The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC150-SW-82

Linux Device Driver

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

Version 1.1.x

User Manual

Issue 1.1.5

December 2014

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

TPMC150-SW-82

Linux Device Driver

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

Supported Modules:

TPMC150

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
1.1.3	New file list: ChangeLog.txt added Corrected description of installation	October 19, 2007
1.1.4	TEWS LLC Address removed	May 17, 2010
1.1.5	General Update	December 04, 2014

Table of Content

1 INTRODUCTION.. 4
2 INSTALLATION.. 5
2.1 Build and install the Device Driver.. 5
2.2 Uninstall the Device Driver.. 6
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 I/O 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 (synchronous) read for multiple converter
- Configure (synchronous) 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

2 Installation

The directory TPMC150-SW-82 on the distribution media contains the following files:

TPMC150-SW-82-1.1.5.pdf	This manual in PDF format
TPMC150-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

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

Directory path './tpmc150/':

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

In order to perform an installation, extract all files of the archive TPMC150-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TPMC150-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tpmc150.h to */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

- Only 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 dependent kernel 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 tpmc150drv

- 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 compatible module found. The first

On success the device driver will create a minor device for each compatible channel found. The first PMC module can be accessed with device node /dev/tpmc150_0, the second module 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 or sysfs (udev), all /dev/tpmc851_* 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 a device file system (devfs, udev, ...) 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.

3 I/O 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);
if (fd == -1)
{
    /* Handle 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

Error Code	Description
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

Error Code	Description
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 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 *tpmc150.h* :

Symbol	**Meaning**
TP150_IOC_READ_DIGINPUT	Read the state of the digital input channels
TP150_IOC_READ_CONDATA	Read the value of a specified converter channel
TP150_IOC_READ_ENCDATA	Read the value of a specified encoder counter channel
TP150_IOC_WRITE_ENCPRELD	Set the value of the encoder preload register
TP150_IOC_CONFIG_CON	Configure a converter channel
TP150_IOC_CONFIG_ENC	Configure an encoder channel
TP150_IOC_CONFIG_MULTICON	Configure converter channels for multiple (synchron) read
TP150_IOC_CONFIG_MULTIENC	Configure converter channels for multiple (synchron) read
TP150_IOC_WAIT_DIGINEVENT	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

Error Code	Description
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 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 current 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.

```
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

```
#include <tpmc150.h>

int                         hCurrent;
int                         result;
TP150_READ_DIGINPUT_BUF     digInBuf;

/*
** 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

Error Code	Description
EFAULT	Invalid pointer to the buffer.

3.3.2 TP150_IOC_READ_CONDATA

NAME

TP150_IOC_READ_CONDATA – Read value of a converter channel

DESCRIPTION

This ioctl function reads the current 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.

```
typedef struct
{
        unsigned char           channel;
        unsigned short          value;
        unsigned long           status;
} 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 current (or latched) value of the converter channel.

status

This value returns the current 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: up (counter clockwise, angle increasing)

EXAMPLE

```
#include <tpmc150.h>

int                         hCurrent;
int                         result;
TP150_READ_CONDATA_BUF      rdConBuf;

/*
** 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

Error Code	Description
EFAULT	Invalid pointer to the buffer.
EACCES	Invalid channel number specified

3.3.3 TP150_IOC_READ_ENCDATA

NAME

TP150_IOC_READ_ENCDATA – Read value of an encoder channel

DESCRIPTION

This ioctl function reads the current 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.

```
typedef struct
{
        unsigned char           channel;/
        unsigned long           value;
} 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 current (or latched) value of the encoder channel.

EXAMPLE

```
#include <tpmc150.h>

int                         hCurrent;
int                         result;
TP150_READ_ENCDATA_BUF      rdEncBuf;

…
```

```
…

/*
** 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

Error Code	Description
EFAULT	Invalid pointer to the buffer.
EACCES	Invalid channel number specified

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.

```
typedef struct
{
        unsigned char           channel;
        unsigned long           value;
        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_IMMPRELOAD	execute preload immediately.

EXAMPLE

```
#include <tpmc150.h>

int                         hCurrent;
int                         result;
TP150_WRITE_ENCPRELD_BUF    wrPrldBuf;

/*
** 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

Error Code	Description
EFAULT	Invalid pointer to the buffer.
EACCES	Invalid channel number specified

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.

```
typedef struct
{
        unsigned char           channel;
        unsigned long           convRes;
        unsigned long           synchStatLatch;
        unsigned long           synchConv;
} 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

```
#include <tpmc150.h>

int                         hCurrent;
int                         result;
TP150_CONFIG_CON_BUF        cfConBuf;

/*
** Set converter channel 2 for 16 bit resolution
** synchronous latch on conversion disabled
*/
cfConBuf.channel = 2;
cfConBuf.convRes = TP150_IO_M_CONCRES_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

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

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.

```
typedef struct
{
        unsigned char           channel;
        unsigned long           sigAnaMode;
        unsigned long           refMode;
        unsigned long           enable;
        unsigned long           enableOutput;
} 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

```
#include <tpmc150.h>

int                            hCurrent;
int                            result;
TP150_CONFIG_ENC_BUF           cfEncBuf;

/*
** 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 = TRUE;
cfEncBuf.enableOutput = TRUE;
result = ioctl(hCurrent, TP150_IOC_CONFIG_ENC, &cfEncBuf);
if(result >= 0)
{
    /* Configure encoder channel successful */
}
else
{
    /* Configure encoder channel failed */
}
```

ERRORS

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

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.

```
typedef struct
{
        unsigned long           enable;
        unsigned long           chEnable[4];
} 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

```
#include <tpmc150.h>

int                         hCurrent;
int                         result;
TP150_CONFIG_MULTI_BUF      cfMltiConBuf;

/*
** 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

Error Code	Description
EFAULT	Invalid pointer to the buffer.

3.3.8 TP150_IOC_CONFIG_MULTIENC

NAME

TP150_IOC_CONFIG_MULTIENC – 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.

```
typedef struct
{
        unsigned long           enable;
        unsigned long           chEnable[4];
} 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

```
#include <tpmc150.h>

int                         hCurrent;
int                         result;
TP150_CONFIG_MULTI_BUF      cfMltiEncBuf;

/*
** 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_MULTIENC, & cfMltiEncBuf);
if(result >= 0)
{
     /* Configure multiple encoder successful */
}
else
{
     /* Configure multiple encoder failed */
}
```

ERRORS

Error Code	Description
EFAULT	Invalid pointer to the buffer.

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.

```
typedef struct
{
        unsigned char           channel;
        unsigned long           transition;
        unsigned long           timeout;
} 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

```
#include <tpmc150.h>

int                         hCurrent;
int                         result;
TP150_WAIT_EVENT_BUF        wtBuf;

…
```

```
…

/*
** Wait for a low-to-high transition on channel 2
** Timout 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

Error Code	Description
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

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/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  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   CPU1
  0:   121    0         IO-APIC-edge     timer
  1:   5      5         IO-APIC-edge     i8042
  8:   0      1         IO-APIC-edge     rtc0
  9:   0      0         IO-APIC-fasteoi  acpi
 12:   164    165       IO-APIC-edge     i8042
 14:   44     1096      IO-APIC-edge     ata_piix
 15:   0      0         IO-APIC-edge     ata_piix
 16:   0      0         IO-APIC          16-fasteoi uhci_hc d:usb5, TPMC150
 18:   0      0         IO-APIC          18-fasteoi uhci_hc d:usb4

 …

 ERR:  0
 MIS:  0


# lspci -v
```

```
00:00.0 Host bridge: Intel Corporation 82G33/G31/P35/P31 Express DRAM
Controller (rev 10)
        Subsystem: ASUSTeK Computer Inc. P5KPL-VM Motherboard
        Flags: bus master, fast devsel, latency 0
        Capabilities: [e0] Vendor Specific Information: Len=0b <?>

…

00:1e.0 PCI bridge: Intel Corporation 82801 PCI Bridge (rev e1) (prog-if 01
[Subtractive decode])
        Flags: bus master, fast devsel, latency 0
        Bus: primary=00, secondary=04, subordinate=04, sec-latency=32
        I/O behind bridge: 0000e000-0000efff
        Memory behind bridge: feb00000-febfffff
        Capabilities: [50] Subsystem: ASUSTeK Computer Inc. Device 8179

04:01.0 Signal processing controller: TEWS Technologies GmbH Device 0096
(rev 0a)
        Subsystem: TEWS Technologies GmbH Device 000a
        Flags: medium devsel, IRQ 16
        Memory at feb9fc00 (32-bit, non-prefetchable) [size=128]
        I/O ports at e880 [size=128]
        Memory at feb9f800 (32-bit, non-prefetchable) [size=128]
        Kernel driver in use: TEWS TECHNOLOGIES TPMC150 4, 3, 2 or 1
Channel Synchro Resolver-to-Digital Converter
```