The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC851-SW-72

LynxOS Device Driver

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

Version 2.0.x

User Manual

Issue 2.0.0

December 2006

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
25469 Halstenbek, Germany
www.tews.com
Phone: +49 (0) 4101 4058 0
Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com

TEWS TECHNOLOGIES LLC
9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com
Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

TPMC851-SW-72

LynxOS Device Driver

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

Issue	Description	Date
1.0.0	First Issue	February 21, 2005
2.0.0	DAC Sequencer Start structure modified, General Revision	December 13, 2006

Table of Content

1 Introduction

The TPMC851-SW-72 LynxOS device driver allows the operation of a TPMC851 Multifunction I/O PMC on LynxOS platforms with DRM based PCI interface.

The standard file (I/O) functions (open, close, ioctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and configuration operations.

The TPMC851-SW-72 device driver supports the following features:

- Reading an ADC input value from a specified channel
- Configuring and using the ADC input sequencer
- Setting a DAC output value to a specified channel
- Configuring and using the DAC output sequencer
- Reading from digital I/O input register
- Writing to digital I/O output register
- Waiting for input I/O input event (high, low or any transition on input line)
- Configuring I/O line direction
- Reading counter value
- Reset counter value
- Setting counter preload and match value
- Configuring counter mode
- Wait for counter match and control event

The TPMC851-SW-72 device driver supports the modules listed below:

TPMC851	Multifunction I/O (16 bit ADC/DAC, TTL I/O, Counter)	(PMC)

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

TPMC851 User manual

TPMC851 Engineering Manual

2 Installation

The directory TPMC851-SW-72 on the distribution media contains the following files:

TPMC851-SW-72-2.0.0.pdf	This manual in PDF format
TPMC851-SW-72-SRC.tar	Device Driver and Example sources
ChangeLog.txt	Release history
Release.txt	Release information

The TAR archive TPMC851-SW-72-SRC.tar contains the following files and directories:

tpmc851/tpmc851.c	Driver source code
tpmc851/tpmc851.h	Definitions and data structures for driver and application
tpmc851/tpmc851def.h	Definitions and data structures for the driver
tpmc851/tpmc851_info.c	Device information definition
tpmc851/tpmc851_info.h	Device information definition header
tpmc851/tpmc851.cfg	Driver configuration file include
tpmc851/tpmc851.import	Linker import file
tpmc851/Makefile	Device driver make file
tpmc851/example/tpmc851exa.c	Example application source
tpmc851/example/Makefile	Example application make file

In order to perform a driver installation first extract the TAR file to a temporary directory then copy the following files to their target directories:

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/tpmc851 or /sys/drivers.cpci_x86/tpmc851

2. Copy the following files to this directory:
   - tpmc851.c
   - tpmc851def.h
   - tpmc851.import
   - Makefile

3. Copy tpmc851.h to /usr/include/

4. Copy tpmc851_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tpmc851_info.h to /sys/dheaders/

6. Copy tpmc851.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform. For example: /sys/cfg.ppc or /sys/cfg.x86 ....

2.1 Device Driver Installation

The two methods of driver installation are as follows:

- Static Installation
- Dynamic Installation (only native LynxOS systems)

2.1.1 Static Installation

With this method, the driver object code is linked with the kernel routines and is installed during system start-up.

2.1.1.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tpmc851, where xxx represents the BSP that supports the target hardware.
2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

2.1.1.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).
2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = ... tpmc851_info.x
   ```

   And at the end of the Makefile

   ```
   tpmc851_info.o:$(DHEADERS)/tpmc851_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

2.1.1.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os/ respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry at the end of this file.

   ```
   I:tpmc851.cfg
   ```

2.1.1.4 Rebuild the Kernel

1. Change to the directory /sys/lynx.os/ (/sys/bsp.xxx)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

   The N flag instructs init to run mknod and create all the nodes mentioned in the new nodetab.

4. After reboot you should find the following new devices (depends on the device configuration): /dev/tp851a, /dev/tp851b, /dev/tp851c, …

2.1.2 Dynamic Installation

This method allows you to install the driver after the operating system is booted. The driver object code is attached to the end of the kernel image and the operating system dynamically adds this driver to its internal structures. The driver can also be removed dynamically.

2.1.2.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tpmc851, where xxx represents the BSP that supports the target hardware.
2. To make the dynamic link-able driver enter:

   ```
   make dldd
   ```

2.1.2.2 Create Device Information Declaration

1. Change to the directory /sys/drivers.xxx/tpmc851, where xxx represents the BSP that supports the target hardware.
2. To create a device definition file for the major device (this works only on native system)

   ```
   make t851info
   ```

3. To install the driver enter:

   ```
   drinstall –c tpmc851.obj
   ```

   If successful, drinstall returns a unique <driver-ID>
4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> t851info
   ```

   The <driver-ID> is returned by the drinstall command
5. To create nodes for the devices enter:

   ```
   mknod /dev/tp851a c <major_no> 0
   ```

   …

The <major_no> is returned by the devinstall command.

If all steps are completed successfully the TPMC851 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TPMC851 device enter the following commands:

```
devinstall –u –c <device-ID>
drinstall –u <driver-ID>
```

2.1.3 Device Information Definition File

The device information definition contains information necessary to install the TPMC851 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tpmc851_info.h*.

This structure contains the following parameter:

PCIBusNumber Contains the PCI bus number at which the TPMC851 is connected. Valid bus numbers are in range from 0 to 255.

PCIDeviceNumber Contains the device number (slot) at which the TPMC851 is connected. Valid device numbers are in range from 0 to 31.

If both PCIBusNumber and PCIDeviceNumber are –1 then the driver will auto scan for the TPMC851 device. The first device found in the scan order will be allocated by the driver for this major device.

Already allocated devices can't be allocated twice. This is important to know if there are more than one TMPC851 major devices.

A device information definition is unique for every TPMC851 major device. The file *tpmc851_info.c* on the distribution disk contains two device information declarations, **tp851a_info** for the first major device and **tp851b_info** for the second major device.

If the driver should support more than two major devices it is necessary to copy and paste an existing declaration and rename it with a unique name, for example **tp851c_info**, **tp851d_info** and so on.

It is also necessary to modify the device and driver configuration file, respectively the configuration include file *tpmc851.cfg*.

The following device declaration information uses the auto find method to detect the TPMC851 module on PCI bus.

```
TP851_INFO tp851a_info = {

    -1,                 /*  Auto find the TPMC851 on any PCI bus   */
    -1,

};
```

2.1.4 Configuration File: CONFIG.TBL

The device and driver configuration file CONFIG.TBL contains entries for device drivers and its major and minor device declarations. Each time the system is rebuild, the config utility read this file and produces a new set of driver and device configuration tables and a corresponding nodetab.

To install the TPMC851 driver and devices into the LynxOS system, the configuration include file tpmc851.cfg must be included in the CONFIG.TBL (see also 2.1.1.3).

The file tpmc851.cfg on the distribution disk contains the driver entry (*C:tpmc851:*\...) and a major device entry (*D:TPMC851 1:t851a_info::*) with one minor device entry (*"N: tp851a"*).

If the driver should support more than one major device, the following entries for major and minor devices must be enabled by removing the comment character (#). By copy and paste an existing major and minor entry and renaming the new entries, it is possible to add any number of additional TPMC851 devices.

This example shows a driver entry with one major device and one minor device:

```
#    Format:
#    C:driver-name:open:close:read:write:select:control:install:uninstall
#    D:device-name:info-block-name:raw-partner-name
#    N:node-name:minor-dev

C:tpmc851:\
     :tp851open:tp851close:tp851read:tp851write:\
     ::tp851ioctl:tp851install:tp851uninstall
D:TPMC851 1:tp851a_info::
N:tp851a:0
```

The configuration above creates the following node in the /dev directory.

/dev/tp851a

3 TPMC851 Device Driver Programming

LynxOS system calls are all available directly to any C program. They are implemented as ordinary function calls to "glue" routines in the system library, which trap to the OS code.

Note that many system calls use data structures, which should be obtained in a program from appropriate header files. Necessary header files are listed with the system call synopsis.

3.1 open()

NAME

open() - open a file

SYNOPSIS

#include <sys/file.h>
#include <sys/types.h>
#include <fcntl.h>

int open (char *path, int oflags[, mode_t mode])

DESCRIPTION

Opens a file (TPMC851 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TPMC851 devices ***oflags*** must be set to **O_RDWR** to open the file for both reading and writing.

The ***mode*** argument is required only when a file is created. Because a TPMC851 device already exists this argument is ignored.

EXAMPLE

```
int fd

/* open the device named "/dev/tp851a" for I/O */
fd = open ("/dev/tp851a", O_RDWR);
if (!fd)
{
     /* handle error */
}
```

RETURNS

open returns a file descriptor number if successful, or –1 on error.

SEE ALSO

LynxOS System Call - open()

3.2 close()

NAME

close() – close a file

SYNOPSIS

int close(int fd)

DESCRIPTION

This function closes an opened device.

EXAMPLE

```
int  result;

/*
**   close the device
*/
result = close(fd);
if (result < 0)
{
     /* handle error */
}
```

RETURNS

close returns 0 (OK) if successful, or –1 on error

SEE ALSO

LynxOS System Call - close()

3.3 ioctl()

NAME

ioctl() – I/O device control

SYNOPSIS

#include <ioctl.h>
#include <tpmc851.h>

int ioctl (int fd, int request, char *arg)

DESCRIPTION

ioctl provides a way of sending special commands to a device driver. The call sends the value of **request** and the pointer **arg** to the device associated with the descriptor **fd**.

The following ioctl codes are supported by the driver and are defined in tpmc851.h:

Symbol	Meaning
TP851_C_ADC_READ	Read value from ADC channel
TP851_C_ADC_SEQCONFIG	Configure ADC sequencer channel
TP851_C_ADC_SEQSTART	Start ADC sequencer
TP851_C_ADC_SEQSTOP	Stop ADC sequencer
TP851_C_ADC_SEQREAD	Read values from ADC sequencer buffrer
TP851_C_DAC_WRITE	Write value to DAC channel
TP851_C_DAC_SEQCONFIG	Configure ADC sequencer channel
TP851_C_DAC_SEQSTART	Start ADC sequencer
TP851_C_DAC_SEQSTOP	Stop ADC sequencer
TP851_C_DAC_SEQWRITE	Write values to DAC sequencer buffer
TP851_C_IO_READ	Read from digital I/O
TP851_C_IO_WRITE	Write to digital I/O
TP851_C_IO_EVENTWAIT	Wait for I/O event
TP851_C_IO_CONFIG	Configure digital I/O
TP851_C_IO_DEBCONFIG	Configure digital I/O (input) debouncer
TP851_C_CNT_READ	Read value from counter/timer
TP851_C_CNT_MATCHWAIT	Wait for counter match event
TP851_C_CNT_CTRLWAIT	Wait for counter control event
TP851_C_CNT_CONFIG	Configure counter
TP851_C_CNT_RESET	Reset counter
TP851_C_CNT_SETPRELD	Set counter preload value
TP851_C_CNT_SETMATCH	Set counter match value

See behind for more detailed information on each control code.

RETURNS

ioctl returns 0 if successful, or –1 on error.

On error, *errno* will contain a standard error code (see also LynxOS System Call – ioctl) or a special error code. Function specific error codes will be described below with the function.

SEE ALSO

LynxOS System Call - ioctl().

3.3.1 TP851_C_ADC_READ

NAME

TP851_C_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 (*TP851_ADC_READ_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
    int             channel;
    int             gain;
    unsigned long   flags;
    short           adcValue;
} TP851_ADC_READ_BUF, *PTP851_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
TP851_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. If not set, the raw value read from the module will be returned in *adcValue*.
TP851_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). If not set the driver will use the automatic mode, which sets interface configuration, waits settling time and then starts the conversion.

TP851_F_DIFF	If set the input channel will be a differential input. 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;
TP851_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   = TP851_F_CORR | TP851_F_DIFF;

printf("Read from ADC ... ");
result = ioctl(    fd,
                   TP851_C_ADC_READ,
                   (char*)&adcReadBuf);
if (result == OK)
{
     printf("OK\n");
     printf("    ADC-value: %d", adcReadBuf.adcValue);
} else {
     /* process ioctl error */
}
```

Error Codes

EBUSY	The ADC sequencer is currently running.
ECHRNG	Specified channel is invalid.
EINVAL	Specified gain level is invalid.
ETIME	The ADC conversion timed out.
EINTR	Function was interrupted.

All other returned error codes are system error conditions.

3.3.2 TP851_C_ADC_SEQCONFIG

NAME

TP851_C_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 (*TP851_ADC_SEQCONFIG_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
    int             channel;
    int             enable;
    int             gain;
    unsigned long   flags;
} TP851_ADC_SEQCONFIG_BUF, *PTP851_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
TP851_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. If not set, the raw value read from the module will be returned.
TP851_F_DIFF	If set the input channel will be a differential input. If not set the input channel will be a single-ended input.

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;
TP851_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     = TP851_F_CORR;

printf("Configure channel for Sequencer ... ");
result = ioctl(    fd,
                   TP851_C_ADC_SEQCONFIG,
                   (char*)&adcSeqConfBuf);
if (result == OK)
{
     printf("OK\n");
} else {
     /* process ioctl error */
}
```

Error Codes

EBUSY	The ADC sequencer is currently running.
ECHRNG	Specified channel is invalid.
EINVAL	Specified flags or gain level is invalid.

All other returned error codes are system error conditions.

3.3.3 TP851_C_ADC_SEQSTART

NAME

TP851_C_ADC_SEQSTART – Start ADC sequencer

DESCRIPTION

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

A pointer to the start structure (*TP851_ADC_SEQSTART_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        unsigned short          cycTime;
        unsigned long           flags;
        long                    bufSize;
} TP851_ADC_SEQSTART_BUF, *PTP851_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
TP851_F_EXTTRIGSRC	If set the ADC sequencer is trigger with digital I/O line 0. If not set, the ADC sequencer uses the ADC cycle counter.
TP851_F_EXTTRIGOUT	If set the ADC trigger is used as output on digital I/O line 0.

TP851_F_EXTTRIGSRC* and *TP851_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 TP851_C_ADC_SEQREAD).

EXAMPLE

```
#include <tpmc851.h>

int                    fd;
int                    result;
TP851_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,
                   TP851_C_ADC_SEQSTART,
                   (char*)&adcSeqStartBuf);
if (result == OK)
{
     printf("OK\n");
} else {
     /* process ioctl error */
}
```

Error Codes

EBUSY	The ADC sequencer is currently running.
EINVAL	Specified flags are invalid.
ENOMEM	No memory is available to allocate the internal buffer.

All other returned error codes are system error conditions.

3.3.4 TP851_C_ADC_SEQSTOP

NAME

TP851_C_ADC_SEQSTOP – Stop ADC sequencer

DESCRIPTION

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

No additional parameter is necessary.

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;

/*
** Stop the sequencer
*/
printf("Stop ADC Sequencer ... ");
result = ioctl(    fd,
                   TP851_C_ADC_SEQSTOP,
                   NULL);
if (result == OK)
{
     printf("OK\n");
} else {
     /* process ioctl error */
}
```

Error Codes

EACCES	The ADC sequencer is not running.

All other returned error codes are system error conditions.

3.3.5 TP851_C_ADC_SEQREAD

NAME

TP851_C_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 (*TP851_ADC_SEQREAD_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
    long   seqState;
    short  buffer[32];
} TP851_ADC_SEQREAD_BUF, *PTP851_ADC_SEQREAD_BUF;
```

seqState

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

flag	description
TP851_SF_SEQACTIVE	If set the ADC sequencer is started. If not set, the ADC sequencer stopped.
TP851_SF_SEQOVERFLOWERR	If set the ADC sequencer has detected an overflow error. (Hardware detected)
TP851_SF_SEQTIMERERROR	If set the ADC sequencer has detected a timer error. (Hardware detected)
TP851_SF_SEQIRAMERROR	If set the ADC sequencer has detected an instruction RAM error. (Hardware detected)
TP851_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;
TP851_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,
                        TP851_C_ADC_SEQREAD,
                        (char*)&adcSeqReadBuf);
     if (result == OK)
     {
          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 */
     }
}
```

Error Codes

EACCES	The ADC sequencer is not running.
ENODATA	No data is available inside the internal buffer.

All other returned error codes are system error conditions.

3.3.6 TP851_C_DAC_WRITE

NAME

TP851_C_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 (*TP851_DAC_WRITE_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
    int             channel;
    unsigned long   flags;
    short           dacValue;
} TP851_DAC_WRITE_BUF, *PTP851_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
TP851_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. If not set, *dacValue* is written to the DAC channel.
TP851_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 *TP851_F_NOUPDATE* flag. 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;
TP851_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,
                   TP851_C_DAC_WRITE,
                   (char*)&dacWriteBuf);
if (result == OK)
{
     printf("OK\n");
} else {
     /* process ioctl error */
}
```

Error Codes

EBUSY	The DAC sequencer is currently running.
ECHRNG	Specified channel is invalid.
EINVAL	Specified gain level is invalid.

All other returned error codes are system error conditions.

3.3.7 TP851_C_DAC_SEQCONFIG

NAME

TP851_C_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 (*TP851_DAC_SEQCONFIG_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
    int             channel;
    int             enable;
    unsigned long   flags;
} TP851_DAC_SEQCONFIG_BUF, *PTP851_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
TP851_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. If not set, *dacValue* is written to the DAC channel.

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;
TP851_DAC_SEQCONFIG_BUF dacSeqConfBuf;

/*
** Configure DAC channel 1, using corrected data
** when the sequencer is running
*/
dacSeqConfBuf.channel   = 1;
dacSeqConfBuf.enable    = TRUE;
dacSeqConfBuf.flags     = TP851_F_CORR;

printf("Configure channel for Sequencer ... ");
result = ioctl(    fd,
                   TP851_C_DAC_SEQCONFIG,
                   (char*)&dacSeqConfBuf);
if (result == OK)
{
     printf("OK\n");
} else {
     /* process ioctl error */
}
```

Error Codes

EBUSY	The DAC sequencer is currently running.
ECHRNG	Specified channel is invalid.
EINVAL	Specified flags are invalid.

All other returned error codes are system error conditions.

3.3.8 TP851_C_DAC_SEQSTART

NAME

TP851_C_DAC_SEQSTART – Start DAC sequencer

DESCRIPTION

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

A pointer to the start structure (*TP851_DAC_SEQSTART_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        unsigned short          cycTime;
        unsigned long           flags;
        long                    bufSize;
        short                   buffer[1];
} TP851_DAC_SEQSTART_BUF, *PTP851_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
TP851_F_EXTTRIGSRC	If set the DAC sequencer is trigger with digital I/O line 1. If not set, the DAC sequencer uses the DAC cycle counter.
TP851_F_EXTTRIGOUT	If set the DAC trigger is used as output on digital I/O line 1.
TP851_F_DACSEQREPEAT	If set the DAC will repeat data when the end of the buffer is reached, the *TP851_SF_SEQFIFOUNDERFLOW* error will be suppressed.

TP851_F_EXTTRIGSRC* and *TP851_F_EXTTRIGOUT can not be used at the same time.

bufSize

Specifies the array size of buffer. This value must be the same as used for *s* in *TP851_CALC_SIZE_DAC_SEQDATA_BUF(s)* when calculating the allocation size for *adcSeqBuf*.

buffer

Array used for DAC sequencer data FIFO.
The DAC data is stored by the application into this 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;
TP851_DAC_SEQSTART_BUF dacSeqStartBuf;

/*
** Start sequencer with a buffer of 100 word and a cycle time of 100 ms,
** do not use external trigger
*/
dacSeqStartBuf.cycTime      = 1000;
dacSeqStartBuf.flags        = TP851_F_DACSEQREPEAT;
dacSeqStartBuf.bufSize      = 100;

/* Fill buffer */
dacSeqStartBuf.buffer[0] = …;
dacSeqStartBuf.buffer[1] = …;
dacSeqStartBuf.buffer[2] = …;

…
```

```
...

printf("Start DAC Sequencer ... ");
result = ioctl(    fd,
                   TP851_C_DAC_SEQSTART,
                   (char*)&dacSeqStartBuf);
if (result == OK)
{
     printf("OK\n");
} else {
     /* process ioctl error */
}
```

Error Codes

EBUSY	The DAC sequencer is already running.
EINVAL	Specified flags are invalid.
ENOMEM	No memory is available to allocate the internal buffer.

All other returned error codes are system error conditions.

3.3.9 TP851_C_DAC_SEQSTOP

NAME

TP851_C_DAC_SEQSTOP – Stop DAC sequencer

DESCRIPTION

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

No additional parameter is necessary.

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;

/*
** Stop the sequencer
*/
printf("Stop DAC Sequencer ... ");
result = ioctl(    fd,
                   TP851_C_DAC_SEQSTOP,
                   NULL);
if (result == OK)
{
     printf("OK\n");
} else {
     /* process ioctl error */
}
```

Error Codes

EACCES	The DAC sequencer is not running.

All other returned error codes are system error conditions.

3.3.10 TP851_C_DAC_SEQWRITE

NAME

TP851_C_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 (*TP851_DAC_SEQWRITE_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
    short  buffer[8];
} TP851_DAC_SEQWRITE_BUF, *PTP851_DAC_SEQWRITE_BUF;
```

buffer[]

This array contains data for the activated channels. Only the previously selected channels must be supplied with valid data. Array index 0 contains values for channel 1, array index 1 corresponds to channel 2 and so on.

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;
TP851_DAC_SEQWRITE_BUF  dacSeqWriteBuf;

/*
** Write values to internal sequencer buffer (1000 times)
** assuming that channel 1 and 3 are enabled.
*/
/* fill first cycle */
dacSeqWriteBuf.buffer[0] = …;
dacSeqWriteBuf.buffer[2] = …;
for (cycle=0; cycle<1000; cycle++)
{
    result = ioctl(    fd,
                       TP851_C_DAC_SEQWRITE,
                       (char*)&dacSeqWriteBuf);
```

```
        if (result == OK)
        {
             /* OK, fill next cycle */
             dacSeqWriteBuf.buffer[0] = …;
             dacSeqWriteBuf.buffer[2] = …;

        }
        if (result == ENOSPC)
        {
             /* wait a short time for new space available */
        }
}
```

Error Codes

EACCES	The DAC sequencer is not running.
ENOSPC	No space is available for new data inside the internal buffer.

All other returned error codes are system error conditions.

3.3.11 TP851_C_IO_READ

NAME

TP851_C_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 (*TP851_IO_BUF*) is passed by the parameter *arg* to the driver.

typedef struct
{
    unsigned short   value;
} TP851_IO_BUF, *PTP851_IO_BUF;

value

Returns the current digital I/O input value.

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;
TP851_IO_BUF            ioBuf;

/* Read I/O input value */
printf("Read I/O input value ... ");
result = ioctl(    fd,
                   TP851_C_IO_READ,
                   (char*)&ioBuf);
if (result == OK)
{
     printf("    I/O input: %04X", ioBuf.value);
} else {
     /* process ioctl error */
}
```

Error Codes

All returned error codes are system error conditions.

3.3.12 TP851_C_IO_WRITE

NAME

TP851_C_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 (*TP851_IO_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
      unsigned short          value;
} TP851_IO_BUF, *PTP851_IO_BUF;
```

value

Specifies the new digital I/O output value.

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;
TP851_IO_BUF            ioBuf;

/* Write 0x1234 to I/O output */
ioBuf.value = 0x1234;
printf("Write I/O output value ... ");
result = ioctl(    fd,
                   TP851_C_IO_WRITE,
                   (char*)&ioBuf);
if (result == OK)
{
     printf("OK\n);
} else {
     /* process ioctl error */
}
```

Error Codes

All returned error codes are system error conditions.

3.3.13 TP851_C_IO_EVENTWAIT

NAME

TP851_C_IO_EVENTWAIT – Wait for I/O event

DESCRIPTION

This function waits for an I/O input event.

A pointer to the event structure (*TP851_IO_EVENTWAIT_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        int             ioLine;
        unsigned long   flags;
        long            timeout;
} TP851_IO_EVENTWAIT_BUF, *PTP851_IO_EVENTWAIT_BUF;
```

ioLine

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

flags

Specifies the event to wait for. This is an ored value of the following flags:

flag	description
TP851_F_HI2LOTRANS	If set, the function will return after a high to low transition occurs.
TP851_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 -1 to wait indefinitely for the given event.

EXAMPLE

```
#include <tpmc851.h>

int                    fd;
int                    result;
TP851_IO_EVENTWAIT_BUF waitBuf;

/*
** Wait for a transition on I/O line 12 (max wait 10000 ticks)
*/
waitBuf.ioLine =   12;
waitBuf.flags =    TP851_F_HI2LOTRANS | TP851_F_LO2HITRANS;
waitBuf.timeout =  10000;

printf("Wait for an I/O event ... ");
result = ioctl(    fd,
                   TP851_C_IO_EVENTWAIT,
                   (char*)&waitBuf);
if (result == OK)
{
     printf("OK\n);
} else {
     /* process ioctl error */
}
```

Error Codes

ENOSPC	No space is available for new wait requests.
EINVAL	Invalid I/O line specified.
ETIMEDOUT	The timer expired.

All other returned error codes are system error conditions.

3.3.14 TP851_C_IO_CONFIG

NAME

TP851_C_IO_CONFIG – Configure digital I/O

DESCRIPTION

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

A pointer to the configuration structure (*TP851_IO_CONF_BUF*) is passed by the parameter *arg* to the driver.

typedef struct
{
    unsigned short   direction;
} TP851_IO_CONF_BUF, *PTP851_IO_CONF_BUF;

direction

Specifies the new direction setting 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;
TP851_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,
                   TP851_C_IO_CONFIG,
                   (char*)&ioConfBuf);
if (result == OK)
{
     printf("OK\n);
} else {
     /* process ioctl error */
}
```

Error Codes

All returned error codes are system error conditions.

3.3.15 TP851_C_IO_DEBCONFIG

NAME

TP851_C_IO_DEBCONFIG – Configure digital I/O (input) debouncer

DESCRIPTION

This function configures the digital I/O debouncing circuit.

A pointer to the configure structure (*TP851_IO_DEBCONF_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        unsigned short          enableMask;
        unsigned short          debTime;
} TP851_IO_DEBCONF_BUF, *PTP851_IO_DEBCONF_BUF;
```

enableMask

Specifies digital I/O lines which shall be used with 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;
TP851_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,
                   TP851_C_IO_DEBCONFIG,
                   (char*)&ioDebConfBuf);
```

```
if (result == OK)
{
     printf("OK\n);
} else {
     /* process ioctl error */
}
```

Error Codes

All returned error codes are system error conditions.

3.3.16 TP851_C_CNT_READ

NAME

TP851_C_CNT_READ – Read value from counter/timer

DESCRIPTION

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

A pointer to the read structure (*TP851_CNT_READ_ BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
        unsigned long           count;          /* Counter value */
        unsigned long           state;          /* Counter state information (cleared after read) */
} TP851_CNT_READ_BUF, *PTP851_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
TP851_SF_CNTBORROW	Counter borrow bit set (actual state)
TP851_SF_CNTCARRY	Counter carry bit set (actual state)
TP851_SF_CNTMATCH	Counter match event has occurred since last read.
TP851_SF_CNTSIGN	Counter sign bit (actual state)
TP851_SF_CNTDIRECTION	If set, counter direction is upward. If not set, counter direction is downward.
TP851_SF_CNTLATCH	Counter value has been latched.
TP851_SF_CNTLATCHOVERFLOW	Counter latch overflow has occurred.
TP851_SF_CNTSNGLCYC	Counter Single Cycle is active

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;
TP851_CNT_READ_BUF      cntBuf;

/* Read counter value */
printf("Read counter ... ");
result = ioctl(    fd,
                   TP851_C_CNT_READ,
                   (char*)&cntBuf);
if (result == OK)
{
     printf("    Counter: %ld", cntBuf.counter);
     printf("    State:   %lXh", cntBuf.state);
} else {
     /* process ioctl error */
}
```

Error Codes

All returned error codes are system error conditions.

3.3.17 TP851_C_CNT_MATCHWAIT

NAME

TP851_C_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 (*TP851_CNT_WAIT_BUF*) is passed by the parameter *arg* to the driver.

typedef struct
{
    long   timeout;
} TP851_CNT_WAIT_BUF, *PTP851_CNT_WAIT_BUF;

timeout

Specifies the maximum time the function will wait for the match event. The time is specified in ticks. Specify -1 to wait indefinitely for the given event.

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;
TP851_CNT_WAIT_BUF      cntWaitBuf;

/*
** Wait for counter match event (max wait 10000 ticks)
*/
waitBuf.timeout =  10000;
printf("Wait for counter match event ... ");
result = ioctl(    fd,
                   TP851_C_CNT_MATCHWAIT,
                   (char*)&cntWaitBuf);
if (result == OK)
{
     printf("OK\n);
} else {
     /* process ioctl error */
}
```

Error Codes

ENOSPC	No space is available for new wait requests.
ETIMEDOUT	The timer expired.

All other returned error codes are system error conditions.

3.3.18 TP851_C_CNT_CTRLWAIT

NAME

TP851_C_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 *TP851_C_CNT_CONFIG* specifying the parameter *controlMode*.

A pointer to the wait structure (*TP851_CNT_WAIT_BUF*) is passed by the parameter *arg* to the driver.

typedef struct
{
    long timeout;
} TP851_CNT_WAIT_BUF, *PTP851_CNT_WAIT_BUF;

timeout

Specifies the maximum time the function will wait for the match event. The time is specified in ticks. Specify -1 to wait indefinitely for the given event.

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;
TP851_CNT_WAIT_BUF      cntWaitBuf;

/*
** Wait for counter control event (max wait 10000 ticks)
*/
waitBuf.timeout =  10000;
printf("Wait for counter control event ... ");
result = ioctl(    fd,
                   TP851_C_CNT_CTRLWAIT,
                   (char*)&cntWaitBuf);
if (result == OK)
{
     printf("OK\n);
} else {
     /* process ioctl error */
}
```

Error Codes

ENOSPC	No space is available for new wait requests.
ETIMEDOUT	The timer expired.

All other returned error codes are system error conditions.

3.3.19 TP851_C_CNT_CONFIG

NAME

TP851_C_CNT_CONFIG – Configure counter

DESCRIPTION

This function configures the counter.

A pointer to the configuration structure (*TP851_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;
} TP851_CNT_CONFIG_BUF, *PTP851_CNT_CONFIG_BUF;
```

inputMode

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

flag	description
TP851_M_CNTIN_DISABLE	Counter disabled
TP851_M_CNTIN_TIMERUP	Timer Mode Up
TP851_M_CNTIN_TIMERDOWN	Timer Mode Down
TP851_M_CNTIN_DIRCOUNT	Direction Count
TP851_M_CNTIN_UPDOWNCOUNT	Up/Down Count
TP851_M_CNTIN_QUAD1X	Quadrature Count 1x
TP851_M_CNTIN_QUAD2X	Quadrature Count 2x
TP851_M_CNTIN_QUAD3X	Quadrature Count 4x

clockDivider

Specifies clock divider. 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
TP851_M_CNT_CYCLE	Cycling Counter
TP851_M_CNT_DIVN	Divide-by-N
TP851_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
TP851_M_CNTCTRL_NONE	No Control Mode
TP851_M_CNTCTRL_LOAD	Load Mode
TP851_M_CNTCTRL_LATCH	Latch Mode
TP851_M_CNTCTRL_GATE	Gate Mode
TP851_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
TP851_F_CNTINVINP2	If set, input line 2 is low active If not set, input line 2 is high active
TP851_F_CNTINVINP3	If set, input line 3 is low active If not set, input line 3 is high active
TP851_F_CNTINVINP4	If set, input line 4 is low active If not set, input line 4 is high active

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;
TP851_CNT_CONFIG_BUF    cntConfBuf;

/*
** Setup counter for direction count, clock divider 1, cycling count,
** no control mode and all line high active
*/
cntConfBuf.inputMode =      TP851_M_CNTIN_DIRCOUNT;
cntConfBuf.clockDivider =   1;
cntConfBuf.countMode =      TP851_M_CNT_CYCLE;
cntConfBuf.controlMode =    TP851_M_CNTCTRL_NONE;
cntConfBuf.invFlags =       0;

printf("Set counter configuration ... ");
result = ioctl(    fd,
                   TP851_C_CNT_CONFIG,
                   (char*)&cntConfBuf);
```

```
if (result == OK)
{
     printf("OK\n");
} else {
     /* process ioctl error */
}
```

Error Codes

EINVAL	Specified flag or mode is invalid.

All other returned error codes are system error conditions.

3.3.20 TP851_C_CNT_RESET

NAME

TP851_C_CNT_RESET – Reset counter

DESCRIPTION

This function resets the counter value to 0x00000000.

No additional parameter is necessary for this function.

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;

/* Reset counter */
printf("Reset counter ... ");
result = ioctl(    fd,
                   TP851_C_CNT_RESET,
                   NULL);
if (result == OK)
{
     printf("OK\n");
} else {
     /* process ioctl error */
}
```

Error Codes

All returned error codes are system error conditions.

3.3.21 TP851_C_CNT_SETPRELD

NAME

TP851_C_CNT_SETPRELD – Set counter preload value

DESCRIPTION

This function sets the counter preload register.

A pointer to the preload structure (*TP851_CNT_SETPRELD_BUF*) is passed by the parameter *arg* to the driver.

The *TP851_CNT_SETPRELD _BUF* structure has the following layout:

```
typedef struct
{
        unsigned long           value;
        unsigned long           flags;
} TP851_CNT_SETPRELD_BUF, *PTP851_CNT_SETPRELD_BUF;
```

value

Specifies the new counter preload value.

flags

Is an ored value of the following flags:

flag	description
TP851_F_IMMPRELOAD	If set, the function will immediate load the preload value into the counter If not set, preload value will be used for the next preload condition.

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;
TP851_CNT_SETPRELD_BUF  cntPrldBuf;

/*
** Immediately load 0x11223344 into the counter and preload register
*/
cntPrldBuf. value       = 0x11223344;
cntPrldBuf.flags        = TP851_F_IMMPRELOAD;

…
```

```
…

printf("Set preload value ... ");
result = ioctl(    fd,
                   TP851_C_CNT_SETPRELD,
                   (char*)&cntPrldBuf);
if (result == OK)
{
     printf("OK\n");
} else {
     /* process ioctl error */
}
```

Error Codes

All returned error codes are system error conditions.

3.3.22 TP851_C_CNT_SETMATCH

NAME

TP851_C_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 TP851_C_CNT_MATCHWAIT).

A pointer to the match structure (*TP851_CNT_SETMATCH_BUF*) is passed by the parameter *arg* to the driver.

```
typedef struct
{
      unsigned long          value;
} TP851_CNT_SETMATCH_BUF, *PTP851_CNT_SETMATCH_BUF;
```

value

Specifies the new counter match value.

EXAMPLE

```
#include <tpmc851.h>

int                     fd;
int                     result;
TP851_CNT_SETMATCH_BUF  cntMatchBuf;

/* Set match value to 0x10000 */
cntMatchBuf.value       = 0x10000;
printf("Set counter match value ... ");
result = ioctl(    fd,
                   TP851_C_CNT_SETMATCH,
                   (char*)&cntMatchBuf);
if (result == OK)
{
     printf("OK\n");
} else {
     /* process ioctl error */
}
```

Error Codes

All returned error codes are system error conditions.

4 Debugging and Diagnostic

If the driver does not work properly, please enable debug outputs by defining the symbols *DEBUG, DEBUG_TPMC, DEBUG_PCI* and *DEBUG_INT.*

The debug output should appear on the console. If not, please check the symbol *KKPF_PORT* in *uparam.h*. This symbol should be configured to a valid COM port (e.g. *SKDB_COM1*).

The debug output displays the device information data for the current major device, and a memory dump of the PCI base address registers like this.

```
Bus = 0  Dev = 17  Func = 0
[00] = 03531498
[04] = 02800003
[08] = 11800000
[0C] = 00000008
[10] = CFFFEF80
[14] = 0000D801
[18] = CFFFEC00
[1C] = CFFFEF40
[20] = CFFFEF00
[24] = 00000000
[28] = 00000000
[2C] = 000A1498
[30] = 00000000
[34] = 00000040
[38] = 00000000
[3C] = 0000010B

PCI Base Address 0 (PCI_RESID_BAR0)

CBFFEF80 : 00 FE FF 0F C0 FF FF 0F C0 FF FF 0F 00 00 00 00
CBFFEF90 : 00 00 00 00 01 00 00 00 01 02 00 00 01 03 00 00
CBFFEFA0 : 00 00 00 00 00 00 00 00 22 00 80 01 22 00 40 01
CBFFEFB0 : 22 00 40 01 00 00 00 00 00 00 00 00 01 01 00 00
CBFFEFC0 : 21 02 00 00 21 03 00 00 00 00 00 00 61 00 30 00
CBFFEFD0 : 00 00 78 18 D2 B6 6D 02 00 00 00 00 00 00 00 00
CBFFEFE0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEFF0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

PCI Base Address 1 (PCI_RESID_BAR1)

0000D800 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000D810 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000D820 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000D830 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000D840 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
0000D850 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000D860 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000D870 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

PCI Base Address 2 (PCI_RESID_BAR2)

CBFFEC00 : 00 00 00 20 00 00 13 7A 00 00 00 00 00 00 00 00
CBFFEC10 : 00 00 00 00 00 00 00 00 00 00 03 E8 00 00 00 00
CBFFEC20 : 00 00 00 02 00 00 00 02 00 00 00 00 00 00 00 00
CBFFEC30 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEC40 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEC50 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEC60 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEC70 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

PCI Base Address 3 (PCI_RESID_BAR3)

CBFFEF40 : 13 72 03 DB 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEF50 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEF60 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEF70 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEF80 : 00 FE FF 0F C0 FF FF 0F C0 FF FF 0F 00 00 00 00
CBFFEF90 : 00 00 00 00 01 00 00 00 01 02 00 00 01 03 00 00
CBFFEFA0 : 00 00 00 00 00 00 00 00 22 00 80 01 22 00 40 01
CBFFEFB0 : 22 00 40 01 00 00 00 00 00 00 00 00 01 01 00 00

PCI Base Address 4 (PCI_RESID_BAR4)

CBFFEF00 : 00 84 FF FD 00 89 00 04 00 96 FF EF 00 AE FF CA
CBFFEF10 : FF 2B FF EB FF 4F 00 2C FF 60 FF E9 FF 4D FF B1
CBFFEF20 : FF 3D 00 05 FF 3A 00 30 FF 31 FF C8 FF 3D 00 20
CBFFEF30 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEF40 : 13 72 03 DB 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEF50 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEF60 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CBFFEF70 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Found a TPMC851, BusNo=0, DevNo=17
        Calibration Data:
            ADC [gain 1]: Offset: 132 / Gain: -3
            ADC [gain 2]: Offset: 137 / Gain: 4
            ADC [gain 3]: Offset: 150 / Gain: -17
            ADC [gain 4]: Offset: 174 / Gain: -54
            ADC [chan 1]: Offset: -213 / Gain: -21
            ADC [chan 2]: Offset: -177 / Gain: 44
```

```
ADC [chan 3]: Offset: -160 / Gain: -23
ADC [chan 4]: Offset: -179 / Gain: -79
ADC [chan 5]: Offset: -195 / Gain: 5
ADC [chan 6]: Offset: -198 / Gain: 48
ADC [chan 7]: Offset: -207 / Gain: -56
ADC [chan 8]: Offset: -195 / Gain: 32
```

The debug output above is only an example. Debug output on other systems may be different for addresses and data in some locations.