The Embedded I/O Company

TEWS TECHNOLOGIES

TIP150-SW-65

Windows 2000/XP Device Driver

1 or 2 Channel Synchro/Resolver-To-Digital Converter

Version 1.0.x

User Manual

Issue 1.0.1

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

TIP150-SW-65

Betriebssystem Device Driver

1 or 2 Channel Synchro/Resolver-To-Digital Converter

Supported Modules:
TIP150-3x
TIP150-4x

Issue	Description	Date
1.0.0	First Issue	January 24, 2007
1.0.1	Files moved to subdirectory, Carrier Driver description added	June 20, 2008

Table of Contents

1 Introduction

1.1 Device Driver

The TIP150-SW-65 Windows WDM (Windows Driver Model) device driver is a kernel mode driver which allows the operation of the TIP150 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 TIP150 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 TIP150-SW-65 device driver supports the following features:

- Read position and status synchronously from one channel device
- Read position and status synchronously from both channels of a device
- Configure resolution of the channels

The TIP150-SW-65 device driver supports the modules listed below:

TIP150-3x	1 Channel Synchro/Resolver-To-digital Converter	(IPAC)
TIP150-4x	2 Channel Synchro/Resolver-To-digital Converter	(IPAC)

To get more information about the features and use of TIP150 devices it is recommended to read the manuals listed below.

TIP150 User manual

TIP150 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 TIP150-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 TIP150-SW-65 on the distribution media:

tip150.sys	Device driver binary
tip150.h	Header file with IOCTL code definitions and driver specific data types
tip150.inf	Installation script
TIP150-SW-65-1.0.1.pdf	This document in PDF format
example\tip150exa.c	example application C source file
ChangeLog.txt	Release history
Release.txt	Release information

2.1 Software Installation

The TIP150 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 TIP150 Device Driver on a Windows 2000/XP operating system.

After installing the TIP150 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 TIP150 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 (tip150.h, ...) to the desired target directories.

After successful installation the TIP150 device driver will start immediately and create devices (tip150_1, tip150_2, ...) for all recognized TIP150 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 TIP150 Synchro/Resolver to Digital Converter***" should appear.

3 TIP150 Device Driver Programming

The TIP150-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 TIP150 Files and I/O Functions

The following section doesn't contain a full description of the Win32 functions for interaction with the TIP150 device driver. Only the required parameters are described in detail.

3.1.1 Opening a TIP150 Device

Before you can perform any I/O the TIP150 device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the TIP150 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 TIP150 to open.
The *lpFileName* string should be of the form **\\.\tip150_*x*** to open the device *x.* The ending x is a one-based number. The first device found by the driver is \\.\tip150_1, the second \\.\tip150_2 and so on.

dwDesiredAccess

Specifies the type of access to the TIP150. For the TIP150 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 TIP150, set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for TIP150 devices.

dwCreationDistribution

Specifies which action to take on files that exist and which action to take when files that do not exist. TIP150 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 TIP150 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TIP150 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call **GetLastError**.

Example

```
HANDLE    hDevice;

hDevice = CreateFile(
     "\\\\.\\tip150_1",
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // TIP150 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 TIP150 Device

The **CloseHandle** function closes an open TIP150 handle.

```
BOOL CloseHandle
(
      HANDLE hDevice;                          // handle to a TIP150 device to close
)
```

Parameters

hDevice

Identifies an already opened TIP150 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 TIP150 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 TIP150 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 *tip150.h*:

Value	Meaning
IOCTL_TIP150_READ_SNGL	Synchronously read position and status of a specified channel
IOCTL_TIP150_READ_DBL	Synchronously read position and status of both channels of a specified device
IOCTL_TIP150_CONTROL	Configure resolution of a specified channel

See behind for more detailed information on each control code.

To use these TIP150 specific control codes, the header file tip150.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_TIP150_READ_SNGL

This function reads the current position and status of the specified channel. The data value (position) and the status are synchronized by hardware. A pointer to the single channel read buffer (*TIP150_READ_SNGL_BUF*) must be passed by the arguments *lpInBuffer* and *lpOutBuffer* to the driver. The arguments *nInBufferSize* and *nOutBufferSize* specifies the size of this buffer.

The *TIP150_READ_SNGL_BUF* structure has the following layout:

```
typedef struct
{
        int                     channel;
        unsigned short          data;
        unsigned char           status;
} TIP150_READ_SNGL_BUF;
```

channel

This parameter specifies the channel to read from. Valid channel numbers are 1 and 2, if a two channel TIP150 is used.

data

This parameter returns the data value of the specified channel. Depending on the selected resolution for this channel, the lower bits are set to 0.

status

This parameter returns the value of the status register. The following ORed status flags (defined in 'tip150.h') can be returned:

flag	description
TIP150_STAT_BIT	Built-In-Test failed (refer to TIP150 User Manual)
TIP150_STAT_LOS	Loss of signal (refer to TIP150 User Manual)

Example

```
#include "tip150.h"

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP150_READ_SNGL_BUF snglBuf;

/* Read counter from channel 1 */
snglBuf.channel = 1;

...
```

…

```
success = DeviceIoControl (
     hDevice,                          // TIP150 handle
     IOCTL_TIP150_READ_SNGL,
     &snglBuf,                         // input buffer
     sizeof(TIP150_READ_SNGL_BUF),     // size of input buffer
     &snglBuf,                         // output buffer
     sizeof(TIP150_READ_SNGL_BUF),     // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     printf("Data: %d – Status: %02Xh\n",
                    snglBuf.data,
                    snglBuf.status);
} else {
     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	The specified channel number is out of range

See Also

Win32 documentation DeviceIoControl()

3.1.3.2 IOCTL_TIP150_READ_DBL

This function reads the current position and status of both channels of the specified device. The both data values (position) and both status values are synchronized by hardware. A pointer to the double channel read buffer (*TIP150_READ_DBL_BUF*) must be passed by the argument *lpOutBuffer* to the driver. The argument *nOutBufferSize* specifies the size of this buffer. The argument lpInBuffer is not used and must be set to *NULL*. The argument *nInBufferSize* must be set to 0.

This function is only allowed for two channel TIP150 (TIP150-4x).

The *TIP150_READ_DBL_BUF* structure has the following layout:

```
typedef struct
{
        unsigned short          data[2];
        unsigned char           status[2];
} TIP150_READ_DBL_BUF;
```

data[]

This array returns the data values of the channels. Depending on the selected resolution for the channels, the lower bits are set to 0. Array index 0 returns the value of channel 1 and index 1 returns the value of channel 2.

status[]

This array returns the values of the status registers. Array index 0 returns the value of channel 1 and index 1 returns the value of channel 2.The following ORed status flags (defined in 'tip150.h') can be returned:

flag	description
TIP150_STAT_BIT	Built-In-Test failed (refer to TIP150 User Manual)
TIP150_STAT_LOS	Loss of signal (refer to TIP150 User Manual)

Example

```
#include "tip150.h"

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP150_READ_DBL_BUF dblBuf;

...
```

…

```
/* Read counter from both channels */
success = DeviceIoControl (
     hDevice,                          // TIP150 handle
     IOCTL_TIP150_READ_DBL,
     NULL,                             // input buffer
     0,                                // size of input buffer
     &dblBuf,                          // output buffer
     sizeof(TIP150_READ_DBL_BUF),      // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     printf("#1: Data: %d – Status: %02Xh\n",
                  dblBuf[0].data,
                  dblBuf[0].status);
     printf("#2: Data: %d – Status: %02Xh\n",
                  dblBuf[1].data,
                  dblBuf[1].status);
} else {
     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_ACCESS_DENIED	The specified device is a single channel version, this function will only work for two channel versions.

See Also

Win32 documentation DeviceIoControl()

3.1.3.3 IOCTL_TIP150_CONTROL

This function configures the resolution of the specified channel. A pointer to the control buffer (*TIP150_CONTROL_BUF*) must be passed by the argument *lpInBuffer* to the driver. The argument *nInBufferSize* specifies the size of this buffer. The argument *lpOutBuffer* is not used and should be set to *NULL*. The argument *nOutBufferSize* should be set to 0.

The *TIP150_CONTROL_BUF* structure has the following layout:

```
typedef struct
{
        int                     channel;
        unsigned short          resolution;
} TIP150_CONTROL_BUF;
```

channel

This parameter specifies the channel to configure. Valid channel numbers are 1 and 2, if a two channel TIP150 is used.

resolution

This parameter specifies the new resolution (in bit) of the specified channel. Valid values are 10, 12, 14, and 16.

Example

```
#include “tip150.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP150_CONTROL_BUF cntlBuf;

/* Set resolution from channel 1 to 14-bit */
cntlBuf.channel = 1;
cntlBuf.channel = 14;

…
```

…

```
success = DeviceIoControl (
     hDevice,                          // TIP150 handle
     IOCTL_TIP150_READ_SNGL,
     &cntlBuf,                         // input buffer
     sizeof(TIP150_CONTROL_BUF),       // size of input buffer
     NULL,                             // output buffer
     0,                                // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     printf("OK\n");
} else {
     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	The specified channel number is out of range
ERROR_INVALID_PARAMETER	Illegal resolution specified

See Also

Win32 documentation DeviceIoControl()