

TIP116-SW-82

Linux Device Driver

4 Channel Quadrature / General Purpose Counter

Version 1.0.x

User Manual

Issue 1.0.2

July 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

TIP116-SW-82

Linux Device Driver

4 Channel Quadrature/General Purpose Counter

Supported Modules:

TIP116

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2005-2008 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	May 11, 2005
1.0.1	General Revision	February 05, 2008
1.0.2	Carrier Driver description added	July 17, 2008

Table of Contents

1	INTRODUCTION.....	4
	1.1 Device Driver	4
	1.2 IPAC Carrier Driver	5
2	INSTALLATION.....	6
	2.1 Build and install the device driver.....	7
	2.2 Uninstall the device driver	7
	2.3 Install device driver into the running kernel	8
	2.4 Remove device driver from the running kernel	8
	2.5 Change Major Device Number	9
3	DEVICE INPUT/OUTPUT FUNCTIONS	10
	3.1 open()	10
	3.2 close().....	12
	3.3 ioctl()	13
	3.3.1 T116_IOC_G_READ_COUNTER.....	15
	3.3.2 T116_IOC_S_SET_PRELOAD	17
	3.3.3 T116_IOC_S_SET_COMPARE	19
	3.3.4 T116_IOC_S_CONFIG_COUNTER	21
	3.3.5 T116_IOC_S_RESET_COUNTER	24
	3.3.6 T116_IOC_LATCH_COUNTER.....	25
	3.3.7 T116_IOC_G_READ_ALL_COUNTER	26
	3.3.8 T116_IOC_G_INPUT_STATE	28
	3.3.9 T116_IOC_G_READ_TIMER	30
	3.3.10 T116_IOC_S_CONFIG_TIMER.....	31
	3.3.11 T116_IOC_G_READ_GPIO	33
	3.3.12 T116_IOC_S_SET_GPIO.....	34
	3.3.13 T116_IOC_S_CONFIG_GPIO.....	35
	3.3.14 T116_IOC_WAIT_EVENT	37

1 Introduction

1.1 Device Driver

The TIP116-SW-82 Linux device driver allows the operation of a TIP116 IPAC module on Linux operating systems.

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-SW-82 device driver supports the following features:

- Read counter value and state from a specified channel
- Set counter preload value of a specified channel
- Set counter compare value of a specified channel
- Reset counter
- Configure counter mode
- Read all counter values and states
- Latch all channels at the same time
- Read input state of X, Y, Z lines of channel 1 to 4
- Read timer counter
- Configure timer counter
- Read state of the GPIO pin (input)
- Set state of the GPIO pin (output)
- Configure function of the GPIO pin
- Wait for counter match, counter latch, or timer interrupt events

The TIP116-SW-82 device driver supports the modules listed below:

TIP116	4 Channel Quadrature / General Purpose Counter	(IndustryPack ®)
--------	---	------------------

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
IPAC Carrier Driver 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-82 is part of this TIP116-SW-82 distribution. It is located in directory CARRIER-SW-82 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-82 User Manual for a detailed description how to install and setup the CARRIER-SW-82 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

Following files are located on the distribution media:

Directory path 'TIP116-SW-82':

TIP116-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
TIP116-SW-82-1.0.2.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

For installation the files have to be copied to the desired target directory.

The GZIP compressed archive TIP116-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tip116/':

tip116.c	Driver source code
tip116def.h	Driver include file
tip116.h	Driver include file for application program
makenode	Script to create device nodes on the file system
Makefile	Device driver make file
example/tip116exa.c	Example application
example/Makefile	Example application make file
include/config.h	Driver independent library header file
include/tpmodule.h	Kernel independent library header file
include/tpmodule.c	Kernel independent library source code file

In order to perform an installation, extract all files of the archive TIP116-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzf TIP116-SW-82-SRC.tar.gz' will extract the files into the local directory.

Before building a new device driver, the TEWS TECHNOLOGIES IPAC carrier driver must be installed properly, because this driver includes the header file *ipac_carrier.h*, which is part of the IPAC carrier driver distribution. Please refer to the IPAC carrier driver user manual in the directory path *CARRIER-SW-82* on the separate distribution media.

2.1 Build and install the device driver

- Login as root
- Change to the target directory
- To create and install the driver in the module directory `/lib/modules/<version>/misc` enter:

make install

For Linux kernel 2.6.x, there may be compiler warnings claiming some undefined `ipac_*` symbols. These warnings are caused by the IPAC carrier driver, which is unknown during compilation of this TIP driver. The warnings can be ignored.

- Also after the first build we have to execute `depmod` to create a new dependency description for loadable kernel modules. This dependency file is later used by `modprobe` to automatically load the correct IPAC carrier driver modules.

depmod -aq

2.2 Uninstall the device driver

- Login as root
- Change to the target directory
- To remove the driver from the module directory `/lib/modules/<version>/misc` enter:

make uninstall

- Update kernel module dependency description file

depmod -aq

2.3 Install device driver into the running kernel

- To load the device driver into the running kernel, login as root and execute the following commands:

```
# modprobe tip116drv
```

- After the first build or if you are using dynamic major device allocation it is necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled a device file system (devfs or sysfs with udev) then you have to skip running the *makenode* script. Instead of creating device nodes from the script the driver itself takes creating and destroying of device nodes in its responsibility.

```
# sh makenode
```

On success the device driver will create a minor device for each TIP116 module found. The first TIP116 can be accessed with device node `/dev/tip116_0`, the second TIP116 with device node `/dev/tip116_1`, the third TIP116 with device node `/dev/tip116_2` and so on.

The allocation of device nodes to physical TIP116 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP116 device driver will only work if kernel KMOD support is installed, necessary carrier board drivers already installed and the kernel dependency file is up to date. If KMOD support isn't available you have to build either a new kernel with KMOD installed or you have to install the IPAC carrier kernel modules manually in the correct order (please refer to the IPAC carrier driver user manual).

2.4 Remove device driver from the running kernel

- To remove the device driver from the running kernel login as root and execute the following command:

```
# modprobe tip116drv -r
```

If your kernel has enabled devfs or sysfs (udev), all `/dev/tip116_x` nodes will be automatically removed from your file system after this.

Be sure that the driver isn't opened by any application program. If opened you will get the response "*tip116drv: Device or resource busy*" and the driver will still remain in the system until you close all opened files and execute *modprobe -r* again.

2.5 Change Major Device Number

The TIP116 driver uses dynamic allocation of major device numbers per default. If this isn't suitable for the application it's possible to define a major number for the driver.

To change the major number edit the file tip116.c, change the following symbol to appropriate value and enter make install to create a new driver.

TIP116_MAJOR

Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP116_MAJOR      122
```

3 Device Input/Output functions

This chapter describes the interface to the device driver I/O system.

3.1 open()

NAME

open() - open a file descriptor

SYNOPSIS

```
#include <fcntl.h>
```

```
int open (const char *filename, int flags)
```

DESCRIPTION

The open function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file is to be opened. This is a bit mask; you create the value by the bitwise OR of the appropriate parameters (using the | operator in C). See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int fd;
fd = open("/dev/tip116_0", O_RDWR);
if (fd == -1)
{
    /* handle error condition */
}
```

RETURNS

The normal return value from open is a non-negative integer file descriptor. In the case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

ENODEV

The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during open. For more information about open error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.2 close()

NAME

close() – close a file descriptor

SYNOPSIS

```
#include <unistd.h>
```

```
int close (int filedes)
```

DESCRIPTION

The close function closes the file descriptor *filedes*.

EXAMPLE

```
int fd;

if (close(fd) != 0)
{
    /* handle close error conditions */
}
```

RETURNS

The normal return value from close is 0. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

ENODEV

The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during close. For more information about close error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 ioctl()

NAME

ioctl() – device control functions

SYNOPSIS

```
#include <sys/ioctl.h>
```

```
int ioctl(int filedes, int request [, void *argp])
```

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *filedes*, causing the corresponding device to perform the requested operation.

The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *tip116.h*:

Symbol	Meaning
T116_IOCTLG_READ_COUNTER	Read counter value and state of a specified channel
T116_IOCS_SET_PRELOAD	Set counter preload value of a specified channel
T116_IOCS_SET_COMPARE	Set counter compare value of a specified channel
T116_IOCS_CONFIG_COUNTER	Configure operation mode of a specified channel
T116_IOCS_RESET_COUNTER	Reset counter value of a specified channel
T116_IOCTL_LATCH_COUNTER	Latch all counter values
T116_IOCTLG_READ_ALL_COUNTER	Read counter value and state of all channels
T116_IOCTLG_INPUT_STATE	Get line input state of all X, Y, and Z lines
T116_IOCTLG_READ_TIMER	Read current timer counter value
T116_IOCS_CONFIG_TIMER	Configure timer counter
T116_IOCTLG_READ_GPIO	Read state of GPIO pin
T116_IOCS_SET_GPIO	Set state of GPIO pin
T116_IOCS_CONFIG_GPIO	Configure function of GPIO pin
T116_IOCTL_WAIT_EVENT	Wait for a counter latch, counter match, or timer interrupt event

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.

RETURNS

On success, zero is returned. In the case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL

Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument request.

Other function dependant error codes will be described for each ioctl code separately. Note, the TIP116 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 T116_IOC_G_READ_COUNTER

NAME

T116_IOC_G_READ_COUNTER – Reads the counter value and state

DESCRIPTION

This ioctl function reads the current counter value and state of the specified channel of the TIP116 associated with the open file descriptor, *filedes* and returns the value in the parameter buffer (*T116_COUNTER_BUFFER*) pointed to by *argp*.

```
typedef struct
{
    int      channel;
    long     data;
    long     flags;
} T116_COUNTER_BUFFER, *PT116_COUNTER_BUFFER;
```

channel

This parameter specifies the counter channel number. Allowed values are 1 up to 4.

data

This parameter returns the counter value.

flags

This parameter returns flags specifying the state of the counter channel. The following flags can be returned by the driver.

Status flag	Description
T116_BORROW	The counter has changed from 0 to -1 (0 → 0xFFFFFFFF)
T116_CARRY	The counter has changed from -1 to 0 (0xFFFFFFFF → 0)
T116_MATCH	The counter has hit the set match value of the channel.
T116_SIGN	This flag displays the sign of the value (if value is used as an unsigned value). This flag always shows the direction of the last overflow (set flag) or underflow (unset flag) condition.
T116_DIRECTION	This flag displays the last count direction. If the flag is set the last count has incremented the counter, if it is unset it has decremented the counter.
T116_LATCHED	This flag is set if the returned value has been latched before.
T116_LATCH_OVERFLOW	This flag displays if more than one latch has been executed since the read out of the value. (Data has been lost)

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
T116_COUNTER_BUFFER countBuf;

/* Read value and state of counter channel 2 */
countBuf.channel = 2;

result = ioctl(fd, T116_IOCTL_READ_COUNTER, &countBuf);
if (result >= 0)
{
    /* OK */
    printf("Value: %ld\n", countBuf.data);
    printf("Stateflags: %s%s%s%s%s\n",
           (countBuf.flags & T116_BORROW) ? "BORROW " : "",
           (countBuf.flags & T116_CARRY) ? "CARRY " : "",
           (countBuf.flags & T116_MATCH) ? "MATCH " : "",
           (countBuf.flags & T116_LATCHED) ? "LATCHED " : "",
           (countBuf.flags & T116_OVERFLOW) ? "LATCHOVERFLOW " : "");
    printf("Sign: %s\n",
           (countBuf.flags & T116_SIGN) ? "OVERFLOW" : "UNDERFLOW");
    printf("Direction: %s\n\n",
           (countBuf.flags & T116_DIRECTION) ? "UP" : "DOWN");
}
else
{
    /* ERROR */
}
}
```

ERRORS

EFAULT

Invalid pointer to the read buffer.
Invalid channel number specified.

3.3.2 T116_IOC_SET_PRELOAD

NAME

T116_IOC_SET_PRELOAD – Set the counter preload value

DESCRIPTION

This ioctl function sets the counter preload value of the specified channel of the TIP116 associated with the open file descriptor, *filedes*. The preload value and flags are supplied in the parameter buffer (*T116_COUNTER_BUFFER*) pointed to by *argp*.

typedef struct

```
{
    int      channel;
    long     data;
    long     flags;
} T116_COUNTER_BUFFER, *PT116_COUNTER_BUFFER;
```

channel

This parameter specifies the counter channel number. Allowed values are 1 up to 4.

data

This parameter specifies the new counter preload value.

flags

This parameter specifies ORed flags, used when loading the preload value. The following flags are defined.

Status flag	Description
T116_IMMEDIATE_PRELOAD	The preload value will be loaded into the counter when setting the preload value.

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
T116_COUNTER_BUFFER countBuf;

/* Set preload value to 50000 and load the counter of channel 3 */
/* immediately */
countBuf.channel = 3;
countBuf.data = 50000;
countBuf.flags = T116_IMMEDIATE_PRELOAD;

result = ioctl(fd, T116_IOCTL_SET_PRELOAD, &countBuf);
if (result >= 0)
{
    /* OK */
}
else
{
    /* ERROR */
}
```

ERRORS

EFAULT

Invalid pointer to the preload buffer.
Invalid channel number specified.
Invalid flag specified.

3.3.3 T116_IOCS_SET_COMPARE

NAME

T116_IOCS_SET_COMPARE – Set the counter compare value

DESCRIPTION

This ioctl function sets the counter compare value of the specified channel of the TIP116 associated with the open file descriptor, *filedes*. The compare value is supplied in the parameter buffer (*T116_COUNTER_BUFFER*) pointed to by *argp*.

typedef struct

```
{
    int      channel;
    long     data;
    long     flags;
} T116_COUNTER_BUFFER, *PT116_COUNTER_BUFFER;
```

channel

This parameter specifies the counter channel number. Allowed values are 1 up to 4.

data

This parameter specifies the new counter preload value.

flags

This parameter is unused and ignored for this function.

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
T116_COUNTER_BUFFER countBuf;

/* Set compare value of channel 2 to 10000 */
/* immediately */
countBuf.channel = 2;
countBuf.data = 10000;

result = ioctl(fd, T116_IOC_SET_COMPARE, &countBuf);
if (result >= 0)
{
    /* OK */
}
else
{
    /* ERROR */
}
```

ERRORS

EFAULT

Invalid pointer to the compare buffer.
Invalid channel number specified.

3.3.4 T116_IOCS_CONFIG_COUNTER

NAME

T116_IOCS_CONFIG_COUNTER – Configures the specified counter channel

DESCRIPTION

This ioctl function configures the specified counter channel of the TIP116 associated with the open file descriptor, *filedes*. The compare value is supplied in the parameter buffer (*T116_COUNTER_CONFIG_BUFFER*) pointed to by *argp*.

typedef struct

```
{
    int          channel;
    unsigned long inputMode;
    unsigned long specMode;
    unsigned long ZControl;
    unsigned long QuadControl;
    unsigned long XYZPolarity;
} T116_COUNTER_CONFIG_BUFFER, *PT116_COUNTER_CONFIG_BUFFER;
```

channel

This parameter specifies the counter channel number. Allowed values are 1 up to 4.

inputMode

This parameter specifies the input mode. The following input modes are defined:

Counter Input Mode	Description
T116_COUNTOFF	Counter disabled
T116_QUADCOUNT	Quadrature count
T116_UPDOWNCOUNT	Up/Down count
T116_DIRECTCOUNT	Direction count

specMode

This parameter specifies the special count mode. The following special count modes are defined:

Special Count Mode	Description
T116_SPECMODENO	No special mode active
T116_SPECMODEDIVN	Divide-by-N
T116_SPECMODESNGLCYC	Single cycle

ZControl

This parameter specifies the Z-control mode. The following Z-control modes are defined:

Z-Control Mode	Description
T116_ZNO	No Z-control
T116_ZLOAD	Load on Z
T116_ZLATCH	Latch on Z
T116_ZGATE	Gate on Z
T116_ZRESET	Reset on Z

QuadControl

This parameter specifies the quadrature control mode. This value is only used in quadrature counter mode. The following quadrature control modes are defined:

Quadrature Control Mode	Description
T116_1X	1X counting
T116_2X	2X counting
T116_4X	4X counting

XYZPolarity

This parameter field specifies the input polarity of the X, Y, and Z input lines. Following values can be combined by OR:

Mode	Description
T116_XHIGHACTIV	X input signals are interpreted as high active. (overridden by <i>T116_XLOWACTIV</i>)
T116_XLOWACTIV	X input signals are interpreted as low active. (overrides <i>T116_XHIGHACTIV</i>)
T116_YHIGHACTIV	Y input signals are interpreted as high active. (overridden by <i>T116_YLOWACTIV</i>)
T116_YLOWACTIV	Y input signals are interpreted as low active. (overrides <i>T116_YHIGHACTIV</i>)
T116_ZHIGHACTIV	Z input signals are interpreted as high active. (overridden by <i>T116_ZLOWACTIV</i>)
T116_ZLOWACTIV	Z input signals are interpreted as low active. (overrides <i>T116_ZHIGHACTIV</i>)

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
T116_COUNTER_CONFIG_BUFFER countConfBuf;

/* Configure counter channel 1 as follows*/
/* - quadrature counter (4X) */
/* - no special mode */
/* - Latch on Z signal */
/* - X, Y, Z are low active */
countBuf.channel = 2;
countBuf.inputMode = T116_QUADCOUNT;
countBuf.specMode = T116_SPECMODENO;
countBuf.ZControl = T116_ZLATCH;
countBuf.QuadControl = T116_4X;
countBuf.XYZPolarity = T116_XLOWACTIV | T116_YLOWACTIV | T116_ZLOWACTIV;

result = ioctl(fd, T116_IOCS_CONFIG_COUNTER, &countConfBuf);
if (result >= 0)
{
    /* OK */
}
else
{
    /* ERROR */
}
```

ERRORS

EFAULT

Invalid pointer to the configuration buffer.
Invalid channel number specified.
Invalid parameter value specified.

3.3.5 T116_IOC_RESET_COUNTER

NAME

T116_IOC_RESET_COUNTER – Reset the specified counter

DESCRIPTION

This ioctl function resets the counter of the specified channel from the TIP116 associated with the open file descriptor, *filedes*. The channel is supplied in the parameter buffer (*T116_COUNTER_RESET_BUFFER*) pointed to by *argp*.

typedef struct

```
{
    int      channel;
} T116_COUNTER_RESET_BUFFER, *PT116_COUNTER_RESET_BUFFER;
```

channel

This parameter specifies the counter channel number. Allowed values are 1 up to 4.

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
T116_COUNTER_RESET_BUFFER resetBuf;

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

result = ioctl(fd, T116_COUNTER_RESET_BUFFER, &resetBuf);
if (result >= 0)
{
    /* OK */
} else {
    /* ERROR */
}
```

ERRORS

EFAULT

Invalid pointer to the buffer.
Invalid channel number specified.

3.3.6 T116_IOC_LATCH_COUNTER

NAME

T116_IOC_LATCH_COUNTER – Latch all counters

DESCRIPTION

This ioctl function latches all counters of the TIP116 associated with the open file descriptor, *filedes*. The buffer pointed *argp* will not be used and should be set to *NULL*.

EXAMPLE

```
#include <tip116.h>

int fd;
int result;

/* Latch all counter values */
result = ioctl(fd, T116_IOC_LATCH_COUNTER, NULL);
if (result >= 0)
{
    /* OK */
}
else
{
    /* ERROR */
}
```

ERRORS

EBUSY

Data get lost, previous latched values have not been read.

3.3.7 T116_IOC_G_READ_ALL_COUNTER

NAME

T116_IOC_G_READ_ALL_COUNTER – Reads the counter values and states of all channels

DESCRIPTION

This ioctl function reads the current counter values and states of all channels from the TIP116 associated with the open file descriptor, *filedes* and it returns the values in the parameter buffer (*T116_READ_ALL_BUFFER*) pointed to by *argp*.

typedef struct

```
{
    long      data[4];
    long      flags[4];
} T116_READ_ALL_BUFFER, *PT116_READ_ALL_BUFFER;
```

data[]

This array returns the counter values for all four channels. The array index specifies the counter channel. (Index 0 specifies counter channel 1, Index 1 specifies counter channel 2, and so on.)

flags[]

This parameter returns flags specifying the state of the counter channels. The array index specifies the counter channel. The index corresponds to the index used for *data[]*. (Index 0 specifies counter channel 1, Index 1 specifies counter channel 2, and so on.) The following flags can be returned by the driver.

Status flag	Description
T116_BORROW	The counter has changed from 0 to -1 (0 → 0xFFFFFFFF)
T116_CARRY	The counter has changed from -1 to 0 (0xFFFFFFFF → 0)
T116_MATCH	The counter has hit the set match value of the channel.
T116_SIGN	This flag displays the sign of the value (if value is used as an unsigned value). This flag always shows the direction of the last overflow (set flag) or underflow (unset flag) condition.
T116_DIRECTION	This flag displays the last count direction. If the flag is set the last count has incremented the counter, if it is unset it has decremented the counter.
T116_LATCHED	This flag is set if the returned value has been latched before.
T116_LATCH_OVERFLOW	This flag displays if more than one latch has been executed since the read out of the value. (Data has been lost)

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
int channel;
T116_READ_ALL_BUFFER allBuf;

/* Read value and state of all counter channels */
result = ioctl(fd, T116_IOCTL_READ_ALL_COUNTER, &allBuf);
if (result >= 0)
{
    /* OK */
    for (channel = 0; channel < 4; channel++)
    {
        printf("Channel %d\n", channel + 1);
        printf("Value: %d\n", allBuf.data);
        printf("Stateflags: %s%s%s%s\n",
            (allBuf.flags & T116_BORROW) ? "BORROW " : "",
            (allBuf.flags & T116_CARRY) ? "CARRY " : "",
            (allBuf.flags & T116_MATCH) ? "MATCH " : "",
            (allBuf.flags & T116_LATCHED) ? "LATCHED " : "",
            (allBuf.flags & T116_CARRY) ? "LATCHOVERFLOW " : "");
        printf("Sign: %s\n",
            (allBuf.flags & T116_SIGN) ? "OVERFLOW" : "UNDERFLOW");
        printf("Direction: %s\n\n",
            (allBuf.flags & T116_DIRECTION) ? "UP" : "DOWN");
    }
}
else
{
    /* ERROR */
}
}
```

ERRORS

EFAULT

Invalid pointer to the read buffer.

3.3.8 T116_IOCTL_INPUT_STATE

NAME

T116_IOCTL_INPUT_STATE – Reads the current state of the X, Y, and Z lines of the input counters.

DESCRIPTION

This ioctl function reads the current state of the X, Y, and Z lines of the input counters from the TIP116 associated with the open file descriptor, *filedes* and returns the values in the parameter buffer (*T116_INPUT_STATE_BUFFER*) pointed to by *argp*.

typedef struct

```
{
    unsigned short    inputState;
} T116_INPUT_STATE_BUFFER, *PT116_INPUT_STATE_BUFFER;
```

inputState

This parameter returns the states of the X, Y, and Z lines. The bits are assigned as shown below.

Bit	Input Line
0	X-Input Channel 1
1	Y-Input Channel 1
2	Z-Input Channel 1
3	X-Input Channel 2
4	Y-Input Channel 2
5	Z-Input Channel 2
6	X-Input Channel 3
7	Y-Input Channel 3
8	Z-Input Channel 3
9	X-Input Channel 4
10	Y-Input Channel 4
11	Z-Input Channel 4

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
T116_IOCTL_INPUT_STATE statBuf;

/* Read value and state of all counter channels */
result = ioctl(fd, T116_IOCTL_INPUT_STATE, &statBuf);
if (result >= 0)
{
    /* OK */
    printf("InputState: %X\n", statBuf.inputState);
}
else
{
    /* ERROR */
}
```

ERRORS

EFAULT

Invalid pointer to the read buffer.

3.3.9 T116_IOCTL_READ_TIMER

NAME

T116_IOCTL_READ_TIMER – Reads the current timer counter value

DESCRIPTION

This ioctl function reads the current counter value and state of the specified channel from the TIP116 associated with the open file descriptor, *filedes* and returns the value in the parameter buffer (*T116_TIMER_BUFFER*) pointed to by *argp*.

```
typedef struct
{
    unsigned short    value;
} T116_TIMER_BUFFER, *PT116_TIMER_BUFFER;
```

value

This parameter returns the timer counter value.

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
T116_TIMER_BUFFER timerBuf;

/* Read timer counter value */
result = ioctl(fd, T116_IOCTL_READ_TIMER, &timerBuf);
if (result >= 0)
{
    /* OK */
    printf("Value: %ld\n", timerBuf.value);
}
else
{
    /* ERROR */
}
```

ERRORS

EFAULT

Invalid pointer to the read buffer.

3.3.10 T116_IOCS_CONFIG_TIMER

NAME

T116_IOCS_CONFIG_TIMER – Configures the timer counter

DESCRIPTION

This ioctl function configures the timer counter of the TIP116 associated with the open file descriptor, *filedes*. The compare value is supplied in the parameter buffer (*T116_TIMER_CONFIG_BUFFER*) pointed to by *argp*.

```
typedef struct
{
    unsigned short    preloadVal;
    int               prescaler;
    int               enable;
} T116_TIMER_CONFIG_BUFFER, *PT116_TIMER_CONFIG_BUFFER;
```

preloadVal

This parameter specifies the timer counter preload value.

prescaler

This parameter specifies clock prescaler of timer counter input clock. Allowed values are 1, 2, 4, and 8.

Prescaler	Clock
1	8 MHz
2	4 MHz
4	2 MHz
8	1 MHz

enable

This parameter specifies if the timer counter should be enabled or not. *TRUE* will enable the timer counter, *FALSE* will disable it.

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
T116_TIMER_CONFIG_BUFFER timerConfBuf;

/* Enable timer and set preload value to 1000 and */
/* count with a clock of 1Mhz */
timerConfBuf.preloadVal = 10000;
timerConfBuf.prescaler = 8;
timerConfBuf.enable = TRUE;

result = ioctl(fd, T116_IOCS_CONFIG_TIMER, &timerConfBuf);
if (result >= 0)
{
    /* OK */
}
else
{
    /* ERROR */
}
```

ERRORS

EFAULT

Invalid pointer to the configuration buffer.

3.3.11 T116_IOCTL_READ_GPIO

NAME

T116_IOCTL_READ_GPIO – Reads the current state of the GPIO pin

DESCRIPTION

This ioctl function reads the current state of the GPIO pin of the TIP116 associated with the open file descriptor, *filedes* and returns the value in the parameter buffer (*T116_GPIO_BUFFER*) pointed to by *argp*. The state is only valid in input mode.

```
typedef struct
{
    unsigned char    value;
} T116_GPIO_BUFFER, *PT116_GPIO_BUFFER;
```

value

This parameter returns the GPIO state. If GPIO input signal is high the returned value is 1, if the signal is low the returned value is 0.

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
T116_GPIO_BUFFER gpioBuf;

/* Read state of GPIO pin */
result = ioctl(fd, T116_IOCTL_READ_GPIO, &gpioBuf);
if (result >= 0)
{
    /* OK */
    printf("GPIO-pin %s\n", gpioBuf.value ? "HIGH" : "LOW");
}
else
{
    /* ERROR */
}
```

ERRORS

EFAULT

Invalid pointer to the read buffer.

3.3.12 T116_IOCSET_GPIO

NAME

T116_IOCSET_GPIO – Sets the state of the GPIO pin

DESCRIPTION

This ioctl function sets the state of the GPIO pin of the TIP116 associated with the open file descriptor, *filedes* and returns the value in the parameter buffer (*T116_GPIO_BUFFER*) pointed to by *argp*. This function is only used in GPIO output mode, otherwise it is ignored.

```
typedef struct
{
    unsigned char    value;
} T116_GPIO_BUFFER, *PT116_GPIO_BUFFER;
```

value

This parameter specifies the new GPIO output state. 1 set the GPIO signal high, 0 sets it low.

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
T116_GPIO_BUFFER gpioBuf;

/* Set GPIO pin to low */
gpioBuf.value = 0;

result = ioctl(fd, T116_IOCSET_GPIO, &gpioBuf);
if (result >= 0)
{
    /* OK */
}
else
{
    /* ERROR */
}
```

ERRORS

EFAULT

Invalid pointer to the set buffer.

3.3.13 T116_IOCS_CONFIG_GPIO

NAME

T116_IOCS_CONFIG_GPIO – Configures the GPIO pin

DESCRIPTION

This ioctl function configures the GPIO pin of the TIP116 associated with the open file descriptor, *filedes*. The compare value is supplied in the parameter buffer (*T116_CONFIG_GPIO_BUFFER*) pointed to by *argp*.

```
typedef struct
{
    int         mode;
    int         direction;
    int         clockDivider;
} T116_CONFIG_GPIO_BUFFER, *PT116_CONFIG_GPIO_BUFFER;
```

mode

This parameter specifies the GPIO mode. The following GPIO modes are defined:

GPIO Mode	Description
T116_GPIO_GPIO	The GPIO pin is used as general purpose I/O.
T116_GPIO_COUNTLATCH	The GPIO pin is used as external counter latch signal.
T116_GPIO_CLK	The GPIO pin is used as clock output.

direction

This parameter specifies the direction of the GPIO pin in GPIO and latched mode, this value is ignored for clock mode. The following special count modes are defined:

GPIO Direction	GPIO Mode	Latch Mode
T116_GPIO_IN	general purpose input	latch slave (triggers latch)
T116_GPIO_OUT	general purpose output	latch master (signal is generated on latch command)

clockDivider

This parameter field specifies the output clock divider. This parameter is only used for clock output mode. Following values can be used:

Divider	Description
1	1X – 8MHz
2	2X – 4MHz
4	4X – 2MHz
8	8X – 1MHz

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
T116_CONFIG_GPIO_BUFFER gpioConfBuf;

/* Configure GPIO as cock output pin (2 MHz) */
countBuf.mode = T116_GPIO_CLK;
countBuf.clockDivider = 4;

result = ioctl(fd, T116_IOCS_CONFIG_GPIO, &gpioConfBuf);
if (result >= 0)
{
    /* OK */
}
else
{
    /* ERROR */
}

/* Configure GPIO output pin */
countBuf.mode = T116_GPIO_GPIO;
countBuf.direction = T116_GPIO_OUT;

result = ioctl(fd, T116_IOCS_CONFIG_GPIO, &gpioConfBuf);
if (result >= 0)
{
    /* OK */
}
else
{
    /* ERROR */
}
```

ERRORS

EFAULT

Invalid pointer to the event wait buffer.
Invalid parameter value specified.

3.3.14 T116_IOC_WAIT_EVENT

NAME

T116_IOC_WAIT_EVENT – Waits for interrupt event

DESCRIPTION

This ioctl function waits for a specified interrupt event of the TIP116 associated with the open file descriptor, *filedes*. The compare value is supplied in the parameter buffer (*T116_EVENT_BUFFER*) pointed to by *argp*.

```
typedef struct
{
    int      event;
    int      channel;
    int      timeout;
} T116_EVENT_BUFFER, *PT116_EVENT_BUFFER;
```

event

This parameter specifies the event to wait for. The following events are defined:

Event	Description
T116_EV_COUNTER_LATCH	The event mode will wait for a latch event (via Z signal in latch on Z mode) on the specified channel.
T116_EV_COUNTER_MATCH	The event mode will wait for a match event on the specified channel. (Compare and counter value are matching)
T116_EV_TIMER	This event mode will wait for a timer event. The specified channel number is ignored.

channel

This parameter specifies the counter channel number. It is unused for timer events. Allowed values are 1 up to 4.

timeout

This parameter specifies the maximum time the function should wait for the event. The time is specified in ticks.

EXAMPLE

```
#include <tip116.h>

int fd;
int result;
T116_EVENT_BUFFER eventBuf;

/* Wait for a match event on channel 4 (Timeout after 10000 ticks) */
eventBuf.event = T116_EV_COUNTER_MATCH;
eventBuf.channel = 4;
eventBuf.timeout = 10000;

result = ioctl(fd, T116_IOC_WAIT_EVENT, &eventBuf);
if (result >= 0)
{
    /* OK */
}
else
{
    /* ERROR */
}
```

ERRORS

EFAULT	Invalid pointer to the configuration buffer. Invalid channel number specified. Invalid parameter specified.
ETIME	The specified timeout time has expired without an event.