*The Embedded I/O Company*

TEWS TECHNOLOGIES

# TPMC851-SW-82

## Linux Device Driver

Multifunction I/O (16-bit DAC/ADC, TTL I/O, Counter)

Version 1.1.x

## User Manual

Issue 1.1.1

September 2019

**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

## TPMC851-SW-82

Linux Device Driver

Multifunction I/O
(16-bit DAC/ADC, TTL I/O, Counter)

Supported Modules:
TPMC851



| Issue | Description | Date |
|---|---|---|
| 1.0.0 | First Issue | December 12, 2005 |
| 1.0.1 | New Address TEWS LLC, ChangeLog.txt added to file list | December 3, 2006 |
| 1.0.2 | General revision | November 26, 2008 |
| 1.0.3 | Address of TEWS LLC removed | August 5, 2009 |
| 1.0.4 | General revision | February 14, 2012 |
| 1.1.0 | Correction of Counter Input Mode definition<br>(TPMC851_M_CNTIN_QUAD3X to TPMC851_M_CNTIN_QUAD4X)<br>Correction of value for wait without timeout | December 3, 2014 |
| 1.1.1 | File-List modified | September 24, 2019 |

# Table of Contents

# 1 Introduction

The TPMC851-SW-82 Linux device driver allows the operation of the TPMC851 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 TPMC851-SW-82 device driver supports the following features:

- Executing AD conversion and reading input value
- Setting up, Starting and Stopping ADC Input Sequencer
- Configuring ADC Sequencer Trigger I/O
- Reading ADC Sequencer input data
- Setting output value and starting DA conversion
- Setting up, Starting and Stopping DAC Sequencer
- Configuring DAC Sequencer Trigger I/O
- Setting DAC Sequencer Data
- Reading digital I/O data
- Setting digital output data
- Configuring I/O direction and input debouncer
- Waiting for input events
- Reading counter value
- Resetting counter value
- Configuring counter mode and controls
- Setting preload and match value
- Waiting for counter events

The TPMC851-SW-82 device driver supports the modules listed below:

| | | |
|---|---|---|
| TPMC851 | 16(32) ADC, 8 DAC, 16 I/O, 1 counter | (PMC) |

To get more information about the features and use of TPMC851 device it is recommended to read the manuals listed below.

| |
|---|
| TPMC851 User Manual |

# 2 Installation

The directory TPMC851-SW-82 on the distribution media contains the following files:

| | |
|---|---|
| TPMC851-SW-82-SRC.tar.gz | GZIP compressed archive with driver source code |
| TPMC851-SW-82-1.1.1.pdf | PDF copy of this manual |
| Release.txt | Release information |
| ChangeLog.txt | Release history |

The GZIP compressed archive TPMC851-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tpmc851/':

| | |
|---|---|
| tpmc851.c | TPMC851 device driver source |
| tpmc851def.h | TPMC851 driver include file |
| tpmc851.h | TPMC851 include file for driver and application |
| makenode | Script to create device nodes on the file system |
| Makefile | Device driver make file |
| include/config.h | Driver independent library header file |
| 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/tpmc851exa.c | Example application |
| example/Makefile | Example application make file |
| COPYING | Copy of the GNU Public License (GPL) |

In order to perform an installation, extract all files of the archive TPMC851-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TPMC851-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tpmc851.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 tpmc851drv**

- 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 channel found. The first PMC module can be accessed with device node /dev/tpmc851_0, the second module with device node /dev/tpmc851_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 tpmc851drv**

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 "*tpmc851drv: 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 TDRV011 driver uses dynamic allocation of major device numbers by default. If this isn't suitable for the application it is possible to define a major number for the driver. If the kernel has enabled devfs the driver will not use the symbol TPMC851_MAJOR.

To change the major number edit the file *tpmc851def.h*, change the following symbol to appropriate value and enter **make install** to create a new driver.

| | |
|---|---|
| TPMC851_MAJOR | Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation. |

Example:

```
#define TPMC851_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/tpmc851_0”, O_RDWR);
if (fd == -1)
{
     /* handle error condition */
}
```

### 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 *tpmc851.h*:

| Symbol | Meaning |
|---|---|
| TPMC851_IOC_ADC_READ | Read value from ADC channel |
| TPMC851_IOC_ADC_SEQCONFIG | Configure ADC sequencer channel |
| TPMC851_IOC_ADC_SEQSTART | Start ADC sequencer |
| TPMC851_IOC_ADC_SEQSTOP | Stop ADC sequencer |
| TPMC851_IOC_ADC_SEQREAD | Read values from ADC sequencer buffer |
| TPMC851_IOC_DAC_WRITE | Write value to DAC channel |
| TPMC851_IOC_DAC_SEQCONFIG | Configure DAC sequencer channel |
| TPMC851_IOC_DAC_SEQSTART | Start DAC sequencer |
| TPMC851_IOC_DAC_SEQSTOP | Stop DAC sequencer |
| TPMC851_IOC_DAC_SEQWRITE | Write values to DAC sequencer buffer |
| TPMC851_IOC_DAC_SEQSTATE | Get DAC sequencer and information |
| TPMC851_IOC_IO_READ | Read from digital I/O |
| TPMC851_IOC_IO_WRITE | Write to digital I/O |
| TPMC851_IOC_IO_EVENTWAIT | Wait for I/O event |
| TPMC851_IOC_IO_CONFIG | Configure digital I/O |
| TPMC851_IOC_IO_DEBCONFIG | Configure digital I/O (input) debouncer |

*(continued on the next page)*

*(… continued)*

| | |
|---|---|
| TPMC851_IOC_CNT_READ | Read value from counter/timer |
| TPMC851_IOC_CNT_MATCHWAIT | Wait for counter match event |
| TPMC851_IOC_CNT_CTRLWAIT | Wait for counter control event |
| TPMC851_IOC_CNT_CONFIG | Configure counter |
| TPMC851_IOC_CNT_RESET | Reset counter |
| TPMC851_IOC_CNT_SETPRELD | Set counter preload value |
| TPMC851_IOC_CNT_SETMATCH | Set counter match value |

See behind for more detailed information on each control code.

**To use these TPMC851 specific control codes the header file tpmc851.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* |
| EFAULT | Parameter data can not be copied to the drivers context |

Other function dependent error codes will be described for each ioctl code separately. Note, the TPMC851 driver always returns standard Linux error codes.

## SEE ALSO

ioctl man pages

## 3.3.1 TPMC851_IOC_ADC_READ

### NAME

TPMC851_IOC_ADC_READ – Read value from ADC channel

### DESCRIPTION

This function starts an ADC conversion with specified parameters, waits for completion and returns the value.

**The ADC sequencer must be stopped for single ADC conversions.**

A pointer to the read structure (*TPMC851_ADC_READ_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        int             channel;
        int             gain;
        unsigned long   flags;
        short           adcValue;
} TPMC851_ADC_READ_BUF;
```

*channel*

Specifies the ADC channel number. Valid values are 1..16 for differential input and 1..32 for single-ended input.

*gain*

Specifies the input gain. Valid gain values are 1, 2, 4, and 8.

*flags*

Is an ored value of the following flags:

| Flag | Description |
|---|---|
| TPMC851_F_CORR | If set the function will return a corrected value of the input data in adcValue. Factory set and module dependent correction data is used for correction.<br>If not set, the raw value read from the module will be returned in adcValue. |
| TPMC851_F_IMMREAD | If set the driver will start the conversion without waiting for settling time. This should only be used if the previous conversion has used the same interface parameters (channel, gain, differential/single-ended).<br>If not set the driver will use the automatic mode, which sets interface configuration, waits settling time and then starts the conversion. |
| TPMC851_F_DIFF | If set the input channel will be a differential input.<br>If not set the input channel will be a single-ended input. |

*adcValue*

This value will return the read ADC value.

## EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     result;
TPMC851_ADC_READ_BUF    adcReadBuf;

/* Read a corrected value from differential channel 2, use a gain of 4 */
adcReadBuf.channel = 2;
adcReadBuf.gain    = 4;
adcReadBuf.flags   = TPMC851_F_CORR | TPMC851_F_DIFF;

…
```

…

```
printf("Read from ADC ... ");
result = ioctl(    fd,
                   TPMC851_IOC_ADC_READ,
                   &adcReadBuf);
if (result >= 1)
{
     printf("OK\n");
     printf("    ADC-value: %d", adcReadBuf.adcValue);
}
else
{
     /* process ioctl error */
}
```

## ERRORS

| Error Code | Description |
|---|---|
| EBUSY | The ADC sequencer is currently running |
| ECHRNG | Specified channel is invalid |
| EINVAL | Specified gain level is invalid |
| ETIME | The ADC conversion timed out |

## 3.3.2 TPMC851_IOC_ADC_SEQCONFIG

### NAME

TPMC851_IOC_ADC_SEQCONFIG – Configure ADC sequencer channel

### DESCRIPTION

This function enables and configures, or disables an ADC channel for sequence use.

**The ADC sequencer must be stopped to execute this function.**

A pointer to the configuration structure (*TPMC851_ADC_SEQCONFIG_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        int             channel;
        int             enable;
        int             gain;
        unsigned long   flags;
} TPMC851_ADC_SEQCONFIG_BUF;
```

*channel*

Specifies the ADC channel number to configure. Valid values are 1..16 for differential input and 1..32 for single-ended input.

*enable*

Specifies if the channel shall be used in sequencer mode or not. (0 disables the channel any other value will enable the channel)

*gain*

Specifies the input gain. Valid gain values are 1, 2, 4, and 8.

*flags*

Is an ored value of the following flags:

| Flag | Description |
|---|---|
| TPMC851_F_CORR | If set the sequencer will return a corrected value for the specified channel. Factory set and module dependent correction data is used for correction.<br>If not set, the raw value read from the module will be returned. |
| TPMC851_F_DIFF | If set the input channel will be a differential input.<br>If not set the input channel will be a single-ended input. |

## EXAMPLE

```
#include “tpmc851.h”

int                          fd;
int                          result;
TPMC851_ADC_SEQCONFIG_BUF    adcSeqConfBuf;

/*
** Configure single-ended channel 3, using a gain of 4 and returning
** corrected data when the sequencer is running
*/
adcSeqConfBuf.channel   = 3;
adcSeqConfBuf.enable    = TRUE;
adcSeqConfBuf.gain      = 4;
adcSeqConfBuf.flags     = TPMC851_F_CORR;

printf("Configure channel for Sequencer ... ");
result = ioctl(    fd,
                   TPMC851_IOC_ADC_SEQCONFIG,
                   &adcSeqConfBuf);
if (result >= 1)
{
     printf("OK\n");
}
else
{
     /* process ioctl error */
}
```

## ERRORS

| Error Code | Description |
|---|---|
| EBUSY | The ADC sequencer is currently running |
| ECHRNG | Specified channel is invalid |
| EINVAL | Specified gain level or flags are invalid |

## 3.3.3 TPMC851_IOC_ADC_SEQSTART

### NAME

TPMC851_IOC_ADC_SEQSTART – Start ADC sequencer

### DESCRIPTION

This function configures the ADC sequencer time and starts the ADC sequencer.

A pointer to the start structure (*TPMC851_ADC_SEQSTART_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        unsigned short          cycTime;
        unsigned long           flags;
        long                    bufSize;
} TPMC851_ADC_SEQSTART_BUF;
```

*cycTime*

Specifies the ADC sequencer cycle time. The sequencer time is specified in 100µs steps. With a value of 0, the “Sequencer Continuous Mode” is selected.

*flags*

Is an ored value of the following flags:

| Flag | Description |
|---|---|
| TPMC851_F_EXTTRIGSRC | If set the ADC sequencer is trigger with digital I/O line 0.<br>If not set, the ADC sequencer uses the ADC cycle counter. |
| TPMC851_F_EXTTRIGOUT | If set the ADC trigger is used as output on digital I/O line 0. |

***TPMC851_F_EXTTRIGSRC* and *TPMC851_F_EXTTRIGOUT cannot be used at the same time.***

*bufSize*

Specifies the internal ADC sequencer buffer size. The sequencer stores the incoming values inside an internal buffer, from where the user application retrieves the data (refer to ioctl function TPMC851_C_ADC_SEQREAD).

### EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         result;
TPMC851_ADC_SEQSTART_BUF    adcSeqStartBuf;

/*
** Start sequencer with a buffer of 100 word and a cycle time of 100 ms,
** do not use external trigger
*/
adcSeqStartBuf.cycTime      = 1000;
adcSeqStartBuf.flags        = 0;
adcSeqStartBuf.bufSize      = 100;

printf("Start ADC Sequencer ... ");
result = ioctl(    fd,
                   TPMC851_C_ADC_SEQSTART,
                   &adcSeqStartBuf);
if (result >= 1)
{
     printf("OK\n");
}
else
{
     /* process ioctl error */
}
```

### ERRORS

| Error Code | Description |
|---|---|
| EBUSY | The ADC sequencer is currently running |
| EINVAL | Specified gain level or flags are invalid |
| ENOMEM | No memory is available to allocate the internal buffer |

## 3.3.4 TPMC851_IOC_ADC_SEQSTOP

### NAME

TPMC851_IOC_ADC_SEQSTOP – Stop ADC sequencer

### DESCRIPTION

This function stops the ADC sequencer. All sequencer channel configurations are still valid after stopping.

### EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     result;

/*
** Stop the sequencer
*/
printf("Stop ADC Sequencer ... ");
result = ioctl(    fd,
                   TPMC851_IOC_ADC_SEQSTOP,
                   NULL);
if (result >= 1)
{
     printf("OK\n");
}
else
{
     /* process ioctl error */
}
```

### ERRORS

| Error Code | Description |
|---|---|
| EACCES | The ADC sequencer is not running |

## 3.3.5 TPMC851_IOC_ADC_SEQREAD

### NAME

TPMC851_IOC_ADC_SEQREAD – Read values from ADC sequencer buffer

### DESCRIPTION

This function reads values from the internal ADC sequencer buffer.

A pointer to the read structure (*TPMC851_ADC_SEQREAD_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        long                    seqState;
        short                   buffer[32];
} TPMC851_ADC_SEQREAD_BUF;
```

*seqState*

Displays the sequencer state. This is an ored value of the following status flags.

| Flag | Description |
|---|---|
| TPMC851_SF_SEQACTIVE | If set the ADC sequencer is started.<br>If not set, the ADC sequencer stopped. |
| TPMC851_SF_SEQOVERFLOWERR | If set the ADC sequencer has detected an overflow error. (Hardware detected) |
| TPMC851_SF_SEQTIMERERROR | If set the ADC sequencer has detected a timer error. (Hardware detected) |
| TPMC851_SF_SEQIRAMERROR | If set the ADC sequencer has detected an instruction RAM error. (Hardware detected) |
| TPMC851_SF_SEQFIFOOVERFLOW | If set the internal FIFO (buffer) has overrun. Data got lost. |

*buffer*

This array contains data from the activated channels. Only the previously selected channels contain valid data. Array index 0 contains values from channel 1, array index 1 corresponds to channel 2 and so on.

## EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     result;
TPMC851_ADC_SEQREAD_BUF adcSeqReadBuf;

/*
** Read values from internal sequencer buffer (1000 times)
** assuming that channel 1 and 3 are enabled.
*/
for (cycle=0; cycle<1000; cycle++)
{
     result = ioctl(    fd,
                        TPMC851_IOC_ADC_SEQREAD,
                        (char*)&adcSeqReadBuf);
     if (result >= 1)
     {
          printf(“   Channel(1)=%d     Channel(3)=%d  \n”,
                    adcSeqReadBuf.buffer[0],
                    adcSeqReadBuf.buffer[2] );
     }
     if (result == ENODATA)
     {
          /* wait a short time for new data to arrive */
     }
}
```

## ERRORS

| Error Code | Description |
|---|---|
| EACCES | The ADC sequencer is not running |
| ENODATA | No data is available inside the internal buffer |

## 3.3.6 TPMC851_IOC_DAC_WRITE

### NAME

TPMC851_IOC_DAC_WRITE – Write value to DAC channel

### DESCRIPTION

This function writes a value to the DAC register.

**The DAC sequencer must be stopped for single DAC writes.**

A pointer to the write structure (*TPMC851_DAC_WRITE_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        int             channel;
        unsigned long   flags;
        short           dacValue;
} TPMC851_DAC_WRITE_BUF;
```

*channel*

Specifies the DAC channel number. Valid values are 1..8.

*flags*

Is an ORed value of the following flags:

| Flag | Description |
|---|---|
| TPMC851_F_CORR | If set the function will correct the dacValue before writing to DAC channel. Factory set and module dependent correction data is used for correction.<br>If not set, dacValue is written to the DAC channel. |
| TPMC851_F_NOUPDATE | If set the DACs will not update after changing the DAC value. The output voltage will change with the next write with unset TPMC851_F_NOUPDATE flag.<br>If not set the DAC will immediately convert and output the new voltage. |

*dacValue*

This value is written to the DAC channel.

### EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     result;
TPMC851_DAC_WRITE_BUF   dacWriteBuf;

/*
** Write uncorrected 0x4000 to DAC channel 5, immediate convert
*/
dacWriteBuf.channel     = 5;
dacWriteBuf.flags       = 0;
dacWriteBuf.dacValue    = 0x4000;

printf("Write to DAC ... ");
result = ioctl(    fd,
                   TPMC851_IOC_DAC_WRITE,
                   (char*)&dacWriteBuf);
if (result >= 1)
{
     printf("OK\n");
}
else
{
     /* process ioctl error */
}
```

### ERRORS

| Error Code | Description |
|---|---|
| EBUSY | The DAC sequencer is currently running |
| ECHRNG | Specified channel is invalid |
| EINVAL | Specified gain level is invalid |

## 3.3.7 TPMC851_IOC_DAC_SEQCONFIG

### NAME

TPMC851_IOC_DAC_SEQCONFIG – Configure DAC sequencer channel

### DESCRIPTION

This function enables and configures, or disables a DAC channel for sequence use.

**The DAC sequencer must be stopped to execute this function.**

A pointer to the configuration structure (*TPMC851_DAC_SEQCONFIG_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
    int             channel;
    int             enable;
    unsigned long   flags;
} TPMC851_DAC_SEQCONFIG_BUF;
```

*channel*

Specifies the DAC channel number to configure. Valid values are 1..8.

*enable*

Specifies if the channel shall be used in sequencer mode or not. (0 disables the channel, any other value will enable the channel)

*flags*

Is an ored value of the following flags:

| Flag | Description |
|---|---|
| TPMC851_F_CORR | If set the function will correct the dacValue before writing to DAC channel. Factory set and module dependent correction data is used for correction.<br>If not set, dacValue is written to the DAC channel. |

## EXAMPLE

```
#include “tpmc851.h”

int                          fd;
int                          result;
TPMC851_DAC_SEQCONFIG_BUF    dacSeqConfBuf;

/*
** Configure DAC channel 1, using corrected data
** when the sequencer is running
*/
dacSeqConfBuf.channel   = 1;
dacSeqConfBuf.enable    = TRUE;
dacSeqConfBuf.flags     = TPMC851_F_CORR;

printf("Configure channel for Sequencer ... ");
result = ioctl(    fd,
                   TPMC851_IOC_DAC_SEQCONFIG,
                   (char*)&dacSeqConfBuf);
if (result >= 1)
{
     printf("OK\n");
}
else
{
     /* process ioctl error */
}
```

## ERRORS

| Error Code | Description |
|---|---|
| EBUSY | The DAC sequencer is currently running |
| ECHRNG | Specified channel is invalid |
| EINVAL | Specified gain level is invalid |

## 3.3.8 TPMC851_IOC_DAC_SEQSTART

### NAME

TPMC851_IOC_DAC_SEQSTART – Start DAC sequencer

### DESCRIPTION

This function configures the DAC sequencer time and starts the DAC sequencer.

A pointer to the start structure (*TPMC851_DAC_SEQSTART_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        unsigned short          cycTime;
        unsigned long           flags;
        long                    bufSize;
        short                   *buffer;
} TPMC851_DAC_SEQSTART_BUF;
```

*cycTime*

Specifies the DAC sequencer cycle time. The sequencer time is specified in 100µs steps. With a value of 0, the “Sequencer Continuous Mode” is selected.

*flags*

Is an ORed value of the following flags:

| Flag | Description |
|---|---|
| TPMC851_F_EXTTRIGSRC | If set the DAC sequencer is trigger with digital I/O line 1.<br>If not set, the DAC sequencer uses the DAC cycle counter. |
| TPMC851_F_EXTTRIGOUT | If set the DAC trigger is used as output on digital I/O line 1. |
| TPMC851_F_DACSEQREPEAT | If set the DAC will repeat data when the end of the buffer is reached, the TPMC851_SF_SEQFIFOUNDERFLOW error will be suppressed. |

***TPMC851_F_EXTTRIGSRC* and *TPMC851_F_EXTTRIGOUT cannot be used at the same time.***

*bufSize*

This value specifies the size of the DAC sequencer FIFO. The value is specified in number of data words.

*buffer*

Pointer to a buffer of short values used for initial DAC sequencer data.
The DAC data is stored by the application into this buffer and copied to the drivers FIFO. The assignment from data to channel is done as follows. The first data will be used for the lowest enabled channel, the second from the next enabled channel and so on. There will be no data used for disabled channels. If the end of *buffer* is reached the next data will be read again from the beginning of the buffer.

Example:
Enabled channels: 1, 2, 5
Buffer size: 10
The table shows the index the data is used to for channel and cycle.

| Sequencer Cycle | Channel 1 | Channel 2 | Channel 3 |
|---|---|---|---|
| **1st** | 0 | 1 | 2 |
| **2nd** | 3 | 4 | 5 |
| **3rd** | 6 | 7 | 8 |
| **4th** | 9 | 0 | 1 |
| **5th** | 2 | 3 | 4 |
| **…** | … | … | … |

## EXAMPLE

```
#include “tpmc851.h”

int                          fd;
int                          result;
TPMC851_DAC_SEQSTART_BUF     dacSeqStartBuf;
short                        buffer[1000];

/*
** Start sequencer with a buffer of 100 word and a cycle time of 100 ms,
** do not use external trigger
*/
/* Fill buffer */
buffer[0] = …;
buffer[1] = …;
buffer[2] = …;

dacSeqStartBuf.cycTime       = 1000;
dacSeqStartBuf.flags         = TPMC851_F_DACSEQREPEAT;
dacSeqStartBuf.bufSize       = 1000;
dacSeqStartBuf.buffer        = buffer;

…
```

…

```
printf("Start DAC Sequencer ... ");
result = ioctl(    fd,
                   TPMC851_IOC_DAC_SEQSTART,
                   (char*)&dacSeqStartBuf);
if (result >= 1)
{
     printf("OK\n");
}
else
{
     /* process ioctl error */
}
```

## ERRORS

| Error Code | Description |
|---|---|
| EBUSY | The DAC sequencer is already running |
| EINVAL | Specified flags are invalid |
| ENOMEM | No memory is available to allocate the internal buffer |

## 3.3.9 TPMC851_IOC_DAC_SEQSTOP

### NAME

TPMC851_IOC_DAC_SEQSTOP – Stop DAC sequencer

### DESCRIPTION

This function stops the DAC sequencer. All sequencer channel configurations are still valid after stopping.

### EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     result;

/*
** Stop the sequencer
*/
printf("Stop DAC Sequencer ... ");
result = ioctl(    fd,
                   TPMC851_IOC_DAC_SEQSTOP,
                   NULL);
if (result >= 1)
{
     printf("OK\n");
}
else
{
     /* process ioctl error */
}
```

### ERRORS

| Error Code | Description |
|---|---|
| EACCES | The DAC sequencer is not running |

### 3.3.10 TPMC851_IOC_DAC_SEQWRITE

#### NAME

TPMC851_IOC_DAC_SEQWRITE – Write values to DAC sequencer buffer

#### DESCRIPTION

This function writes values to the internal DAC sequencer buffer.

A pointer to the write structure (*TPMC851_DAC_SEQWRITE_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        long            bufSize;
        short           *buffer;
} TPMC851_DAC_SEQWRITE_BUF;
```

*bufSize*

This value specifies the size of the data buffer. The driver will only accept buffer sizes smaller or equal to the free number of element in the drivers FIFO. The number of free elements can be read with *TPMC851_IOC_DAC_SEQSTATE*.

*buffer*

This pointer points the buffer containing the new DAC data values for the activated channels. The data is supplied in the way as described in *TPMC851_IOC_DAC_SEQSTART*.

#### EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         result;
TPMC851_DAC_SEQWRITE_BUF    dacSeqWriteBuf;
short                       buffer[100];

/*
** Fill up 100 data values
*/
/* fill first cycle */
buffer[0] = …;
buffer[1] = …;
buffer[2] = …;

…
```

…

```
dacSeqWriteBuf.bufSize  = 100;
dacSeqWriteBuf.buffer   = buffer;
result = ioctl(    fd,
                   TPMC851_IOC_DAC_SEQWRITE,
                   (char*)&dacSeqWriteBuf);
if (result >= 1)
{
     /* OK, FIFO filled up */
}
else
{
     /* Fillung up failed */
}
```

## ERRORS

| Error Code | Description |
|---|---|
| EACCES | The DAC sequencer is not running |
| EINVAL | Invalid buffer size specified |
| EFAULT | Additional: There is not enough space in FIFO to copy the supplied data buffer |

## 3.3.11 TPMC851_IOC_DAC_SEQSTATE

### NAME

TPMC851_IOC_DAC_SEQSTATE – Get DAC sequencer and information

### DESCRIPTION

This function reads the state and number of free elements of the DAC sequencer.

A pointer to the state structure (*TPMC851_DAC_SEQSTATE_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        unsigned long           state;
        short                   freeElems;
} TPMC851_DAC_SEQSTATE_BUF;
```

*state*

This value returns the actual state of the DAC sequencer. The following flags can be ored in the value:

| Flag | Description |
|---|---|
| TPMC851_SF_SEQACTIVE | If set the DAC sequencer is started.<br>If not set, the DAC sequencer stopped. |
| TPMC851_SF_SEQUNDERFLOWERR | If set the DAC sequencer has detected an underrun error. (Hardware detected) |
| TPMC851_SF_SEQFIFOUNDERFLOW | If set the application supplied FIFO (buffer) is empty and the sequencer could not write new data. |

*freeElems*

This value returns the number of free data elements in the DAC sequencer FIFO.

### EXAMPLE

```
#include “tpmc851.h”

int                             fd;
int                             result;
TPMC851_DAC_SEQSTATE_BUF        dacSeqStatBuf;

…
```

```
…

/*
** read DAC sequencer state
*/
result = ioctl(    fd,
                   TPMC851_IOC_DAC_SEQSTATE,
                   (char*)&dacSeqStatBuf);
if (result >= 1)
{
     /* OK */
     printf (“State: %Xh, free: %d\n”,
               dacSeqStatBuf.state,
               dacSeqStatBuf.freeElems);
}
else
{
     /* Failed */
}
```

### 3.3.12 TPMC851_IOC_IO_READ

#### NAME

TPMC851_IOC_IO_READ – Read from digital I/O

#### DESCRIPTION

This function reads the current value of the digital I/O input. Only bits previously configured to *input* are valid.

A pointer to the read structure (*TPMC851_IO_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
      unsigned short          value;
} TPMC851_IO_BUF;
```

*value*

Returns the current digital I/O input value.

#### EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     result;
TPMC851_IO_BUF          ioBuf;

/* Read I/O input value */
printf("Read I/O input value ... ");
result = ioctl(    fd,
                   TPMC851_IOC_IO_READ,
                   (char*)&ioBuf);
if (result >= 1)
{
     printf("    I/O input: %04X", ioBuf.value);
}
else
{
     /* process ioctl error */
}
```

### 3.3.13 TPMC851_IOC_IO_WRITE

#### NAME

TPMC851_IOC_IO_WRITE – Write to digital I/O

#### DESCRIPTION

This function writes a value to the digital I/O output. Only bits previously configured to *output* are valid.

A pointer to the write structure (*TPMC851_IO_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
      unsigned short          value;
} TPMC851_IO_BUF;
```

*value*

Specifies the new digital I/O output value.

#### EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     result;
TPMC851_IO_BUF          ioBuf;

/* Write 0x1234 to I/O output */
ioBuf.value = 0x1234;
printf("Write I/O output value ... ");
result = ioctl(    fd,
                   TPMC851_IOC_IO_WRITE,
                   (char*)&ioBuf);
if (result >= 1)
{
     printf("OK\n);
}
else
{
     /* process ioctl error */
}
```

## 3.3.14 TPMC851_IOC_IO_EVENTWAIT

### NAME

TPMC851_IOC_IO_EVENTWAIT – Wait for digital I/O event

### DESCRIPTION

This function waits for an I/O input event.

A pointer to the event structure (*TPMC851_IO_EVENTWAIT_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        int             ioLine;
        unsigned long   flags;
        long            timeout;
} TPMC851_IO_EVENTWAIT_BUF;
```

*ioLine*

Specifies the digital I/O line where the event shall occur. Valid values are 0..15.

*flags*

Specifies the event that shall occur. This is an ORed value of the following flags:

| Flag | Description |
|---|---|
| TPMC851_F_HI2LOTRANS | If set, the function will return after a high to low transition occurs. |
| TPMC851_F_LO2HITRANS | If set, the function will return after a low to high transition occurs. |

**At least one flag must be specified.**

*timeout*

Specifies the maximum time the function will wait for the specified event. The time is specified in ticks. Specify 0 to wait indefinitely for the given event.

### EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         result;
TPMC851_IO_EVENTWAIT_BUF    waitBuf;

/*
** Wait for a transition on I/O line 12 (max wait 10000 ticks)
*/
waitBuf.ioLine =   12;
waitBuf.flags =    TPMC851_F_HI2LOTRANS | TPMC851_F_LO2HITRANS;
waitBuf.timeout =  10000;

printf("Wait for an I/O event ... ");
result = ioctl(    fd,
                   TPMC851_IOC_IO_EVENTWAIT,
                   (char*)&waitBuf);
if (result >= 1)
{
     printf("OK\n);
}
else
{
     /* process ioctl error */
}
```

### ERRORS

| Error Code | Description |
|---|---|
| ENOSPC | No space is available for new wait requests |
| ETIMEDOUT | The timer expired |

### 3.3.15 TPMC851_IOC_IO_CONFIG

#### NAME

TPMC851_IOC_IO_CONFIG – Configure digital I/O direction

#### DESCRIPTION

This function configures digital I/O lines to input or output (direction).

A pointer to the configure structure (*TPMC851_IO_CONF_BUF*) is passed by the parameter *arg* to the driver.

typedef struct
{
    unsigned short  direction;
} TPMC851_IO_CONF_BUF;

*direction*

Specifies the new direction for digital I/O. A bit set to 1 enables output, a 0 means that the I/O line is input.

#### EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         result;
TPMC851_IO_CONF_BUF         ioConfBuf;

/* Enable line 0,2,8,9 for output, all other lines are input */
ioConfBuf.direction = (1 << 0) | (1 << 2) | (1 << 8) | (1 << 9);
printf("Set new I/O configuration ... ");
result = ioctl(    fd,
                   TPMC851_IOC_IO_CONFIG,
                   (char*)&ioConfBuf);
if (result >= 1)
{
     printf("OK\n);
}
else
{
     /* process ioctl error */
}
```

### 3.3.16 TPMC851_IOC_IO_DEBCONFIG

#### NAME

TPMC851_IOC_IO_DEBCONFIG – Configure digital input debouncer

#### DESCRIPTION

This function configures the digital I/O debouncing circuit.

A pointer to the configure structure (*TPMC851_IO_DEBCONF_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        unsigned short          enableMask;
        unsigned short          debTime;
} TPMC851_IO_DEBCONF_BUF;
```

*enableMask*

Specifies digital I/O lines which shall observed by the debouncer. A bit set to 1 enables the debouncer, and a 0 disables the debouncer for the adequate I/O line.

*debTime*

Specifies the debounce time. The time is specified in 100ns steps.

#### EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     result;
TPMC851_IO_DEBCONF_BUF  ioDebConfBuf;

/*
** Enable Debouncer for line 0 and 2 (debounce time 1ms)
*/
ioDebConfBuf.enableMask =    (1 << 0) | (1 << 2);
ioDebConfBuf.debTime =       10000;

…
```

```
…

printf("Set debouncer configuration ... ");
result = ioctl(    fd,
                   TPMC851_IOC_IO_DEBCONFIG,
                   (char*)&ioDebConfBuf);
if (result >= 1)
{
     printf("OK\n);
}
else
{
     /* process ioctl error */
}
```

## 3.3.17 TPMC851_IOC_CNT_READ

### NAME

TPMC851_IOC_CNT_READ – Read value from counter/timer

### DESCRIPTION

This function reads the current value of the counter/timer.

A pointer to the read structure (*TPMC851_CNT_READ_ BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        unsigned long           count;
        unsigned long           state;
} TPMC851_CNT_READ_BUF;
```

*count*

Returns the current counter value.

*state*

Returns the counter state. If possible the flags are cleared after read. This is an ORed value of the following flags.

| Flag | Description |
|---|---|
| TPMC851_SF_CNTBORROW | Counter borrow bit set (actual state) |
| TPMC851_SF_CNTCARRY | Counter carry bit set (actual state) |
| TPMC851_SF_CNTMATCH | Counter match event has occurred since last read. |
| TPMC851_SF_CNTSIGN | Counter sign bit (actual state) |
| TPMC851_SF_CNTDIRECTION | If set, counter direction is upward.<br>If not set, counter direction is downward. |
| TPMC851_SF_CNTLATCH | Counter value has been latched. |
| TPMC851_SF_CNTLATCHOVERFLOW | Counter latch overflow has occurred. |
| TPMC851_SF_CNTSNGLCYC | Counter Single Cycle is active |

### EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         result;
TPMC851_CNT_READ_BUF        cntBuf;

/* Read counter value */
printf("Read counter ... ");
result = ioctl(    fd,
                   TPMC851_IOC_CNT_READ,
                   (char*)&cntBuf);
if (result >= 1)
{
     printf("    Counter: %ld", cntBuf.counter);
     printf("    State:   %lXh", cntBuf.state);
}
else
{
     /* process ioctl error */
}
```

### 3.3.18 TPMC851_IOC_CNT_MATCHWAIT

#### NAME

TPMC851_IOC_CNT_MATCHWAIT – Wait for counter match event

#### DESCRIPTION

This function waits for a counter match event. This event occurs if the current timer/counter value matches the previously setup counter-match-register.

A pointer to the wait structure (*TPMC851_CNT_WAIT_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
    long                timeout;
} TPMC851_CNT_WAIT_BUF;
```

*timeout*

Specifies the maximum time the function will wait for the match event. The time is specified in ticks. Specify 0 to wait indefinitely for the given event.

#### EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         result;
TPMC851_CNT_WAIT_BUF        cntWaitBuf;

/*
** Wait for counter match event (max wait 10000 ticks)
*/
cntWaitBuf.timeout =    10000;

…
```

…

```
printf("Wait for counter match event ... ");
result = ioctl(    fd,
                   TPMC851_IOC_CNT_MATCHWAIT,
                   (char*)&cntWaitBuf);
if (result >= 1)
{
     printf("OK\n);
}
else
{
     /* process ioctl error */
}
```

## ERRORS

| Error Code | Description |
|---|---|
| ENOSPC | No space is available for new wait requests |
| ETIMEDOUT | The timer expired |

## 3.3.19 TPMC851_IOC_CNT_CTRLWAIT

### NAME

TPMC851_IOC_CNT_CTRLWAIT – Wait for counter control event

### DESCRIPTION

This function waits for a counter control event. The event to wait for is chosen with ioctl() function *TPMC851_IOC_CNT_CONFIG* specifying the parameter *controlMode*.

A pointer to the wait structure (*TPMC851_CNT_WAIT_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        long                    timeout;
} TPMC851_CNT_WAIT_BUF;
```

*timeout*

Specifies the maximum time the function will wait for the match event. The time is specified in ticks. Specify 0 to wait indefinitely for the given event.

### EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         result;
TPMC851_CNT_WAIT_BUF        cntWaitBuf;

/*
** Wait for counter control event (max wait 10000 ticks)
*/
cntWaitBuf.timeout =    10000;

…
```

…

```
printf("Wait for counter control event ... ");
result = ioctl(    fd,
                   TPMC851_IOC_CNT_CTRLWAIT,
                   (char*)&cntWaitBuf);
if (result >= 1)
{
     printf("OK\n);
}
else
{
     /* process ioctl error */
}
```

## ERRORS

| Error Code | Description |
|---|---|
| ENOSPC | No space is available for new wait requests |
| ETIMEDOUT | The timer expired |

## 3.3.20 TPMC851_IOC_CNT_CONFIG

### NAME

TPMC851_IOC_CNT_CONFIG – Configure counter

### DESCRIPTION

This function configures the counter.

A pointer to the configuration structure (*TPMC851_CNT_CONFIG_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        unsigned long       inputMode;
        int                 clockDivider;
        unsigned long       countMode;
        unsigned long       controlMode;
        unsigned long       invFlags;
} TPMC851_CNT_CONFIG_BUF;
```

*inputMode*

Specifies the counter input mode. The following modes are defined and valid:

| Flag | Description |
|---|---|
| TPMC851_M_CNTIN_DISABLE | Counter disabled |
| TPMC851_M_CNTIN_TIMERUP | Timer Mode Up |
| TPMC851_M_CNTIN_TIMERDOWN | Timer Mode Down |
| TPMC851_M_CNTIN_DIRCOUNT | Direction Count |
| TPMC851_M_CNTIN_UPDOWNCOUNT | Up/Down Count |
| TPMC851_M_CNTIN_QUAD1X | Quadrature Count 1x |
| TPMC851_M_CNTIN_QUAD2X | Quadrature Count 2x |
| TPMC851_M_CNTIN_QUAD4X | Quadrature Count 4x |

*clockDivider*

Specifies the clock divider value. Allowed clock divider values are 1 (40MHz), 2 (20MHz), 4 (10MHz) and 8 (5MHz).

*countMode*

Specifies the count mode. The following modes are defined and valid:

| Flag | Description |
|---|---|
| TPMC851_M_CNT_CYCLE | Cycling Counter |
| TPMC851_M_CNT_DIVN | Divide-by-N |
| TPMC851_M_CNT_SINGLE | Single Cycle |

*controlMode*

Specifies the counter control mode. These events can generate counter control events. The following modes are defined and valid:

| Flag | Description |
|---|---|
| TPMC851_M_CNTCTRL_NONE | No Control Mode |
| TPMC851_M_CNTCTRL_LOAD | Load Mode |
| TPMC851_M_CNTCTRL_LATCH | Latch Mode |
| TPMC851_M_CNTCTRL_GATE | Gate Mode |
| TPMC851_M_CNTCTRL_RESET | Reset Mode |

*invFlags*

Specifies if counter input lines shall be inverted or not. This is an ored value of the following flags:

| Flag | Description |
|---|---|
| TPMC851_F_CNTINVINP2 | If set, input line 2 is low active<br>If not set, input line 2 is high active |
| TPMC851_F_CNTINVINP3 | If set, input line 3 is low active<br>If not set, input line 3 is high active |
| TPMC851_F_CNTINVINP4 | If set, input line 4 is low active<br>If not set, input line 4 is high active |

## EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         result;
TPMC851_CNT_CONFIG_BUF      cntConfBuf;

/*
** Setup counter for direction count, clock divider 1, cycling count,
** no control mode and all line high active
*/
cntConfBuf. inputMode =     TPMC851_M_CNTIN_DIRCOUNT;
cntConfBuf. clockDivider =  1;
cntConfBuf. countMode =     TPMC851_M_CNT_CYCLE;
cntConfBuf. controlMode =   TPMC851_M_CNTCTRL_NONE;
cntConfBuf. invFlags =      0;

…
```

…

```
printf("Set counter configuration ... ");
result = ioctl(    fd,
                   TPMC851_IOC_CNT_CONFIG,
                   (char*)&cntConfBuf);
if (result >= 1)
{
     printf("OK\n");
}
else
{
     /* process ioctl error */
}
```

## ERRORS

| Error Code | Description |
|---|---|
| EINVAL | Specified flag or mode is invalid. |

### 3.3.21 TPMC851_IOC_CNT_RESET

#### NAME

TPMC851_IOC_CNT_RESET – Reset counter value

#### DESCRIPTION

This function resets the counter value to 0x00000000.

#### EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     result;

/* Reset counter */
printf("Reset counter ... ");
result = ioctl(    fd,
                   TPMC851_IOC_CNT_RESET,
                   NULL);
if (result >= 1)
{
     printf("OK\n");
}
else
{
     /* process ioctl error */
}
```

## 3.3.22 TPMC851_IOC_CNT_SETPRELD

### NAME

TPMC851_IOC_CNT_SETPRELD – Set counter preload value

### DESCRIPTION

This function sets the counter preload register.

A pointer to the preload structure (*TPMC851_CNT_SETPRELD_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        unsigned long           value;
        unsigned long           flags;
} TPMC851_CNT_SETPRELD_BUF;
```

*value*

Specifies the new counter preload value.

*flags*

Is an ORed value of the following flags:

| Flag | Description |
|---|---|
| TPMC851_F_IMMPRELOAD | If set, the function will immediate load the preload value into the counter<br>If not set, preload value will be used for the next preload condition. |

### EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         result;
TPMC851_CNT_SETPRELD_BUF    cntPrldBuf;

…
```

```
…

/*
** Immediately load 0x11223344 into the counter and preload register
*/
cntPrldBuf.value        = 0x11223344;
cntPrldBuf.flags        = TPMC851_F_IMMPRELOAD;

printf("Set preload value ... ");
result = ioctl(    fd,
                   TPMC851_IOC_CNT_SETPRELD,
                   (char*)&cntPrldBuf);
if (result >= 1)
{
     printf("OK\n");
}
else
{
     /* process ioctl error */
}
```

### 3.3.23 TPMC851_IOC_CNT_SETMATCH

#### NAME

TPMC851_IOC_CNT_SETMATCH – Set counter match value

#### DESCRIPTION

This function sets the counter match register. If counter and match value are the same, a match event occurs. The driver can wait for this event (refer to ioctl function *TPMC851_IOC_CNT_MATCHWAIT*).

A pointer to the match structure (*TPMC851_CNT_SETMATCH_BUF*) is passed by the parameter *arg* to the driver.

typedef struct
{
    unsigned long    value;
} TPMC851_CNT_SETMATCH_BUF;

*value*

Specifies the new counter match value.

#### EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         result;
TPMC851_CNT_SETMATCH_BUF    cntMatchBuf;

/* Set match value to 0x10000 */
cntMatchBuf.value       = 0x10000;
printf("Set counter match value ... ");
result = ioctl(    fd,
                   TPMC851_IOC_CNT_SETMATCH,
                   (char*)&cntMatchBuf);
if (result >= 1)
{
     printf("OK\n");
}
else
{
     /* process ioctl error */
}
```

# 4 Diagnostic

If the TPMC851 does not work properly it is helpful to get some status information from the driver respective kernel.

Check TPMC851 PCI information with lspci, which displays the PCI location of the TPMC851 and its addresses.

```
lspci –v
…
04:01.0 Signal processing controller: TEWS Technologies GmbH Device 0353
        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=512]
        Memory at feb9f400 (32-bit, non-prefetchable) [size=64]
        Memory at feb9f000 (32-bit, non-prefetchable) [size=64]
        Kernel driver in use: TEWS TECHNOLOGIES TPMC851 AD-DA-Converter,
Digital IO and Counter
        Kernel modules: tpmc851drv
…
```

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 TPMC851 driver (see also the proc man pages).

```
cat /proc/devices
Character devices:
  1 mem
  2 pty
…
162 raw
254 tpmc851drv
…

# cat /proc/iomem
…
80000000-ffffffff : PCI Bus 0000:00
  feb00000-febfffff : PCI Bus 0000:04
    feb9f000-feb9f03f : 0000:04:01.0
      feb9f000-feb9f03f : TPMC851
    feb9f400-feb9f43f : 0000:04:01.0
      feb9f400-feb9f43f : TPMC851
    feb9f800-feb9f9ff : 0000:04:01.0
      feb9f800-feb9f9ff : TPMC851
    feb9fc00-feb9fc7f : 0000:04:01.0
…
```