The Embedded I/O Company

TEWS TECHNOLOGIES

TIP850-SW-82

Linux Device Driver

16 Channel 12 bit A/D + 4 Channel 12 bit D/A

Version 1.0.x

User Manual

Issue 1.0.0

November 2004

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

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Reno, Nevada 89504 / USA
Phone: +1 (775) 686 6077
Fax: +1 (775) 686 6024
e-mail: usasales@tews.com
www.tews.com

TIP850-SW-82

16 Channel 12 bit A/D + 4 Channel 12 bit D/A

Linux Device Driver

Issue	Description	Date
1.0.0	First Issue	November 29, 2004

Table of Contents

1 Introduction

The TIP850-SW-82 Linux device driver allows the operation of a TIP850 IPAC module on Linux operating systems.

Because the TIP850 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 TIP850 device driver includes the following features:

- Reading converted AD values from a specified analog input channel with or without data correction
- Writing DA values to a specified analog output channel with or without data correction
- Reading module type and correction values out of the ID PROM

2 Installation

The directory TIP850-SW-82 on the distribution media contains the following files:

TIP850-SW-82.pdf	This manual in PDF format
TIP850-SW-82.tar.gz	GZIP compressed archive with driver source code

The GZIP compressed archive TIP850-SW-82.tar.gz contains the following files and directories:

tip850/tip850.c	Driver source code
tip850/tip850def.h	Driver include file
tip850/tip850.h	Driver include file for application program
tip850/tpmodule.c	Driver independent library
tip850/tpmodule.h	Driver independent library header file
tip850/makenode	Script to create device nodes on the file system
tip850/Makefile	Device driver make file
tip850/example/example.c	Example application
tip850/example/Makefile	Example application make file

In order to perform an installation, extract all files of the archive TIP850-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 *CARRIER-SW-82* on the 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

- 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 tip850drv

- 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 new device file system (devfs) 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 TIP850 module found. The first TIP850 can be accessed with device node /dev/tip850_0, the second TIP850 or the second channel of the first TIP850 with device node /dev/tip850_1 and so on.

The allocation of device nodes to physical TIP850 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP850 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 tip850drv –r

If your kernel has enabled devfs, all /dev/tip850_... 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 "*tip850drv: 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 TIP850 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 TIP850_MAJOR.

To change the major number edit the file tip850drv.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP850_MAJOR Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP850_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/tip850_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

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

A pointer to the callers read buffer (*T850_READ_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;
        long                timeout;
} T850_READ_BUFFER, *PT850_READ_BUFFER;
```

channel

Specifies the channel number at which 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:

T850_GAIN_1	Select Gain 1x	Valid for TIP850-10/-11
T850_GAIN_10	Select Gain 10x	Valid for TIP850-10
T850_GAIN_100	Select Gain 100x	Valid for TIP850-10
T850_GAIN_2	Select Gain 2x	Valid for TIP850-11
T850_GAIN_4	Select Gain 4x	Valid for TIP850-11
T850_GAIN_8	Select Gain 8x	Valid for TIP850-11

mode

Specifies the channel input interface. If it should be used with a differential interface, this member must have the value *T850_DIFF*, otherwise the value should be *T850_SINGLE,* if it should be used with a single-ended input.

correction

If this parameter is *T850_CORR* the driver performs an automatic offset and gain correction with factory calibration data stored in the TIP850 ID-PROM, otherwise the value should be *T850_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 range is T850_MIN_ADC_VALUE to T850_MAX_ADC_VALUE (See tip850.h). The corresponding voltage value depends on the selected gain. (See TIP850 hardware manual for further details)

timeout

This parameter describes the maximum time the user is willing to wait for an ADC conversion ready event. A value of 0 means wait for ever.

EXAMPLE

```
{
  int fd;
  ssize_t  num_bytes;
  T850_READ_BUFFER r_buf;
  ...
  r_buf.gain       = T850_GAIN_1;
  r_buf.mode       = T850_SINGLE;
  r_buf.channel    = 1;
  r_buf.correction = T850_CORR;
  r_buf.timeout    = 10;

  num_bytes = read(fd, &r_buf, sizeof(r_buf));

  /*
  **  Check the result of the last device I/O operation
  */
  if (num_bytes > 0) {
    printf("ADC Value = %d\n", r_buf.data);
  }
  else {
    printf("Read failed -->  Error = %d\n", errno );
  }

  ...

}
```

RETURNS

On success read returns the size of the structure T850_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	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	The conversion was not completed within the given timeout. The hardware seems to be faulty or the device mapping is incorrect.
EFAULT	Invalid pointer to the read buffer.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.4 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 TIP850 associated with the file descriptor *filedes* from a structure (*T850_WRITE_BUFFER)* pointed by *buffer*. The argument *size* specifies the length of the buffer and must be set to the length of the structure *T850_WRITE_BUFFER*.

The *T850_WRITE_BUFFER* structure has the following layout:

```
typedef struct
{
        int             data[4];
        unsigned int    correction[4];
        unsigned int    refresh[4];
} T850_WRITE_BUFFER, *PT850_WRITE_BUFFER;
```

data

This array parameter contains the output values for all DAC channels. The data for DAC X (0…3) is accessed through data[X].

Output Mode	data range	voltage range
Bipolar	-2048...2047	-10V...10V

correction

Set parameter correction[X] to T850_CORR to perform an automatic gain and offset correction for DAC channel X (0…3) with calibration data stored in the IDPROM. T850_NOCORR means do not perform any correction and write directly to the certain DAC output.

refresh

This array parameter enables or disables the certain DAC channel. To enable updating of DAC channel X (0…3) set refresh[X] to T850_REFRESH otherwise set it to T850_NOREFRESH.

EXAMPLE

```
int fd;
ssize_t  num_bytes;
T850_WRITE_BUFFER  w_buf;

/* write to DAC channel 0, precondition: w_buf is zeroed */
w_buf.data[0]        = 1024;          /* set DAC 0 output to +5V */
w_buf.correction[0]  = T850_NOCORR;   /* no data correction */
w_buf.refresh[0]     = T850_REFRESH;  /* enable channel 0 */

num_bytes = write(fd, &w_buf, sizeof(w_buf));

if (num_bytes != sizeof(w_buf)) {
   // process error;
}
```

RETURNS

On success write returns the size of *T850_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.
EFAULT	The pointer to the user buffer is not valid.
EBUSY	Another concurrent process is writing to the device at the moment.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.5 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 *TIP850.h*:

Symbol	Meaning
T850_IOCG_READ_PARAM	Read module parameter

See behind for more detailed information on each control code.

To use these TIP850 specific control codes the header file TIP850.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 TIP850 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.5.1 T850_IOCG_READ_PARAM

NAME

T850_IOCG_READ_PARAM - Read module parameter

DESCRIPTION

This ioctl function attempts to read the module type and calibration data of the TIP850 associated with the open file descriptor, *filedes*, into the parameter buffer pointed to by *argp.*

The parameter buffer (*T850_PARAM_BUFFER*) has the following layout:

```
typedef struct {
        unsigned int            model;
        unsigned int            revision;
        T850_CAL_BUFFER         dac_cal[4];
        T850_CAL_BUFFER         adc_cal[4];
} T850_PARAM_BUFFER, *PT850_PARAM_BUFFER;


typedef struct
{
        int offset;
        int gain;
} T850_CAL_BUFFER;
```

model

This parameter receives the model code of the associated TIP850. (0x09 = TIP850-10, 0x11 = TIP850-11)

revision

This parameter receives the revision code of the associated TIP850. (0x10 = V 1.0, 0x11 = V1.1+)

dac_cal

This parameter receives four T850_CAL_BUFFER structures for each DAC chip mounted on the certain TIP850. The gain error of the output amplifier for DAC X (0..3) is accessed through dac_cal[X].gain. The offset (zero) error of the output amplifier for DAC X (0..3) is accessed through dac_cal[X].offset. The T850_CAL_BUFFER values are stored in the unit ¼ LSB (see also Hardware User Manual).

adc_cal

This parameter receives four T850_CAL_BUFFER structures for all possible ADC gain stages used on the certain TIP850. The gain error of the input amplifier for T850_GAIN_X is accessed through adc_cal[T850_GAIN_X].gain. The offset (zero) error of the input amplifier for T850_GAIN_X is accessed through adc_cal[T850_GAIN_X].offset. The T850_CAL_BUFFER values are stored in the unit ¼ LSB (see also Hardware User Manual).

EXAMPLE

```
{
  int fd;
  int result;
  T850_PARAM_BUFFER  ParamBuf;

  ...

  result = ioctl(fd, T850_IOCG_READ_PARAM, &ParamBuf);

  /*
  **  Check the result of the last device I/O control operation
  */
  if (result >= 0) {
    printf("Read module parameter successful\n");
  }
  else {
     printf("Read parameter failed --> Error = %d\n", errno);
  }

  ...

}
```

ERRORS

EFAULT	Invalid pointer to the read buffer.

SEE ALSO

ioctl man pages