The Embedded I/O Company

TEWS TECHNOLOGIES

TIP150-SW-82

Linux Device Driver

1 or 2 Channel Synchro / Resolver-To-Digital Converter

Version 1.0.x

User Manual

Issue 1.0

July 2003

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

TIP150-SW-82

1 or 2 Channel Synchro/Resolver-To-Digital Converter

Linux Device Driver

Issue	Description	Date
1.0	First Issue	February 21, 2003

Table of Contents

1 Introduction

The TIP150-SW-82 Linux device driver allows the operation of a TIP150 IPAC module on Linux operating systems with kernel version 2.4.4 or higher installed.

Because the TIP150 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 TIP150 device driver includes the following features:

- reading one channel
- reading both channels at once (if supported by module)
- configure module

2 Installation

The software is delivered on a PC formatted 3½" HD diskette.

The directory A:\TIP150-SW-82 contains the following files:

TIP150-SW-82.pdf	This manual in PDF format
TIP150-SW-82.tar.gz	GZIP compressed archive with driver source code

The GZIP compressed archive TIP150-SW-82.tar.gz contains the following files and directories:

tip150/tip159drv.c	Driver source code
tip150/tip150def.h	Driver include file
tip150/tip150.h	Driver include file for application program
tip150/makenode	Script to create device nodes on the file system
tip150/makefile	Device driver make file
tip150/example/example.c	Example application
tip150/example/makefile	Example application make file

In order to perform an installation, extract all files of the archive TIP150-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 *A:\CARRIER-SW-82* on the distribution diskette.

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 tip150drv

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.

 # sh makenode

On success the device driver will create a minor device for each TIP150 module found. The first TIP150 can be accessed with device node /dev/tip150_0, the second TIP150 with device node /dev/tip150_1 and so on.

The allocation of device nodes to physical TIP150 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP111 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 tip150drv

Be sure that the driver isn't opened by any application program. If opened you will get the response "*tip150drv: 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 TIP150 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.

To change the major number edit the file tip150drv.c, change the following symbol to appropriate value and enter **make install** to create a new driver.

TIP150_MAJOR Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP150_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/tip150_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

E_NODEV	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

E_NODEV	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 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 *TIP150.h*:

Symbol	Meaning
T150_IOCX_READ_SINGLE	Read single channel
T150_IOCG_READ_DOUBLE	Read both available channels (if supported)
T150_IOCS_CONFIG	Configure TIP150-module

See behind for more detailed information on each control code.

To use these TIP150 specific control codes the header file TIP150.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 TIP150 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 T150_IOCX_READ_SINGLE

NAME

T150_IOCG_READ_SINGLE - Read single channel

DESCRIPTION

This ioctl function reads the current value of the specified channel. The value is returned in the I/O buffer structure at the corresponding channel-position. The other value is invalid.

A pointer to the I/O-Buffer structure *T150_BUFFER* is passed by the parameter *argp* to the driver. For the existing status-values please refer to the TIP150 manual or use the pre-defined macros from *tip150.h*.

The *T150_BUFFER* structure has the following layout:

```
typedef struct {
        unsigned char       channel;
        unsigned short      data[2];
        unsigned char       status[2];
} T150_BUFFER;
```

unsigned char channel

Specifies the channel number from which the driver should read the data value. If the channel is unsupported by the TIP150-module, the error code *ECHRNG* is generated.

unsigned short data[2]

Array of data-values. The data value is stored at the array-position corresponding to the specified channel-number. The other array-field contains an invalid value and should not be used.

unsigned char status[2]

Array of status-values. The status value is stored at the array-position corresponding to the specified channel-number. The other array-field contains an invalid value and should not be used.

EXAMPLE

```
int          fd;
int          result;
T150_BUFFER  ioBuf;

/***   specify channel number to be read  ***/
ioBuf.channel = 0;

result = ioctl(fd, T150_IOCG_READ_SINGLE, &ioBuf);

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

ERRORS

EFAULT	Invalid argument pointer. Please check the argument *argp*.
ECHRNG	specified channel not supported by attached TIP150-device

SEE ALSO

ioctl man pages

3.3.2 T150_IOCG_READ_DOUBLE

NAME

T150_IOCG_READ_DOUBLE - Read both available channels (if supported)

DESCRIPTION

This ioctl function reads both channels (if available) on the TIP150-module in a synchronous way.

A pointer to the I/O-Buffer structure *T150_BUFFER* is passed by the parameter *argp* to the driver.

The *T150_BUFFER* structure has the following layout:

```
typedef struct {
        unsigned char           channel;
        unsigned short          data[2];
        unsigned char           status[2];
} T150_BUFFER;
```

unsigned char channel

Specifies the number of the channel to be read by the driver. Is not used here, because both available channels should be read. If the second channel is not supported by the TIP150-module, the error code *ECHRNG* is generated.

unsigned short data[2]

Array of data-values. The array-position marks the corresponding channel-number.

unsigned char status[2]

Array of status-values. The array-position marks the corresponding channel-number.

EXAMPLE

```
int          fd;
int          result;
T150_BUFFER  ioBuf;


result = ioctl(fd, T150_IOCG_READ_DOUBLE, &ioBuf);

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

ERRORS

EFAULT	Invalid argument pointer. Please check the argument *argp*.
ECHRNG	Two Channels not supported by attached TIP150-device

SEE ALSO

ioctl man pages

3.3.3 T150_IOCS_CONFIG

NAME

T150_IOCS_CONFIG – configure TIP150-module

DESCRIPTION

This ioctl function configures the operation-mode and resolution of the TIP150-module.

The new control value is passed through a pointer to the configuration-structure TIP150_CONFIG by the argument *argp* to the driver.

To create a valid control-value, please use the pre-defined macros in *tip150.h.* The options are OR'ed together to create the control value (see the following example).

The *T150_CONFIG* structure has the following layout:

```
typedef struct {
        unsigned char       channel;
        unsigned char       control;
} T150_CONFIG;
```

unsigned char channel

Specifies the number of the channel to be configured by the driver.

unsigned char control

Specifies the value to be written into the corresponding channel-configuration-register. As an extra option a spinlock-secured synchronous access to the data and status values can be specified. If the (multiprocessor-) system is spinlocked, the read-access cannot be interrupted by any other scheduled process. This option is channel-independent.

EXAMPLE

```
int             fd;
int             result;
TIP150_CONFIG   configBuf;

/***
 ** set resolution of channel0 to 12bit with syncronous status bit,
 ** add spinlock-secured access
***/
configBuf.channel = 0;
configBuf.control = ( T150_CFG_RES_12BIT | T150_CFG_SYN_STAT |
                      T150_CFG_SYN_LOCK );

result = ioctl(fd, T150_IOCS_CONFIG, &configBuf);

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

ERRORS

EFAULT	Invalid argument pointer. Please check the argument *argp*.
ECHRNG	specified channel not supported by attached TIP150-device

SEE ALSO

ioctl man pages