

TIP551-SW-82

Linux Device Driver

4 Channel 16-Bit DAC

Version 1.1.x

User Manual

Issue 1.1.3

September 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

TIP551-SW-82

Linux Device Driver

4 Channel 16-Bit DAC

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.

©2003-2008 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	January 30, 2003
1.1	Support for DEVFS and SMP	September 25, 2003
1.1.1	Introduction and installation modified	April 06, 2006
1.1.2	New Address TEWS TECHNOLOGIES LLC, ChangeLog.txt added	January 2, 2007
1.1.3	Carrier Driver description added	September 25, 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.....	6
2.2	Uninstall the device driver	7
2.3	Install device driver into the running kernel	7
2.4	Remove device driver from the running kernel	8
2.5	Change Major Device Number	8
3	DEVICE INPUT/OUTPUT FUNCTIONS	9
3.1	open()	9
3.2	close().....	11
3.3	write()	12
3.4	ioctl()	15
3.4.1	T551_IOCTL_INFO	16

1 Introduction

1.1 Device Driver

The TIP551-SW-82 Linux device driver allows the operation of TIP551 IPAC modules on Linux operating systems.

Because the TIP551 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 TIP551 device driver includes the following features:

- writing new values to a specified DAC channel
- writing new values to multiple DAC channels with simultaneous update
- reading device information data

The TIP551-SW-82 supports the modules listed below:

TIP551-10	Optically isolated 4 Channel 16 bit D/A, 0V to +10V or +/-10V Output Voltage Range	IndustryPack® compatible
-----------	---	--------------------------

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

TIP551 User manual
TIP551 Engineering Manual
CARRIER-SW-82 IPAC Carrier 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 TIP551-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

The directory TIP551-SW-82 on the distribution media contains the following files:

TIP551-SW-82-1.1.3.pdf	This manual in PDF format
TIP551-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
ChangeLog.txt	Release history
Release.txt	Release information

The GZIP compressed archive TIP551-SW-82-SRC.tar.gz contains the following files and directories:

tip551/tip551.c	Driver source code
tip551/tip551def.h	Driver include file
tip551/tip551.h	Driver include file for application program
tip551/makenode	Script to create device nodes on the file system
tip551/Makefile	Device driver make file
tip551/example/tip551exa.c	Example application
tip551/example/Makefile	Example application make file
tip551/include/tpmodule.h	Kernel independent library header file
tip551/include/tpmodule.c	Kernel independent library source code file

In order to perform an installation, extract all files of the archive TIP551-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzf TIP551-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
Copy file *tip551.h* to */usr/include* to allow user application access
- 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 tip551drv
```

- 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 TIP551 module found. The first TIP551 can be accessed with device node */dev/tip551_0*, the second TIP551 with device node */dev/tip551_1*, the third TIP551 with device node */dev/tip551_2* and so on.

The allocation of device nodes to physical TIP551 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP551 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 tip551drv -r
```

If your kernel has enabled devfs or sysfs (udev), all /dev/tip551_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 "*tip551drv: 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 TIP551 driver use 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. If the kernel has enabled devfs the driver will not use the symbol TIP551_MAJOR.

To change the major number edit the file tip551.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP551_MAJOR Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP551_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/tip551_0", O_RDWR);
if (fd < 0) {
    /* handle open error */
}
```

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

<code>ENODEV</code>	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 write()

NAME

write() – write to a device

SYNOPSIS

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

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

DESCRIPTION

This function attempts to write to the specified DAC channel(s) of the TIP551 associated with the file descriptor *fildes* from a structure (*T551_WRITE_BUFFER*) pointed by *buffer*. The argument *size* specifies the length of the write buffer.

```
typedef struct
{
    int    chan;
    int    corr;
    int    data;
} T551_WRITE_BUFFER;
```

The write function also supports the simultaneous update feature of the TIP551. It's possible to update the DAC outputs of up to 4 channels with one write. In this case the internal DAC data registers of the specified DAC channels will be loaded with new data and then the data conversion is started simultaneously at all DAC channels.

Using this mode requires an array of the structure *T551_WRITE_BUFFER* with a length of the number of channels to update. Each array item must be setup to appropriate values for the DAC channel (*chan*), data correction (*corr*) and the new DAC value (*data*). The order of channels in this array can be accidental. To tell the driver that this is a multi channels write, the write function argument *size* must be the set to the length of the array (see also the example below).

chan

Selects the DAC channel. Valid channel numbers are 1...4.

corr

Set this parameter to TRUE (1) to perform an automatic gain and offset correction with calibration data stored in the IDPROM. FALSE (0) means do not perform any correction and write directly to the DAC output.

data

Contains the new DAC output value.

Output Mode	data range	voltage range
Unipolar	0...65535	0...10V
Bipolar	-32768...32767	-10V...10V

EXAMPLE

```
#include <tip551.h>

int fd;
ssize_t NumBytes;
T551_WRITE_BUFFER wrBuf;
T551_WRITE_BUFFER wrBufArray[T551_MAX_CHAN];

wrBuf.chan = 1;
wrBuf.corr = TRUE;           /* data correction */
wrBuf.data = 65535;         /* full-scale if unipolar output */

NumBytes = write(fd, &wrBuf, sizeof(T551_WRITE_BUFFER));

if (NumBytes != sizeof(T551_WRITE_BUFFER)) {
    // process error;
}

/*
** Update channel 1, 3 and 4 with new data and let channel 2
** unchanged.
*/
wrBufArray[0].chan = 3;
wrBufArray[0].corr = TRUE;   /* data correction */
wrBufArray[0].data = -32768; /* -full-scale if bipolar output */

wrBufArray[1].chan = 1;
wrBufArray[1].corr = TRUE;   /* data correction */
wrBufArray[1].data = 0;      /* mid-scale if bipolar output */

wrBufArray[2].chan = 4;
wrBufArray[2].corr = TRUE;   /* data correction */
wrBufArray[2].data = 32767;  /* +full-scale if bipolar output */
```

```
/*
** Please note the size of the write buffer array is 3 times the
** size of structure T551_WRITE_BUFFER
*/
NumBytes = write(fd, &wrBufArray[0], 3 * sizeof(T551_WRITE_BUFFER));

if (NumBytes != (3 * sizeof(T551_WRITE_BUFFER))) {
    // process error;
}
```

RETURNS

On success write returns the value of the write argument size. 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 too small or if the specified channel number is out of range.
ETIME	Timeout occurred during conversion.

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 fildes, int request [, void *argp])
```

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *fildes*, 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 *tip551.h*:

Symbol	Meaning
T551_IOCTLG_INFO	Read device information data

See behind for more detailed information on each control code.

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 <i>request</i> .
--------	--

Other function dependant error codes will be described for each ioctl code separately. Note, the TIP551 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.4.1 T551_IOCG_INFO

NAME

T551_IOCG_INFO - Read device information data

DESCRIPTION

This ioctl function reads the module variant, the current output voltage range and the factory calibration data from the specified device and returns this information in the *T551_INFO_BUFFER* structure to the caller.

A pointer to the *T551_INFO_BUFFER* structure is passed by the parameter *argp* to the driver.

typedef struct

```
{
    int    variant;
    int    voltage_range;
    short  offset_corr[T551_MAX_CHAN];
    short  gain_corr[T551_MAX_CHAN];
} T551_INFO_BUFFER;
```

variant

Returns the module variant.

value	module variant
10	TIP551-10

voltage_range

Returns the actual output voltage range for all channels.

value	symbol	voltage range
1	T551_UNIPOLAR	0V ... 10V
2	T551_BIPOLAR	-10V ... 10V

offset_corr

Returns the factory offset calibration data for channel 1...4 in the unit ¼ LSB.

gain_corr

Returns the factory gain calibration data for channel 1...4 in the unit ¼ LSB.

EXAMPLE

```
#include <tip551.h>

int fd;
int result;
T551_INFO_BUFFER infoBuf;

result = ioctl(fd, T551_IOCTL_INFO, &infoBuf);

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

ERRORS

EFAULT

Invalid pointer to the T551_INFO_BUFFER. Please check the argument *argp*.

SEE ALSO

ioctl man pages