The Embedded I/O Company

TEWS TECHNOLOGIES

TIP500-SW-82

Linux Device Driver

Optically Isolated 16 Channel 12 Bit ADC

Version 1.2.x

User Manual

Issue 1.2.1

February 2009

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

TIP500-SW-82

Linux Device Driver

Optically Isolated 16 Channel 12 Bit ADC

Supported Modules:
TIP500

Issue	Description	Date
1.0	First Issue	July 15, 2001
1.1	Support for CARRIER CLASS DRIVER, DEVFS and SMP	September 17, 2003
1.2.0	General Revision	November 13, 2006
1.2.1	Introduction for IPAC Carrier Support	February 2, 2009

Table of Contents

1 Introduction

1.1 Device Driver

The TIP500-SW-82 Linux device driver allows the operation of a TIP500 IPAC module on Linux operating systems.

Because the TIP500 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 TIP500-SW-82 device driver supports the following features:

- reading analog input values from specified channels
- single-ended and differential input mode
- automatic offset and gain correction with factory calibration data
- obtaining module variant and factory calibration data out of the IDPROM

The TIP500-SW-82 device driver supports the modules listed below:

TIP500-10	Optically isolated 16 channel 16 bit ADC input voltage range +/-10V, gain 1, 2, 5, 10	IndustryPack® compatible
TIP500-11	Optically isolated 16 channel 16 bit ADC input voltage range +/-10V, gain 1, 2, 4, 8	IndustryPack® compatible
TIP500-20	Optically isolated 16 channel 16 bit ADC input voltage range 0V to +10V, gain 1, 2, 5, 10	IndustryPack® compatible
TIP500-21	Optically isolated 16 channel 16 bit ADC input voltage range 0V to +10V, gain 1, 2, 4, 8	IndustryPack® compatible

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

TIP500 User manual

TIP500 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 TIP500-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

Following files are located on the distribution media:

Directory path '.\TIP500-SW-82\':

TIP500-SW-82-1.2.1.pdf	This manual in PDF format
TIP500-SW-82-SRC.tar.gz	Device Driver and Example sources
ChangeLog.txt	Release history
Release.txt	Release information

The GZIP compressed archive TIP500-SW-82-SRC.tar.gz contains the following files and directories:

tip500/tip500.c	Driver source code
tip500/tip500def.h	Driver include file
tip500/tip500.h	Driver include file for application program
tip500/makenode	Script to create device nodes on the file system
tip500/Makefile	Device driver make file
tip500/example/tip500exa.c	Example application
tip500/example/Makefile	Example application make file
tip500/include/config.h	Driver independent library header file
tip500/include/tpmodule.h	Kernel independent library header file
tip500/include/tpmodule.c	Kernel independent library source code file

In order to perform an installation, extract all files of the archive TIP500-SW-82.tar.gz to the desired target directory.

- Login as *root* and change to the target directory
- Copy tip500.h to */usr/include*

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 tip500drv

- 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 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 TIP500 found. The first TIP500 module can be accessed with device node /dev/tip500_0, the second module with device node /dev/tip500_1, the third TIP500 module with device node /dev/tip500_2 and so on.

The allocation of device nodes to physical TIP500 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP500 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 –r tip500drv

If your kernel has enabled a dynamic device file system, all /dev/tip500_x nodes will be removed automatically 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 "*tip500drv: 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 TIP500 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 TIP500_MAJOR.

To change the major number edit the file tip500drv.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP500_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP500_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/tip500_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 */
}
```

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 read()

NAME

read() – read from a device

SYNOPSIS

#include <unistd.h>

#include <tip500.h>

ssize_t read(*int filedes, void *buffer, size_t size*)

DESCRIPTION

The read function attempts to start an AD conversion on the specified channel and return the converted value in a read buffer to the caller.

A pointer to the callers read buffer (*T500_IO_BUFFER*) and the size of this structure is passed by the parameters *buffer* and *size* to the device.

```
typedef struct
{
        unsigned int            channel;
        unsigned int            gain;
        unsigned int            mode;
        unsigned int            correction;
        int                     data;
} T500_IO_BUFFER, *PT500_IO_BUFFER;
```

channel

Specifies the channel number to read the AD value. Valid channel numbers are 1..16 if Single-Ended is selected for, if differential is selected the valid channel numbers are in the range of 1..8.

gain

Specifies the gain, which shall be used to read the AD value. Valid gains are:

T500_GAIN_1	Select Gain 1	Valid for TIP500-10/-11/-20/-21
T500_GAIN_2	Select Gain 2	Valid for TIP500-10/-11/-20/-21
T500_GAIN_4	Select Gain 4	Valid for TIP500-11/-21
T500_GAIN_5	Select Gain 5	Valid for TIP500-10/-20
T500_GAIN_8	Select Gain 8	Valid for TIP500-11/-21
T500_GAIN_10	Select Gain 10	Valid for TIP500-10/-20

mode

Specifies the channel input interface. If it should be used with a differential interface, this member must have the value *T500_DIFF*, otherwise the value should be *T500_SINGLE,* if it should be used with a single-ended input.

correction

If this parameter is *T500_CORR* the driver performs an automatic offset and gain correction with factory calibration data stored in the TIP500 ID-PROM, otherwise the value should be *T500_NOCORR*.

data

Analog input value read from the specified ADC channel. The analog data is returned as sign extended two's complement integer value with 16-bit resolution. The lower four bits are always 0 (see TIP500 Hardware User Manual).

EXAMPLE

```
#include <uinstd.h>
#include <tip500.h>

int fd;
ssize_t  NumBytes;
T500_IO_BUFFER ADCBuf;

ADCBuf.gain       = T500_GAIN_1;
ADCBuf.mode       = T500_SINGLE;
ADCBuf.channel    = 1;
ADCBuf.correction = T500_CORR;

NumBytes = read(fd, &ioBuf, sizeof(ioBuf));
if (NumBytes < 0)
{
     /* process read error */
}
```

RETURNS

On success read returns the size of the structure T500_IO_BUFFER. 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 size of the read buffer is too small or if the gain or channel parameter out of range.
ETIME	Timeout during AD conversion.
EBUSY	AD conversion is in progress. Only relevant if the device is opened more than once.
EFAULT	Invalid pointer to the read buffer.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.4 ioctl()

NAME

ioctl() – device control functions

SYNOPSIS

#include <sys/ioctl.h>
#include <tip500.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 tip500.h:

Symbol	Meaning
T500_IOCG_READ_PARAM	Read module parameter

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 request.

Other function dependant error codes will be described for each ioctl code separately. Note, the TIP500 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.5 T500_IOCG_READ_PARAM

NAME

T500_IOCG_READ_PARAM - Read module parameter

DESCRIPTION

This ioctl function attempts to read the module type and calibration data of the TIP500 associated with the open file descriptor, *filedes*, into the parameter buffer pointed to by *argp.*

The parameter buffer (*T500_PARAM_BUFFER*) has the following layout:

```
typedef struct
{
        unsigned int            ModuleType;
        int                     calGain[4];
        int                     calOffs[4];
} T500_PARAM_BUFFER, *PT500_PARAM_BUFFER;
```

ModuleType

Receives the type code (10/11/20/21) of the associated TIP500.

calGain

Receives the gain error of the input amplifier for four possible gain selections in the unit ¼ LSB (see also Hardware User Manual).

calOffs

Receives the offset (zero) error of the input amplifier for four possible gain selections in the unit ¼ LSB (see also Hardware User Manual).

EXAMPLE

```
include <sys/ioctl.h>
include <tip500.h>

int fd;
int result;
T500_PARAM_BUFFER  ParamBuf;

result = ioctl(fd, T500_IOCG_READ_PARAM, &ParamBuf);
if (result < 0)
{
     /* handle ioctl error */
}
```

ERRORS

EFAULT	Invalid pointer to the read buffer.

SEE ALSO

ioctl man pages