl (
     hDevice,               // TIP501 handle
     IOCTL_TIP501_READ,
     &rdBuf,                // parameter for the driver
     sizeof(T501_READ_BUFFER),
     &rdBuf,                // data from the driver
     sizeof(T501_READ_BUFFER),
     &NumBytes,             // size of returned Buffer
     0);

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

Error Codes

ERROR_INSUFFICIENT_BUFFER	The input or output buffer is too small. Please check the parameters *nInBufferSize* and *nOutBufferSize* of the DeviceIoControl () function call
ERROR_MEMBER_NOT_IN_GROUP	Invalid channel number
ERROR_INVALID_PARAMETER	Some parameters are out of range or invalid (gain, mode or correction)
ERROR_SEM_TIMEOUT	Timeout occurred during conversion

See Also

Win32 documentation DeviceIoControl(), TIP501 Hardware User Manual

4.1.3.2 IOCTL_TIP501_INFO

This control function reads the module variant and the factory calibration data from the specified device and returns this information in the *T501_INFO_BUFFER* structure to the caller.

A pointer to the *T501_INFO_BUFFER* structure is passed by the argument *lpOutBuffer* to the driver. The argument *nOutBufferSize* specifies the length of this buffer.

```
typedef struct {
    int         variant;
    short       offset_corr[4];
    short       gain_corr[4];
} T501_INFO_BUFFER;
```

variant

Returns the module variant.

value	module variant
10	TIP501-10
11	TIP501-11
20	TIP501-20
21	TIP501-21

offset_corr

The factory programmed correction data for offset correction is returned in this array. The index of the array specifies the input gain. (See table below)

gain_corr

The factory programmed correction data for gain correction is returned in this array. The index of the array specifies the input gain. (See table below)

Index	Gain (TIP501-10/20)	Gain (TIP501-11/21)
0	1	1
1	2	2
2	5	4
3	10	8

Example

```
#include “tip501.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
T501_INFO_BUFFER    infoBuf;

…
```

…

```
success = DeviceIoControl (
     hDevice,                // TIP501 handle
     IOCTL_TIP501_INFO,
     NULL,                   // not used, set to NULL
     0,                      // not used, set to 0
     &infoBuf,
     sizeof(T501_INFO_BUFFER),
     &NumBytes,
     0);


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

Error Codes

ERROR_INSUFFICIENT_BUFFER	The output buffer is too small. Please check the argument *nOutBufferSize.*

ERROR_INSUFFICIENT_BUFFER	The output buffer is too small. Please check the argument *nOutBufferSize*

See Also

Win32 documentation DeviceIoControl(), TIP501 Hardware User Manual