

The Embedded I/O Company



TPMC150-SW-82

Linux Device Driver

4, 3, 2 or 1 Channel Synchro/Resolver-to-Digital Converter

Version 1.1.x

User Manual

Issue 1.1.2

January 2006

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
Phone: +49-(0)4101-4058-0
e-mail: info@tews.com

25469 Halstenbek / Germany
Fax: +49-(0)4101-4058-19
www.tews.com

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Phone: +1 (775) 686 6077
e-mail: usasales@tews.com

Reno, Nevada 89504 / USA
Fax: +1 (775) 686 6024
www.tews.com

TPMC150-SW-82

4, 3, 2 or 1 Channel Synchro/Resolver-to-Digital Converter

Linux Device Driver

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2005 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	July 1, 2003
1.1.0	General Revision	November 9, 2005
1.1.1	Corrected file list and install description	December 12, 2005
1.1.2	Corrected install description	January 16, 2006

Table of Content

1	INTRODUCTION.....	4
2	INSTALLATION.....	5
	2.1 Build and install the device driver.....	5
	2.2 Uninstall the device driver	5
	2.3 Install device driver into the running kernel	6
	2.4 Remove device driver from the running kernel	7
	2.5 Change Major Device Number	7
3	DEVICE INPUT/OUTPUT FUNCTIONS	8
	3.1 open()	8
	3.2 close().....	10
	3.3 ioctl()	11
	3.3.1 TP150_IOC_READ_DIGINPUT	13
	3.3.2 TP150_IOC_READ_CONDATA	15
	3.3.3 TP150_IOC_READ_ENCDATA	17
	3.3.4 TP150_IOC_WRITE_ENCPRELD	19
	3.3.5 TP150_IOC_CONFIG_CON	21
	3.3.6 TP150_IOC_CONFIG_ENC	23
	3.3.7 TP150_IOC_CONFIG_MULTICON	26
	3.3.8 TP150_IOC_CONFIG_MULTIENC	28
	3.3.9 TP150_IOC_WAIT_DIGINEVENT	30
4	DIAGNOSTIC.....	32

1 Introduction

The TPMC150-SW-82 Linux device driver allows the operation of the TPMC150 PMC conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close()* and *ioctl()* functions.

Special I/O operation that do not fit to the standard I/O calls will be performed by calling the *ioctl()* function with a specific function code and an optional function dependent argument.

The TPMC150-SW-82 device driver supports the following features:

- Read digital input values
- Read converter data
- Read encoder data
- Wait for event on digital input
- Set preload encoder value
- Configure converter
- Configure encoder
- Configure (synchron) read for multiple converter
- Configure (synchron) read for multiple encoder

The TPMC150-SW-82 device driver supports the modules listed below:

TPMC150	4, 3, 2 or 1 Channel Synchro/Resolver-to-Digital Converter	(PMC)
---------	--	-------

To get more information about the features and use of TPMC150 device it is recommended to read the manuals listed below.

TPMC150 User manual
TPMC150 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path `.\TPMC150-SW-82\`:

TPMC150-SW-82-1.1.1.pdf	This manual in PDF format
TPMC150-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information

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

The GZIP compressed archive `TPMC150-SW-82-SRC.tar.gz` contains the following files and directories:

Directory path `./TPMC150/`:

<code>tpmc150.c</code>	Driver source code
<code>tpmc150def.h</code>	Driver include file
<code>tpmc150.h</code>	Driver include file for application program
<code>makenode</code>	Script to create device nodes on the file system
<code>Makefile</code>	Device driver make file
<code>example/tpmc150exa.c</code>	Example application
<code>example/Makefile</code>	Example application make file
<code>include/tpmodule.h</code>	Driver and kernel independent library header file
<code>include/tpmodule.c</code>	Driver and kernel independent library source file
<code>include/tpxxxhwdep.h</code>	HAL library header file
<code>include/tpxxxhwdep.c</code>	HAL library source file

In order to perform an installation, extract all files of the archive `TPMC150-SW-82-SRC.tar.gz` to the desired target directory.

- Login as *root* and change to the target directory
- Copy `tpmc150.h` to `/lib/modules/<version>/build/include` and `/usr/include`

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

- To update the device driver's module dependencies, enter:

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

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 tpmc150drv

- 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 device file system (devfs) then 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 compatible channel found. The first channel of the first PMC module can be accessed with device node `/dev/tpmc150_0`, the second channel with device node `/dev/tpmc150_1` and so on. The assignment of device nodes to physical PMC modules depends on the search order of the PCI bus driver.

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 tpmc150drv
```

If your kernel has enabled devfs, all `/dev/tpmc150_*` 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 "*tpmc150drv: 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

This paragraph is only for Linux kernels without DEVFS installed.

The TPCM150 driver uses dynamic allocation of major device numbers per 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 `tpmc150def.h`, change the following symbol to appropriate value and enter `make install` to create a new driver.

TPMC150_MAJOR Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TPMC150_MAJOR    122
```

Be sure that the desired major number isn't used by other drivers. Please check `/proc/devices` to see which numbers are free.

Keep in mind that's necessary to create new device nodes if the major number for the TPCM150 driver has changed and the `makenode` script isn't used.

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/tpmc150_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 fildes, int request [, void *argp])
```

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *fildes*, 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 *tpmc150.h* :

Value	Meaning
<i>TP150_IOC_READ_DIGINPUT</i>	Read the state of the digital input channels
<i>TP150_IOC_READ_CONDATA</i>	Read the value of a specified converter channel
<i>TP150_IOC_READ_ENCDATA</i>	Read the value of a specified encoder counter channel
<i>TP150_IOC_WRITE_ENCPRELD</i>	Set the value of the encoder preload register
<i>TP150_IOC_CONFIG_CON</i>	Configure a converter channel
<i>TP150_IOC_CONFIG_ENC</i>	Configure an encoder channel
<i>TP150_IOC_CONFIG_MULTICON</i>	Configure converter channels for multiple (synchron) read
<i>TP150_IOC_CONFIG_MULTIENC</i>	Configure converter channels for multiple (synchron) read
<i>TP150_IOC_WAIT_DIGINEVENT</i>	Wait for a specified transition on a specified digital input channel

See behind for more detailed information on each control code.

To use these TPMC150 specific control codes the header file TPMC150.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 TPMC150 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TP150_IOC_READ_DIGINPUT

NAME

TP150_IOC_READ_DIGINPUT – Read the states of the digital inputs

DESCRIPTION

This ioctl function reads the actual state of the digital input channels.

A pointer to the callers digital read buffer (*TP150_READ_DIGINPUT_BUF*) is passed by the parameter *argp* to the driver. The optional argument can be omitted for this ioctl function.

The *TP150_READ_DIGINPUT_BUF* structure has the following layout:

typedef struct

```
{  
    unsigned char    value;        /* digital input state (R) */  
} TP150_READ_DIGINPUT_BUF, *PTP150_READ_DIGINPUT_BUF;
```

value

This value returns the states of the digital input channels. Bit 0 specifies the state of digital channel 1, bit 1 the state of channel 2 and so on. There will be four digital input channels available independent on the model type.

EXAMPLE

```
int          hCurrent = 0;
int          result;
TP150_READ_DIGINPUT_BUF  digInBuf;

hCurrent = open(...);

...

/*
** Read the value of the digital input channels
*/
result = ioctl(hCurrent, TP150_IOC_READ_DIGINPUT, &digInBuf);
if(result >= 0)
{
    /* Reading digital inputs successful */
    printf("Input: 1: %s \n", (digInBuf.value & (1<<0)) ? "ON" : "OFF");
    printf("Input: 2: %s \n", (digInBuf.value & (1<<1)) ? "ON" : "OFF");
    printf("Input: 3: %s \n", (digInBuf.value & (1<<2)) ? "ON" : "OFF");
    printf("Input: 4: %s \n", (digInBuf.value & (1<<3)) ? "ON" : "OFF");
}
else
{
    /* Reading digital inputs failed */
}
}
```

ERRORS

EFAULT Invalid pointer to the buffer.

SEE ALSO

ioctl man pages

3.3.2 TP150_IOC_READ_CONDATA

NAME

TP150_IOC_READ_CONDATA – Read value of a converter channel

DESCRIPTION

This ioctl function reads the actual or latched value of a specified converter channel.

A pointer to the callers read buffer (*TP150_READ_CONDATA_BUF*) is passed by the parameter *argp* to the driver. The optional argument can be omitted for this ioctl function.

The *TP150_READ_CONDATA_BUF* structure has the following layout:

typedef struct

```
{
    unsigned char    channel;    /* channel number (W) */
    unsigned short   value;      /* converter data (R) */
    unsigned long    status;     /* channel status (R) */
} TP150_READ_CONDATA_BUF, *PTP150_READ_CONDATA_BUF;
```

channel

This value specifies the converter channel on the TPMC150. The valid values depend on the model type.

Model	Valid channel numbers
TPMC150-10	1...4
TPMC150-11	1...3
TPMC150-12	1...2
TPMC150-13	1

value

This value returns the actual (or latched) value of the converter channel.

status

This value returns the actual state of the converter channel. The value is an ORed value of the following flags:

Flag	Description
TP150_IO_F_CONDATA_BIT	If this bit is set the Build-in-Self-Test failed.
TP150_IO_F_CONDATA_NOADAMOUNT	If this bit is set there is no adapter mounted.
TP150_IO_F_CONDATA_MOTDIR	This bit indicates the direction of the motion. 0: down (clockwise, angle decreasing) 1: down (counter clockwise, angle increasing)

EXAMPLE

```
int                hCurrent = 0;
int                result;
TP150_READ_CONDATA_BUF  rdConBuf;

hCurrent = open(...);

...

/*
** Read value and status from channel 2
*/
rdConBuf.channel = 2;
result = ioctl(hCurrent, TP150_IOC_READ_CONDATA, &rdConBuf);
if(result >= 0)
{
    /* Reading converter data successful */
    printf("    Value:  %d (%04Xh)\n", rdConBuf.value, rdConBuf.value);
    printf("    Motion: %s\n",
           (rdConBuf.status & TP150_IO_F_CONDATA_MOTDIR) ? "up" : "down");
    ...
}
else
{
    /* Reading converter data failed */
}
```

ERRORS

EFAULT	Invalid pointer to the buffer.
EACCES	Invalid channel number specified

SEE ALSO

ioctl man pages

3.3.3 TP150_IOC_READ_ENCDATA

NAME

TP150_IOC_READ_ENCDATA – Read value of an encoder channel

DESCRIPTION

This ioctl function reads the actual or latched value of a specified encoder channel.

A pointer to the callers read buffer (*TP150_READ_ENCDATA_BUF*) is passed by the parameter *argp* to the driver. The optional argument can be omitted for this ioctl function.

The *TP150_READ_ENCDATA_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char    channel;    /* channel number (W) */
    unsigned long    value;      /* encoder data (R) */
} TP150_READ_ENCDATA_BUF, *PTP150_READ_ENCDATA_BUF;
```

channel

This value specifies the encoder channel on the TPMC150. The valid values depend on the model type.

Model	Valid channel numbers
TPMC150-10	1...4
TPMC150-11	1...3
TPMC150-12	1...2
TPMC150-13	1

value

This value returns the actual (or latched) value of the encoder channel.

EXAMPLE

```
int                hCurrent = 0;
int                result;
TP150_READ_ENCDATA_BUF  rdEncBuf;

hCurrent = open(...);

...

/*
** Read value of encoder channel 2
*/
rdEncBuf.channel = 2;
result = ioctl(hCurrent, TP150_IOC_READ_ENCDATA, &rdEncBuf);
if(result >= 0)
{
    /* Reading encoder value successful */
    printf("    Value:  %ld (%08lXh)\n", rdEncBuf.value, rdEncBuf.value);
}
else
{
    /* Reading encoder value failed */
}
```

ERRORS

EFAULT	Invalid pointer to the buffer.
EACCES	Invalid channel number specified.

SEE ALSO

ioctl man pages

3.3.4 TP150_IOC_WRITE_ENCPRELD

NAME

TP150_IOC_WRITE_ENCPRELD – Set the preload value of an encoder channel

DESCRIPTION

This ioctl function sets the preload value of the specified encoder channel.

A pointer to the callers buffer (*TP150_WRITE_ENCPRELD_BUF*) is passed by the parameter *argp* to the driver. The optional argument can be omitted for this ioctl function.

The *TP150_WRITE_ENCPRELD_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char    channel;    /* channel number (W) */
    unsigned long    value;      /* encoder preload data (W) */
    unsigned long    flags;
} TP150_WRITE_ENCPRELD_BUF, *PTP150_WRITE_ENCPRELD_BUF;
```

channel

This value specifies the encoder channel on the TPMC150. The valid values depend on the model type.

Model	Valid channel numbers
TPMC150-10	1...4
TPMC150-11	1...3
TPMC150-12	1...2
TPMC150-13	1

value

This value specifies the encoder preload value.

flags

This value is an ORed value of the following flags.

Flag	Valid channel numbers
TP150_IO_F_IMMEDIATE	execute preload immediately.

EXAMPLE

```
int          hCurrent = 0;
int          result;
TP150_WRITE_ENCPRELD_BUF wrPrldBuf;

hCurrent = open(...);

...

/*
** Set Preload value of encoder channel 2
** Immediate Preload
*/
wrPrldBuf.channel = 2;
wrPrldBuf.value   = 0x11223344;
wrPrldBuf.channel = TP150_IO_F_IMMPRELOAD;
result = ioctl(hCurrent, TP150_IOC_WRITE_ENCPRELD, &wrPrldBuf);
if(result >= 0)
{
    /* Setting encoder preload value successful */
}
else
{
    /* Setting encoder preload value failed */
}
```

ERRORS

EFAULT	Invalid pointer to the buffer.
EACCES	Invalid channel number specified.

SEE ALSO

ioctl man pages

3.3.5 TP150_IOC_CONFIG_CON

NAME

TP150_IOC_CONFIG_CON – Configure converter channel

DESCRIPTION

This ioctl function configures a specified converter channel.

A pointer to the callers buffer (*TP150_CONFIG_CON_BUF*) is passed by the parameter *argp* to the driver. The optional argument can be omitted for this ioctl function.

The *TP150_CONFIG_CON_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char    channel;        /* channel number (W) */
    unsigned long   convRes;        /* Converter Resolution (W) */
    unsigned long   synchStatLatch; /* synchronous Status Latch (TRUE/FALSE) (W) */
    unsigned long   synchConv;      /* synchronous Conversion (TRUE/FALSE) (W) */
} TP150_CONFIG_CON_BUF, *PTP150_CONFIG_CON_BUF;
```

channel

This value specifies the converter channel on the TPMC150. The valid values depend on the model type.

Model	Valid channel numbers
TPMC150-10	1...4
TPMC150-11	1...3
TPMC150-12	1...2
TPMC150-13	1

convRes

This parameter specifies the converter resolution. Allowed values are:

Flag	Description
TP150_IO_M_CONCRES_10BIT	Set converter resolution to 10bit.
TP150_IO_M_CONCRES_12BIT	Set converter resolution to 12bit.
TP150_IO_M_CONCRES_14BIT	Set converter resolution to 14bit.
TP150_IO_M_CONCRES_16BIT	Set converter resolution to 16bit.

synchStatLatch

This parameter specifies if the synchronous status latch shall be enabled (*TRUE*) or not (*FALSE*).

synchConv

This parameter specifies if the synchronous conversion shall be enabled (*TRUE*) or not (*FALSE*). (Ignored for channel 2 & 4)

EXAMPLE

```
int                hCurrent = 0;
int                result;
TP150_CONFIG_CON_BUF  cfConBuf;

hCurrent = open(...);

...

/*
** Set converter channel 2 for 16 bit resolution
** synchronous latch on conversion disabled
*/
cfConBuf.channel = 2;
cfConBuf.convRes = TP150_IO_M_CONGRES_16BIT;
cfConBuf.synchStatLatch= FALSE;
cfConBuf.synchConv= FALSE;
result = ioctl(hCurrent, TP150_IOC_CONFIG_CON, &cfConBuf);
if(result >= 0)
{
    /* Configure converter channel successful */
}
else
{
    /* Configure converter channel failed */
}
```

ERRORS

EFAULT	Invalid pointer to the buffer.
EACCES	Invalid channel number specified.
EINVAL	Invalid parameter specified

SEE ALSO

ioctl man pages

3.3.6 TP150_IOC_CONFIG_ENC

NAME

TP150_IOC_CONFIG_ENC – Configure an encoder channel

DESCRIPTION

This ioctl function configures a specified encoder channel.

A pointer to the callers buffer (*TP150_CONFIG_ENC_BUF*) is passed by the parameter *argp* to the driver. The optional argument can be omitted for this ioctl function.

The *TP150_CONFIG_ENC_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char    channel;    /* channel number (W) */
    unsigned long    sigAnaMode; /* Signal Analysis Mode (W) */
    unsigned long    refMode;    /* Encoder Reference Mode (W) */
    unsigned long    enable;     /* Incremental Encoder Emulation
                                (TRUE/FALSE) (W) */
    unsigned long    enableOutput; /* Incremental Encoder Emulation Output
                                (TRUE/FALSE) (W) */
} TP150_CONFIG_ENC_BUF, *PTP150_CONFIG_ENC_BUF;
```

channel

This value specifies the encoder channel on the TPMC150. The valid values depend on the model type.

Model	Valid channel numbers
TPMC150-10	1...4
TPMC150-11	1...3
TPMC150-12	1...2
TPMC150-13	1

sigAnaMode

This parameter specifies the encoder signal analysis mode. Allowed values are:

Flag	Description
TP150_IO_M_CONESIGANA_OFF	disable counter
TP150_IO_M_CONESIGANA_1X	1x - single
TP150_IO_M_CONESIGANA_2X	1x – double
TP150_IO_M_CONESIGANA_4X	4x - quad

refMode

This parameter specifies the encoder reference mode. Allowed values are:

Flag	Description
TP150_IO_M_CONEREF_NONE	None reference mode
TP150_IO_M_CONEREF_REF	Reference mode
TP150_IO_M_CONEREF_AUTOREF	Auto reference mode
TP150_IO_M_CONEREF_INDEX	Index mode

enable

This parameter specifies if the incremental encoder emulation shall be enabled (*TRUE*) or not (*FALSE*).

enableOutput

This parameter specifies if the incremental encoder emulation output shall be enabled (*TRUE*) or not (*FALSE*).

EXAMPLE

```

int          hCurrent = 0;
int          result;
TP150_CONFIG_ENC_BUF  cfEncBuf;

hCurrent = open(...);

...

/*
** Enable encoder channel 2 in 4x none reference mode with output
*/
cfEncBuf.channel = 2;
cfEncBuf.sigAnaMode = TP150_IO_M_CONESIGANA_4X;
cfEncBuf.refMode = TP150_IO_M_CONEREF_NONE;
cfEncBuf.enable = FALSE;
cfEncBuf.enableOutput = FALSE;
result = ioctl(hCurrent, TP150_IOC_CONFIG_ENC, &cfEncBuf);
if(result >= 0)
{
    /* Configure encoder channel successful */
}
else
{
    /* Configure encoder channel failed */
}

```

ERRORS

EFAULT	Invalid pointer to the buffer.
EACCES	Invalid channel number specified.
EINVAL	Invalid parameter specified

SEE ALSO

ioctl man pages

3.3.7 TP150_IOC_CONFIG_MULTICON

NAME

TP150_IOC_CONFIG_MULTICON – Configure multiple converter read

DESCRIPTION

This ioctl function configures multiple converter reads.

A pointer to the callers buffer (*TP150_CONFIG_MULTI_BUF*) is passed by the parameter *argp* to the driver. The optional argument can be omitted for this ioctl function.

The *TP150_CONFIG_MULTI_BUF* structure has the following layout:

typedef struct

```
{
    unsigned long    enable;        /* Multiple Read enable (TRUE/FALSE) (W) */
    unsigned long    chEnable[4];  /* Multiple Read enable for Channel
                                   (TRUE/FALSE) (W) */
} TP150_CONFIG_MULTI_BUF, *PTP150_CONFIG_MULTI_BUF;
```

enable

This parameter specifies multiple converter read shall be enabled (*TRUE*) or not (*FALSE*).

enableCh[]

The array entries specifies which channels shall be enabled (*TRUE*) or not (*FALSE*) for the multiple read. Set index 0 for channel 1, index 1 for channel 2 and so on.

EXAMPLE

```
int          hCurrent = 0;
int          result;
TP150_CONFIG_MULTI_BUF  cfMltiConBuf;

hCurrent = open(...);

...

/*
** Enable multiple read for channel 1, 2 and 4
*/
cfMltiConBuf.enable = TRUE;
cfMltiConBuf.enableCh[0] = TRUE;
cfMltiConBuf.enableCh[1] = TRUE;
cfMltiConBuf.enableCh[2] = FALSE;
cfMltiConBuf.enableCh[3] = TRUE;
result = ioctl(hCurrent, TP150_IOC_CONFIG_MULTICON, & cfMltiConBuf);
if(result >= 0)
{
    /* Configure multiple converter successful */
}
else
{
    /* Configure multiple converter failed */
}
```

ERRORS

EFAULT Invalid pointer to the buffer.

SEE ALSO

ioctl man pages

3.3.8 TP150_IOC_CONFIG_MULTIENTC

NAME

TP150_IOC_CONFIG_MULTIENTC – Configure multiple encoder read

DESCRIPTION

This ioctl function configures multiple encoder reads.

A pointer to the callers buffer (*TP150_CONFIG_MULTI_BUF*) is passed by the parameter *argp* to the driver. The optional argument can be omitted for this ioctl function.

The *TP150_CONFIG_MULTI_BUF* structure has the following layout:

typedef struct

```
{
    unsigned long    enable;        /* Multiple Read enable (TRUE/FALSE) (W) */
    unsigned long    chEnable[4];  /* Multiple Read enable for Channel
                                   (TRUE/FALSE) (W) */
} TP150_CONFIG_MULTI_BUF, *PTP150_CONFIG_MULTI_BUF;
```

enable

This parameter specifies multiple encoder read shall be enabled (*TRUE*) or not (*FALSE*).

enableCh[]

The array entries specifies which channels shall be enabled (*TRUE*) or not (*FALSE*) for the multiple read. Set index 0 for channel 1, index 1 for channel 2 and so on.

EXAMPLE

```
int          hCurrent = 0;
int          result;
TP150_CONFIG_MULTI_BUF  cfMltiEncBuf;

hCurrent = open(...);

...

/*
** Enable multiple read for channel 1, 2 and 4
*/
cfMltiEncBuf.enable = TRUE;
cfMltiEncBuf.enableCh[0] = TRUE;
cfMltiEncBuf.enableCh[1] = TRUE;
cfMltiEncBuf.enableCh[2] = FALSE;
cfMltiEncBuf.enableCh[3] = TRUE;
result = ioctl(hCurrent, TP150_IOC_CONFIG_MULTIENT, & cfMltiEncBuf);
if(result >= 0)
{
    /* Configure multiple encoder successful */
}
else
{
    /* Configure multiple encoder failed */
}
```

ERRORS

EFAULT Invalid pointer to the buffer.

SEE ALSO

ioctl man pages

3.3.9 TP150_IOC_WAIT_DIGINEVENT

NAME

TP150_IOC_WAIT_DIGINEVENT – Wait for a digital input event

DESCRIPTION

This ioctl function waits for a specified digital input event.

A pointer to the callers buffer (*TP150_WAIT_EVENT_BUF*) is passed by the parameter *argp* to the driver. The optional argument can be omitted for this ioctl function.

The *TP150_WAIT_EVENT_BUF* structure has the following layout:

```
typedef struct
{
    unsigned char    channel;    /* channel number (W) */
    unsigned long    transition; /* Specify transition (W) */
    unsigned long    timeout;    /* Timeout in ticks (W) */
} TP150_WAIT_EVENT_BUF, *PTP150_WAIT_EVENT_BUF;
```

channel

This parameter specifies the digital input channel the event shall occur Valid channel numbers are 1 up to 4.

transition

This parameter specifies the transition which triggers the event. Allowed values are:

Value	Description
TP150_IO_M_WAITEVTRANS_LO	Event occurs if a High-to-Low transition occurs.
TP150_IO_M_WAITEVTRANS_HI	Event occurs if a Low-to-High transition occurs.

timeout

This value specifies maximum time the function should block before it returns if the event does not occur.

EXAMPLE

```
int                hCurrent = 0;
int                result;
TP150_WAIT_EVENT_BUF wtBuf;

hCurrent = open(...);

...

/*
** Wait for a low-to-high transition on channel 2
** Timeout after 10000 ticks
*/
wtBuf.channel = 2;
wtBuf.transition = TP150_IO_M_WAITEVTRANS_HI;
wtBuf.timeout = 10000;
result = ioctl(hCurrent, TP150_IOC_WAIT_DIGINEVENT, &wtBuf);
if(result >= 0)
{
    /* Event has occurred */
}
else
{
    /* Error or Timeout */
}
```

ERRORS

EFAULT	Invalid pointer to the buffer.
EACCES	Invalid channel number specified.
EINVAL	Invalid parameter specified
EBUSY	There is already a process waiting for an event on this channel
ETIME	System call timed out
ERESTARTSYS	System restarts

SEE ALSO

ioctl man pages

4 Diagnostic

If the TPMC150 does not work properly it is helpful to get some status information from the driver respective kernel.

The Linux */proc* file system provides information about kernel, resources, driver, devices and so on. The following screen dumps displays information of a correct running TPMC150 driver (see also the *proc* man pages).

```
cat /proc/pci
PCI devices found:
...
  Bus 0, device 15, function 0:
    Signal processing controller: PCI device 1498:0096 (TEWS Datentechnik
    GmbH) (rev 10).
      IRQ 11.
      Non-prefetchable 32 bit memory at 0xcfffd80 [0xcfffdfff].
      I/O at 0xd000 [0xd07f].
      Non-prefetchable 32 bit memory at 0xcfffd00 [0xcfffd7f].
...

```

```
cat /proc/devices
Character devices:
 1 mem
 2 pty
 3 tty
 4
 5 cua
 7 vcs
10 misc
13 input
14 sound
29 fb
36 netlink
162 raw
180 usb
226 drm
254 tpmc150drv

```

```
# cat /proc/interrupts
          CPU0
 0:      9231009          XT-PIC  timer
 1:         845          XT-PIC  keyboard
 2:          0          XT-PIC  cascade
 5:      18974          XT-PIC  eth0
 8:          1          XT-PIC  rtc
11:      11339          XT-PIC  usb-ohci, usb-ohci, SiS 7012, TPMC150
12:         2176          XT-PIC  PS/2 Mouse
14:      22696          XT-PIC  ide0
15:          1          XT-PIC  ide1
NMI:          0
ERR:          0
```

```
# cat /proc/ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
02f8-02ff : serial(auto)
0376-0376 : ide1
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(auto)
0cf8-0cff : PCI conf1
a000-afff : PCI Bus #01
  a800-a8ff : ATI Technologies Inc Rage 128 Pro Ultra TF
d000-d07f : PCI device 1498:0096 (TEWS Datentechnik GmBH)
d400-d4ff : Standard Microsystems Corp [SMC] 83C170QF
  d400-d4ff : epic100
d800-d83f : Silicon Integrated Systems [SiS] SiS7012 PCI Audio Accelerator
  d800-d83f : SiS 7012
dc00-dcff : Silicon Integrated Systems [SiS] SiS7012 PCI Audio Accelerator
  dc00-dcff : SiS 7012
ff00-ff0f : Silicon Integrated Systems [SiS] 5513 [IDE]
  ff00-ff07 : ide0
  ff08-ff0f : ide1
```

```
# cat /proc/iomem
00000000-0009fbff : System RAM
0009fc00-0009ffff : reserved
000a0000-000bffff : Video RAM area
000c0000-000c7fff : Video ROM
000f0000-000fffff : System ROM
00100000-0ffeffff : System RAM
    00100000-00247f2e : Kernel code
    00247f2f-0033ed03 : Kernel data
0fff0000-0fff7fff : ACPI Tables
0fff8000-0fffffff : ACPI Non-volatile Storage
c7c00000-cfcfffff : PCI Bus #01
    c8000000-cbffffff : ATI Technologies Inc Rage 128 Pro Ultra TF
cfe00000-cfefffff : PCI Bus #01
    cfefc000-cfefffff : ATI Technologies Inc Rage 128 Pro Ultra TF
cfff0000-cfff0fff : Standard Microsystems Corp [SMC] 83C170QF
    cfff0000-cfff0fff : epic100
cfffd00-cfffd07f : PCI device 1498:0096 (TEWS Datentechnik GmBH)
cfffd00-cfffd07f : TPMC150
cfffd80-cfffdfff : PCI device 1498:0096 (TEWS Datentechnik GmBH)
cffff000-cffff0ff : Silicon Integrated Systems [SiS] 7001
    cffff000-cffff0ff : usb-ohci
cffff000-cffff0ff : Silicon Integrated Systems [SiS] 7001 (#2)
    cffff000-cffff0ff : usb-ohci
d0000000-d3ffffff : Silicon Integrated Systems [SiS] 735 Host
fec00000-fec00fff : reserved
fee00000-fee00fff : reserved
ff000000-ff000fff : reserved
ff000000-ff000fff : reserved
fffc0000-ffffff : reserved
```