
TIP700-SW-82

Linux Device Driver

Digital Output 24V DC

Version 1.0.x

User Manual

Issue 1.0

June 2003

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7
Phone: +49-(0)4101-4058-0
e-mail: info@tews.com

25469 Halstenbek / Germany
Fax: +49-(0)4101-4058-19
www.tews.com

TEWS TECHNOLOGIES LLC

1 E. Liberty Street, Sixth Floor
Phone: +1 (775) 686 6077
e-mail: usasales@tews.com

Reno, Nevada 89504 / USA
Fax: +1 (775) 686 6024
www.tews.com

TIP700-SW-82

Digital Output 24V DC

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.

©2003 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	July 10, 2003

Table of Contents

1	INTRODUCTION.....	4
2	INSTALLATION.....	5
	2.1 Build and install the device driver.....	5
	2.2 Uninstall the device driver	5
	2.3 Install device driver into the running kernel	6
	2.4 Remove device driver from the running kernel	6
	2.5 Change Major Device Number	7
3	DEVICE INPUT/OUTPUT FUNCTIONS	8
	3.1 open()	8
	3.2 close().....	9
	3.3 write()	10
	3.4 ioctl()	12
	3.4.1 T700_IOC_ENABLE_WD	14
	3.4.2 T700_IOC_DISABLE_WD	15
	3.4.3 T700_IOC_TRIGGER_WD	16

1 Introduction

The TIP700-SW-82 Linux device driver allows the operation of a TIP700 IPAC module on Linux operating systems with kernel version 2.4.4 or higher installed.

Because the TIP700 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 TIP700 device driver includes the following features:

- writing digital output value
- enable and disable output watchdog

2 Installation

The software is delivered on a PC formatted 3½" HD diskette.

The directory A:\TIP700-SW-82 contains the following files:

TIP700-SW-82.pdf	This manual in PDF format
TIP700-SW-82.tar.gz	GZIP compressed archive with driver source code

The GZIP compressed archive TIP700-SW-82.tar.gz contains the following files and directories:

tip700/tip700drv.c	Driver source code
tip700/tip700def.h	Driver include file
tip700/tip700.h	Driver include file for application program
tip700/makenode	Script to create device nodes on the file system
tip700/makefile	Device driver make file
tip700/example/example.c	Example application
tip700/example/makefile	Example application make file

In order to perform an installation, extract all files of the archive TIP700-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 A:\CARRIER-SW-82 on the distribution diskette.

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 tip700drv
```

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.

```
# sh makenode
```

On success the device driver will create a minor device for each TIP700 module found. The first TIP700 can be accessed with device node `/dev/tip700_0`, the second TIP700 with device node `/dev/tip700_1`, the third TIP700 with device node `/dev/tip700_2` and so on.

The allocation of device nodes to physical TIP700 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP700 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 tip700drv -r
```

Be sure that the driver isn't opened by any application program. If opened you will get the response "*tip700drv: 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 TIP700 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.

To change the major number edit the file `tip700drv.c`, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP700_MAJOR Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP700_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/tip700_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

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 write()

NAME

write() – write to a device

SYNOPSIS

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

```
ssize_t write(int filedes, void *buffer, size_t size)
```

DESCRIPTION

This function attempts to write to the output registers of the TIP700 associated with the file descriptor *filedes* from a structure (*T700_BUFFER*) pointed by *buffer*. The argument *size* specifies the length of the buffer and must be set to the length of the structure *T700_BUFFER*.

The *T700_BUFFER* structure has the following layout:

```
typedef struct {
    unsigned short    value;
} T700_BUFFER;
```

unsigned short value

Holds the new value for output lines 1 to 16. Where bit 2^0 corresponds to output line 1, bit 2^1 to output line 2, and so on.

EXAMPLE

```
int fd;
ssize_t NumBytes;
T700_BUFFER ioBuf;

ioBuf.value = 0x1234;

NumBytes = write(fd, &ioBuf, sizeof(T700_BUFFER));

if (NumBytes != sizeof(T700_BUFFER)) {
    // process error;
}
```

RETURNS

On success write returns the size of bytes written (always the size of *T700_BUFFER*). In the case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL This error code is returned if the size of the buffer is wrong.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.4 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 *TIP700.h*:

Symbol	Meaning
<i>T700_IOC_ENABLE_WD</i>	Enable output watchdog
<i>T700_IOC_DISABLE_WD</i>	Disable output watchdog
<i>T700_IOC_TRIGGER_WD</i>	Trigger output watchdog

See behind for more detailed information on each control code.

To use these TIP700 specific control codes the header file TIP700.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 TIP700 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.4.1 T700_IOC_ENABLE_WD

NAME

T700_IOC_ENABLE_WD - Enable output watchdog

DESCRIPTION

This ioctl function enables the output watchdog facility of the TIP700. If the output watchdog is not retriggered within approximately 120 milliseconds the state of the output register will be set to inactive. The watchdog is triggered implicitly by a write to the output register or by executing the ioctl function *T700_IOC_TRIGGER_WD*.

The initialization state (driver startup) of the output watchdog is disabled.

EXAMPLE

```
int fd;
int result;

result = ioctl(fd, T700_IOC_ENABLE_WD);

if (result < 0) {
    /* handle ioctl error */
}
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages

3.4.2 T700_IOC_DISABLE_WD

NAME

T700_IOC_DISABLE_WD - Disable output watchdog

DESCRIPTION

This ioctl function disables the output watchdog facility of the TIP700.

The initialization state (driver startup) of the output watchdog is disabled.

EXAMPLE

```
int fd;
int result;

result = ioctl(fd, T700_IOC_DISABLE_WD);

if (result < 0) {
    /* handle ioctl error */
}
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages

3.4.3 T700_IOC_TRIGGER_WD

NAME

T700_IOC_TRIGGER_WD - Trigger output watchdog

DESCRIPTION

This ioctl function triggers the output watchdog of the TIP700. If the output watchdog is not retriggered within approximately 120 milliseconds the state of the output register will be set to inactive. The watchdog is triggered implicitly by a write to the output register or by executing this ioctl function.

The initialization state (driver startup) of the output watchdog is disabled.

EXAMPLE

```
int fd;
int result;

result = ioctl(fd, T700_IOC_TRIGGER_WD);

if (result < 0) {
    /* handle ioctl error */
}
```

ERRORS

This ioctl function returns no function specific error codes.

SEE ALSO

ioctl man pages