The Embedded I/O Company

TEWS TECHNOLOGIES

TIP700-SW-65

Windows 2000/XP Device Driver

Digital Output 24V DC

Version 1.2.x

User Manual

Issue 1.2.1

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

TIP700-SW-65

Windows 2000/XP Device Driver

Digital Output 24V DC

Supported Modules:
TIP700

Issue	Description	Date
1.0	First Issue	October 28, 2003
1.1	Description of *IOCTL_TIP700_READ* corrected	May 17, 2004
1.2	WinXP description added, Win98/Me description removed	June 15, 2004
1.2.1	General Revision, new address TEWS LLC	May 29, 2009

Table of Contents

1 Introduction

1.1 Device Driver

The TIP700-SW-65 Windows WDM (Windows Driver Model) device driver is a kernel mode driver which allows the operation of the TIP700 on Intel or Intel-compatible x86 Windows 2000 or Windows XP operating systems.

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.

Because the TIP700 device driver is stacked on the TEWS TECHNOLOGIES IPAC Carrier Driver, it is necessary to install also the appropriate IPAC Carrier Driver. Please refer to the IPAC Carrier Driver user manual for further information.

The TIP700-SW-65 device driver supports the following features:

- writing a new output value
- read actual output value
- start, stop and trigger the output watchdog

The TIP700-SW-65 device driver supports the modules listed below:

TIP700-10	16 isolated digital outputs 24V DC	IndustryPack® compatible
TIP700-20	8 isolated digital outputs 24V DC	IndustryPack® compatible

To get more information about the features and use of TIP700 devices it is recommended to read the manuals listed below.

TIP700 User manual

TIP700 Engineering 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 TIP700-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 on the distribution media:

Directory path 'TIP700-SW-65':

tip700.sys	Device driver binary
tip700.h	Header file with IOCTL code definitions and driver specific data types
tip700.inf	Installation script
TIP700-SW-65-1.2.1.pdf	This document in PDF format
example\tip700exa.c	example application C source file
ChangeLog.txt	Release history
Release.txt	Release information

2.1 Software Installation

The TIP700-SW-65 Device Driver software assumes a correctly installed and active TEWS TECHNOLOGIES IPAC Carrier Driver.

2.1.1 Windows 2000/XP

This section describes how to install the TIP700 Device Driver on a Windows 2000/XP operating system.

After installing the TIP700 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 TIP700 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 (tip700.h, ...) to the desired target directories.

After successful installation the TIP700 device driver will start immediately and create devices (tip700_1, tip700_2, ...) for all recognized TIP700 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 " ***TEWS TECHNOLOGIES TIP700 (Digital Output 24V)***" should appear.

3 TIP700 Device Driver Programming

The TIP700-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.)

3.1 TIP700 Files and I/O Functions

The following section doesn't contain a full description of the Win32 functions for interaction with the TIP700 device driver. Only the required parameters are described in detail.

3.1.1 Opening a TIP700 Device

Before you can perform any I/O the TIP700 device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the TIP700 device.

```
HANDLE CreateFile
(
    LPCTSTR lpFileName,                          // pointer to filename
    DWORD dwDesiredAccess,                       // access (read-write) mode
    DWORD dwShareMode,                           // share mode
    LPSECURITY_ATTRIBUTES lpSecurityAttributes,  // pointer to security attributes
    DWORD dwCreationDistribution,                // how to create
    DWORD dwFlagsAndAttributes,                  // file attributes
    HANDLE hTemplateFile                         // handle to file with attributes to copy
)
```

Parameters

lpFileName

Points to a null-terminated string that specifies the name of the TIP700 to open.
The *lpFileName* string should be of the form **\\.\tip700_*x*** to open the device *x*. The ending x is a one-based number. The first device found by the driver is \\.\tip700_1, the second \\.\tip700_2 and so on.

dwDesiredAccess

Specifies the type of access to the TIP700. For the TIP700 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. Not used for TIP700, set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for TIP700 devices.

dwCreationDistribution

Specifies which action to take on files that exist and which action to take when files that do not exist. TIP700 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 TIP700 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TIP700 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call **GetLastError**.

Example

```
HANDLE    hDevice;

hDevice = CreateFile(
     "\\\\.\\tip700_1",
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // TIP700 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()

3.1.2 Closing a TIP700 Device

The **CloseHandle** function closes an open TIP700 handle.

```
BOOL CloseHandle
(
      HANDLE hDevice;                          // handle to a TIP700 device to close
)
```

Parameters

hDevice

Identifies an already opened TIP700 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 TIP700 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 TIP700 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 *tip700.h*:

Value	Meaning
IOCTL_TIP700_READ	Read the output value
IOCTL_TIP700_WRITE	Write a new value to the output port
IOCTL_TIP700_ENABLE_WD	Enable output watchdog
IOCTL_TIP700_DISABLE_WD	Disable output watchdog
IOCTL_TIP700_TRIGGER_WD	Trigger output watchdog

See behind for more detailed information on each control code.

To use these TIP700 specific control codes, the header file tip700.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()

3.1.3.1 IOCTL_TIP700_READ

The read function reads back the actual content of the output register.

The parameter *lpInBuffer* must pass a NULL pointer to the device driver to the device driver. The parameter and *lpOutBuffer* must pass a pointer to the read buffer (*USHORT*) to the device driver.

The value of the output register will be copied into the specified buffer.

Example

```
#include “tip700.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;
USHORT    RWBuf;

/*
**  Read content of output register
*/
success = DeviceIoControl (
     hDevice,               // TIP700 handle
     IOCTL_TIP700_READ,
     NULL,                  // parameter for the driver
     0,
     &RWBuf,                // contains the read data
     sizeof(USHORT),
     &NumBytes,             // size of returned Buffer
     0
);
if( success ) {
     printf(“Output Register = 0x%x\n”, RWBuf);
}
else {
     ErrorHandler ( "Device I/O control error” );    // process error
}
```

3.1.3.2 IOCTL_TIP700_WRITE

The Write function writes a value to the output port, and triggers the output watchdog (if enabled).

The parameter *lpInBuffer* must pass a pointer to the write buffer (*USHORT*) to the device driver. *lpOutBuffer* must pass a NULL pointer to the device driver.

The new output value must be stored into the specified buffer.

Example

```
#include “tip700.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;
USHORT    RWBuf;

/*
**  Write a new value (0x1234) to the output Port
*/
RWBuf = 0x1234;

success = DeviceIoControl (
     hDevice,               // TIP700 handle
     IOCTL_TIP700_WRITE,
     &RWBuf,                // parameter for the driver
     sizeof(USHORT),
     NULL,                  // no data returned
     0,
     &NumBytes,             // size of returned Buffer
     0
);
if( success ) {
     printf(“Write successfully completed\n”);
}
else {
     ErrorHandler ("Device I/O control error”); // process error
}
```

3.1.3.3 IOCTL_TIP700_ENABLE_WD

This function enables the output watchdog. The watchdog is disabled after power up or reset.

The parameters *lpInBuffer* and *lpOutBuffer* must pass a NULL pointer to the device driver.

Example

```
#include “tip700.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;

/*
**  Enable watchdog
*/
success = DeviceIoControl (
     hDevice,                    // TIP700 handle
     IOCTL_TIP700_ENABLE_WD,
     NULL,                       // parameter for the driver
     0,
     NULL,                       // no data returned
     0,
     &NumBytes,                  // size of returned Buffer
     0
);

if( success ) {
     printf( “Enable watchdog successfully completed\n” );
}
else {
     ErrorHandler ( "Device I/O control error” );   // process error
}
```

3.1.3.4 IOCTL_TIP700_DISABLE_WD

This function disables the output watchdog.

The parameter *lpInBuffer* and *lpOutBuffer* must pass a NULL pointer to the device driver.

Example

```
#include “tip700.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;

/*
**  Disable watchdog
*/
success = DeviceIoControl (
     hDevice,                    // TIP700 handle
     IOCTL_TIP700_DISABLE_WD,
     NULL,                       // no input data
     0,
     NULL,                       // no data returned
     0,
     &NumBytes,                  // size of returned Buffer
     0
);
if( success ) {
      printf( “Disable watchdog successfully completed\n” );
}
else {
     ErrorHandler ( "Device I/O control error” );    // process error
}
```

3.1.3.5 IOCTL_TIP700_TRIGGER_WD

This function triggers the output watchdog. This let start the watchdog time to run again. This function must be called at least every 120ms to prevent the watchdog from disabling the output lines.

The parameter *lpInBuffer* and *lpOutBuffer* must pass a NULL pointer to the device.

Example

```
#include “tip700.h”

HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;

/*
**  Trigger watchdog
*/
success = DeviceIoControl (
     hDevice,                    // TIP700 handle
     IOCTL_TIP700_TRIGGER_WD,
     NULL,                       // no input data
     0,
     NULL,                       // no data returned
     0,
     &NumBytes,                  // size of returned Buffer
     0
);
if( success ) {
     printf( “Trigger watchdog successfully completed\n” );
}
else {
     ErrorHandler ( "Device I/O control error” );   // process error
}
```