
TIP115-SW-82

Linux Device Driver

5 Channel SSI Interface with ,Listen Only' Mode

Version 1.0.x

User Manual

Issue 1.0.0

May 2005

TIP115-SW-82

5 Channel SSI Interface with ,Listen Only' Mode

Linux Device Driver

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 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First issue	May 11, 2005

Table of Content

1	INTRODUCTION.....	4
2	INSTALLATION.....	5
	2.1 Build and install the device driver.....	5
	2.2 Uninstall the device driver	6
	2.3 Install device driver into the running kernel	6
	2.4 Remove device driver from the running kernel	7
	2.5 Change Major Device Number	7
3	DEVICE INPUT/OUTPUT FUNCTIONS	8
	3.1 open()	8
	3.2 close().....	10
	3.3 ioctl()	11
	3.3.1 T115_IOCTL_READ_DATA	13
	3.3.2 T115_IOCTL_READ_ALL_DATA	15
	3.3.3 T115_IOCTL_CONFIG	17

1 Introduction

The TIP115-SW-82 Linux device driver allows the operation of a TIP115 IPAC module on Linux operating systems.

Because the TIP115 device driver is stacked on the TEWS TECHNOLOGIES IPAC carrier driver, it's necessary to install also the appropriate IPAC carrier driver. Please refer to the IPAC carrier driver user manual for further information.

The TIP115 device driver includes the following features:

- Start transmission, wait, and read value on a specified channel
- Start transmission, wait, and read values on all channels (except channels in 'listen only' mode)
- Configuration of channels

2 Installation

The directory TIP115-SW-82 on the distribution media contains the following files:

TIP115-SW-82.pdf	This manual in PDF format
TIP115-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Information about the Device Driver Release

The GZIP compressed archive TIP115-SW-82.tar.gz contains the following files and directories:

tip115/tip115.c	Driver source code
tip115/tip115def.h	Driver include file
tip115/tip115.h	Driver include file for application program
tip115/tpmodule.c	Driver independent library
tip115/tpmodule.h	Driver independent library header file
tip115/makenode	Script to create device nodes on the file system
tip115/Makefile	Device driver make file
tip115/example/tip115example.c	Example application
tip115/example/Makefile	Example application make file

In order to perform an installation, extract all files of the archive TIP115-SW-82.tar.gz to the desired target 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 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

- 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 tip115drv
```

- 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 the new device file system (devfs) 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 TIP115 module found. The first TIP115 can be accessed with device node */dev/tip115_0*, the second TIP115 or the second channel of the first TIP115 with device node */dev/tip115_1* and so on.

The allocation of device nodes to physical TIP115 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP115 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 tip115drv -r
```

If your kernel has enabled devfs, all /dev/tip115_... 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 "*tip115drv: 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 TIP115 driver use 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 TIP115_MAJOR.

To change the major number edit the file tip115drv.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP115_MAJOR Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP115_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/tip115_0", O_RDWR);
```

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 *tip115.h*:

Symbol	Meaning
<i>T115_IOCTL_READ_DATA</i>	Execute a read out value from specified channel
<i>T115_IOC_READ_ALL_DATA</i>	Execute a synchronous read out on all channels (except channel in 'listen only' mode)
<i>T115_IOC_CONFIG</i>	Configure specified channel

See behind for more detailed information on each control code.

To use these TIP115 specific control codes the header file tip115.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 TIP115 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 T115_IOCTL_READ_DATA

NAME

T115_IOCTL_READ_DATA – Start transmission and read input value

DESCRIPTION

This ioctl function starts a transmission and reads the actual value from the TIP115 associated with the open file descriptor, *filedes* and returns the value in the parameter buffer pointed to by *argp*.

The function will start a transmission if the TIP115 channel is in normal SSI mode, if it is in 'listen only' mode, the function will wait until a data transmission is detected.

The parameter buffer (*T115_DATA_BUFFER*) has the following layout:

```
typedef struct
{
    int      channel;
    long     data;
    long     timeout;
} T115_DATA_BUFFER, *PT115_DATA_BUFFER;
```

channel

This parameter specifies the channel number. Allowed values are 0 up to 4.

data

This parameter returns the value read via SSI interface.

timeout

This parameter specifies the maximum time to wait for data. The timeout time is specified in ticks.

EXAMPLE

```
int fd;
int result;
T102_DATA_BUFFER dataBuf;

...

/* Read value from of channel 2, timeout after 1500 ticks */
dataBuf.channel = 2;
dataBuf.timeout = 1500;

result = ioctl(fd, T115_IOCTL_READ_DATA, &dataBuf);
if (result >= 0)
{
    /* OK */
    printf("Value: %ld\n", dataBuf.data);
}
else
{
    /* ERROR */
}

...
```

ERRORS

EFAULT	Invalid pointer to the read buffer. Invalid channel number specified.
ETIME	The function call has timed out.
EBUSY	Global read is active.
EIO	A parity or read error occurred.

SEE ALSO

ioctl man pages

3.3.2 T115_IOC_READ_ALL_DATA

NAME

T115_IOC_READ_ALL_DATA – Start transmission on all channels

DESCRIPTION

This ioctl function starts a transmission and reads the actual value from all channels of the TIP115 associated with the open file descriptor, *filedes* and returns the values in the parameter buffer pointed to by *argp*.

A valid value will be returned for all channels which are configured and not in 'listen only' mode.

The parameter buffer (*T115_ALL_DATA_BUFFER*) has the following layout:

```
typedef struct
{
    long    timeout;
    long    data[5];
    int     status[5];
} T115_ALL_DATA_BUFFER, *PT115_ALL_DATA_BUFFER;
```

timeout

This parameter specifies the maximum time to wait for data. The timeout time is specified in ticks.

data[]

In this array the values read via SSI interfaces will be returned. The index of the array specified the channel number.

status[]

In this array the status of data transmission will be returned. The index of the array specified the channel number. The status can be one of the following values (defined in tip115.h):

Status	Description
<i>T115_STATUS_ERROR</i>	An read or parity error occurred
<i>T115_STATUS_IDLE</i>	Internal use
<i>T115_STATUS_OK</i>	The data returned in <i>data[]</i> is valid.
<i>T115_STATUS_LISTEN_ONLY</i>	No data read, channel is in 'listen only' mode
<i>T115_STATUS_DISABLED</i>	No data read, channel is off (not configured)

EXAMPLE

```
int fd;
int result;
T115_ALL_DATA_BUFFER allBuf;

...

/* Read values of all channels, timeout after 100 ticks */
allBuf.timeout = 100;

result = ioctl(fd, T115_IOCTL_READ_ALL_DATA, &allBuf);
if (result >= 0)
{
    /* OK */
    printf("[0] Val: %ld - Stat: %d\n", allBuf.data[0], allBuf.status[0]);
    printf("[1] Val: %ld - Stat: %d\n", allBuf.data[1], allBuf.status[1]);
    ...
}
else
{
    /* ERROR */
}

...
```

ERRORS

EFAULT	Invalid pointer to the read buffer.
ETIME	The function call has timed out.
EBUSY	A transmission is active.

SEE ALSO

ioctl man pages

3.3.3 T115_IOCS_CONFIG

NAME

T115_IOCS_CONFIG – Configure channel

DESCRIPTION

This ioctl function configure a specified channel of the TIP115 associated with the open file descriptor, *filedes*. The parameters are supplied in the parameter buffer pointed to by *argp*.

The parameter buffer (*T115_CONFIG_BUFFER*) has the following layout:

```
typedef struct
{
    int          channel;
    unsigned long flags;
    int          clockRate;
    int          dataBits;
} T115_CONFIG_BUFFER, *PT115_CONFIG_BUFFER;
```

channel

This parameter specifies the channel number. Allowed values are 0 up to 4.

flags

This parameter is an ORed value of flags specifying the configuration of the channel. Following value are defined.

Value	Description
T115_CONFIG_F_NORMAL_MODE	Selects normal SSI mode.
T115_CONFIG_F_LISTEN_ONLY_MODE	Selects 'listen only' mode.
T115_CONFIG_F_BINARY_CODING	Select binary interpretation for input data stream.
T115_CONFIG_F_GRAY_CODING	Select gray interpretation for input data stream.
T115_CONFIG_F_ZEROBIT_1	1 clock cycle for zero bits.
T115_CONFIG_F_ZEROBIT_2	1 clock cycle for zero bits.
T115_CONFIG_F_PARITY_OFF	Parity checking is off.
T115_CONFIG_F_PARITY_ON	Parity checking is on.
T115_CONFIG_F_PARITY_EVEN	Check on even parity, only valid if parity checking is enabled.
T115_CONFIG_F_PARITY_ODD	Check on odd parity, only valid if parity checking is enabled.

clockRate

This value specifies the clock rate in normal SSI mode. In 'listen only' mode this value will be ignored. Valid values for this parameter are 1 to 15, specifying the clockRate in steps of 1µsec, and 0 if the channel shall be disabled.

dataBits

This value specified the number of valid data bits in the value. Valid values for this parameter are 1 up to 32.

EXAMPLE

```
int fd;
int result;
T115_CONFIG_BUFFER confBuf;

...

/* Configur channel 3: */
/* 24 bit, 5µsec, graycode, no parity, 1 zero bit, SSI mode */
confBuf.channel = 3;
confBuf.dataBits = 24;
confBuf.clockRate = 5;
confBuf.flags = T115_CONFIG_F_PARITY_OFF |
                T115_CONFIG_F_NORMAL_MODE |
                T115_CONFIG_F_ZEROBIT_1 |
                T115_CONFIG_F_GRAY_CODING;

result = ioctl(fd, T115_IOCTL_CONFIG, &confBuf);
if (result >= 0)
{
    /* OK */
    ...
}
else
{
    /* ERROR */
}

...
```

ERRORS

EFAULT

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

SEE ALSO

ioctl man pages