The Embedded I/O Company

TEWS TECHNOLOGIES

TIP501-SW-82

Linux Device Driver

16 Channel 16-Bit ADC

Version 1.2.x

User Manual

Issue 1.2.4

September 2009

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7 25469 Halstenbek, Germany

Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19

e-mail: info@tews.com www.tews.com

TIP501-SW-82

Linux Device Driver

16 Channel 16-Bit ADC

Supported Modules:
TIP501

Issue	Description	Date
1.0	First Issue	January 28, 2003
1.1	Support for DEVFS and SMP	September 18, 2003
1.2.0	Introduction and install description modified	February 13, 2006
1.2.1	New Address TEWS TECHNOLOGIES LLC, ChangeLog.txt	December 15, 2006
1.2.2	Filelist modified	February 07, 2008
1.2.3	Carrier Driver description added	July 7, 2008
1.2.4	Address TEWS LLC removed	September 18, 2009

Table of Contents

1 Introduction

1.1 Device Driver

The TIP501-SW-82 Linux device driver allows the operation of a TIP501 IPAC module on Linux operating systems.

Because the TIP501 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 TIP501-SW-82 device driver includes the following features:

- reading analog input values from specified channels
- standard and pipeline mode
- single-ended and differential input mode
- automatic offset and gain correction with factory calibration data
- TEWS TECHNOLOGIES IPAC carrier driver support.

The TIP501-SW-82 supports the modules listed below:

TIP501-10	Optically isolated 16 channel 16 bit ADC input voltage range +/-10V, gain 1, 2, 5, 10	IndustryPack® compatible
TIP501-11	Optically isolated 16 channel 16 bit ADC input voltage range +/-10V, gain 1, 2, 4, 8	IndustryPack® compatible
TIP501-20	Optically isolated 16 channel 16 bit ADC input voltage range 0V to +10V, gain 1, 2, 5, 10	IndustryPack® compatible
TIP501-21	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.

TIP501 User manual

TIP501 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 TIP501-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 TIP501-SW-82 on the distribution media contains the following files:

TIP501-SW-82-1.2.4.pdf	This manual in PDF format
TIP501-SW-82-SRC.tar.gz	Device Driver and Example sources
ChangeLog.txt	Release history
Release.txt	Release information

The GZIP compressed archive TIP501-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tip501/':

tip501.c	Driver source code
tip501def.h	Driver include file
tip501.h	Driver include file for application program
makenode	Script to create device nodes on the file system
Makefile	Device driver make file
example/tip501exa.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 TIP501-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TIP501-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 *tip501.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 tip501drv

- After the first build or if you are using dynamic major device allocation it is 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 TIP501 module found. The first TIP501 can be accessed with device node /dev/tip501_0, the second TIP501 with device node /dev/tip501_1, the third TIP501 with device node /dev/tip501_2 and so on.

The allocation of device nodes to physical TIP501 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP501 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 tip501drv –r

If your kernel has enabled devfs or sysfs (udev), all /dev/tip501_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 "*tip501drv: 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 TIP501 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 TIP501_MAJOR.

To change the major number edit the file tip501drv.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP501_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP501_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/tip501_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 read()

NAME

read() – read from a device

SYNOPSIS

```
#include <unistd.h>

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

DESCRIPTION

The read function starts an AD conversion at the specified input channel of the TIP501 associated with the file descriptor *fildes*. A pointer to the read buffer structure (*T501_READ_BUFFER*) is passed by read function argument *buffer* to the driver. The argument *size* specifies the length of the read buffer and must be set to the length of the structure *T501_READ_BUFFER*.

Before calling the read function some elements of the read buffer must be set to appropriate values (see below for a detailed description of each element). After successful execution the element *data* returns the converted analog input value as a two's complement integer value.

The *T501_READ_BUFFER* structure has the following layout:

```
typedef struct
{
        int     chan;
        int     gain;
        int     flags;
        int     data;
} T501_READ_BUFFER;
```

chan

Specifies the channel number at which the conversion will be started. Valid channel numbers are 1...16 if single-ended is selected or 1...8 for differential input.

gain

This parameter specifies the gain for the input voltage amplifier. Valid gain constants are listed below:

Definition	Gain	Valid for TIP501 Variant
T501_GAIN_1	1	TIP501-10/-11/-20/-21
T501_GAIN_2	2	TIP501-10/-11/-20/-21
T501_GAIN_4	4	TIP501-11/-21
T501_GAIN_5	5	TIP501-10/-20
T501_GAIN_8	8	TIP501-11/-21
T501_GAIN_10	10	TIP501-10/-20

flags

This bit mask controls the read operation; you create the value by the bitwise OR of the appropriate parameters (using the **|** operator in C).

T501_DIFF	Use differential analog inputs. If this flag is omitted single-ended inputs will be selected.
T501_PIPELINE	Use data pipeline mode. In this mode converted data from the previous conversion will be read and a new conversion is started. If this flag is omitted, data from the new started conversion will be read (see also the TIP501 User Manual for more information).
T501_CORRECTION	Perform an automatic offset and gain correction with factory calibration data stored in the TIP501 ID-PROM. If this flag is omitted the converted data will be read directly.

data

This parameter receives the converted analog input value (two's complement).

EXAMPLE

```
int  fd;
ssize_t  NumBytes;
T501_READ_BUFFER  rdBuf;

rdBuf.chan  = 1;
rdBuf.gain  = T501_GAIN_1;
rdBuf.flags = T501_DIFF | T501_CORRECTION;

NumBytes = read(fd, &rdBuf, sizeof(rdBuf));

if (NumBytes != sizeof(T501_READ_BUFFER)) {
   // process error;
}
```

RETURNS

On success read returns always the size of the structure *T501_READ_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, or the gain or channel number is out of range.
ETIME	Timeout during AD 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 *tip501.h*:

Symbol	**Meaning**
T501_IOCG_INFO	Read device information data

See behind for more detailed information on each control code.

To use these TIP501 specific control codes the header file tip501.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 TIP501 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.4.1 T501_IOCG_INFO

NAME

T501_IOCG_INFO - Read device information data

DESCRIPTION

This ioctl function reads the module variant and the factory calibration data from the specified device and returns this information in the *T501_INFO_BUFFER* structure to the caller.

A pointer to the *T501_INFO_BUFFER* structure is passed by the parameter *argp* to the driver.

The *T501_INFO_BUFFER* structure has the following layout:

```
typedef struct
{
        int     variant;
        short   offset_corr[4];
        short   gain_corr[4];
} T501_INFO_BUFFER;
```

variant

Returns the module variant.

value	module variant
10	TIP501-10
11	TIP501-11
20	TIP501-20
21	TIP501-21

offset_corr

The factory programmed correction data for offset correction is returned in this array. The index of the array specifies the input gain. (See table below)

gain_corr

The factory programmed correction data for gain correction is returned in this array. The index of the array specifies the input gain. (See table below)

Index	Gain (TIP501-10/20)	Gain (TIP501-11/21)
0	1	1
1	2	2
2	5	4
3	10	8

EXAMPLE

```
int  fd;
int  result;
T501_INFO_BUFFER  infoBuf;

result = ioctl(fd, T501_IOCG_INFO, &infoBuf);

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

ERRORS

EFAULT	Invalid argument pointer. Please check the argument argp.

SEE ALSO

ioctl man pages