The Embedded I/O Company

TEWS TECHNOLOGIES

TIP830-SW-82

Linux Device Driver

8 Channel Simultaneous Sampling ADC

Version 1.2.x

User Manual

Issue 1.2.0

July 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

TIP830-SW-82

Linux Device Driver

8 Channel Simultaneous Sampling ADC

Supported Modules:
TIP830

Issue	Description	Date
1.0	First Issue	November 20, 2001
1.1	Support for IPAC Carrier Driver, DEVFS and SMP	February 24, 2004
1.2.0	General revision	July 1, 2009

Table of Contents

1 Introduction

1.1 TIP830 Device Driver

The TIP830-SW-82 Linux device driver allows the operation of the TIP830 8 Channel ADC IndustryPack® modules conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close()*, *read()* and *ioctl()* functions.

Because the TIP830 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 TIP830-SW-82 device driver supports the following features:

- reading converted AD values from the eight input channels with or without data correction
- reading module type and correction data from the boards ID PROM

The TIP830-SW-82 device driver supports the modules listed below:

TIP830-10	8 Channel Simultaneous 12-bit ADC	(IndustryPack®)
TIP830-20	8 Channel Simultaneous 16-bit ADC	(IndustryPack®)

To get more information about the features and use of TIP830 devices it is recommended to read the manuals listed below.

TIP830 User manual

TIP830 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 TIP830-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 '.\TIP830-SW-82\':

TIP830-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
TIP830-SW-82-1.2.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

For installation the files have to be copied to the desired target directory.

The GZIP compressed archive TIP830-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tip830/':

tip830drv.c	TIP830 device driver source
tip830def.h	TIP830 driver include file
tip830.h	TIP830 include file for driver and application
makenode	Script to create device nodes on the file system
Makefile	Device driver make file
example/tip830exa.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 TIP830-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TIP830-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tip830.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 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 tip830drv

- 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 TIP830 module found. The first TIP830 can be accessed with device node /dev/tip830_0, the second TIP830 can be accessed with device node /dev/tip830_1 and so on.

The allocation of device nodes to physical TIP830 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP830 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 is not 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 tip830drv -r

If your kernel has enabled a dynamic device file system all /dev/tip830_x nodes will be automatically removed from your file system after this.

Be sure that the driver is not opened by any application program. If opened you will get the response ´´*tip830drv: 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 TIP830 driver uses dynamic allocation of major device numbers by default. If this is not suitable for the application it is possible to define a major number for the driver. If the kernel has enabled a dynamic device file system the driver will not use the symbol TIP830_MAJOR.

To change the major number edit the file tip830.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP830_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP830_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/tip830_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 <tip830.h>

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

DESCRIPTION

The read function starts an AD conversion and returns the converted values in a read buffer to the caller.

A pointer to the callers read buffer (T830_RW_BUFFER) and the size of this structure is passed by the parameters buffer and size to the device.

```
typedef struct
{
        unsigned int        correction[T830_MAX_CHAN];
        int                 data[T830_MAX_CHAN];
} T830_IO_BUFFER, *PT830_IO_BUFFER;
```

correction

If the corresponding entry of this array contains the value of *T830_CORR* the driver performs an automatic offset and gain correction with factory calibration data stored in the TIP830 ID-PROM, otherwise the value should be *T830_NOCORR*. Index 0 specifies channel 1, index 1 specifies channel 2 and so on.

data

The analog input values read from the ADC channels are returned in this array. The analog data is returned as sign extended two's complement integer value with 16-bit resolution, also if there is a 12bit-ADC if using TIP830-10. Index 0 specifies channel 1, index 1 specifies channel 2 and so on.

EXAMPLE

```
#include      <tip830.h>

int                 fd;
ssize_t             num_bytes;
int                 i;

…
```

…

```
/* use correction for channel 1, 3, 5, 7 */
T830_IO_BUFFER adc_buf =
     {{T830_CORR , 0, T830_CORR , 0, T830_CORR , 0, T830_CORR , 0},
      {,,,,,,,}};

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

/* Check the result of the last device I/O operation */
if (num_bytes > 0) {
     for (i = 0; i < T830_MAX_CHAN; i++)   {
          printf("ADC Ch%d Value = %d\n", i, adc_buf.data[i]);
     }
}
else {
     printf("Read failed -->  Error = %d\n", errno);
}
```

RETURNS

On success read returns the size of the structure T830_IO_BUFFER. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EBUSY	Device busy. This error code is returned if another task or process is still running an AD conversion.
EFAULT	Invalid pointer to the read buffer.
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 50 microseconds. The hardware seems to be faulty or the device mapping is incorrect.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.4 ioctl()

NAME

ioctl() – device control functions

SYNOPSIS

#include <sys/ioctl.h>
#include <tip830.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 tip830.h:

Symbol	Meaning
TIP830_IOCG_READ_PARAM	Read module parameters

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 dependent error codes will be described for each ioctl code separately. Note, the TIP830 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.4.1 T830_IOCG_READ_PARAM

NAME

T830_IOCG_READ_PARAM - Read module parameter

DESCRIPTION

This ioctl function reads the module type and calibration data of the TIP830 associated with the open file descriptor, *filedes*, into the parameter buffer pointed to by *argp.*

The parameter buffer (*T830_PARAM_BUFFER*) has the following layout:

```
typedef struct
{
        unsigned int        module_type;
        int                 cal_gain[T830_MAX_CHAN];
        int                 cal_offs[T830_MAX_CHAN];
} T830_PARAM_BUFFER, *PT830_PARAM_BUFFER;
```

module_type

This parameter receives the type code (10 or 20) of the associated TIP830.

cal_gain

Receives the gain error of the channel specified with the array index in the unit 1 LSB (see also Hardware User Manual). Index 0 specifies the gain error of channel 1, index 1 specifies the gain error of channel 2 and so on.

cal_offs

Receives the offset (zero) error of the channel specified with the array index in the unit 1 LSB (see also Hardware User Manual). Index 0 specifies the offset error of channel 1, index 1 specifies the offset error of channel 2 and so on.

EXAMPLE

```
#include      <tip830.h>

int               fd;
int               result;
T830_PARAM_BUFFER  param_buf;

result = ioctl(fd, T830_IOCG_READ_PARAM, &param_buf);

/* 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

4 Debugging

For debugging output see tip830drv.c. You will find the following symbol:

```
#undef TIP830_DEBUG_VIEW
```

To enable a debug output replace “undef” with “define”.

The TIP830_DEBUG_VIEW symbol controls debugging output of the driver.

```
TIP830 - 8 Channel Simultaneous Sampling ADC - version 1.2.0 (2009-07-01)
TIP830  :  Probe new TIP830 mounted on <TEWS TECHNOLOGIES - (Compact)PCI
IPAC Carrier> at slot A

TIP830  :  IP I/O Memory Space
00000000 : FF 12 FF C0 FF 00 FF 04 FF 12 FF C2 FF 00 FF 01
00000010 : FF FF FF FD FF 12 FF CC FF 12 FF CE FF FF FF FE
00000020 : FF FF FF FE

TIP830  :  IP ID Memory Space
00000000 : FF 49 FF 50 FF 41 FF 43 FF B3 FF 07 FF 10 FF 00
00000010 : FF 00 FF 00 FF 1C FF 89 FF 0A FF 08 FF 08 FF 09
00000020 : FF 08 FF 07 FF 08 FF 07 FF D2 FF B4 FF BC FF A9
00000030 : FF 0C FF DA FF 00 FF FE FF 00 FF 00 FF 00 FF 00
```