The Embedded I/O Company

TEWS TECHNOLOGIES

TIP119-SW-82

Linux Device Driver

Six Channel 16 bit Quadrature Decoder Counter

Version 1.0.x

User Manual

Issue 1.0.0

July 2007

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

TIP119-SW-82

Linux Device Driver

Six Channel 16 bit Quadrature Decoder Counter

Issue	Description	Date
1.0.0	First Issue	July 6, 2007

Table of Contents

1 Introduction

The TIP119-SW-82 Linux device driver allows the operation of TIP119 IPAC modules on Linux operating systems.

Because the TIP119 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 TIP119 device driver includes the following features:

- read counter channel value
- reset counter channels
- wait for interrupts (overflow, index)
- counter width enhanced to 32bit by driver
- counter value rrad on index interrupt by driver

The TIP119-SW-82 supports the modules listed below:

TIP119-10	Six Channel 16 bit Quadrature Decoder Counter	IndustryPack® compatible

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

TIP119 User manual

TIP119 Engineering Manual

CARRIER-SW-82 IPAC Carrier User Manual

2 Installation

The directory TIP119-SW-82 on the distribution media contains the following files:

TIP119-SW-82-1.0.0.pdf	This manual in PDF format
TIP119-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

The GZIP compressed archive TIP119-SW-82-SRC.tar.gz contains the following files and directories:

tip119/tip119.c	Driver source code
tip119/tip119def.h	Driver include file
tip119/tip119.h	Driver include file for application program
tip119/makenode	Script to create device nodes on the file system
tip119/Makefile	Device driver make file
tip119/example/tip119exa.c	Example application
tip119/example/Makefile	Example application make file
tip119/include/tpmodule.h	Kernel independent library header file
tip119/include/tpmodule.c	Kernel independent library source code file

In order to perform an installation, extract all files of the archive TIP119-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TIP119-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 tip119drv

- After the first build or if you are using dynamic major device allocation it's 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 TIP119 module found. The first TIP119 can be accessed with device node /dev/tip119_0, the second TIP119 with device node /dev/tip119_1, the third TIP119 with device node /dev/tip119_2 and so on.

The allocation of device nodes to physical TIP119 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP119 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 tip119drv –r

If your kernel has enabled devfs or sysfs (udev), all /dev/tip119_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 "*tip119drv: 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 TIP119 driver uses dynamic allocation of major device numbers by default. If this isn't suitable for the application it's possible to define a major number for the driver. If the kernel has enabled devfs the driver will not use the symbol TIP119_MAJOR.

To change the major number edit the file tip119.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP119_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP119_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/tip119_0", O_RDWR);
if (fd < 0)
{
  /* 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

E_NODEV	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.

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

E_NODEV	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 *tip119.h*:

Symbol	Meaning
TIP119_IOCX_COUNTERREAD	Read Counter Value of specific channel
TIP119_IOC_COUNTERRESET	Reset all counter channels
TIP119_IOCX_WAITFORINDEX	Wait for Index interrupt
TIP119_IOCX_WAITFOROVERFLOWSW	Wait for Software Counter Overflow
TIP119_IOCX_WAITFOROVERFLOWHW	Wait for Hardware Counter Overflow

See behind for more detailed information on each control code.

To use these TIP119 specific control codes the header file tip119.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 TIP119 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TIP119_IOCX_COUNTERREAD

NAME

TIP119_IOCX_COUNTERREAD - Read Counter Value of specific channel

DESCRIPTION

This ioctl function returns the counter value of the specified channel. A pointer to the caller's buffer (*TIP119_COUNTER_DATA*) is passed by the parameter *argp* to the driver.

```
typedef struct
{
        unsigned char       Channel;
        unsigned long       Data;
} TIP119_COUNTER_DATA, *PTIP119_COUNTER_DATA;
```

Channel

Specifies the counter channel number to be read. The channel number is 1-based, i.e. specify "1" for channel 1, "2" for channel 2 and so on. Possible values are 1 to 6.

Data

Returns the counter data of the specified channel. The actual 16bit counter hardware value is virtually enhanced to 32bit by evaluating overflow interrupts.

EXAMPLE

```
#include "tip119.h"

int  fd;
int  result;
TIP119_COUNTER_DATA CounterData;

/*
** Read Counter Value of Channel 1
*/
CounterData.Channel = 1;
result = ioctl(fd, TIP119_IOCX_COUNTERREAD, &CounterData);

if (result >= 0) {
   printf("Counter = 0x%08lX\n", CounterData.Data);
} else {
   /* handle ioctl error */
}
```

ERRORS

Error code	Description
EFAULT	Error copying data to or from user space.
EINVAL	Invalid channel specified.

3.3.2 TIP119_IOCX_WAITFORINDEX

NAME

TIP119_IOCX_WAITFORINDEX - Wait for Index interrupt

DESCRIPTION

This ioctl waits for an incoming Index interrupt. The counter value is read by the interrupt service routine on the interrupt. A pointer to the caller's buffer (*TIP119_WAIT_STRUCT*) is passed by the parameter *argp* to the driver.

```
typedef struct
{
        unsigned char       Channel;
        int                 Timeout;
        unsigned long       CounterValue;
        unsigned long       OverflowDirection;
} TIP119_WAIT_STRUCT, *PTIP119_WAIT_STRUCT;
```

Channel

Specifies the counter channel number. The channel number is 1-based, i.e. specify "1" for channel 1, "2" for channel 2 and so on. Possible values are 1 to 6.

Timeout

Specifies the number of system ticks to wait for this interrupt. Specify 0 to wait indefinitely.

CounterValue

Returns the counter data of the specified channel, read by the driver interrupt service routine. The actual 16bit counter hardware value is virtually enhanced to 32bit by evaluating overflow interrupts.

Note that there is a delay between the real index and the reading of the counter value caused by the system interrupt latency.

OverflowDirection

Not used for this function.

EXAMPLE

```
#include “tip119.h”

int  fd;
int  result;
TIP119_WAIT_STRUCT WaitStruct;

/*
** Wait at least 100 system ticks for an Index Interrupt on Channel 1
*/
WaitStruct.Channel = 1;
WaitStruct.Timeout = 100;

result = ioctl(fd, TIP119_IOCX_WAITFORINDEX, &WaitStruct);

if (result >= 0) {
  printf(“Index Interrupt occurred.\n”);
  printf(“Read Counter = 0x%08lX\n”, WaitStruct.CounterValue);
} else {
  /* handle ioctl error */
}
```

ERRORS

Error code	Description
EFAULT	Error copying data to or from user space.
EINVAL	Invalid channel specified.
ETIME	Timeout happened before Interrupt.
EBUSY	There is already a pending job waiting for this interrupt on this channel.

3.3.3 TIP119_IOCX_WAITFOROVERFLOWSW

NAME

TIP119_IOCX_WAITFOROVERFLOWSW - Wait for Software Counter Overflow

DESCRIPTION

This ioctl waits for an incoming Software Counter (32bit) Overflow interrupt. The counter value is read by the interrupt service routine on the interrupt. A pointer to the caller's buffer (*TIP119_WAIT_STRUCT*) is passed by the parameter *argp* to the driver.

```
typedef struct
{
        unsigned char       Channel;
        int                 Timeout;
        unsigned long       CounterValue;
        unsigned long       OverflowDirection;
} TIP119_WAIT_STRUCT, *PTIP119_WAIT_STRUCT;
```

Channel

Specifies the counter channel number. The channel number is 1-based, i.e. specify "1" for channel 1, "2" for channel 2 and so on. Possible values are 1 to 6.

Timeout

Specifies the number of system ticks to wait for this interrupt. Specify 0 to wait indefinitely.

CounterValue

Returns the counter data of the specified channel, read by the driver interrupt service routine. The actual 16bit counter hardware value is virtually enhanced to 32bit by evaluating overflow interrupts.

Note that there is a delay between the real index and the reading of the counter value caused by the system interrupt latency.

OverflowDirection

This value returns the overflow direction. The following values are possible:

Value	Description
TIP119_OVERFLOW_UP	Counter Overflow from 0xFFFFFFFF to 0x00000000 (up).
TIP119_OVERFLOW_DOWN	Counter Overflow from 0x00000000 to 0xFFFFFFFF (down).

EXAMPLE

```
#include “tip119.h”

int  fd;
int  result;
TIP119_WAIT_STRUCT WaitStruct;

/*
** Wait at least 100 system ticks for a Software-Counter Overflow
** on Channel 1
*/
WaitStruct.Channel = 1;
WaitStruct.Timeout = 100;

result = ioctl(fd, TIP119_IOCX_WAITFOROVERFLOWSW, &WaitStruct);

if (result >= 0) {
  printf(“Overflow Interrupt (SW) occurred.\n”);
  printf(“Read Counter = 0x%08lX\n”, WaitStruct.CounterValue);
  printf(“Overflow Direction: %s\n”,
       (WaitStruct.OverflowDirection==TIP119_OVERFLOW_UP)?”UP”:”DOWN”);
} else {
  /* handle ioctl error */
}
```

ERRORS

Error code	Description
EFAULT	Error copying data to or from user space.
EINVAL	Invalid channel specified.
ETIME	Timeout happened before Interrupt.
EBUSY	There is already a pending job waiting for this interrupt on this channel.

3.3.4 TIP119_IOCX_WAITFOROVERFLOWHW

NAME

TIP119_IOCX_WAITFOROVERFLOWHW - Wait for Hardware Counter Overflow

DESCRIPTION

This ioctl waits for an incoming Hardware Counter (16bit) Overflow interrupt. The counter value is read by the interrupt service routine on the interrupt. A pointer to the caller's buffer (*TIP119_WAIT_STRUCT*) is passed by the parameter *argp* to the driver.

```
typedef struct
{
        unsigned char     Channel;
        int               Timeout;
        unsigned long     CounterValue;
        unsigned long     OverflowDirection;
} TIP119_WAIT_STRUCT, *PTIP119_WAIT_STRUCT;
```

Channel

Specifies the counter channel number. The channel number is 1-based, i.e. specify "1" for channel 1, "2" for channel 2 and so on. Possible values are 1 to 6.

Timeout

Specifies the number of system ticks to wait for this interrupt. Specify 0 to wait indefinitely.

CounterValue

Returns the counter data of the specified channel, read by the driver interrupt service routine. The actual 16bit counter hardware value is virtually enhanced to 32bit by evaluating overflow interrupts.

Note that there is a delay between the real index and the reading of the counter value caused by the system interrupt latency.

OverflowDirection

This value returns the overflow direction. The following values are possible:

Value	Description
TIP119_OVERFLOW_UP	Counter Overflow from 0xFFFF to 0x0000 (up).
TIP119_OVERFLOW_DOWN	Counter Overflow from 0x0000 to 0xFFFF (down).

EXAMPLE

```
#include “tip119.h”

int  fd;
int  result;
TIP119_WAIT_STRUCT WaitStruct;

/*
** Wait at least 100 system ticks for a Hardware-Counter Overflow
** on Channel 1
*/
WaitStruct.Channel = 1;
WaitStruct.Timeout = 100;

result = ioctl(fd, TIP119_IOCX_WAITFOROVERFLOWHW, &WaitStruct);

if (result >= 0) {
  printf(“Overflow Interrupt (HW) occurred.\n”);
  printf(“Read Counter = 0x%08lX\n”, WaitStruct.CounterValue);
  printf(“Overflow Direction: %s\n”,
       (WaitStruct.OverflowDirection==TIP119_OVERFLOW_UP)?”UP”:”DOWN”);
} else {
  /* handle ioctl error */
}
```

ERRORS

Error code	Description
EFAULT	Error copying data to or from user space.
EINVAL	Invalid channel specified.
ETIME	Timeout happened before Interrupt.
EBUSY	There is already a pending job waiting for this interrupt on this channel.

4 Debugging

For debugging output see tip119.c. You will find the following symbol:

```
#undef TIP119_DEBUG_VIEW
```

To enable a debug output replace “undef” with “define”.

The TIP119_DEBUG_VIEW symbol controls debugging output from the whole driver.

You can retrieve these messages from the /proc file system using the following command:

 # cat /proc/kmsg

```
TIP119 - Quadrature Decoder Counter - version 1.0.0 (2007-07-06)<6>
TIP119: Probe new TIP119 mounted on <TEWS TECHNOLOGIES - (Compact)PCI IPAC
Carrier> at slot A

TIP119: IP I/O Memory Space
00000000 : FF 7B 00 00 00 00 00 00 00 00 00 00 FF FF FF FF
00000010 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```