The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC551-SW-72

LynxOS Device Driver

8/4 Channels of Isolated 16 Bit D/A

Version 1.0.x

User Manual

Issue 1.0.0

November 2009

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

TPMC551-SW-72

LynxOS Device Driver

8/4 Channels of Isolated 16 Bit D/A

Supported Modules:
TPMC551

Issue	Description	Date
1.0.0	First Issue	November 4, 2009

Table of Contents

1 Introduction

The TPMC551-SW-72 LynxOS device driver allows the operation of the TPMC551 digital analog converter 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 TPMC551-SW-72 device driver supports the following features:

- setting output voltage
- latched conversion and simultaneous output
- output data correction with factory calibration data
- configuration, start and stop of the sequencer
- write sequencer data sets
- reading board information

The TPMC551-SW-72 device driver supports the modules listed below:

TPMC551-10	8 Channel 16-bit ADC (Front I/O)	(PMC)
TPMC551-11	4 Channel 16-bit ADC (Front I/O)	(PMC)
TPMC551-20	8 Channel 16-bit ADC (Back I/O)	(PMC)
TPMC551-21	4 Channel 16-bit ADC (Back I/O)	(PMC)

To get more information about the features and use of TPMC551 devices it is recommended to read the manuals listed below.

TPMC551 User Manual

TPMC551 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC551-SW-72':

TPMC551-SW-72-SRC.tar.gz	GZIP compressed archive with driver source code
TPMC551-SW-72-1.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

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

The GZIP compressed archive TPMC551-SW-72-SRC.tar.gz contains the following files and directories:

Directory path 'tpmc551':

tpmc551.c	TPMC551 device driver source
tpmc551def.h	TPMC551 driver include file
tpmc551.h	TPMC551 include file for driver and application
tpmc551_info.c	TPMC551 Device information definition
tpmc551_info.h	TPMC551 Device information definition header
tpmc551.cfg	TPMC551 Driver configuration file include
tpmc551.import	Linker import file
Makefile	Device driver make file
example/tpmc551exa.c	Example application
example/Makefile	Example application makefile

In order to perform a driver installation, first extract the TAR file to a temporary directory, than follow the steps below:

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/tpmc551 or /sys/drivers.cpci_x86/tpmc551

2. Copy the following files to this directory:
   - tpmc551.c
   - tpmc551def.h
   - tpmc551.import
   - Makefile

3. Copy tpmc551.h to /usr/include/

4. Copy tpmc551_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tpmc551_info.h to /sys/dheaders/

Copy tpmc551.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:

(1) Static Installation
(2) Dynamic Installation (only native LynxOS 4 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/tpmc551, 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 = … tpmc551_info.x
   ```

   And at the end of the Makefile

   ```
   tpmc551_info.o:$(DHEADERS)/tpmc551_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:tpmc551.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/tpmc551a, /dev/tpmc551b, …

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/tpmc551, 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/tpmc551, 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 systems)

   ```
   make t551info
   ```

3. To install the driver enter:

   ```
   drinstall –c tpmc551.obj
   ```

   If successful, drinstall returns a unique <driver-ID>

4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> t551info
   ```

   The <driver-ID> is returned by the drinstall command

5. To create the node for the devices enter:

   ```
   mknod /dev/tpmc551a c <major_no> 0
   ```

   The <major_no> is returned by the devinstall command.

If all steps are successfully completed, the TPMC551 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TPMC551 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 TPMC551 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tpmc551_info.h*.

This structure contains the following parameter:

PCIBusNumber	Contains the PCI bus number at which the supported device is connected. Valid bus numbers are in range from 0 to 255.
PCIDeviceNumber	Contains the device number (slot) at which the supported device 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 supported devices. 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 TPMC551 major devices.

A device information definition is unique for every TPMC551 major device. The file *tpmc551_info.c* on the distribution media contains two device information declarations, **tpmc551A** for the first major device and **tpmc551B** 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 **tpmc551C**, **tpmc551D** and so on.

It is also necessary to modify the device and driver configuration file, respectively the configuration include file *tpmc551.cfg*.

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

```
TDRV551_INFO tpmc551A = {

     -1,            /*  Auto find the device on any PCI bus    */
     -1
};
```

2.1.4 Configuration File: CONFIG.TBL

The device and driver configuration file CONFIG.TBL (respective config.tbl on LynxOS 5.0 systems) 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 TPMC551 driver and devices into the LynxOS system, the configuration include file tpmc551.cfg must be included in the CONFIG.TBL (see also chapter 2.1.1.3).

The file tpmc551.cfg on the distribution disk contains the driver entry (*C:tpmc551:\....*) and two major device entries (*D:TPMC551 1:tpmc551A:: and D:TPMC551 2:tpmc551B::*).

If the driver should support more than one major device, the following entries for major devices must be enabled by removing the comment character (#). By copy and paste an existing major and minor entries and renaming the new entries, it is possible to add any number of additional TPMC551 devices.

This example shows a driver entry with two major devices 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:tpmc551:tpmc551open:tpmc551close: \
::: \
::tpmc551ioctl: \
:tpmc551install:tpmc551uninstall
D:TPMC551 1:tpmc551A::
N:tpmc551a:0
D:TPMC551 2:tpmc551B::
N:tpmc551b:0
```

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

```
/dev/tpmc551a /dev/tpmc551b
```

3 TPMC551 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 (TPMC551 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TPMC551 device ***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 TPMC551 device already exists this argument is ignored.

EXAMPLE

```
int fd;

fd = open ("/dev/tpmc551a", O_RDWR);
if (fd == -1)
{
     /* 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;

result = close(fd);
if (result == -1)
{
     /* 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 <tpmc551.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 *tpmc551.h*:

Symbol	Meaning
TPMC551_WRITE	Set output data
TPMC551_SEQSETUP	Setup sequencer configuration and start sequencer mode
TPMC551_SEQSTOP	Stop sequencer mode
TPMC551_SEQWRITE	Write sequencer new output data
TPMC551_GETINFO	Get board specific information

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).

SEE ALSO

LynxOS System Call - ioctl().

3.3.1 TPMC551_WRITE

NAME

TPMC551_WRITE – Writes a new output value to the DAC

DESCRIPTION

This function writes a new DAC output value for a specified channel. The function allows writes that immediately output the new voltage, or which just fetch the value and output the voltage with a later write access. A pointer to the callers write buffer (*TPMC551_WRITE_BUFFER*) must be passed by the parameter *arg* to the device.

```
typedef struct
{
    int         channel;
    int         convMode;
    int         corrMode;
    int         value;
} TPMC551_WRITE_BUFFER, *PTPMC551_WRITE_BUFFER;
```

Members

channel

This argument specifies the DAC channel to use. Allowed values are 1 up to 8 for TPMC551-x0 and 1 up to 4 for TPMC551-x1.

convMode

This argument specifies the conversion mode. Symbols for the conversion mode are defined in tpmc551.h:

Define	Description
TPMC551_NORMAL	The value will be written to the specified channel and the output voltage will be updated for the channel
TPMC551_LATCH	The value will be written to the specified channel, but output voltage is not updated
TPMC551_SIMCONV	The value will be written to the specified channel and the output voltage will be updated for all channels

corrMode

This argument specifies if output value correction will be used. Symbols for 'TPMC551_ON' and 'TPMC551_OFF' are defined in tpmc551.h:

Define	Description
TPMC551_OFF	This value specifies that the value will be written to the DAC without correction.
TPMC551_ON	This value specifies that the DAC raw value shall be corrected with the factory correction data before it is written to the DAC.

value

This parameter specifies the new DAC output value. Values are allowed between 0 and 65535 for unipolar (0V…10V) channels and between -32768 and 32767 for bipolar (-/+10V) channels.

EXAMPLE

```
#include <tpmc551.h>

int                     fd;
int                     result;
TPMC551_WRITE_BUFFER    wrBuf;

/* --- write new output value --- */
wrBuf.channel      = 4;              /* DAC channel 4 */
wrBuf.convMode     = TPMC551_NORMAL; /* immidiate conversion */
wrBuf.corrMode     = TPMC551_ON;     /* output value corrrection on */
wrBuf.value        = 0x4000;         /* 5V for bipolar channel */

result = ioctl(fd, TPMC551_WRITE, (char*)&wrBuf);
if (result >= 0)
{
    /* Write has been successful */
}
else
{
    /* Write failed */
}
```

ERRORS

EINVAL	An unsupported input parameter value has been specified. Check input parameters
EBUSY	The sequencer mode is active for the device and single channel conversions are not allowed.

Other returned error codes are system error conditions.

3.3.2 TPMC551_SEQSETUP

NAME

TPMC551_SEQSETUP – Configure sequencer mode and start execution

DESCRIPTION

This function configures channels and timing for sequencer mode and starts the execution. A pointer to the callers configuration buffer (*TPMC551_SEQSETUP_BUFFER*) must be passed by the parameter *arg* to the device.

```
typedef struct
{
        int             cycleTime;
        int             synchMode;
        int             timerMode;
        int             waitMode;
        struct
        {
                int     enable;
                int     corrMode;
                int     value;
        }               chanCfg[MAX_NUM_CHANS];
} TPMC551_SEQSETUP_BUFFER, *PTPMC551_SEQSETUP_BUFFER;
```

Members

cycleTime

This argument specifies the cycle time for sequencer mode. The time is specified in steps of 100µs. Allowed values are 0 up to 65535. A value of 0 specifies that the sequencer will work in continuous mode. If the *timerMode* specifies continuous mode, this value will be ignored.

synchMode

This argument specifies if the channels shall update their output synchronously or as fast as possible. Symbols for *synchMode* are defined in tpmc551.h:

Define	Description
TPMC551_TRANSPARENT	If this value is specified, the outputs will be updated as fast as possible for every channel
TPMC551_SYNCHRON	If this value is specified, all channels used in sequencer mode will update outputs synchronously all channels were written.

timerMode

This argument specifies if the sequencer shall use the cycle timer or run in continuous mode. Symbols for *timerMode* are defined in tpmc551.h:

Define	**Description**
TPMC551_TIMER	The sequencer will run in timer mode. The output voltages will be updated in a fixed grid which is defined by cycleTime.
TPMC551_CONTINUOUS	The sequencer will start the next conversion cycle immediately after the previous has been completed. The speed depends on the number of used channels.

waitMode

This argument specifies if the driver shall wait until the current cycle was completed and the new cycle was started.
Symbols for *timerMode* are defined in tpmc551.h:

Define	**Description**
TPMC551_OFF	The output data will be written and the driver will return.
TPMC551_ON	The output data will be written and the driver will wait until the previous cycle (maybe from other configuration) is completed and the current configuration is used.

chanCfg[]

This argument is an array containing the channel configurations. The array element with index 0 specifies the configuration of channel 1, index 1 specifies the configuration of channel 2, and so on.

chanCfg[].enable

This array element specifies if the assigned channel shall be enabled for sequencer mode. Symbols for ‘TPMC551_ON’ and ‘TPMC551_OFF’ are defined in tpmc551.h:

Define	**Description**
TPMC551_OFF	If this value is specified, the channel will not be used.
TPMC551_ON	If this value is specified, the channel will be enabled in sequencer mode and the output data will be updated with every sequencer cycle.

chanCfg[].corrMode

This array element specifies if output value correction shall be used for the assigned channel. Symbols for ‘TPMC551_ON’ and ‘TPMC551_OFF’ are defined in tpmc551.h:

Define	**Description**
TPMC551_OFF	This value specifies that values for the channel will be used as a raw value.
TPMC551_ON	This value specifies that values for the channel will be used as a corrected value. For correction factory stored values will be used.

chanCfg[].value

This parameter specifies the initial output value of the channel. Values are allowed between 0 and 65535 for unipolar (0V…10V) channels and between -32768 and 32767 for bipolar (-/+10V) channels.

EXAMPLE

```
#include <tpmc551.h>

int                        fd;
int                        chanIdx;
int                        result;
TPMC551_SEQSETUP_BUFFER seqBuf;

/* --- initialize structure --- */
for (chanIdx = 0; chanIdx < MAX_NUM_CHANS; chanIdx++)
{
     seqBuf.chanCfg[].enable = TPMC551_OFF;     /* disable channel */
}

/* --- configure and start sequencer --- */
seqBuf.cycleTime        = 5000;                 /* cycle time: 0.5 sec */
seqBuf.synchMode        = TPMC551_SYNCHRON;     /* synchronous output */
seqBuf.timerMode        = TPMC551_TIMER;        /* user cycle timer */
seqBuf.waitTime         = TPMC551_OFF;          /* return immidiately */
/* Channel 1 */
seqBuf.chanCfg[0].enable    = TPMC551_ON;       /* enable channel */
seqBuf.chanCfg[0].corrMode  = TPMC551_ON;       /* corrrection on */
seqBuf.chanCfg[0].value     = 0;                /* initial out: 0V */
/* Channel 4 */
seqBuf.chanCfg[3].enable    = TPMC551_ON;       /* enable channel */
seqBuf.chanCfg[3].corrMode  = TPMC551_ON;       /* corrrection on */
seqBuf.chanCfg[3].value     = 0x4000;           /* initial out: 5V/2.5V */
/* Channel 8 */
seqBuf.chanCfg[7].enable    = TPMC551_ON;       /* enable channel */
seqBuf.chanCfg[7].corrMode  = TPMC551_ON;       /* corrrection on */
seqBuf.chanCfg[7].value     = 0;                /* initial out: 0V */

result = ioctl(fd, TPMC551_SEQSETUP, (char*)&seqBuf);
if (result >= 0)
{
     /* Sequencer mode successfully started */
}
else
{
     /* Sequencer mode start failed */
}
```

ERRORS

EINVAL	An unsupported input parameter has been specified. Check input parameters
EBUSY	The sequencer mode is already active for the device. The sequencer must be stopped.
EINTR	The function was cancelled.
ETIMEDOUT	The wait time has exceeded the maximum time of a sequencer cycle. (no interrupts or HW-problem)

Other returned error codes are system error conditions.

3.3.3 TPMC551_SEQSTOP

NAME

TPMC551_SEQSTOP – Stop sequencer mode

DESCRIPTION

This function stops the sequencer mode. The function dependent parameter *arg* can be set to NULL.

EXAMPLE

```
#include <tpmc551.h>

int                     fd;
int                     result;

/* --- stop sequencer --- */
result = ioctl(fd, TPMC551_SEQSTOP, NULL);
if (result >= 0)
{
     /* Sequencer successfully stopped */
}
else
{
     /* Sequencer stop failed */
}
```

3.3.4 TPMC551_SEQWRITE

NAME

TPMC551_SEQWRITE – Write sequencer data

DESCRIPTION

This function writes a set of sequencer data if the sequencer is started. A pointer to the callers write buffer (*TPMC551_SEQWRITE_BUFFER*) must be passed by the parameter *arg* to the device.

```
typedef struct
{
        int             waitMode;
        int             value[MAX_NUM_CHANS];
        unsigned int    status;
} TPMC551_SEQWRITE_BUFFER, *PTPMC551_SEQWRITE_BUFFER;
```

Members

waitMode

This argument specifies how data shall be handled and if data can be overwritten or data output of the application is synchronized with the sequencer cycle. Symbols for 'TPMC551_ON' and 'TPMC551_OFF' are defined in tpmc551.h:

Define	**Description**
TPMC551_OFF	The function stores the data for the next cycle immediately. Previous written data which has not been transferred to the DAC will be overwritten.
TPMC551_ON	The function waits until there is space to store data for a new cycle. The function will always wait until data for a new cycle has been transferred to the DACs. Data will not be overwritten. This function allows synchronizing with the cycle timer.

value[]

This argument specifies the new output values. Values are allowed between 0 and 65535 for unipolar (0V…10V) channels and between -32768 and 32767 for bipolar (-/+10V) channels. Only array elements of enabled channels are used. The array element with index 0 specifies the configuration of channel 1, index 1 specifies the configuration of channel 2, and so on.

status

This parameter returns the status of the sequencer. Symbols for the returned flags are defined in tpmc551.h. The status is a value of OR'ed flags:

Flag	Description
TPMC551_FL_UNDERRUN	This status is detected by hardware and signals a data under run. Data has not been updated in a complete sequencer cycle and old data will be used again. This status will announce in common that data has not been provided in time. There are two cases let this status occur: (1) the application has not provided data in time (2) the interrupt has not been handled within the cycle time, data is available, but has not been updated in hardware
TPMC551_FL_OVERRUN	This flag indicates that data has been overwritten and only the new data will be used.

EXAMPLE

```
#include <tpmc551.h>

int                           fd;
int                           chanIdx;
int                           result;
TPMC551_SEQWRITE_BUFFER       seqWrBuf;

/* --- write new set of data --- */
seqWrBuf.waitMode  = TPMC551_ON;      /* synchronize with cycle timer */
seqWrBuf.value[0]  = 0x1000;          /* channel 1: new data */
seqWrBuf.value[3]  = 0x1000;          /* channel 4: new data */
seqWrBuf.value[7]  = 0x1000;          /* channel 8: new data */

result = ioctl(fd, TPMC551_SEQREAD, (char*)&seqWrBuf);
if (result >= 0)
{
     printf("Write successful, status = %x\n", seqWrBuf.status);
}
else
{
     /* Sequencer write failed */
}
```

ERRORS

EINTR	The function was cancelled.
ETIMEDOUT	The wait time has exceeded the maximum time of a sequencer cycle. (no interrupts or HW-problem)
EINVAL	An unsupported input parameter has been specified. Check input parameters.
EBUSY	The sequencer mode has not been started for the device. The sequencer must be started first.

Other returned error codes are system error conditions.

3.3.5 TPMC551_INFO

NAME

TPMC551_INFO – Returns board data and information about the configuration

DESCRIPTION

This function returns board specific data like number of channels, correction data and configuration of the channels voltage range. A pointer to the callers write buffer (*TPMC551_INFO_BUFFER*) must be passed by the parameter *arg* to the device.

```
typedef struct
{
        int         availChans;
        int         polMode[MAX_NUM_CHANS];
        int         corrDataOffset[MAX_NUM_CHANS];
        int         corrDataGain[MAX_NUM_CHANS];
} TPMC551_INFO_BUFFER, *PTPMC551_INFO_BUFFER;
```

Members

availChans

The returned value specifies the number channels available on the board. This will be 4 for TPMC551-x1 and 8 for TPMC551-x0.

polMode[]

This array returns the voltage configuration of the channel. The array index specifies the channel, 0 for channel 1, 1 for channel 2, and so on. Symbols for the polarity mode are defined in tpmc551.h:

Define	Description
TPMC551_UNDEF	This value specifies, that the channel is not available
TPMC551_UNIPOL	This value specifies that the channel is configured for a voltage range from 0V up to +10V (unipolar)
TPMC551_BIPOL	This value specifies that the channel is configured for a voltage range from -10V up to +10V (bipolar)

corrDataOffset[]

This array returns the factory calibration offset values for the channels. The array index specifies the channel, 0 for channel 1, 1 for channel 2, and so on.

corrDataGain []

This array returns the factory calibration gain values for the channels. The array index specifies the channel, 0 for channel 1, 1 for channel 2, and so on.

EXAMPLE

```
#include <tpmc551.h>

int                     fd;
int                     result;
TPMC551_INFO_BUFFER     infoBuf;

/* --- read current input value --- */
result = ioctl(fd, TPMC551_INFO, (char*)&infoBuf);
if (result >= 0)
{
     /* Info read has been successful */
     printf(“%d channels available\n”, infoBuf. availChans);
     …
}
else
{
     /* Info read failed */
}
```

4 Debugging and Diagnostic

If the driver will not work properly, please enable debug outputs by defining the symbols *DEBUG, DEBUG_TPMC,* and *DEBUG_PCI* in file tpmc551.c.

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 like this.

```
TPMC551: Device Driver Install
Bus = 0  Dev = 16  Func = 0
[00] = 905010B5
[04] = 02800000
[08] = 11800001
[0C] = 00000000
[10] = 80003000
[14] = 00802001
[18] = 00803001
[1C] = 80004000
[20] = 00000000
[24] = 00000000
[28] = 00000000
[2C] = 02271498
[30] = 00000000
[34] = 00000000
[38] = 00000000
[3C] = 00000109

PCI Base Address 0 (PCI_RESID_BAR0)

70603000 : E1 FF FF 0F C0 FF FF 0F 00 00 00 00 00 00 00 00
70603010 : 00 00 00 00 01 00 00 00 01 01 00 00 00 00 00 00
70603020 : 00 00 00 00 00 00 00 00 80 20 41 51 80 62 05 55
70603030 : 00 00 00 00 00 00 00 00 00 00 00 00 09 00 00 00

PCI Base Address 1 (PCI_RESID_BAR1)

PCI Base Address 2 (PCI_RESID_BAR2)

70203000 : 0000 0000 000A 0000 0000 0000 0000 0000
70203010 : 0000 0000 0000 0000 0000 0000 0000 0000
70203020 : FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
70203030 : FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF
```

```
PCI Base Address 3 (PCI_RESID_BAR3)

70604000 : 00 14 FF FC FF F6 00 0C 00 0B 00 12 FF ED 00 01
70604010 : 00 07 00 07 00 06 00 07 FF B4 FF BB FF B0 FF B2
TPMC551: Found TPMC551-xx on Bus 0, Device 16
    Correction data: Offset/Gain
                          7/20
                          7/-4
                          6/-10
                          7/12
                        -76/11
                        -69/18
                        -80/-19
                        -78/1
```

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