The Embedded I/O Company

TEWS TECHNOLOGIES

TIP570-SW-82

Linux Device Driver

16/8 Channel 12 bit ADC and 8 Channel 12 bit DAC

Version 1.3.x

User Manual

Issue 1.3.0

April 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

TIP570-SW-82

Linux Device Driver

16/8 Channel 12 bit ADC
8 Channel 12 bit DAC

Supported Modules:
TIP570

Issue	Description	Date
1.0	First Issue	July 15, 2001
1.1	Support for CARRIER CLASS DRIVER, DEVFS and SMP	September 25, 2003
1.2.0	Linux Kernel 2.6.x Revision	May 4, 2005
1.3.0	General Revision, TIP570-11 support added	April 28, 2009

Table of Contents

1 Introduction

1.1 Device Driver

The TIP570-SW-82 Linux device driver allows the operation of the TIP570 Analog Input and Output IPAC module conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, close*()*, *read()*, *write()* and *ioctl()* functions.

Because the TIP570 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 TIP570-SW-82 device driver supports the following features:

- start AD conversion and return input value
- support of different ADC input gains
- write values into DAC and output new value
- support of latch and transparent DAC mode
- data correction for ADC and DAC with factory set data
- TEWS TECHNOLOGIES IPAC carrier driver support.

The TIP570-SW-82 device driver supports the modules listed below:

TIP570-10	8 DAC channels 16 ADC channels with gain 1, 2, 5 and 10	IndustryPack® compatible
TIP570-11	8 DAC channels 16 ADC channels with gain 1, 2, 4 and 8	IndustryPack® compatible

To get more information about the features and use of TIP570 devices it is recommended to read the manuals listed below.

TIP570 User manual

TIP570 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 TIP570-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 'TIP570-SW-82':

TIP570-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
TIP570-SW-82-1.3.0.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 TIP570-SW-82-SRC.tar.gz contains the following files and directories:

Directory path 'tip570':

tip570.c	Driver source code
tip570def.h	Driver include file
tip570.h	Driver include file for application program
makenode	Script to create device nodes on the file system
Makefile	Device driver make file
example/tip570exa.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 TIP570-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TIP570-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 *tip570.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 tip570drv

- 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 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 TIP570 module found. The first TIP570 can be accessed with device node /dev/tip570_0, the second TIP570 with device node /dev/tip570_1, the third TIP570 with device node /dev/tip570_2 and so on.

The allocation of device nodes to physical TIP570 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP570 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 tip570drv –r

If your kernel has enabled a dynamic device file system like devfs or sysfs (udev), all /dev/tip570_... 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 "*tip570drv: 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 TIP570 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 TIP570_MAJOR.

To change the major number edit the file tip570drv.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP570_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP570_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/tip570_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 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 (*T570_RW_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;
} T570_IO_BUFFER, *PT570_IO_BUFFER;
```

channel

Specifies the ADC channel number from where to read the AD value. Valid channel numbers are 1..16 if Single-Ended is selected. 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:

Value	Description	
T570_GAIN_1	Select Gain 1	
T570_GAIN_2	Select Gain 2	
T570_GAIN_4	Select Gain 4	(only TIP570-11)
T570_GAIN_5	Select Gain 5	(only TIP570-10)
T570_GAIN_8	Select Gain 8	(only TIP570-11)
T570_GAIN_10	Select Gain 10	(only TIP570-10)

mode

Specifies the channel input interface. Allowed values are:

Value	Description
T570_SINGLE	Input Interface is used in single-ended mode
T570_DIFF	Input Interface is used in differential mode

correction

Specifies if data correction with the factory stored calibration data should be performed for the input value.

Value	Description
T570_NOCORR	Disable data correction. The raw value will be returned.
T570_CORR	Enable data correction. A corrected value will be returned.

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 TIP570 Hardware User Manual).

EXAMPLE

```
#include      <tip570.h>

int               fd;
ssize_t           NumBytes;
T570_IO_BUFFER    ADCBuf;

ADCBuf.gain       = T570_GAIN_1;
ADCBuf.mode       = T570_SINGLE;
ADCBuf.channel    = 1;
ADCBuf.correction = T570_CORR;

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

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

RETURNS

On success read returns the size of the structure T570_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	The conversion was not completed within 20 microseconds. 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

read() – write to a device

SYNOPSIS

#include <unistd.h>

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

DESCRIPTION

The write function attempts to start a DA conversion on the specified channel.

A pointer to the callers write buffer (*T570_RW_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;
} T570_IO_BUFFER, *PT570_IO_BUFFER;
```

channel

Specifies the DAC channel number where to write the DA value. Valid channel numbers are 1..8.

gain

This parameter is not used for the write command.

mode

Specifies the DAC mode. The following values and bits can be set:

Value	Description
T570_TRANS	The output will be written in transparent mode. The value will be written to the specified DAC channel and the output voltage will change immediately.
T570_LATCH	The output will be written in latched mode. The value will be written to the specified DAC channel, but there will be no change at the output line. The output line will be set if the *T570_CONV* bit is set, independent of the specified channel.
T570_CONV	This bit can be set additional to the latched mode. (*T570_LATCH* | *T570_CONV).* Setting this bit will execute a write as described for the latched mode, but it will also trigger a conversion of current values stored in the DACs data registers. The output voltages of all DAC channels will be updated.

correction

Specifies if data correction with the factory stored and channel dependent calibration data should be performed for the output value.

Value	Description
T570_NOCORR	Disable data correction. The specified value will be written to the DAC.
T570_CORR	Enable data correction. A corrected value will be written to the DAC.

data

Analog output value that will be written to the specified DAC channel. The analog data must be a sign extended two's complement integer value with 16-bit resolution. The lower four bits are always ignored (see TIP570 Hardware User Manual).

EXAMPLE

```
#include <tip570.h>

int                fd;
ssize_t            NumBytes;
T570_IO_BUFFER     DACBuf;

DACBuf.data        = 0x400;      /* 5V output */
DACBuf.mode        = T570_TRANS; /* transparent mode */
DACBuf.channel     = 1;
DACBuf.correction  = T570_CORR;

NumBytes = write(fd, &ioBuf, sizeof(ioBuf));

…
```

…

```
/* Check the result of the last device I/O operation */
if (NumBytes > 0)
{
     printf("DAC write OK \n");
}
else
{
     printf("write failed -->  Error = %d\n", errno );
}
```

RETURNS

On success read returns a positive value. 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 write buffer is too small or if the gain or channel parameter out of range.
ETIME	The conversion was not completed within 20 microseconds. The hardware seems to be faulty or the device mapping is incorrect.
EFAULT	Invalid pointer to the write buffer.

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 *tip570.h*:

Symbol	Meaning
T570_IOCG_READ_PARAM	Read module parameter

See behind for more detailed information on each control code.

To use these TIP570 specific control codes the header file tip570.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 TIP570 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.5.1 T570_IOCG_READ_PARAM

NAME

T570_IOCG_READ_PARAM - Read module parameter

DESCRIPTION

This ioctl function attempts to read the module type and calibration data of the TIP570 associated with the open file descriptor, *filedes*, into the parameter buffer pointed to by *argp.*

```
typedef struct
{
        unsigned int        ModuleType;
        int                 ADCcalGain[4];
        int                 ADCcalOffs[4];
        int                 DACcalGain[8];
        int                 DACcalOffs[8];
} T570_PARAM_BUFFER, *PT570_PARAM_BUFFER;
```

ModuleType

Returns the type code of the associated TIP570. (10/11)

ADCcalGain[4]

Receives the gain error of the input amplifier for the possible four gain selections in the unit ¼ LSB (see also TIP570 Hardware User Manual).

ADCcalOffs[4]

Returns the offset (zero) error of the input amplifier for the possible four gain selections in the unit ¼ LSB (see also TIP570 Hardware User Manual).

DACcalGain[8]

Returns the gain error for all the DAC channels in the unit ¼ LSB (see also TIP570 Hardware User Manual).

DACcalOffs[8]

Returns the offset (zero) error for all the DAC channels in the unit ¼ LSB (see also TIP570 Hardware User Manual).

EXAMPLE

```
#include <tip570.h>

int                fd;
int                result;
T570_PARAM_BUFFER  ParamBuf;

result = ioctl(fd, T570_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