The Embedded I/O Company

TEWS TECHNOLOGIES

TIP550-SW-82

Linux Device Driver

8(4) Channel 12-Bit DAC

Version 1.2.x

User Manual

Issue 1.2.2

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

TIP550-SW-82

Linux Device Driver

8(4) Channel 12-Bit DAC

Supported Modules:
TIP550

Issue	Description	Date
1.0	First Issue	January 24, 2003
1.1	Support for DEVFS and SMP	September 18, 2003
1.2	missing sign for bipolar output value	December 8, 2003
1.2.0	New Address TEWS TECHNOLOGIES LLC, ChangeLog.txt	January 3, 2006
1.2.1	Description added for dynamic device filesystems, file list modified	April 25, 2008
1.2.2	Carrier Driver description added	July 7, 2008

Table of Contents

1 Introduction

1.1 Device Driver

The TIP550-SW-82 Linux device driver allows the operation of a TIP550 IPAC module on Linux operating systems.

Because the TIP550 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 TIP550 device driver includes the following features:

- writing new values to the specified DAC channel
- reading device information data
- automatic offset and gain correction with factory calibration data
- TEWS TECHNOLOGIES IPAC carrier driver support.

The TIP550-SW-82 supports the modules listed below:

TIP550-10	Optically isolated 8 channel 12 bit DAC	IndustryPack® compatible
TIP550-11	Optically isolated 4 channel 12 bit DAC	IndustryPack® compatible

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

TIP550 User manual

TIP550 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 TIP550-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 TIP550-SW-82 on the distribution media contains the following files:

TIP550-SW-82-1.2.2.pdf	This manual in PDF format
TIP550-SW-82-SRC.tar.gz	Device Driver and Example sources
ChangeLog.txt	Release history
Release.txt	Release information

The GZIP compressed archive TIP550-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tip550/':

tip550.c	Driver source code
tip550def.h	Driver include file
tip550.h	Driver include file for application program
makenode	Script to create device nodes on the file system
Makefile	Device driver make file
example/tip550exa.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 TIP550-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TIP550-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 *tip550.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

- 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 tip550drv

- 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 dynamic 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 TIP550 module found. The first TIP550 can be accessed with device node /dev/tip550_0, the second TIP550 with device node /dev/tip550_1, the third TIP550 with device node /dev/tip550_2 and so on.

The allocation of device nodes to physical TIP550 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP550 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 tip550drv –r

If your kernel has enabled a dynamic device file system like devfs or sysfs (udev), all /dev/tip550_... 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 "*tip550drv: 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 TIP550 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 TIP550_MAJOR.

To change the major number edit the file tip550drv.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP550_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP550_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/tip550_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

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 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 specified DAC channel of the TIP550 associated with the file descriptor filedes from a structure (*T550_WRITE_BUFFER*) pointed by buffer. The argument size specifies the length of the buffer and must be set to the length of the structure *T550_WRITE_BUFFER*.

```
typedef struct
{
        int     chan;
        int     corr;
        int     data;
} T550_WRITE_BUFFER;
```

chan

Selects the DAC channel. Valid channel numbers are 1...8 for TIP550-10 and channel numbers 1...4 for TIP550-11.

corr

Set this parameter to TRUE (1) to perform an automatic gain and offset correction with calibration data stored in the IDPROM. FLASE (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…4095	0…10V
Bipolar	-2048…2047	-10V…10V

EXAMPLE

```
#include <tip550.h>

int  fd;
ssize_t  NumBytes;
T550_WRITE_BUFFER  wrBuf;


wrBuf.chan  = 1;
wrBuf.corr  = 0;     /* no data correction */
wrBuf.data  = 4095;  /* full-scale for unipolar output */

NumBytes = write(fd, &wrBuf, sizeof(T550_WRITE_BUFFER));

if (NumBytes != sizeof(T550_WRITE_BUFFER)) {
  // process error;
}
```

RETURNS

On success write returns the size of *T550_WRITE_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 too small or if the specified channel number is out of range.
EACCESS	The output voltage mode isn't configured until now. Please call the ioctl function *T550_IOCS_CONFIG* before.
ETIME	Timeout occurred during conversion.

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 *tip550.h*:

Symbol	Meaning
T550_IOCS_CONFIG	Configure output voltage mode
T550_IOCG_INFO	Read device information data

See behind for more detailed information on each control code.

To use these TIP550 specific control codes the header file tip550.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 TIP550 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.4.1 T550_IOCS_CONFIG

NAME

T550_IOCS_CONFIG - Configure output voltage mode

DESCRIPTION

This ioctl function must be called to configure the output voltage mode for channel group 1 and 2 to match the hardware configuration of the device (jumper J1-4). Be sure that this ioctl function is called before the first write; otherwise write will fail (EACCES).

A pointer to the *T550_CONFIG_BUFFER* structure is passed by the parameter *argp* to the driver.

```
typedef struct
{
    int     output_mode_1;
    int     output_mode_2;
} T550_CONFIG_BUFFER;
```

output_mode_1, output_mode_2

These structure elements specifies the output voltage mode for channel group 1 (channel 1...4) and channel group 2 (channel 5...8). For TIP550-11 devices the channel group 2 isn't relevant (only 4 channels at all) and should be set to 0.

Each channel group can be configured separately either for unipolar (*T550_UNIPOLAR*) or bipolar (*T550_BIPOLAR*) output.

Symbol	**Voltage Range**
T550_UNIPOLAR	0…10V
T550_BIPOLAR	-10V…10V

EXAMPLE

```
#include <tip550.h>

int  fd;
int  result;
T550_CONFIG_BUFFER  confBuf;


confBuf.output_mode_1 = T550_UNIPOLAR;
confBuf.output_mode_2 = T550_BIPOLAR;

result = ioctl(fd, T550_IOCS_CONFIG, &confBuf);

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

ERRORS

EFAULT	Invalid argument pointer. Please check the argument *argp*.
EINVAL	Invalid output voltage mode.

SEE ALSO

ioctl man pages

3.4.2 T550_IOCG_INFO

NAME

T550_IOCG_INFO - Read device information data

DESCRIPTION

This ioctl function reads the module variant, the current output voltage setting and the factory calibration data from the specified device and returns this information in the *T550_INFO_BUFFER* structure to the caller.

A pointer to the *T550_INFO_BUFFER* structure is passed by the parameter *argp* to the driver.

```
typedef struct
{
        int     variant;
        int     output_mode_1;
        int     output_mode_2;
        int     offset_corr[8];
        int     gain_corr[8];
} T550_INFO_BUFFER;
```

variant

Returns the module variant.

Value	**Module Variant**
10	TIP550-10
11	TIP550-11

output_mode_1

Returns the actual output voltage mode for channel group 1 (channel 1...4).

Value	**Symbol**	**Voltage Range**
1	T550_UNIPOLAR	0…10V
2	T550_BIPOLAR	-10V…10V

output_mode_2

Returns the actual output voltage mode for channel group 2 (channel 5...8). For TIP550-11 modules -1 is returned.

Value	**Symbol**	**Voltage Range**
1	T550_UNIPOLAR	0…10V
2	T550_BIPOLAR	-10V…10V

offset_corr

Returns the factory offset calibration data for channel 1...8 in the unit ¼ LSB. For TIP550-11 modules only the values for channel 1..4 are valid.

gain_corr

Returns the factory gain calibration data for channel 1...8 in the unit ¼ LSB. For TIP550-11 modules only the values for channel 1..4 are valid.

EXAMPLE

```
#include <tip550.h>

int  fd;
int  result;
T550_INFO_BUFFER  infoBuf;


result = ioctl(fd, T550_IOCG_INFO, &infoBuf);

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

ERRORS

EFAULT	Invalid pointer to the *T550_INFO_BUFFER*. Please check the argument *argp.*

SEE ALSO

ioctl man pages