The Embedded I/O Company

TIP551-SW-65

Windows 2000/XP Device Driver

4 Channel 16-Bit DAC

Version 1.1.x

User Manual

Issue 1.1.0

April 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

TIP551-SW-65

Windows 2000/XP Device Driver

4 Channel 16-Bit DAC

Supported Modules:
TIP551

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.

Issue	Description	Date
1.0	First Issue	May 26, 2004
1.1.0	General Revision, New Address TEWS LLC	April 7, 2009

Table of Contents

1 Introduction

1.1 Device Driver

The TIP551-SW-65 Windows 2000/XP (WDM – Windows Driver Model) device driver allows the operation of the TIP551 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 TIP551-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 TIP551-SW-65 device driver supports the following features:

- set voltage for a single DAC
- set voltages to more than one channel simultaneously
- read module version, current configuration and calibration data

The TIP551-SW-65 device driver supports the modules listed below:

TIP551	4 Channel 16-Bit DAC	IndustryPack®

To get more information about the features and use of TIP551 devices it is recommended to read the manuals listed below.

TIP551 User manual

TIP551 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 TIP551-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 TIP551-SW-65 on the distribution media:

tip551.sys	Device driver binary
tip551.h	Header file with IOCTL code definitions and driver specific data types
tip551.inf	Installation script
TIP551-SW-65-1.1.0.pdf	This document
\example\tip551exa.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 TIP551 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 TIP551 Device Driver on a Windows 2000/XP operating system.

After installing the TIP551 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 TIP551 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 (tip551.h) to the desired target directories.

After successful installation the TIP551 device driver will start immediately and creates devices (TIP551_1, TIP551_2, ...) for all recognized TIP551 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 "***TIP551***" should appear.

3 TIP551 Device Driver Programming

The TIP551-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 TIP551 device driver. Only the required parameters are described in detail.

4.1.1 Opening a TIP551 Device

Before you can perform any I/O the *TIP551* device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the *TIP551* 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 **\\.\tip551_*x*** to open the device *x.* The ending x is a one-based number. The first device found by the driver is \\.\tip551_1, the second \\.\tip551_2 and so on.

dwDesiredAccess

Specifies the type of access to the TIP551. For the TIP551 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 TIP551, set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for TIP551 devices.

dwCreationDistribution

Specifies which action to take on files that exist and which action to take when files that do not exist. TIP551 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 TIP551 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TIP551 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call **GetLastError**.

Example

```
HANDLE    hDevice;

hDevice = CreateFile(
     “\\\\.\\tip551_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // TIP551 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 TIP551 Device

The **CloseHandle** function closes an open TIP551 handle.

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

Parameters

hDevice

Identifies an already opened TIP551 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 TIP551 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 TIP551 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 *tip551.h*:

Value	**Meaning**
IOCTL_TIP551_WRITE	Set analog output value
IOCTL_TIP551_INFO	Read device information data

See behind for more detailed information on each control code.

To use these TIP551 specific control codes the header file tip551.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_TIP551_WRITE

This control function writes to the specified DAC channel(s) of the TIP551 from a structure (*T551_WRITE_BUFFER)* pointed to by *lpInBuffer*. The argument *nInBufferSize* specifies the length of the buffer.

The write function also supports the simultaneous update feature of the TIP551. It is possible to update the DAC outputs of up to 4 channels with one write. In this case the internal DAC data registers of the specified DAC channels will be loaded with new data and then the data conversion is started simultaneously at all DAC channels.

Using this mode requires an array of the structure *T551_WRITE_BUFFER* with a length of the number of channels to update. Each array item must be setup to a appropriate values for the DAC channel (*chan*), data correction (*corr*) and the new DAC value (*data*). The order of channels in this array can be arbitrary. To tell the driver that this is a multi channels write the argument *nInBufferSize* must be set to the length of the array (see also the example below).

```
typedef struct {
        int         chan;
        int         corr;
        int         data;
} T551_WRITE_BUFFER;
```

chan

Selects the DAC channel. Valid channel numbers are 1...4.

corr

Set this parameter to TRUE (1) to perform an automatic gain and offset correction with calibration data stored in the IDPROM. FALSE (0) means do not perform any correction and write directly to the DAC output.

data

Contains the new DAC output value.

Output Mode	data range	voltage range
Unipolar	0...65535	0...10V
Bipolar	-32768...32767	-10V...10V

Example

```
#include “tip551.h”

HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
T551_WRITE_BUFFER   wrBuf;
T551_WRITE_BUFFER   wrBufArray[T551_MAX_CHAN];

…
```

```
…

wrBuf.chan    =   1;            /* first channel */
wrBuf.corr    =   TRUE;         /* data correction */
wrBuf.data    =   65535;        /* full-scale if unipolar output */

success = DeviceIoControl (
     hDevice,                // TIP551 handle
     IOCTL_TIP551_WRITE,
     &wrBuf,                 // parameter for the driver
     sizeof(T551_WRITE_BUFFER),
     NULL,                   // no data returned
     0,
     &NumBytes,              // size of returned Buffer
     0
);
if( !success ) {
     ErrorHandler ("Device I/O control error”); // process error
}

…

/*
**       Update channel 1, 3 and 4 with new data and let channel 2
**       unchanged.
*/
wrBufArray[0].chan =   3;
wrBufArray[0].corr =   TRUE;         /* data correction */
wrBufArray[0].data =   -32768;       /* -full-scale if bipolar output */

wrBufArray[1].chan =   1;
wrBufArray[1].corr =   TRUE;         /* data correction */
wrBufArray[1].data =   0;            /* mid-scale if bipolar output */

wrBufArray[2].chan =   4;
wrBufArray[2].corr =   TRUE;         /* data correction */
wrBufArray[2].data =   32767;        /* +full-scale if bipolar output */

…
```

```
…

/*
**        Please note the size of the write buffer array is 3 times the
**        size of structure T551_WRITE_BUFFER
*/
success = DeviceIoControl (
     hDevice,                   // TIP551 handle
     IOCTL_TIP551_WRITE,
     &wrBufArray[0],            // parameter for the driver
     3 * sizeof(T551_WRITE_BUFFER),
     NULL,                      // no data returned
     0,
     &NumBytes,                 // size of returned Buffer
     0
);

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

Error Codes

ERROR_INSUFFICIENT_BUFFER	The input buffer is too small or if the specified channel number is out of range.
ERROR_SEM_TIMEOUT	Timeout occurred during conversion.

4.1.3.2 IOCTL_TIP551_INFO

This control function reads the module variant, the current output voltage range and the factory calibration data of the specified device and returns this information in the *T551_INFO_BUFFER* structure to the caller.

A pointer to the *T551_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;
        int             voltage_range;
        short           offset_corr[T551_MAX_CHAN];
        short           gain_corr[T551_MAX_CHAN];
} T551_INFO_BUFFER;
```

variant

Returns the module variant.

value	module variant
10	TIP551-10

voltage_range

Returns the actual output voltage range for all channels.

value	symbol	voltage range
1	T551_UNIPOLAR	0V ... 10V
2	T551_BIPOLAR	-10V ... 10V

offset_corr

Returns the factory offset calibration data for channel 1...4 in the unit ¼ LSB.

gain_corr

Returns the factory gain calibration data for channel 1...4 in the unit ¼ LSB.

Example

```
#include “tip551.h”


HANDLE              hDevice;
BOOLEAN             success;
ULONG               NumBytes;
T551_INFO_BUFFER    infoBuf;

…
```

…

```
success = DeviceIoControl (
     hDevice,                 // TIP551 handle
     IOCTL_TIP551_INFO,
     NULL,                    // not used, set to NULL
     0,                       // not used, set to 0
     &infoBuf,
     sizeof(T551_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*.