The Embedded I/O Company

TEWS TECHNOLOGIES

TIP670-SW-65

Windows 2000/XP Device Driver

Digital I/O

Version 1.1.x

User Manual

Issue 1.1.1

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

TIP670-SW-65

Windows 2000/XP Device Driver

Digital I/O

Supported Modules:
TIP670

Issue	Description	Date
1.0	First Issue	October 20, 2003
1.1	WinXP description added, Win98/Me description removed	June 15, 2004
1.1.0	File list modified, return value of CloseHandle corrected, general revision, Examples corrected	May 30, 2007
1.1.1	Files moved to subdirectory, Carrier Driver description added	June 20, 2008

Table of Contents

1 Introduction

1.1 Device Driver

The TIP670-SW-65 Windows WDM (Windows Driver Model) device driver is a kernel mode driver which allows the operation of the TIP670 on Intel or Intel-compatible x86 Windows 2000, Windows XP and Windows XP embedded 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 TIP670 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 TIP670-SW-65 device driver includes the following functions:

- writing a new output value
- read the input port immediately without waiting for a specific input event
- read the input port after the following events occur
  - masked input bits match to the specified pattern
  - high-transition at the specified bit position
  - low-transition at the specified bit position
  - any transition (high or low) at the specified bit position
- start, stop and trigger the output watchdog

The TIP670-SW-65 device driver supports the modules listed below:

TIP670-10	8 channel digital input and 8 channel digital output	IPAC
TIP670-20	4 channel digital input and 4 channel digital output	IPAC

To get more information about the features and use of TIP670 devices it is recommended to read the manuals listed below.

TIP670 User manual

TIP670 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 TIP670-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 TIP670-SW-65 on the distribution media:

tip670.sys	Device driver binary
tip670.h	Header file with IOCTL code definitions and driver specific data types
tip670.inf	Installation script
TIP670-SW-65-1.1.1.pdf	This document in PDF format
example\tip670exa.c	example application C source file
ChangeLog.txt	Release history
Release.txt	Release information

2.1 Software Installation

The TIP670 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 TIP670 Device Driver on a Windows 2000/XP operating system.

After installing the TIP670 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 TIP670 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 driver media.
   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 (tip670.h) to a desired target directories.

After successful installation the TIP670 device driver will start immediately and create devices (TIP670_1, TIP670_2, ...) for all recognized TIP670 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 "***TIP670***" should appear.

2.2 Driver Configuration

2.2.1 Event Queue Configuration

After Installation of the TIP670 Device Driver the number concurrent event controlled read request is limited to 5.

If the default values are not suitable the configuration can be changed by modifying the registry, for instance with regedt32 or regedit.

To change the number of queue entries the following value must be modified.

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\tip670\NumJobEntries
```

Valid values are in range between 1 and 100

3 TIP670 Device Driver Programming

The TIP670-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.) I/O Functions

3.1 TIP670 Files and I/O Functions

The following section doesn't contain a full description of the Win32 functions for interaction with the TIP670 device driver. Only the required parameters are described in detail.

3.1.1 Opening a TIP670 Device

Before you can perform any I/O the *TIP670* device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the *TIP670* 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 TIP670 to open.
The *lpFileName* string should be of the form **\\.\TIP670_*x*** to open the device *x.* The ending x is a one-based number. The first device found by the driver is \\.\TIP670_1, the second \\.\TIP670_2 and so on.

dwDesiredAccess

Specifies the type of access to the TIP670. For the TIP670 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 TIP670, set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for TIP670 devices.

dwCreationDistribution

Specifies which action to take on files that exist and which action to take when files that do not exist. TIP670 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 TIP670 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TIP670 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call **GetLastError**.

Example

```
HANDLE    hDevice;

hDevice = CreateFile(
     “\\\\.\\TIP670_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // TIP670 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 TIP670 Device

The **CloseHandle** function closes an open TIP670 handle.

```
BOOL CloseHandle(
    HANDLE hDevice;                    // handle to a TIP670 device to close
)
```

Parameters

hDevice

Identifies an open TIP670 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()

3.1.3 TIP670 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 TIP670 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 tip670.h:

Value	**Meaning**
IOCTL_TIP670_READ	Read the value of the input port
IOCTL_TIP670_WRITE	Write a new value to the output port
IOCTL_TIP670_READ_INP_POL	Read the configured input polarity
IOCTL_TIP670_WRITE_INP_POL	Configure the input polarity
IOCTL_TIP670_READ_OUTP_POL	Read the configured output polarity
IOCTL_TIP670_WRITE_OUTP_POL	Configure the output polarity
IOCTL_TIP670_EVENT_READ	Read the value of the ports on a specified event
IOCTL_TIP670_ENABLE_WD	Setup up and enable output watchdog
IOCTL_TIP670_DISABLE_WD	Disable output watchdog
IOCTL_TIP670_TRIGGER_WD	Trigger output watchdog

See behind for more detailed information on each control code.

To use these TIP670 specific control codes the header file tip670.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_TIP670_READ

The read function reads the content of the input port immediately.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the read buffer (UCHAR) to the device driver.

The value of the input lines will be copied into the specified buffer.

Example

```
#include “tip670.h”

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

/*
**  Read ports immediately
*/
success = DeviceIoControl (
     hDevice,                // TIP670 handle
     IOCTL_TIP670_READ,
     NULL,
     0,
     &RWBuf,                 // contains the read data
     sizeof(UCHAR),
     &NumBytes,              // size of returned Buffer
     0
);
if( success ) {
     printf(“Input Port = 0x%x\n”, RWBuf);
}
else {
     ErrorHandler ( "Device I/O control error” );   // process error
}
```

Error Codes

All returned error codes are system error conditions.

3.1.3.2 IOCTL_TIP670_WRITE

The Write function writes a value to the output port.

The parameter *lpInBuffer* must pass a pointer to the write buffer (*UCHAR*) 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 "tip670.h"

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

/*
**  Write a new value (0x12) to the output Port
*/
RWBuf = 0x12;

success = DeviceIoControl (
     hDevice,                  // TIP670 handle
     IOCTL_TIP670_WRITE,
     &RWBuf,                   // parameter for the driver
     sizeof(UCHAR),
     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
}
```

Error Codes

All returned error codes are system error conditions.

3.1.3.3 IOCTL_TIP670_READ_INP_POL

The input polarity read function reads the software polarity configuration of the input port.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the read buffer (*UCHAR*) to the device driver.

The polarity of the input lines will be returned in the buffer. If a bit is set for the corresponding input pin the signal will be inverted.

Example

```
#include “tip670.h”

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

/*
**  Read input polarity
*/
success = DeviceIoControl (
     hDevice,                      // TIP670 handle
     IOCTL_TIP670_READ_INP_POL,
     NULL,                         // parameter for the driver
     0,
     &RWBuf,                       // contains the read data
     sizeof(UCHAR),
     &NumBytes,                    // size of returned Buffer
     0
);
if( success ) {
     printf(“Input port polarity = 0x%x\n”, RWBuf);
}
else {
     ErrorHandler ( "Device I/O control error” ); // process error
}
```

Error Codes

All returned error codes are system error conditions.

3.1.3.4 IOCTL_TIP670_WRITE_INP_POL

The input polarity write function configures the polarity of the input port.

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

The polarity of the input lines will be configured to the value stored in the buffer. If a bit is set the signal of the corresponding input pin will be inverted.

Example

```
#include “tip670.h”

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

/*
**  Bit 0 and 1 shall invert the input
*/
RWBuf = 0x03;      /* Bits 0..1 – Inverted */
                   /* Bits 2..7 - Not Inverted */

success = DeviceIoControl (
     hDevice,                    // TIP670 handle
     IOCTL_TIP670_WRITE_INP_POL,
     &RWBuf,                     // parameter for the driver
     sizeof(UCHAR),
     NULL,                       // no data returned
     0,
     &NumBytes,                  // size of returned Buffer
     0
);
if( success ) {
     printf( “Polarity change successfully completed\n” );
}
else {
     ErrorHandler ( "Device I/O control error” );   // process error
}
```

Error Codes

All returned error codes are system error conditions.

3.1.3.5 IOCTL_TIP670_READ_OUTP_POL

The output polarity read function reads the software polarity configuration of the output port.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the read buffer (unsigned char) to the device driver.

The polarity of the output lines will be returned in the buffer. If a bit is set for the corresponding output pin the signal will be inverted.

Example

```
#include “tip670.h”

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

/*
**  Read output polarity
*/
success = DeviceIoControl (
     hDevice,                    // TIP670 handle
     IOCTL_TIP670_READ_OUTP_POL,
     NULL,                       // parameter for the driver
     0,
     &RWBuf,                     // contains the read data
     sizeof(UCHAR),
     &NumBytes,                  // size of returned Buffer
     0
);
if( success ) {
     printf(“Output port polarity = 0x%x\n”, RWBuf);
}
else {
     ErrorHandler ( "Device I/O control error” );   // process error
}
```

Error Codes

All returned error codes are system error conditions.

3.1.3.6 IOCTL_TIP670_WRITE_OUTP_POL

The output polarity write function configures the polarity of the output port.

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

The polarity of the output lines will be configured to the value stored in the buffer. If a bit is set the signal of the corresponding output pin will be inverted.

Example

```
#include “tip670.h”

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

/*
**  Bit 0 and 1 shall invert the output
*/
RWBuf = 0x03;      /* Bits 0..1 – Inverted */
                   /* Bits 2..7 - Not Inverted */

success = DeviceIoControl (
     hDevice,                      // TIP670 handle
     IOCTL_TIP670_WRITE_OUTP_POL,
     &RWBuf,                       // parameter for the driver
     sizeof(UCHAR),
     NULL,                         // no data returned
     0,
     &NumBytes,                    // size of returned Buffer
     0
);
if( success ) {
     printf( “Polarity change successfully completed\n” );
}
else {
     ErrorHandler ( "Device I/O control error” );   // process error
}
```

Error Codes

All returned error codes are system error conditions.

3.1.3.7 IOCTL_TIP670_EVENT_READ

The event read function reads the contents of the input port after a specified event has occurred on the input port. Possible events are rising or falling edge or both, at a specified input bit or a pattern match of masked input bits.

Both parameter *lpInBuffer* and *lpOutBuffer* must pass a pointer to the event read buffer (*TIP670_EVRD_BUFFER*) to the device driver.

```
typedef struct {
        UCHAR           port;
        UCHAR           mask;
        UCHAR           match;
        USHORT          mode;
        LONG            timeout;
} TIP670_EVRD_BUFFER, *PTIP670_EVRD_BUFFER;
```

TIP670_MATCH events may get lost, because data evaluation is delayed to the event detection, and the input value may have changed again and the match con not be recognized any more.

The same delay may let the function return event values that are not the values at event time.

port

Receives the content of the input port.

mask

Specifies a bit mask for the input port. A 1 value marks the corresponding bit position as relevant.

match

Specifies a pattern for the input port that must match to the content input port. Only the bit positions specified by *mask* are compared to the input port.

mode

Specifies the “event” mode for this read request

TIP670_MATCH	The driver reads the input port content, if the masked input bits match to the specified pattern. The input mask must be specified in the parameter mask. A ‘1’ value in mask means that the input bit value “must-match” identically to the corresponding bit in the match parameter.
TIP670_HIGH_TR	The driver reads the input port content, if a high-transition at the specified bit position occurs. A ‘1’ value in mask specifies the bit position of the port. If you specify more than one bit position the events are OR’ed. That means the event read will completed if a high-transition at least at one relevant bit position occurs.
TIP670_LOW_TR	The driver reads the input port content, if a low-transition at the specified bit position occurs. A ‘1’ value in mask specifies the bit position of the port. If you specify more than one bit position the events are OR’ed. That means the event read will completed if a low-transition at least at one relevant bit position occurs.
TIP670_ANY_TR	The driver reads the input port content, if a transition (high or low) at the specified bit position occurs. A ‘1’ value in mask specifies the bit position of the port. If you specify more than one bit position the events are OR’ed. That means the event read will completed if a transition at least at one relevant bit position occurs.

timeout

Specifies the amount of time (in seconds) the caller is willing to wait for the specified event to occur. A value of 0 means wait indefinitely.

Example

```
#include “tip670.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP670_EVRD_BUFFER EvRdBuf;

…
```

```
…

/*
** Read the input port if ..
**   bit 0 = 0
**   bit 1 = 1
**   bit 6 = 0
**   bit 7 = 1
*/
EvRdBuf.mode = T670_MATCH;
EvRdBuf.mask = 0xC3;        // bit 0,1,6,7 are relevant
EvRdBuf.match = 0x82;
EvRdBuf.timeout = 10;       // 10 seconds

success = DeviceIoControl (
     hDevice,                         // TIP670 handle
     IOCTL_TIP670_EVENT_READ,
     &EvRdBuf,                        // parameter for the driver
     sizeof(TIP670_EVRD_BUFFER),
     &EvRdBuf,                        // contains the read data
     sizeof(TIP670_EVRD_BUFFER),
     &NumBytes,                       // size of returned Buffer
     0
);
if( success ) {
     printf(“Input port = 0x%x\n”, EvRdBuf.port);
}
else {
     ErrorHandler ( "Device I/O control error” );    // process error
}

/*
**  Read the input port after a high-transition at
**  bit 7 occurred
*/
ReadBuf.mode = T670_HIGH_TR;
ReadBuf.mask = 1<<7;             // high-transition at bit 7
ReadBuf.timeout = 15;            // 15 seconds

…
```

```
…

success = DeviceIoControl (
     hDevice,                    // TIP670 handle
     IOCTL_T670_EVENT_READ,
     &EvRdBuf,                   // parameter for the driver
     sizeof(T670_EVRD_BUFFER),
     &EvRdBuf,                   // contains the read data
     sizeof(T670_EVRD_BUFFER),
     &NumBytes,                  // size of returned Buffer
     0
);
if( success ) {
     printf(“Input port = 0x%x\n”, EvRdBuf.port);
}
else {
     ErrorHandler ( "Device I/O control error” );   // process error
}
```

Error Codes

ERROR_INVALID_PARAMETER	This error is returned if the size of the read buffer is too small or if the parameter mode contains an invalid value.
ERROR_NO_SYSTEM_RESOURCES	No more free entries in the drivers queue to handle concurrent event-controlled read requests. Increase the queue size.
ERROR_SEM_TIMEOUT	The requested event has not occurred within the specified time (timeout).

All other returned error codes are system error conditions.

3.1.3.8 IOCTL_TIP670_ENABLE_WD

This function configures and enables the output watchdog.

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

The watchdog time is specified in 0.5µs steps in the buffer.

Example

```
#include “tip670.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
unsigned short WDBuf;

/*
**  Enable watchdog with a time of 1ms
*/
WDBuf = 2000;
success = DeviceIoControl (
     hDevice,                    // TIP670 handle
     IOCTL_TIP670_ENABLE_WD,
     &WDBuf,                     // parameter for the driver
     sizeof(unsigned short),
     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
}
```

Error Codes

All returned error codes are system error conditions.

3.1.3.9 IOCTL_TIP670_DISABLE_WD

This function disables the output watchdog.

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

Example

```
#include “tip670.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;

/*
**  Disable watchdog
*/
success = DeviceIoControl (
     hDevice,                     // TIP670 handle
     IOCTL_TIP670_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
}
```

Error Codes

All returned error codes are system error conditions.

3.1.3.10 IOCTL_TIP670_TRIGGER_WD

This function triggers the output watchdog. This let start the watchdog time to run again.

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

Example

```
#include “tip670.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;

/*
**  Trigger watchdog
*/
success = DeviceIoControl (
     hDevice,                    // TIP670 handle
     IOCTL_TIP670_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
}
```

Error Codes

All returned error codes are system error conditions.