The Embedded I/O Company

TEWS TECHNOLOGIES

TIP116-SW-65

Windows 2000/XP Device Driver

4 Channel Quadrature / General Purpose Counter

Version 1.0.x

User Manual

Issue 1.0.0

January 2006

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
25469 Halstenbek / Germany
Phone: +49-(0)4101-4058-0
Fax: +49-(0)4101-4058-19
e-mail: info@tews.com
www.tews.com

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Reno, Nevada 89504 / USA
Phone: +1 (775) 686 6077
Fax: +1 (775) 686 6024
e-mail: usasales@tews.com
www.tews.com

TIP116-SW-65

4 Channel Quadrature / General Purpose Counter

Windows 2000/XP Device Driver

Issue	Description	Date
1.0.0	First Issue	January 29, 2006

Table of Contents

1 Introduction

The TIP116-SW-65 Windows WDM (Windows Driver Model) device driver is a kernel mode driver which allows the operation of the TIP116 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 TIP116 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 TIP116 device driver includes the following functions:

- read counter value and state
- configure counter channels (counting mode, Z-Mode, …)
- setup counter preload and compare values
- latch all 4 counter on one module
- wait for compare match and/or “latch on Z” event
- read timer counter
- setup timer
- wait for timer event
- configure general purpose I/O pin
- set general purpose I/O pin
- read state of general purpose I/O pin

The TIP116-SW-65 supports the modules listed below:

TIP116	4 Channel Quadrature / General Purpose Counter	IPAC

To get more information about the features and use of TIP116 devices it is recommended to read the manuals listed below.

TIP116 User manual

TIP116 Engineering Manual

CARRIER-SW-65 IPAC Carrier User Manual

2 Installation

Following files are located on the distribution media:

tip116.sys	Device driver binary
tip116.h	Header file with IOCTL code definitions and driver specific data types
tip116.inf	Installation script
TIP116-SW-65-1.0.0.pdf	This document in PDF format
\example\tip116exa.c	example application C source file
Release.txt	Release information

2.1 Software Installation

The TIP116 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 TIP116 Device Driver on a Windows 2000/XP operating system.

After installing the TIP116 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 TIP116 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 (tip116.h, ...) to the desired target directories.

After successful installation the TIP116 device driver will start immediately and create devices (tip116_1, tip116_2, ...) for all recognized TIP116 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 TIP116 4 Quad./General Purpose Counter***" should appear.

3 Driver Configuration

3.1 Event Queue Configuration

After Installation of the TIP116 Device Driver, the number of concurrent event wait requests is limited to 5.

If the default value is 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\tip116\NumJobEntries

Valid values are in range between 1 ... 100

4 TIP116 Device Driver Programming

The TIP116-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.)

4.1 TIP116 Files and I/O Functions

The following section doesn't contain a full description of the Win32 functions for interaction with the TIP116 device driver. Only the required parameters are described in detail.

4.1.1 Opening a TIP116 Device

Before you can perform any I/O the *TIP116* device must be opened by invoking the **CreateFile** function. **CreateFile** returns a handle that can be used to access the *TIP116* 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 TIP116 to open.
The *lpFileName* string should be of the form **\\.\tip116_*x*** to open the device *x*. The ending x is a one-based number. The first device found by the driver is \\.\tip116_1, the second \\.\tip116_2 and so on.

dwDesiredAccess

Specifies the type of access to the TIP116. For the TIP116 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 TIP116, set to 0.

lpSecurityAttributes

Pointer to a security structure. Set to NULL for TIP116 devices.

dwCreationDistribution

Specifies which action to take on files that exist and which action to take when files that do not exist. TIP116 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 TIP116 devices.

Return Value

If the function succeeds, the return value is an open handle to the specified TIP116 device. If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call **GetLastError**.

Example

```
HANDLE    hDevice;

hDevice = CreateFile(
     “\\\\.\\tip116_1”,
     GENERIC_READ | GENERIC_WRITE,
     0,
     NULL,              // no security attrs
     OPEN_EXISTING,     // TIP116 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 TIP116 Device

The **CloseHandle** function closes an open TIP116 handle.

```
BOOL CloseHandle(
      HANDLE hDevice;                          // handle to a TIP116 device to close
);
```

Parameters

hDevice

Identifies an already opened TIP116 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()

4.1.3 TIP116 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 TIP116 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 *tip116.h*:

Value	Meaning
IOCTL_TIP116_CNT_READ	Read counter value
IOCTL_TIP116_CNT_SET_PRELOAD	Load a new preload value
IOCTL_TIP116_CNT_SET_COMPARE	Load a new compare value
IOCTL_TIP116_CNT_CONFIGURE	Configure Counter
IOCTL_TIP116_CNT_RESET	Reset Counter
IOCTL_TIP116_CNT_LATCH	Latch all Counters
IOCTL_TIP116_CNT_READALL	Read all Counters
IOCTL_TIP116_CNT_EVENTWAIT	Wait for Counter Event
IOCTL_TIP116_CNT_GETINPUTSTATE	Get State of the Input Lines
IOCTL_TIP116_TIMER_READ	Read Timer Value
IOCTL_TIP116_TIMER_CONFIGURE	Configure Timer
IOCTL_TIP116_TIMER_EVENTWAIT	Wait for Timer Event
IOCTL_TIP116_GPIO_READ	Read value of GPIO pins (input)
IOCTL_TIP116_GPIO_SET	Set value of GPIO pins (output)
IOCTL_TIP116_GPIO_CONFIGURE	Configure GPIO pins

See behind for more detailed information on each control code.

To use these TIP116 specific control codes, the header file tip116.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_TIP116_CNT_READ

This function returns the current counter value and state of a specified counter channel to the caller. A pointer to the counter read buffer (*TIP116_CNT_READ_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 *TIP116_CNT_READ_BUF* structure has the following layout:

```
typedef struct
{
    unsigned long       channel;
    unsigned long       count;
    unsigned long       status;
} TIP116_CNT_READ_BUF;
```

channel

This parameter specifies the counter channel that shall be read. Valid channel numbers are 1…4.

count

This parameter returns the counter value.

status

This parameter returns a set of flags that display the state of the counter and the returned value. The following state flags (defined in 'tip116.h') can be returned:

flag	description
TIP116_FL_STATBORROW	The BORROW flag is set, a 0 → -1 transition has occurred
TIP116_FL_STATCARRY	The CARRY flag is set, a -1 → 0 transition has occurred
TIP116_FL_STATMATCH	The MATCH flag is set, the counter value of the counter matches or has passed the programmed compare value
TIP116_FL_STATSIGN	The SIGN flag is set, this display that an underflow has occurred last, if it not set an overflow has occurred last.
TIP116_FL_STATDIRECTION	This flag display the last count direction, set indicates a count forward and a reset indicates a backward count
TIP116_FL_STATLATCHED	The LATCHED flag indicates that the returned value has been latch before the counter is read, in this case the value can not be actual

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_CNT_READ_BUF cntBuf;

/* Read counter value and state of counter channel 3 */
cntBuf.channel = 3;

success = DeviceIoControl (
     hDevice,                          // TIP116 handle
     IOCTL_TIP116_CNT_READ,
     &cntBuf,                          // input buffer
     sizeof(TIP116_CNT_READ_BUF),      // size of input buffer
     &cntBuf,                          // output buffer
     sizeof(TIP116_CNT_READ_BUF),      // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     printf(“Value: %08lXh – State: %08lXh\n”,
                    cntBuf.count,
                    cntBuf.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()

4.1.3.2 IOCTL_TIP116_CNT_SET_PRELOAD

This function loads a new preload value for a specified counter channel. A pointer to the counter preload buffer (*TIP116_CNT_PRLD_BUF*) must be passed by the argument *lpInBuffer* to the driver. The argument *nInBufferSize* specifies the size of this buffer.

The *TIP116_CNT_PRLD_BUF* structure has the following layout:

```
typedef struct
{
    unsigned long       channel;
    unsigned long       value;
    unsigned long       flags;
} TIP116_CNT_PRLD_BUF;
```

channel

This parameter specifies the counter channel the preload shall be written to. Valid channel numbers are 1…4

value

This parameter specifies the new counter preload value.

flags

The flags parameter specifies how the preload shall be executed. The following flags (defined in 'tip116.h') can be returned:

flag	description
TIP116_FL_IMMPLRD	If this flag is set the preload value will be loaded to the counter immediately. If it is not specified the preload value will be used with the next preload initiated by the counter.

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_CNT_PRLD_BUF prldBuf;

/* Set preload value for counter 2, and load immediately */
cntBuf.channel = 2;
cntBuf.value = 100000;
cntBuf.flags = TIP116_FL_IMMPLRD;

success = DeviceIoControl (
     hDevice,                          // TIP116 handle
     IOCTL_TIP116_CNT_SET_PRELOAD,
     &prldBuf,                         // input buffer
     sizeof(TIP116_CNT_PRLD_BUF),      // size of input buffer
     NULL,                             // output buffer
     0,                                // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     /* OK */
} else {
     ErrorHandler ("Device I/O control error”);  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The input buffer is too small. Please check the parameter *nInBufferSize* of the DeviceIoControl() function call
ERROR_MEMBER_NOT_IN_GROUP	The specified channel number is out of range

See Also

Win32 documentation DeviceIoControl()

4.1.3.3 IOCTL_TIP116_CNT_SET_COMPARE

This function loads a new compare value for a specified counter channel. A pointer to the counter compare set buffer (*TIP116_CNT_COMP_BUF*) must be passed by the argument *lpInBuffer* to the driver. The argument *nInBufferSize* specifies the size of this buffer.

The *TIP116_CNT_COMP_BUF* structure has the following layout:

```
typedef struct
{
    unsigned long       channel;
    unsigned long       value;
} TIP116_CNT_COMP_BUF;
```

channel

This parameter specifies the counter channel the compare value shall be set. Valid channel numbers are 1…4

value

This parameter specifies the new counter compare value.

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_CNT_COMP_BUF compBuf;

/* Set compare value for counter 2 */
compBuf.channel = 2;
compBuf.value = 100000;

success = DeviceIoControl (
     hDevice,                        // TIP116 handle
     IOCTL_TIP116_CNT_SET_COMPARE,
     &compBuf,                       // input buffer
     sizeof(TIP116_CNT_COMP_BUF),    // size of input buffer
     NULL,                           // output buffer
     0,                              // size of output buffer
     &NumBytes,                      // number of returned bytes
     0);
if (success) {
     /* OK */
} else {
     ErrorHandler ("Device I/O control error”);  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The input buffer is too small. Please check the parameter *nInBufferSize* of the DeviceIoControl() function call
ERROR_MEMBER_NOT_IN_GROUP	The specified channel number is out of range

See Also

Win32 documentation DeviceIoControl()

4.1.3.4 IOCTL_TIP116_CNT_CONFIGURE

This function configures a selected counter channel. Input signals and its polarity will be chosen. Other special modes and signals can be setup. A pointer to the counter configure buffer (*TIP116_CNT_CFGSET_BUF*) must be passed by the argument *lpInBuffer* to the driver. The argument *nInBufferSize* specifies the size of this buffer.

The *TIP116_CNT_CFGSET_BUF* structure has the following layout:

```
typedef struct
{
    unsigned long       channel;
    unsigned short      inputMode;
    unsigned short      specMode;
    unsigned short      ZControl;
    unsigned short      QuadControl;
    unsigned short      XPolarity;
    unsigned short      YPolarity;
    unsigned short      ZPolarity;
} TIP116_CNT_CFGSET_BUF;
```

channel

This parameter specifies the counter channel the compare value shall be configured. Valid channel numbers are 1…4

inputMode

This parameter specifies the input mode. The following input modes are defined ('tip116.h'):

input mode	description
TIP116_MD_COUNTOFF	This value disables the specified counter channel
TIP116_MD_QUADCOUNT	This value selects the quadrature counter mode
TIP116_MD_UPDOWNCOUNT	This value selects the up/down counter function
TIP116_MD_DIRECTCOUNT	This mode selects the direction counter function

specMode

This parameter specifies the special count mode. The following special count modes are defined ('tip116.h'):

special count mode	description
TIP116_MD_SPECMODENO	This value selects the 'no special count' mode
TIP116_MD_SPECMODEDIVN	This value selects the 'divide-by-n' mode
TIP116_MD_SPECMODESNGLCYC	This value selects the 'single cycle' mode

ZControl

This parameter specifies the mode of the Z input signal. The following modes are defined ('tip116.h'):

Z mode	description
TIP116_MD_ZNO	The Z signal will not be used
TIP116_MD_ZLOAD	The Z signal initiates a counter load
TIP116_MD_ZLATCH	The Z signal latches the counter value
TIP116_MD_ZGATE	The Z signal gates the counter input
TIP116_MD_ZRESET	The Z signal resets the counter value

QuadControl

This parameter specifies the quadrature mode of the selected channel. This value is only used if *TIP116_MD_QUADCOUNT* is select for *inputMode*. The following modes are defined ('tip116.h'):

Quadrature mode	description
TIP116_MD_1X	Select 1x counting
TIP116_MD_2X	Select 2x counting
TIP116_MD_4X	Select 4x counting

XPolarity

This parameter specifies the input polarity of the X line of the selected channel. The following modes are defined ('tip116.h'):

X polarity	description
TIP116_MD_XHIGHACTIV	Selects high active X signal input
TIP116_MD_XLOWACTIV	Selects low active X signal input

YPolarity

This parameter specifies the input polarity of the Y line of the selected channel. The following modes are defined ('tip116.h'):

Y polarity	description
TIP116_MD_YHIGHACTIV	Selects high active X signal input
TIP116_MD_YLOWACTIV	Selects low active X signal input

ZPolarity

This parameter specifies the input polarity of the Z line of the selected channel. The following modes are defined ('tip116.h'):

Z polarity	description
TIP116_MD_ZHIGHACTIV	Selects high active X signal input
TIP116_MD_ZLOWACTIV	Selects low active X signal input

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_CNT_CFGSET_BUF cfgBuf;

/* Setup counter 2 for 4x quadrature counting */
/* Z gates with a low active signal */
cfgBuf.channel = 2;
cfgBuf.inputMode = TIP116_MD_QUADCOUNT;
cfgBuf.specMode = TIP116_MD_SPECMODENO;
cfgBuf.ZControl = TIP116_MD_ZGATE;
cfgBuf.QuadControl = TIP116_MD_4X;
cfgBuf.XPolarity = TIP116_MD_XHIGHACTIV;
cfgBuf.YPolarity = TIP116_MD_YHIGHACTIV;
cfgBuf.ZPolarity = TIP116_MD_ZLOWACTIV;

success = DeviceIoControl (
     hDevice,                          // TIP116 handle
     IOCTL_TIP116_CNT_CONFIGURE,
     &cfgBuf,                          // input buffer
     sizeof(TIP116_CNT_CFGSET_BUF),    // size of input buffer
     NULL,                             // output buffer
     0,                                // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     /* OK */
} else {
     ErrorHandler ("Device I/O control error”);  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The input buffer is too small. Please check the parameter *nInBufferSize* of the DeviceIoControl() function call
ERROR_MEMBER_NOT_IN_GROUP	The specified channel number is out of range
ERROR_INVALID_PARAMETER	A parameter has been specified with an invalid value

See Also

Win32 documentation DeviceIoControl()

4.1.3.5 IOCTL_TIP116_CNT_RESET

This function resets the value of a specified channel to zero. A pointer to the counter reset buffer (*TIP116_CNT_RESET_BUF*) must be passed by the argument *lpInBuffer* to the driver. The argument *nInBufferSize* specifies the size of this buffer.

The *TIP116_CNT_RESET_BUF* structure has the following layout:

```
typedef struct
{
    unsigned long       channel;
} TIP116_CNT_RESET_BUF;
```

channel

This parameter specifies the counter channel that shall reset. Valid channel numbers are 1…4

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_CNT_RESET_BUF resetBuf;

/* Reset counter 2 */
resetBuf.channel = 2;

success = DeviceIoControl (
    hDevice,                        // TIP116 handle
    IOCTL_TIP116_CNT_RESET,
    &resetBuf,                      // input buffer
    sizeof(TIP116_CNT_RESET_BUF),   // size of input buffer
    NULL,                           // output buffer
    0,                              // size of output buffer
    &NumBytes,                      // number of returned bytes
    0);
if (success) {
    /* OK */
} else {
    ErrorHandler ("Device I/O control error”);  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The input buffer is too small. Please check the parameter *nInBufferSize* of the DeviceIoControl() function call
ERROR_MEMBER_NOT_IN_GROUP	The specified channel number is out of range

See Also

Win32 documentation DeviceIoControl()

4.1.3.6 IOCTL_TIP116_CNT_LATCH

This function latches the values of all channels of the module. There is no input or output parameter needed for this function.

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;

/* Latch all counters */
success = DeviceIoControl (
     hDevice,                          // TIP116 handle
     IOCTL_TIP116_CNT_LATCH,
     NULL,                             // input buffer
     0,                                // size of input buffer
     NULL,                             // output buffer
     0,                                // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     /* OK */
} else {
     ErrorHandler ("Device I/O control error”);  // process error
}
```

See Also

Win32 documentation DeviceIoControl()

4.1.3.7 IOCTL_TIP116_CNT_READALL

This function returns a buffer with the current counter values and states of all counter channels of a module. A pointer to the read all counter buffer (*TIP116_CNT_READALL_BUF*) must be passed by the argument *lpOutBuffer* to the driver. The argument *nOutBufferSize* specifies the size of this buffer.

The *TIP116_CNT_READ_BUF* structure has the following layout:

```
typedef struct
{
    TIP116_CNT_READ_BUF    channel[4];
} TIP116_CNT_READALL_BUF;
```

channel[]

This array contains read data from all 4 channels of a module. The array index 0 specifies channel 1, index 1 selects channel 2 and so on.

The *TIP116_CNT_READ_BUF* structure has the following layout:

```
typedef struct
{
    unsigned long        channel;
    unsigned long        count;
    unsigned long        status;
} TIP116_CNT_READ_BUF;
```

channel

This parameter specifies the counter channel number the data has been read from.

count

This parameter returns the counter value.

status

This parameter returns a set of flags that display the state of the counter and the returned value. The following state flags (defined in 'tip116.h') can be returned:

flag	description
TIP116_FL_STATBORROW	The BORROW flag is set, a 0 → -1 transition has occurred
TIP116_FL_STATCARRY	The CARRY flag is set, a -1 → 0 transition has occurred
TIP116_FL_STATMATCH	The MATCH flag is set, the counter value of the counter matches or has passed the programmed compare value
TIP116_FL_STATSIGN	The SIGN flag is set, this display that an underflow has occurred last, if it not set an overflow has occurred last.
TIP116_FL_STATDIRECTION	This flag display the last count direction, set indicates a count forward and a reset indicates a backward count
TIP116_FL_STATLATCHED	The LATCHED flag indicates that the returned value has been latch before the counter is read, in this case the value can not be actual

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_CNT_READ_BUF rdAllBuf;
int i;

/* Read counter values and states of all counter channels */

success = DeviceIoControl (
     hDevice,                          // TIP116 handle
     IOCTL_TIP116_CNT_READALL,
     NULL,                             // input buffer
     0,                                // size of input buffer
     &rdAllBuf,                        // output buffer
     sizeof(TIP116_CNT_READALL_BUF), // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     for (i = 0; i < 4; i++)
     {
          printf(“Channel %d, Value: %08lXh – State: %08lXh\n”,
                    rdAllBuf.channel[i].channel,
                    rdAllBuf.channel[i].count,
                    rdAllBuf.channel[i].status);
     }
} else {
     ErrorHandler ("Device I/O control error”);  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The output buffer is too small. Please check the parameter *nOutBufferSize* of the DeviceIoControl() function call

See Also

Win32 documentation DeviceIoControl()

4.1.3.8 IOCTL_TIP116_CNT_EVENTWAIT

This function waits for a specified counter event on a specified counter channel. A pointer to the counter wait buffer (*TIP116_CNT_WAIT_BUF*) must be passed by the argument *lpInBuffer* to the driver. The argument *nInBufferSize* specifies the size of this buffer.

The *TIP116_CNT_WAIT_BUF* structure has the following layout:

```
typedef struct
{
    unsigned long       channel;
    unsigned long       eventType;
    int                 timeout;
} TIP116_CNT_WAIT_BUF;
```

channel

This parameter specifies the counter channel the event is expected. Valid channel numbers are 1…4

eventType

This parameter specifies the event the function shall wait for. The following events are defined ('tip116.h'):

Event type	description
TIP116_EV_MATCH	Wait for match event. This will occur if the counter value is equal to the specified compare value
TIP116_EV_LATCH	Wait for a latch event. This event will occur if the counter is 'latched by Z'.

timeout

This parameter specifies the maximum time the function shall wait before it returns even if the specified event has not occurred. The time is specified in seconds.

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_CNT_WAIT_BUF waitBuf;

/* Wait for a match event on channel 2, timeout after a minute */
waitBuf.channel = 2;
waitBuf.eventType = TIP116_EV_MATCH;
waitBuf.timeout = 60;

success = DeviceIoControl (
     hDevice,                          // TIP116 handle
     IOCTL_TIP116_CNT_EVENTWAIT,
     &waitBuf,                         // input buffer
     sizeof(TIP116_CNT_WAIT_BUF),      // size of input buffer
     NULL,                             // output buffer
     0,                                // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     /* OK */
} else {
     ErrorHandler ("Device I/O control error”);  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The input buffer is too small. Please check the parameter *nInBufferSize* of the DeviceIoControl() function call
ERROR_MEMBER_NOT_IN_GROUP	The specified channel number is out of range
ERROR_INVALID_PARAMETER	An invalid event has been specified
ERROR_SEM_TIMEOUT	The timeout time has passed before the specified event has occurred

See Also

Win32 documentation DeviceIoControl()

4.1.3.9 IOCTL_TIP116_CNT_GETINPUTSTATE

This function returns the actual state of the counter input lines of the module. A pointer to the read input state buffer (*TIP116_CNT_GETINPUTSTATE_BUF*) must be passed by the argument *lpOutBuffer* to the driver. The argument *nOutBufferSize* specifies the size of this buffer.

The *TIP116_CNT_GETINPUTSTATE_BUF* structure has the following layout:

```
typedef struct
{
    unsigned short      value;
} TIP116_CNT_GETINPUTSTATE_BUF;
```

value

This argument returns the actual state of the input lines. The value is identical to the contents of the '*Global Input Status Register*' of the TIP116. For bit definitions please refer to the TIP116 User Manual.

Example

```
#include "tip116.h"

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_CNT_GETINPUTSTATE_BUF rdStatBuf;

/* Read state of the input lines */
success = DeviceIoControl (
     hDevice,                          // TIP116 handle
     IOCTL_TIP116_CNT_GETINPUTSTATE,
     NULL,                             // input buffer
     0,                                // size of input buffer
     &rdStatBuf,                       // output buffer
     sizeof(TIP116_CNT_GETINPUTSTATE_BUF), // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     printf("Input state: %04Xh\n",
               rdStatBuf.value);
} else {
     ErrorHandler ("Device I/O control error");  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The output buffer is too small. Please check the parameter *nOutBufferSize* of the DeviceIoControl() function call

See Also

Win32 documentation DeviceIoControl()

4.1.3.10 IOCTL_TIP116_TIMER_READ

This function returns the actual value of the timer counter. A pointer to the read timer buffer (*TIP116_TIMER_READ_BUF*) must be passed by the argument *lpOutBuffer* to the driver. The argument *nOutBufferSize* specifies the size of this buffer.

The *TIP116_TIMER_READ_BUF* structure has the following layout:

```
typedef struct
{
    unsigned short      value;
} TIP116_TIMER_READ_BUF;
```

value

This argument returns the actual value of the timer counter register.

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_TIMER_READ_BUF rdTimeBuf;

/* Read state of the input lines */
success = DeviceIoControl (
     hDevice,                          // TIP116 handle
     IOCTL_TIP116_TIMER_READ,
     NULL,                             // input buffer
     0,                                // size of input buffer
     &rdTimeBuf,                       // output buffer
     sizeof(TIP116_TIMER_READ_BUF),    // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     printf(“Timer count: %04Xh\n”, rdTimeBuf.value);
} else {
     ErrorHandler ("Device I/O control error”);  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The output buffer is too small. Please check the parameter *nOutBufferSize* of the DeviceIoControl() function call

See Also

Win32 documentation DeviceIoControl()

4.1.3.11 IOCTL_TIP116_TIMER_CONFIGURE

This function configures the timer. A pointer to the timer configure buffer (*TIP116_TIMER_CONFIG_BUF*) must be passed by the argument *lpInBuffer* to the driver. The argument *nInBufferSize* specifies the size of this buffer.

The *TIP116_TIMER_CONFIG_BUF* structure has the following layout:

```
typedef struct
{
    unsigned short      value;
    unsigned char       clockDivider;
    unsigned char       enable;
} TIP116_TIMER_CONFIG_BUF;
```

value

This parameter specifies the new preload value of the timer, setting up the cycle timer.

clockDivider

This parameter specifies the speed the timer clocked with. The following clock speeds are defined ('tip116.h'):

clock divider	description
TIP116_T_MD_8MHZ	This clock divider will be set to 1, the clock rate will be 8 MHz
TIP116_T_MD_4MHZ	This clock divider will be set to 2, the clock rate will be 4 MHz
TIP116_T_MD_2MHZ	This clock divider will be set to 4, the clock rate will be 2 MHz
TIP116_T_MD_1MHZ	This clock divider will be set to 8, the clock rate will be 1 MHz

enable

This parameter specifies if the timer shall be enabled or disabled. Use the following defines ('tip116.h'):

mode	description
TIP116_T_MD_DISABLE	This value disabled timer
TIP116_T_MD_ENABLE	This value enables the timer

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_TIMER_CONFIG_BUF cfgBuf;

/* Enable timer with 2 MHz, and preload value of 10000 */
cfgBuf.value = 10000;
cfgBuf.clockDivider = TIP116_T_MD_2MHZ;
cfgBuf.enable = TIP116_T_MD_ENABLE;

success = DeviceIoControl (
     hDevice,                          // TIP116 handle
     IOCTL_TIP116_TIMER_CONFIGURE,
     &cfgBuf,                          // input buffer
     sizeof(TIP116_TIMER_CONFIG_BUF),      // size of input buffer
     NULL,                             // output buffer
     0,                                // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     /* OK */
} else {
     ErrorHandler ("Device I/O control error”);  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The input buffer is too small. Please check the parameter *nInBufferSize* of the DeviceIoControl() function call
ERROR_INVALID_PARAMETER	A parameter has been specified with an invalid value

See Also

Win32 documentation DeviceIoControl()

4.1.3.12 IOCTL_TIP116_TIMER_EVENTWAIT

This function waits for a timer event. A pointer to the counter wait buffer (*TIP116_TIMER_WAIT_BUF*) must be passed by the argument *lpInBuffer* to the driver. The argument *nInBufferSize* specifies the size of this buffer.

The *TIP116_TIMER_WAIT_BUF* structure has the following layout:

```
typedef struct
{
    int                 timeout;
} TIP116_TIMER_WAIT_BUF;
```

timeout

This parameter specifies the maximum time the function shall wait before it returns even if the timer event has not occurred. The time is specified in seconds.

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_TIMER_WAIT_BUF waitBuf;

/* Wait for timer event, timeout after a minute */
waitBuf.timeout = 60;

success = DeviceIoControl (
    hDevice,                        // TIP116 handle
    IOCTL_TIP116_TIMER_EVENTWAIT,
    &waitBuf,                       // input buffer
    sizeof(TIP116_TIMER_WAIT_BUF),  // size of input buffer
    NULL,                           // output buffer
    0,                              // size of output buffer
    &NumBytes,                      // number of returned bytes
    0);
if (success) {
    /* OK */
} else {
    ErrorHandler ("Device I/O control error”);  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The input buffer is too small. Please check the parameter *nInBufferSize* of the DeviceIoControl() function call
ERROR_SEM_TIMEOUT	The timeout time has passed before the specified event has occurred

See Also

Win32 documentation DeviceIoControl()

4.1.3.13 IOCTL_TIP116_GPIO_READ

This function returns the actual value of the GPIO pin. A pointer to the GPIO read buffer (*TIP116_GPIO_BUF*) must be passed by the argument *lpOutBuffer* to the driver. The argument *nOutBufferSize* specifies the size of this buffer.

The GPIO pin must be configured as input pin

The *TIP116_GPIO_BUF* structure has the following layout:

```
typedef struct
{
      unsigned char          value;
} TIP116_GPIO_BUF;
```

value

This argument returns the actual state of the GPIO pin.

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_GPIO_BUF rdGpioBuf;

/* Read state of the GPIO pin */
success = DeviceIoControl (
     hDevice,                          // TIP116 handle
     IOCTL_TIP116_GPIO_READ,
     NULL,                             // input buffer
     0,                                // size of input buffer
     &rdGpioBuf,                       // output buffer
     sizeof(TIP116_GPIO_BUF),          // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     printf(“GPIO: %s\n”,
               rdGpioBuf.value ? “ACTIVE” :”PASSIVE”);
} else {
     ErrorHandler ("Device I/O control error”);  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The output buffer is too small. Please check the parameter *nOutBufferSize* of the DeviceIoControl() function call

See Also

Win32 documentation DeviceIoControl()

4.1.3.14 IOCTL_TIP116_GPIO_SET

This function set the GPIO pin. A pointer to the GPIO set buffer (*TIP116_GPIO_BUF*) must be passed by the argument *lpInBuffer* to the driver. The argument *nInBufferSize* specifies the size of this buffer.

The GPIO pin must be configured as output pin

The *TIP116_GPIO_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char       value;
} TIP116_GPIO_BUF;
```

value

This argument specifies the output of the GPIO pin, a 1 sets an active signal and a 0 sets a passive signal.

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_GPIO_BUF wrGpioBuf;

/* Set GPIO pin active */
wrGpioBuf.value = 1;

success = DeviceIoControl (
     hDevice,                        // TIP116 handle
     IOCTL_TIP116_GPIO_SET,
     &wrGpioBuf,                     // input buffer
     sizeof(TIP116_GPIO_BUF),        // size of input buffer
     NULL,                           // output buffer
     0,                              // size of output buffer
     &NumBytes,                      // number of returned bytes
     0);
if (success) {
     /* OK */
} else {
     ErrorHandler ("Device I/O control error”);  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The input buffer is too small. Please check the parameter *nInBufferSize* of the DeviceIoControl() function call

See Also

Win32 documentation DeviceIoControl()

4.1.3.15 IOCTL_TIP116_GPIO_CONFIGURE

This function configures the function of the GPIO pin. A pointer to the GPIO configure buffer (*TIP116_GPIO_CONFIG_BUF*) must be passed by the argument *lpInBuffer* to the driver. The argument *nInBufferSize* specifies the size of this buffer.

The *TIP116_GPIO_CONFIG_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char       mode;
    unsigned char       direction;
    unsigned char       clockDivider;
} TIP116_GPIO_CONFIG_BUF;
```

mode

This parameter specifies function of the GPIO pin. Use the following modes ('tip116.h'):

mode	description
TIP116_GPIO_MD_IO	This value configures the GPIO pin for simple I/O operations
TIP116_GPIO_MD_LATCH	This value configures the GPIO pin for simultaneous count latch operations
TIP116_GPIO_MD_CLOCK	This value configures the GPIO pin as clock output

direction

This parameter specifies the direction of the GPIO pin. Use the following defines ('tip116.h'):

direction	description
TIP116_GPIO_MD_OUTPUT	This value sets the GPIO pin to be an output
TIP116_GPIO_MD_INPUT	This value sets the GPIO pin to be an input

clockDivider

This parameter specifies frequency of the clock output. The following clock speeds are defined ('tip116.h'):

clock divider	description
TIP116_GPIO_MD_8MHZ	This clock divider will be set to 1, the clock rate will be 8 MHz
TIP116_GPIO_MD_4MHZ	This clock divider will be set to 2, the clock rate will be 4 MHz
TIP116_GPIO_MD_2MHZ	This clock divider will be set to 4, the clock rate will be 2 MHz
TIP116_GPIO_MD_1MHZ	This clock divider will be set to 8, the clock rate will be 1 MHz

Example

```
#include “tip116.h”

HANDLE hDevice;
BOOLEAN success;
ULONG NumBytes;
TIP116_GPIO_CONFIG_BUF cfgBuf;

/* GPIO should generate a clock signal of 4 MHz */
cfgBuf.mode = TIP116_GPIO_MD_CLOCK;
cfgBuf.direction = TIP116_GPIO_MD_OUTPUT;
cfgBuf.clockDivider = TIP116_GPIO_MD_4MHZ;

success = DeviceIoControl (
     hDevice,                          // TIP116 handle
     IOCTL_TIP116_GPIO_CONFIGURE,
     &cfgBuf,                          // input buffer
     sizeof(TIP116_GPIO_CONFIG_BUF), // size of input buffer
     NULL,                             // output buffer
     0,                                // size of output buffer
     &NumBytes,                        // number of returned bytes
     0);
if (success) {
     /* OK */
} else {
     ErrorHandler ("Device I/O control error”);  // process error
}
```

Error Codes

ERROR_INSUFFICIENT_BUFFER	The input buffer is too small. Please check the parameter *nInBufferSize* of the DeviceIoControl() function call
ERROR_INVALID_PARAMETER	A parameter has been specified with an invalid value

See Also

Win32 documentation DeviceIoControl()