The Embedded I/O Company

TEWS TECHNOLOGIES

TIP710-SW-65

Windows 2000/XP Device Driver

16 Digital Output, 6V to 48V DC, High Side Switch

Version 1.0.x

User Manual

Issue 1.0.3

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

TIP710-SW-65

Windows 2000/XP Device Driver

16 Digital Output, 6V to 48V DC, High Side Switch

Issue	Description	Date
1.0	First Issue	May 26, 2004
1.0.1	ChangeLog.txt added, file list modified, close() Routine example corrected, issue number format changed and general review	July 24, 2006
1.0.2	New address TEWS LLC	May 10, 2007
1.0.3	Files moved to subdirectory, Carrier Driver description added	June 23, 2008

Table of Contents

1 Introduction

1.1 Device Driver

The TIP710-SW-65 Windows WDM (Windows Driver Model) device driver is a kernel mode driver which allows the operation of the TIP710 on an 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 TIP710 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 TIP710 device driver includes the following functions:

- writing a new output value
- starting, stopping and triggering the output watchdog

The TIP710-SW-65 supports the modules listed below:

TIP710-10	16 Digital Outputs, 6V to 48V DC, High Side Switch (IPAC)

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

TIP710 User manual

TIP710 Engineering Manual

CARRIER-SW-65 IPAC Carrier 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 TIP710-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 TIP710-SW-65 on the distribution media:

tip710.sys	Device driver binary
tip710.h	Header file with IOCTL code definitions
tip710.inf	Installation script
TIP710-SW-65-1.0.3pdf	This document
\example\tip710exa.c	Microsoft Visual C example application
Release.txt	Information about the Device Driver Release
ChangeLog.txt	Information about the Device Driver Release history

2.1 Software Installation

The TIP710 Device Driver software requires a correctly installed and active TEWS TECHNOLOGIES IPAC carrier driver.

2.1.1 Windows 2000/XP

This section describes how to install the TIP710 Device Driver on a Windows 2000/XP operating system.

After installing the TIP710 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 TIP710 driver distribution 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 diskette.
   Click "***Next***" button to continue.
5. If a window shows up announcing that the Windows Logo Test has failed, click **"continue install"** to continue the installation.
6. Complete the device driver installation by clicking "***Finish***" to take all the changes effect.
7. Now copy all needed files (tip710.h, ...) to the desired target directories.

After successful installation the TIP710 device driver will start immediately and creates devices (TIP710_1, TIP710_2, ...) for all recognized TIP710 modules.

2.1.2 Confirming Windows 2000/XP Installation

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

1. From Windows 2000, 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 – TIP710 (16 Digital Outputs, 6V to 48V DC, High Side Switch)***" should appear.

3 TIP710 Device Driver Programming

The TIP710-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 TIP710 Files and I/O Functions

The following section doesn't contain a full description of the Win32 functions for interaction with the TIP710 device driver. Only the required parameters are described in detail.

3.1.1 Opening a TIP710 Device

Before you can perform any I/O the *TIP710* device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the *TIP710* 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 TIP710 to open.
The *lpFileName* string should be of the form **\\.\TIP710_*x*** to open the device *x.* The ending x is a one-based number. The first device found by the driver is \\.\TIP710_1, the second \\.\TIP710_2 and so on.

dwDesiredAccess

Specifies the type of access to the TIP710. For the TIP710 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 TIP710, set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for TIP710 devices.

dwCreationDistribution

Specifies which action to take on files that exist and which action to take when files that do not exist. TIP710 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 TIP710 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TIP710 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call **GetLastError**.

Example

```
HANDLE    hDevice;

hDevice = CreateFile(
     “\\\\.\\TIP710_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // TIP710 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 TIP710 Device

The **CloseHandle** function closes an open TIP710 handle.

```
BOOL CloseHandle(
      HANDLE hDevice;                          // handle to a TIP710 device to close
);
```

Parameters

hDevice

Identifies an open TIP710 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;

hDevice = CreateFile(
     “\\\\.\\TIP710_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no TIP710 device always open existing
     0,                 // no overlapped I/O
     NULL
);
if(hDevice == INVALID_HANDLE_VALUE) {
 ErrorHandler("Could not open device"); // process error
}

/* ... do some device I/O ...  */

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

See Also

CreateFile(), Win32 documentation CloseHandle()

3.1.3 TIP710 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 TIP710 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 *TIP710.h*:

Value	Meaning
IOCTL_TIP710_WRITE	Write a new value to the output port
IOCTL_TIP710_ENABLE_WD	Enable output watchdog
IOCTL_TIP710_DISABLE_WD	Disable output watchdog
IOCTL_TIP710_TRIGGER_WD	Trigger output watchdog

See behind for more detailed information on each control code.

To use these TIP710 specific control codes the header file tip710.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***.

See Also

Win32 documentation DeviceIoControl()

3.1.3.1 IOCTL_TIP710_WRITE

This control function writes a new value to the digital output port. The parameter *lpInBuffer* passes a pointer to an unsigned short variable, which contains the new output value, to the driver.

The parameter *nInBufferSize* must be set to the size of an unsigned short variable (2 byte).

Example

```
#include “tip710.h”

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

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

success = DeviceIoControl (
     hDevice,               // TIP710 handle
     IOCTL_TIP710_WRITE,
     &RWBuf,                // parameter for the driver
     sizeof(USHORT),
     NULL,                  // no data returned
     0,
     &NumBytes,             // size of returned Buffer
     0
);

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

Error Codes

No driver specific error code is returned by this ioctl function. All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP710 Hardware User Manual

3.1.3.2 IOCTL_TIP710_ENABLE_WD

This control function enables the output watchdog of the specified device.

If the watchdog is enabled, the output must be serviced at least every 120 ms or the watchdog will switch the output lines to the passive signal until a new write access (*IOCTL_TIP710_WRITE* or *IOCTL_TIP710_TRIGGER_WD*) is made to the output register.

Example

```
#include “tip700.h”


HANDLE    hDevice;
BOOLEAN   success;
ULONG     NumBytes;


success = DeviceIoControl (
     hDevice,                // TIP710 handle
     IOCTL_TIP710_ENABLE_WD, // enable output watchdog
     NULL,                   // not used, set to NULL
     0,                      // not used, set to 0
     NULL,                   // not used, set to NULL
     0,                      // not used, set to 0
     &NumBytes,              // unused but required
     0
);


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

Error Codes

No driver specific error code is returned by this ioctl function. All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP710 Hardware User Manual

3.1.3.3 IOCTL_TIP710_DISABLE_WD

This function disables the output watchdog.

Example

```
#include “tip710.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;

success = DeviceIoControl (
     hDevice,                    // TIP710 handle
     IOCTL_TIP710_DISABLE_WD,    // disable output watchdog
     NULL,                       // not used, set to NULL
     0,                          // not used, set to 0
     NULL,                       // not used, set to NULL
     0,                          // not used, set to 0
     &NumBytes,                  // unused but required
     0
);


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

Error Codes

No driver specific error code is returned by this ioctl function. All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP710 Hardware User Manual

3.1.3.4 IOCTL_TIP710_TRIGGER_WD

This function triggers the output watchdog by rewriting the output port with the current value. This let start the watchdog time to run again.

Example

```
#include “tip710.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;


success = DeviceIoControl (
     hDevice,                    // TIP710 handle
     IOCTL_TIP710_TRIGGER_WD,    // trigger the output watchdog
     NULL,                       // not used, set to NULL
     0,                          // not used, set to 0
     NULL,                       // not used, set to NULL
     0,                          // not used, set to 0
     &NumBytes,                  // unused but required
     0
);


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

Error Codes

No driver specific error code is returned by this ioctl function. All returned error codes are system error conditions.

See Also

Win32 documentation DeviceIoControl(), TIP710 Hardware User Manual