*The Embedded I/O Company*

# TPMC500-SW-42

## VxWorks Device Driver

32 Channel 12-bit ADC PMC

Version 4.0.x

## User Manual

Issue 4.0.0

September 2010

**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

# TPMC500-SW-42

VxWorks Device Driver

32 Channel 12-bit ADC PMC

Supported Modules:
TPMC500

| Issue | Description | Date |
|---|---|---|
| 1.0 | First Issue | November, 1999 |
| 1.1 | General Revision | November 24, 2003 |
| 2.0.0 | New Device Initialization Parameters, File list modified, Support for Intel x86 added | October 29, 2004 |
| 3.0.0 | New driver startup functions, File list changed, new prefix for defines and structures, read() function replaced by a new ioctl() function | October 1, 2007 |
| 4.0.0 | Implementation of VxBus Driver, tpmc500init() function added, API added | September 14, 2010 |

# Table of Contents

# 1 Introduction

## 1.1 Device Driver

The TPMC500-SW-42 VxWorks device driver software allows the operation of the supported PMC conforming to the VxWorks I/O system specification.

The TPMC500-SW-42 release contains independent driver sources for the old legacy (pre-VxBus) and the new VxBus-enabled driver model. The VxBus-enabled driver is recommended for new developments with later VxWorks 6.x release and mandatory for VxWorks SMP systems.

Both drivers, legacy and VxBus, share the same application programming interface (API) and device-independent basic I/O interface with open(), close() and ioctl() functions. The basic I/O interface is only for backward compatibility with existing applications and should not be used for new developments.

Both drivers invoke a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

The TPMC500-SW-42 device driver supports the following features:

- start AD conversion and read data
- choosing gain, channel, input interface
- correction of input data with board-specific calibration data
- support of ADC sequencer mode

<u>The TPMC500-SW-42 supports the modules listed below:</u>

| | | |
|---|---|---|
| TPMC500 | 32(16) Channel - 12-bit ADC | (PMC) |

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

| |
|---|
| TPMC500 User Manual |
| TPMC500 Engineering Manual |

# 2 Installation

Following files are located on the distribution media:

Directory path 'TPMC500-SW-42':

| | |
|---|---|
| TPMC500-SW-42-4.0.0.pdf | PDF copy of this manual |
| TPMC500-SW-42-VXBUS.zip | Zip compressed archive with VxBus driver sources |
| TPMC500-SW-42-LEGACY.zip | Zip compressed archive with legacy driver sources |
| ChangeLog.txt | Release history |
| Release.txt | Release information |

The archive TPMC500-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tpmc500':

| | |
|---|---|
| tpmc500drv.c | TPMC500 device driver source |
| tpmc500def.h | TPMC500 driver include file |
| tpmc500.h | TPMC500 include file for driver and application |
| tpmc500api.c | TPMC500 API file |
| Makefile | Driver Makefile |
| 40tpmc500.cdf | Component description file for VxWorks development tools |
| tpmc500.dc | Configuration stub file for direct BSP builds |
| tpmc500.dr | Configuration stub file for direct BSP builds |
| include/tvxbHal.h | Hardware dependent interface functions and definitions |
| apps/tpmc500exa.c | Example application |

The archive TPMC500-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tpmc500':

| | |
|---|---|
| tpmc500drv.c | TPMC500 device driver source |
| tpmc500def.h | TPMC500 driver include file |
| tpmc500.h | TPMC500 include file for driver and application |
| tpmc500pci.c | TPMC500 device driver source for x86 based systems |
| tpmc500api.c | TPMC500 API file |
| tpmc500exa.c | Example application |
| include/tdhal.h | Hardware dependent interface functions and definitions |

## 2.1 Legacy vs. VxBus Driver

In later VxWorks 6.x releases, the old VxWorks 5.x legacy device driver model was replaced by VxBus-enabled device drivers. Legacy device drivers are tightly coupled with the BSP and the board hardware. The VxBus infrastructure hides all BSP and hardware differences under a well defined interface, which improves the portability and reduces the configuration effort. A further advantage is the improved performance of API calls by using the method interface and bypassing the VxWorks basic I/O interface.

VxBus-enabled device drivers are the preferred driver interface for new developments.

The checklist below will help you to make a decision which driver model is suitable and possible for your application:

| Legacy Driver | VxBus Driver |
|---|---|
| ▪ VxWorks 5.x releases<br>▪ VxWorks 6.5 and earlier releases<br>▪ VxWorks 6.x releases without VxBus PCI bus support | ▪ VxWorks 6.6 and later releases with VxBus PCI bus<br>▪ SMP systems (only the VxBus driver is SMP safe!) |

**TEWS TECHNOLOGIES recommends not using the VxBus Driver before VxWorks release 6.6. In previous releases required header files are missing and the support for 3rd-party drivers may not be available.**

## 2.2 VxBus Driver Installation

Because Wind River doesn't provide a standard installation method for 3rd party VxBus device drivers the installation procedure needs to be done manually.

In order to perform a manual installation extract all files from the archive TPMC500-SW-42-VXBUS.zip to the typical 3rd party directory *installDir/vxworks-6.x/target/3rdparty* (whereas *installDir* must be substituted by the VxWorks installation directory).

After successful installation the TPMC500 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tpmc500.*

At this point the TPMC500 driver is not configurable and cannot be included with the kernel configuration tool in a Wind River Workbench project. To make the driver configurable the driver library for the desired processor (CPU) and build tool (TOOL) must be built in the following way:

(1) Open a VxWorks development shell (e.g. C:\WindRiver\wrenv.exe -p vxworks-6.7)

(2) Change into the driver installation directory
*installDir/vxworks-6.x/target/3rdparty/tews/tpmc500*

(3) Invoke the build command for the required processor and build tool
*make CPU=cpuName TOOL=tool*

For Windows hosts this may look like this:

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc500
C:> make CPU=PENTIUM4 TOOL=diab
```

To compile SMP-enabled libraries, the argument VXBUILD=SMP must be added to the command line

```
C:> make CPU=PENTIUM4 TOOL=diab VXBUILD=SMP
```

To integrate the TPMC500 driver with the VxWorks development tools (Workbench), the component configuration file *40tpmc500.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc500
C:> copy 40tpmc500.cdf \Windriver\vxworks-6.7\target\config\comps\vxWorks
```

In VxWorks 6.7 and newer releases the kernel configuration tool scans the CDF file automatically and updates the *CxrCat.txt* cache file to provide component parameter information for the kernel configuration tool as long as the timestamp of the copied CDF file is newer than the one of the *CxrCat.txt*. If your copy command preserves the timestamp, force to update the timestamp by a utility, such as *touch*.

In earlier VxWorks releases the CxrCat.txt file may not be updated automatically. In this case, remove or rename the original *CxrCat.txt* file and invoke the make command to force recreation of this file.

```
C:> cd \Windriver\vxworks-6.7\target\config\comps\vxWorks
C:> del CxrCat.txt
C:> make
```

After successful completion of all steps above and restart of the Wind River Workbench, the TPMC500 driver can be included in VxWorks projects by selecting the *“TEWS TPMC500 Driver“* component in the *“hardware (default) - Device Drivers”* folder with the kernel configuration tool.

### 2.2.1 Direct BSP Builds

In development scenarios with the direct BSP build method without using the Workbench or the vxprj command-line utility, the TPMC500 configuration stub files must be copied to the directory *installDir/vxworks-6.x/target/config/comps/src/hwif.* Afterwards the *vxbUsrCmdLine.c* file must be updated by invoking the appropriate make command.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc500
C:> copy tpmc500.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tpmc500.dr \Windriver\vxworks-6.7\target\config\comps\src\hwif

C:> cd \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> make vxbUsrCmdLine.c
```

## 2.3 Legacy Driver Installation

### 2.3.1 Include device driver in VxWorks projects

For including the TPMC500-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Extract all files from the archive TPMC500-SW-42-LEGACY.zip to your project directory.

(2) Add the device driver's C-files to your project.
Make a right click to your project in the 'Workspace' window and use the 'Add Files ...' topic.
A file select box appears, and the driver files in the tpmc500 directory can be selected.

(3) Now the driver is included in the project and will be built with the project.

**For a more detailed description of the project facility please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)**

### 2.3.2 Special installation for Intel x86 based targets

The TPMC500 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro **CPU_FAMILY**. If the content of this macro is equal to *I80X86* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, the required device memory spaces can't be accessed.

To solve this problem an MMU mapping entry has to be added for the required TPMC500 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tpmc500pci.c** contains the function *tpmc500PciInit().* This routine finds out all TPMC500 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*).

The right place to call the function *tpmc500PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window):

```
    tpmc500PciInit();
```

Be sure that the function is called prior to MMU initialization otherwise the TPMC500 PCI spaces remains unmapped and an access fault occurs during driver initialization.

**Modifying the sysLib.c file will change the sysLib.c in the BSP path. Remember this for future projects and recompilations.**

### 2.3.3 BSP dependent adjustments

The driver includes a file called *include/tdhal.h* which contains functions and definitions for BSP adaptation. It may be necessary to modify them for BSP specific settings. Most settings can be made automatically by conditional compilation set by the BSP header files, but some settings must be configured manually. There are two ways of modification, first you can change the *include/tdhal.h* and define the corresponding definition and its value, or you can do it, using the command line option *–D*.

There are 3 offset definitions (*USERDEFINED_MEM_OFFSET*, *USERDEFINED_IO_OFFSET*, and *USERDEFINED_LEV2VEC*) that must be configured if a corresponding warning message appears during compilation. These definitions always need values. Definition values can be assigned by command line option *-D<definition>=<value>*.

| **definition** | **description** |
|---|---|
| USERDEFINED_MEM_OFFSET | The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI memory space access |
| USERDEFINED_IO_OFFSET | The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI I/O space access |
| USERDEFINED_LEV2VEC | The value of this definition must be set to the difference of the interrupt vector (used to connect the ISR) and the interrupt level (stored to the PCI header ) |

Another definition allows a simple adaptation for BSPs that utilize a *pciIntConnect()* function to connect shared (PCI) interrupts. If this function is defined in the used BSP, the definition of *USERDEFINED_SEL_PCIINTCONNECT* should be enabled. The definition by command line option is made by *-D<definition>.*

**Please refer to the BSP documentation and header files to get information about the interrupt connection function and the required offset values.**

## 2.4 System resource requirement

The table gives an overview over the system resources that will be needed by the driver.

| Resource | Driver requirement | Devices requirement |
|---|---|---|
| Memory | < 1 KB | < 1 KB |
| Stack | < 1 KB | --- |
| Semaphores | --- | 2 |

**Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.**

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

**The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.**

# 3 API Documentation

## 3.1 General Functions

### 3.1.1 tpmc500Open()

#### Name

tpmc500Open() – opens a device.

#### Synopsis

```
TPMC500_DEV tpmc500Open
(
        char            *DeviceName
)
```

#### Description

Before I/O can be performed to a device, a file descriptor must be opened by a call to this function.

#### Parameters

*DeviceName*

This parameter points to a null-terminated string that specifies the name of the device. The first TPMC500 device is named "/tpmc500/0", the second device is named "/tpmc500/1" and so on.

#### Example

```
#include “tpmc500.h”

TPMC500_DEV   pDev;

/*
** open file descriptor to device
*/
pDev = tpmc500Open(“/tpmc500/0”);
if (pDev == NULL)
{
     /* handle open error */
}
```

## RETURNS

A device descriptor pointer, or NULL if the function fails. An error code will be stored in *errno*.

## ERROR CODES

The error codes are stored in *errno.*

The error code is a standard error code set by the I/O system.

### 3.1.2 tpmc500Close()

#### Name

tpmc500Close() – closes a device.

#### Synopsis

```
int tpmc500Close
(
    TPMC500_DEV   pDev
)
```

#### Description

This function closes previously opened devices.

#### Parameters

*pDev*

This value specifies the file descriptor pointer to the hardware module retrieved by a call to the corresponding open-function.

#### Example

```
#include “tpmc500.h”

TPMC500_DEV    pDev;
int            result;

/*
** close file descriptor to device
*/
result = tpmc500Close(pDev);
if (result < 0)
{
    /* handle close error */
}
```

## RETURNS

Zero, or -1 if the function fails. An error code will be stored in *errno*.

## ERROR CODES

The error codes are stored in *errno*.

The error code is a standard error code set by the I/O system.

## 3.2 Device Access Functions

### 3.2.1 tpmc500SetModelType

#### Name

tpmc500SetModelType – configures the TPMC500 board type

#### Synopsis

```
STATUS tpmc500SetModelType
(
        TPMC500_DEV   pDev,
        int           modType
)
```

#### Description

This function configures the TPMC500 board type.

**This function must be called after initialization of the ADC device, before any other function accesses the device.**

#### Parameters

*pDev*

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

*modType*

This parameter specifies the model type of the TPMC500. The following model types are supported.

| Model type | Description |
|---|---|
| 10 | TPMC500-10 --- input range +/- 10V, gains: 1,2,5,10, front panel I/O |
| 11 | TPMC500-11 --- input range +/- 10V, gains: 1,2,4,8, front panel I/O |
| 12 | TPMC500-12 --- input range 0…10V, gains: 1,2,5,10, front panel I/O |
| 13 | TPMC500-13 --- input range 0…10V, gains: 1,2,4,8, front panel I/O |
| 20 | TPMC500-20 --- input range +/- 10V, gains: 1,2,5,10, back I/O |
| 21 | TPMC500-21 --- input range +/- 10V, gains: 1,2,4,8, back I/O |
| 22 | TPMC500-22 --- input range 0…10V, gains: 1,2,5,10, back I/O |
| 23 | TPMC500-23 --- input range 0…10V, gains: 1,2,4,8, back I/O |

## Example

```
#include “tpmc500.h”

TPMC500_DEV         pDev;
STATUS              result;

/*
** tell the driver, this is a TPMC500-10
*/
result = tpmc500SetModelType(pDev,
                             10);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* succesful */
}
```

## RETURN VALUE

OK if function succeeds or ERROR.

## ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

| Error code | Description |
|---|---|
| EINVAL | A NULL pointer is referenced for an input value |
| EBADF | The device handle is invalid |
| ENOTSUP | Unsupported or invalid TPMC500 model type specified |
| S_tpmc500Drv_MODBUSY | The module is busy |

## 3.2.2 tpmc500ReadSE

### Name

tpmc500ReadSE – make AD conversion and read value from a single-ended channel

### Synopsis

```
STATUS tpmc500ReadSE
(
        TPMC500_DEV   pDev,
        int           channel,
        int           gain,
        BOOL          fastConv,
        int           *pAdcVal
)
```

### Description

This function executes an AD conversion on a specified single-ended input channel and returns the value.

### Parameters

*pDev*

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

*channel*

This argument specifies the input channel. Allowed values are 1 up to 32 corresponding to the single-ended channel name.

*gain*

This argument specifies the input gain that shall be used. Allowed values are 1, 2, 5, 10 or 1, 2, 4, 8 depending on the module type.

*fastConv*

This argument specifies if fast reads shall be enabled ore not.

| Value | Description |
|---|---|
| FALSE | The standard (interrupt driven) mode will be used. Waiting the settling time (if necessary) and conversion time will be done interrupt driven. The driver task will be pending, waiting for a semaphore. The processor will be able to handle other tasks and interrupts. |
| TRUE | The fast (polled) mode will be used. The driver will not use interrupts, instead it will wait in a busy loop until the settling time (if necessary) and the conversion is finished. Conversions using this mode will be handled faster, but the processor executes a busy loop and other tasks will not be handled during the loops. |

*pAdcVal*

This argument points to a buffer where the AD value will be returned.

## Example

```
#include “tpmc500.h”

TPMC500_DEV         pDev;
STATUS              result;
int                 in_value;

/*
** read AD value from channel 5 with gain = 2 (use interrupts)
*/
result = tpmc500ReadSE (pDev,
                        5,
                        2,
                        FALSE,
                        &in_value);
if (result == ERROR)
{
     /* handle error */
}
else
{
     printf(“ADC #5: %d\n”, in_value);
}
```

## RETURN VALUE

OK if function succeeds or ERROR.

## ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

| **Error code** | **Description** |
|---|---|
| EINVAL | A NULL pointer is referenced for an input value |
| EBADF | The device handle is invalid |
| S_tpmc500Drv_ICHAN | Invalid channel number specified |
| S_tpmc500Drv_IGAIN | Invalid gain specified |
| S_tpmc500Drv_MODBUSY | The module is busy |
| S_tpmc500Drv_TIMEOUT | The conversion timed out |
| S_tpmc500Drv_UNKNOWNTYPE | Unknown TPMC500 model type |

### 3.2.3 tpmc500ReadCorrSE

#### Name

tpmc500ReadCorrSE – make AD conversion and read corrected value from a single-ended channel

#### Synopsis

```
STATUS tpmc500ReadCorrSE
(
        TPMC500_DEV   pDev,
        int           channel,
        int           gain,
        BOOL          fastConv,
        int           *pAdcVal
)
```

#### Description

This function executes an AD conversion on a specified single-ended input channel and returns a corrected value. The raw value read after the conversion has completed will be corrected with factory set correction data.

#### Parameters

*pDev*

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

*channel*

This argument specifies the input channel. Allowed values are 1 up to 32 corresponding to the single-ended channel name.

*gain*

This argument specifies the input gain that shall be used. Allowed values are 1, 2, 5, 10 or 1, 2, 4, 8 depending on the module type.

*fastConv*

This argument specifies if fast reads shall be enabled ore not.

| Value | Description |
|---|---|
| FALSE | The standard (interrupt driven) mode will be used. Waiting the settling time (if necessary) and conversion time will be done interrupt driven. The driver task will be pending, waiting for a semaphore. The processor will be able handle other tasks and interrupts. |
| TRUE | The fast (polled) mode will be used. The driver will not use interrupts, instead it will wait in a busy loop until the settling time (if necessary) and the conversion is finished. Conversions using this mode will be handled faster, but the processor executes a busy loop and other tasks will not be handled during the loops. |

*pAdcVal*

This argument points to a buffer where the corrected AD value will be returned.

## Example

```
#include “tpmc500.h”

TPMC500_DEV        pDev;
STATUS             result;
int                in_value;

/*
** read corrected AD value from channel 5 with gain = 2 (fast)
*/
result = tpmc500ReadCorrSE (pDev,
                            5,
                            2,
                            TRUE,
                            &in_value);
if (result == ERROR)
{
     /* handle error */
}
else
{
     printf(“ADC #5: %d\n”, in_value);
}
```

## RETURN VALUE

OK if function succeeds or ERROR.

## ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

| **Error code** | **Description** |
|---|---|
| EINVAL | A NULL pointer is referenced for an input value |
| EBADF | The device handle is invalid |
| S_tpmc500Drv_ICHAN | Invalid channel number specified |
| S_tpmc500Drv_IGAIN | Invalid gain specified |
| S_tpmc500Drv_MODBUSY | The module is busy |
| S_tpmc500Drv_TIMEOUT | The conversion timed out |
| S_tpmc500Drv_UNKNOWNTYPE | Unknown TPMC500 model type |

## 3.2.4 tpmc500ReadDiff

### Name

tpmc500ReadDiff – make AD conversion and read value from a differential channel

### Synopsis

```
STATUS tpmc500ReadDiff
(
        TPMC500_DEV   pDev,
        int           channel,
        int           gain,
        BOOL          fastConv,
        int           *pAdcVal
)
```

### Description

This function executes an AD conversion on a specified differential input channel and returns the value.

### Parameters

*pDev*

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

*channel*

This argument specifies the input channel. Allowed values are 1 up to 16 corresponding to the differential channel name.

*gain*

This argument specifies the input gain that shall be used. Allowed values are 1, 2, 5, 10 or 1, 2, 4, 8 depending on the module type.

*fastConv*

This argument specifies if fast reads shall be enabled ore not.

| Value | Description |
|---|---|
| FALSE | The standard (interrupt driven) mode will be used. Waiting the settling time (if necessary) and conversion time will be done interrupt driven. The driver task will be pending, waiting for a semaphore. The processor will be able handle other tasks and interrupts. |
| TRUE | The fast (polled) mode will be used. The driver will not use interrupts, instead it will wait in a busy loop until the settling time (if necessary) and the conversion is finished. Conversions using this mode will be handled faster, but the processor executes a busy loop and other tasks will not be handled during the loops. |

*pAdcVal*

This argument points to a buffer where the AD value will be returned.

## Example

```
#include “tpmc500.h”

TPMC500_DEV        pDev;
STATUS             result;
int                in_value;

/*
** read AD value from channel 5 with gain = 2 (use interrupts)
*/
result = tpmc500ReadDiff (pDev,
                          5,
                          2,
                          FALSE,
                          &in_value);
if (result == ERROR)
{
    /* handle error */
}
else
{
    printf(“ADC #5: %d\n”, in_value);
}
```

### RETURN VALUE

OK if function succeeds or ERROR.

### ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

| Error code | Description |
|---|---|
| EINVAL | A NULL pointer is referenced for an input value |
| EBADF | The device handle is invalid |
| S_tpmc500Drv_ICHAN | Invalid channel number specified |
| S_tpmc500Drv_IGAIN | Invalid gain specified |
| S_tpmc500Drv_MODBUSY | The module is busy |
| S_tpmc500Drv_TIMEOUT | The conversion timed out |
| S_tpmc500Drv_UNKNOWNTYPE | Unknown TPMC500 model type |

## 3.2.5 tpmc500ReadCorrDiff

### Name

tpmc500ReadCorrDiff – make AD conversion and read corrected value from a differential channel

### Synopsis

```
STATUS tpmc500ReadCorrDiff
(
        TPMC500_DEV   pDev,
        int           channel,
        int           gain,
        BOOL          fastConv,
        int           *pAdcVal
)
```

### Description

This function executes an AD conversion on a specified differential input channel and returns a corrected value. The raw value read after the conversion has completed will be corrected with factory set correction data.

### Parameters

*pDev*

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

*channel*

This argument specifies the input channel. Allowed values are 1 up to 16 corresponding to the single-ended channel name.

*gain*

This argument specifies the input gain that shall be used. Allowed values are 1, 2, 5, 10 or 1, 2, 4, 8 depending on the module type.

*fastConv*

This argument specifies if fast reads shall be enabled ore not.

| Value | Description |
|---|---|
| FALSE | The standard (interrupt driven) mode will be used. Waiting the settling time (if necessary) and conversion time will be done interrupt driven The driver task will be pending, waiting for a semaphore. The processor will be able handle other tasks and interrupts. |
| TRUE | The fast (polled) mode will be used. The driver will not use interrupts, instead it will wait in a busy loop until the settling time (if necessary) and the conversion is finished. Conversions using this mode will be handled faster, but the processor executes a busy loop and other tasks will not be handled during the loops. |

*pAdcVal*

This argument points to a buffer where the corrected AD value will be returned.

## Example

```
#include “tpmc500.h”

TPMC500_DEV         pDev;
STATUS              result;
int                 in_value;

/*
** read corrected AD value from channel 5 with gain = 2 (fast)
*/
result = tpmc500ReadCorrDiff (pDev,
                              5,
                              2,
                              TRUE,
                              &in_value);
if (result == ERROR)
{
    /* handle error */
}
else
{
    printf(“ADC #5: %d\n”, in_value);
}
```

## RETURN VALUE

OK if function succeeds or ERROR.

## ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

| **Error code** | **Description** |
|---|---|
| EINVAL | A NULL pointer is referenced for an input value |
| EBADF | The device handle is invalid |
| S_tpmc500Drv_ICHAN | Invalid channel number specified |
| S_tpmc500Drv_IGAIN | Invalid gain specified |
| S_tpmc500Drv_MODBUSY | The module is busy |
| S_tpmc500Drv_TIMEOUT | The conversion timed out |
| S_tpmc500Drv_UNKNOWNTYPE | Unknown TPMC500 model type |

## 3.2.6 tpmc500StartSequencer

### Name

tpmc500StartSequencer – configure TPMC500 for sequencer mode and start it.

### Synopsis

```
STATUS tpmc500StartSequencer
(
        TPMC500_DEV                    pDev,
        TPMC500_SEQ_BUFFER         *pSeqBuf
)
```

### Description

This function configures the sequencer mode and starts its execution.

The data will be stored into a ring buffer. The put and get index will indicate the start of a set of data (a set over all enabled channels). The wrap around will be made if there is not enough space in the buffer to store a complete set, e.g. there is memory for only one value, but we have two active channels, the data will be stored at index 0. Below is a simple example of a buffer which shall show how the buffer is organized. The buffer has a size for 7 values, 2 channels are active.

| **Buffer index** | **get/put index** | **Value (after start)** | **Value (1st wraparound)** | **Value (2nd wraparound)** |
|---|---|---|---|---|
| 0 | 0 | 1st value #1 | 4th value #1 | 7th value #1 |
| 1 | | 1st value #2 | 4th value #2 | 7th value #2 |
| 2 | 1 | 2nd value #1 | 5th value #1 | … |
| 3 | | 2nd value #2 | 5th value #2 | … |
| 4 | 2 | 3rd value #1 | 6th value #1 | … |
| 5 | | 3rd value #2 | 6th value #2 | … |
| 6 | | *Never used* | *Never used* | *Never used* |

## Parameters

*pDev*

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

*pSeqBuf*

This argument points to a structure containing the configuration and buffers for sequencer mode.

```
typedef struct
{
    unsigned short          cycletime;
    unsigned long           act_channels;
    TPMC500_IO_BUFFER       *chan_setup;
    unsigned long           buf_size;
    unsigned long           buf_stat;
    unsigned long           putIdx;
    unsigned long           getIdx;
    int                     *buffer;
} TPMC500_SEQ_BUF
```

*cycletime*

This argument specifies the length of one sequencer cycle. This value is specified in 100µs steps. Allowed values are between 1 and 65535. A specified value of 0 enables the continuous mode.

*act_channels*

This value specifies the number of active channels. Valid values are 1 to 32.

*chan_setup*

This parameter points to an array of data structures specifying the channel setup. The used data structure TPMC500_IO_BUFFER will be described below.

*buf_size*

This value specifies the size of the input FIFO. The value specifies the number of sequences that can be handled without reading before the buffer is filled.

*buf_stat*

This value specifies the current state and errors will be shown in this argument. The state is an ORed value of the following flags:

| **Flags** | **Description** |
|---|---|
| TPMC500_SEQ_BUF_OVERRUN | The user supplied FIFO is full and new data cannot be stored |
| TPMC500_SEQ_DATA_OVERFLOW | Old data not read by the software when new values are ready |
| TPMC500_SEQ_TIMER_ERR | The specified cycle time is not long enough to convert the specified channels |
| TPMC500_SEQ_INST_RAM_ERR | The sequencer is started, but no channel has been selected. |

*putIdx*

This parameter holds the current put index, which specifies the position in the FIFO where the next data will be written to. This index should just be read for information but never be changed by the application.

*getIdx*

This parameter holds the current get index, which specifies the position, where the application shall read the next value from FIFO. This index must be modified by the application after reading a value from the FIFO. This index is not changed by the driver.

*buffer*

This parameter points to the user supplied memory area where the sequencer input data will be stored in.

**data structure TPMC500_IO_BUFFER:**

```
typedef struct
{
        int                 Channel;
        int                 Gain;
        unsigned long       flags;
        int                 value;
} TPMC500_IO_BUFFER;
```

*Channel*

This argument specifies the ADC channel to use. Allowed values are 1..32 for single-ended channels and 1..16 for differential channels.

*Gain*

This argument specifies the input gain that shall be used. Allowed values are 1, 2, 5, 10 or 1, 2, 4, 8 depending on the type of the device.

*flags*

This parameter specifies conversion flags setting the input mode. The value is an ORed value of the following flags:

| **Flags** | **Description** |
|---|---|
| TPMC500_DIFFMODE | If set differential input interface is used, if not set the input interface will be single-ended. |
| TPMC500_CORRENA | If this flag is set input data corrections is enabled and ADC data will be corrected with factory set correction data individual for every TPMC500. If the flag is not set the ADC value will be returned without correction. |

*value*

This argument will be unused.

## Example

```
#include “tpmc500.h”

#define   SBUF_SIZE     0x200

TPMC500_DEV         pDev;
STATUS              result;
TPMC500_SEQ_BUF     seq_buf;
TPMC500_IO_BUFFER   seq_rw_par[2];
long                seq_data_buf[SBUF_SIZE];
int                 oldIndex;
int                 i;
int                 arrayOff;

/*
**  Start sequencer using channel 1 and 3
**  Channel 1:
**      differential, data correction on, gain = 1
**  Channel 3:
**      single-ended, data correction off, gain = 5
*/
seq_buf.cycletime       = 10000;          /* Cycletime 1s */
seq_buf.buffer          = seq_data_buf;
seq_buf.act_channels    = 2;
seq_buf.buf_size        = SBUF_SIZE / seq_buf.act_channels;
seq_buf.chan_setup      = seq_rw_par;

…
```

```
…

seq_rw_par[0].Channel   = 1;
seq_rw_par[0].Gain      = 1;
seq_rw_par[0].flags     = TPMC500_CORRENA | TPMC500_DIFFMODE;
seq_rw_par[1].Channel   = 3;
seq_rw_par[1].Gain      = 5;
seq_rw_par[1].Mode      = TPMC500_CORRDIS | TPMC500_SNGLMODE;

result = tpmc500StartSequencer(pDev,
                              &seq_buf);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* operation successful ==> read data from buffer */
     oldIndex = seq_buf.putIdx;
     while (1)
     {
          taskDelay(2);

          /* Check if error occurred */
          if (seq_buf.buf_stat)
          {
               printf("\n\nERROR: status = %08lXh\n", seq_buf.buf_stat);
               break;        /* Error occurred ==> exit loop */
          }

          /* New data available ? */
          if (oldIndex != seq_buf.putIdx)
          {
               arrayOff = (seq_buf.getIdx * seq_buf.act_channels);

               /* Loop over channels */
               for (i = 0; i < seq_buf.act_channels; i++)
               {
                    printf("%d", seq_data_buf[arrayOff + i]);
               }

               seq_buf.getIdx++;

               …
```

```
                  …

                  /* Wrap around ? */
                  if (seq_buf.getIdx >= seq_buf.buf_size)
                  {
                       seq_buf.getIdx = 0;
                  }
                  oldIndex = seq_buf.putIdx;
             }
        }
   }
```

## RETURN VALUE

OK if function succeeds or ERROR.

## ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

| **Error code** | **Description** |
|---|---|
| EINVAL | A NULL pointer is referenced for an input value |
| EBADF | The device handle is invalid |
| S_tpmc500Drv_ICHAN | Invalid channel number specified |
| S_tpmc500Drv_IGAIN | Invalid gain specified |
| S_tpmc500Drv_MODBUSY | The module is busy |
| S_tpmc500Drv_UNKNOWNTYPE | Unknown TPMC500 model type |

## 3.2.7 tpmc500StopSequencer

### Name

tpmc500StopSequencer – stops sequencer mode of specified device

### Synopsis

```
STATUS tpmc500StopSequencer
(
        TPMC500_DEV   pDev
)
```

### Description

This function stops execution of the sequencer mode on the specified device.

### Parameters

*pDev*

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

### Example

```
#include “tpmc500.h”

TPMC500_DEV        pDev;
STATUS             result;

/*
** stop sequencer mode
*/
result = tpmc500StopSequencer(pDev);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* succesful */
}
```

## RETURN VALUE

OK if function succeeds or ERROR.

## ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

| Error code | Description |
|---|---|
| EINVAL | A NULL pointer is referenced for an input value |
| EBADF | The device handle is invalid |
| S_tpmc500Drv_MODBUSY | The module is busy |

# 4 Legacy I/O system functions

This chapter describes the legacy driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

**The legacy I/O system functions are only relevant for the legacy TPMC500 driver. For the VxBus-enabled TPMC500 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.**

## 4.1 tpmc500Drv()

### NAME

tpmc500Drv() - installs the TPMC500 driver in the I/O system

### SYNOPSIS

#include “tpmc500.h”

STATUS tpmc500Drv(void)

### DESCRIPTION

This function searches for devices on the PCI bus, installs the TPMC500 driver in the I/O system.

**A call to this function is the first thing the user has to do before adding any device to the system or performing any I/O request.**

### EXAMPLE

```
#include  "tpmc500.h”

STATUS             result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tpmc500Drv();
if (result == ERROR)
{
     /* Error handling */
}
```

## RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

## ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

| Error code | Description |
|---|---|
| ENXIO | No TPMC500 module found |

## SEE ALSO

VxWorks Programmer’s Guide: I/O System

## 4.2 tpmc500DevCreate()

### NAME

tpmc500DevCreate() – Add a TPMC500 device to the VxWorks system

### SYNOPSIS

```
#include "tpmc500.h"

STATUS tpmc500DevCreate
(
        char            *name,
        int             devIdx,
        int             funcType
)
```

### DESCRIPTION

This routine adds the selected device to the VxWorks system. The device hardware will be setup and prepared for use.

**This function must be called before performing any I/O request to this device.**

### PARAMETER

*name*

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

*devIdx*

This index number specifies the device to add to the system. The device numbers will be assigned in the order the VxWorks *pciFindDevice()* function will find the devices. A 0 selects the first device, a 1 the second, and so on.

*funcType*

This parameter is unused and should be set to *0*.

## EXAMPLE

```
#include  "tpmc500.h”

STATUS             result;

/*-----------------------------------------------------------
  Create the device "/tpmc500/0" for the first TPMC500 device
  -----------------------------------------------------------*/

result = tpmc500DevCreate(   "/tpmc500/0",
                             0,
                             0);
if (result == OK)
{
     /* Device successfully created */
}
else
{
     /* Error occurred when creating the device */
}
```

## RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

## ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

| Error code | Description |
|---|---|
| S_ioLib_NO_DRIVER | The driver has not been installed (call tpmc500Drv()) |
| ENXIO | Specified device not found |
| EBUSY | The specified device has already been created. |
| ENOTSUP | The specified model type is not supported |

## SEE ALSO

VxWorks Programmer’s Guide: I/O System

## 4.3 tpmc500PciInit()

### NAME

tpmc500PciInit() – Generic PCI device initialization

### SYNOPSIS

void tpmc500PciInit()

### DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC500 PCI spaces (base address register) and to enable the TPMC500 device for access.

The global variable *tpmc500Status* obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

| Value | Meaning |
|---|---|
| > 0 | Initialization successful completed. The value of tpmc500Status is equal to the number of mapped PCI spaces |
| 0 | No TPMC500 device found |
| < 0 | Initialization failed. The value of (tpmc500Status & 0xFF) is equal to the number of mapped spaces until the error occurs.<br>Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[].<br>Remedy: Add dummy entries as necessary (syslib.c). |

### EXAMPLE

```
extern void tpmc500PciInit();

…

tpmc500PciInit();
```

## 4.4 tpmc500Init()

### NAME

tpmc500Init() – initialize TPMC500 driver and devices

### SYNOPSIS

#include “tpmc500.h”

STATUS tpmc500Init(void)

### DESCRIPTION

This function is used by the TPMC500 example application to install the driver and to add all available devices to the VxWorks system.

See also 3.1.1 tpmc500Open() for the device naming convention for legacy devices.

**After calling this function it is not necessary to call tpmc500Drv() and tpmc500DevCreate() explicitly.**

### EXAMPLE

```
#include  "tpmc500.h”

STATUS    result;

result = tpmc500Init();

if (result == ERROR)
{
     /* Error handling */
}
```

## RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

## ERROR CODES

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

See 4.1 tpmc500Drv() and 4.2 tpmc500DevCreate() for a description of possible error codes.

# 5 Basic I/O Functions

The VxWorks basic I/O interface functions are useable with the TPMC500 legacy and VxBus-enabled driver in a uniform manner.

## 5.1 open()

#### NAME

open() - open a device or file.

#### SYNOPSIS

```
int open
(
        const char   *name,
        int          flags,
        int          mode
)
```

#### DESCRIPTION

Before I/O can be performed to the TPMC500 device, a file descriptor must be opened by invoking the basic I/O function *open().*

#### PARAMETER

*name*

Specifies the device which shall be opened, the name specified in tpmc500DevCreate() must be used

*flags*

Not used

*mode*

Not used

## EXAMPLE

```
int       fd;

/*-----------------------------------------
  Open the device named "/tpmc500/0" for I/O
  -----------------------------------------*/
fd = open("/tpmc500/0", 0, 0);
if (fd == ERROR)
{
     /* Handle error */
}
```

## RETURNS

A device descriptor number or ERROR. If the function fails an error code will be stored in *errno*.

## ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

## SEE ALSO

ioLib, basic I/O routine - *open()*

## 5.2 close()

### NAME

close() – close a device or file

### SYNOPSIS

```
STATUS close
(
    int         fd
)
```

### DESCRIPTION

This function closes opened devices.

### PARAMETER

*fd*

This file descriptor specifies the device to be closed. The file descriptor has been returned by the *open()* function.

### EXAMPLE

```
int       fd;
STATUS    retval;

/*----------------
  close the device
  ----------------*/
retval = close(fd);
if (retval == ERROR)
{
     /* Handle error */
}
```

### RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

## ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

## SEE ALSO

ioLib, basic I/O routine - close()

## 5.3 ioctl()

### NAME

ioctl() - performs an I/O control function.

### SYNOPSIS

```
#include “tpmc500.h”

int ioctl
(
        int     fd,
        int     request,
        int     arg
)
```

### DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls (read, write) will be performed by calling the ioctl() function.

### PARAMETER

*fd*

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

*request*

This argument specifies the function that shall be executed. Following functions are defined:

| **Function** | **Description** |
|---|---|
| FIO_TPMC500READ | Set up and start sequencer mode |
| FIO_TPMC500STARTSEQ | Set up and start sequencer mode |
| FIO_TPMC500STOPSEQ | Stop sequencer mode |
| FIO_TPMC500MODULTYPE | Specify TPMC500 model type |

*arg*

This parameter depends on the selected function (request). How to use this parameter is described below with the function.

### RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

## ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual). Function specific error codes will be described with the function.

## SEE ALSO

ioLib, basic I/O routine - ioctl()

## 5.3.1 FIO_TPMC500MODULTYPE

This I/O control function specifies the model type of TPMC500. The function specific control parameter **arg** is a pointer to a TPMC500_CONF_BUFFER.

**This function must be called before any other I/O access is done to the specified device.**

```
typedef struct
{
        int             modelType;      /* specifies the model type of the TPMC500 (TPMC500-<xx>) */
} TPMC500_CONF_BUFFER;
```

*modelType*

This parameter specifies the model type of the TPMC500. The following model types are supported.

| Model type | Description |
|---|---|
| 10 | TPMC500-10 --- input range +/- 10V, gains: 1,2,5,10, front panel I/O |
| 11 | TPMC500-11 --- input range +/- 10V, gains: 1,2,4,8, front panel I/O |
| 12 | TPMC500-12 --- input range 0…10V, gains: 1,2,5,10, front panel I/O |
| 13 | TPMC500-13 --- input range 0…10V, gains: 1,2,4,8, front panel I/O |
| 20 | TPMC500-20 --- input range +/- 10V, gains: 1,2,5,10, back I/O |
| 21 | TPMC500-21 --- input range +/- 10V, gains: 1,2,4,8, back I/O |
| 22 | TPMC500-22 --- input range 0…10V, gains: 1,2,5,10, back I/O |
| 23 | TPMC500-23 --- input range 0…10V, gains: 1,2,4,8, back I/O |

### EXAMPLE

```
#include “tpmc500.h”

int                     fd;
TPMC500_CONF_BUFFER     confBuf;
int                     retval;

/*----------------------------------------
  Identify the TPMC500 device as TPMC500-11
  ----------------------------------------*/
confBuf.modelType = 11;

…
```

```
…

retval = ioctl(fd, FIO_TPMC500MODULTYPE, (int)&confBuf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

## ERROR CODES

| Error code | Description |
|---|---|
| ENOTSUP | Unsupported or invalid TPMC500 model type specified |
| S_tpmc500Drv_MODBUSY | The module is busy |

## 5.3.2 FIO_TPMC500READ

This I/O control function starts a conversion for one input channel and returns the value. The function specific control parameter **arg** is a pointer on a TPMC500_IO_BUF structure specifying the conversion parameters.

```
typedef struct
{
        int                 Channel;
        int                 Gain;
        unsigned long       flags;
        long                value;
} TPMC500_IO_BUFFER;
```

*Channel*

This argument specifies the ADC channel to use. Allowed values are 1..32 for single-ended channels and 1..16 for differential channels.

*Gain*

This argument specifies the input gain that shall be used. Allowed values are 1, 2, 5, 10 or 1, 2, 4, 8 depending on the type of the device.

*flags*

This parameter specifies conversion flags setting the input mode. The value is an ORed value of the following flags:

| **Flags** | **Description** |
|---|---|
| TPMC500_DIFFMODE | If set differential input interface is used, if not set the input interface will be single-ended. |
| TPMC500_CORRENA | If this flag is set input data corrections is enabled and ADC data will be corrected with factory set correction data individual for every TPMC500. If the flag is not set the ADC value will be returned without correction. |

*value*

This value returns the ADC input value. The possible range of the value depends on the device. If it is a bipolar version, input values are between -2048 (-10V) and 2047 (~+10V), unipolar versions return values between 0 (0V) and 4095 (~+10V).

## EXAMPLE

```
#include “tpmc500.h”

int                 fd;
TPMC500_IO_BUFFER   read_buf;
int                 retval;

/*----------------------------------------------------
  Read the current value of differential channel 1,
  the gain shall be 2 and the value shall be corrected
  ----------------------------------------------------*/
read_buf.Channel    = 1;
read_buf.Gain       = 2;
read_buf.flags      = TPMC500_CORRENA | TPMC500_DIFFMODE;
retval = ioctl(fd, FIO_TPMC500READ, (int)&read_buf);
if (retval != ERROR)
{
     /* read successful */
     printf(“ADC value: %ld”, read_buf.value);
}
else
{
     /* handle the error */
}
```

## ERROR CODES

| Error code | Description |
|---|---|
| S_tpmc500Drv_ICHAN | Illegal channel number specified |
| S_tpmc500Drv_IGAIN | Illegal gain specified |
| S_tpmc500Drv_MODBUSY | The device is in use or the sequencer is active |

## 5.3.3 FIO_TPMC500STARTSEQ

This I/O control function sets up the sequencer channels and cycle time and afterwards starts the sequencer. The function specific control parameter **arg** is a pointer on a TPMC500_SEQ_BUF structure that will be used while the sequencer is active.

**Data organization with in the buff is described in 3.2.6 tpmc500StartSequencer.**

```
typedef struct
{
        unsigned short        cycletime;
        unsigned long         act_channels;
        TPMC500_IO_BUFFER     *chan_setup;
        unsigned long         buf_size;
        unsigned long         buf_stat;
        unsigned long         putIdx;
        unsigned long         getIdx;
        long                  *buffer;
} TPMC500_SEQ_BUF
```

*cycletime*

This argument specifies the length of one sequencer cycle. This value is specified in 100µs steps. Allowed values are between 1 and 65535. A specified value of 0 enables the continuous mode.

*act_channels*

This value specifies the number of active channels. Valid values are 1 to 32.

*chan_setup*

This parameter points to an array of data structures specifying the channel setup. The used data structure is the same used by the read command (for more information refer to the description of the ioctl() function *FIO_TPMC500READ (5.3.2)*)

*buf_size*

This value specifies the size of the input FIFO. The value specifies the number of sequences that can be handled without reading before the buffer is filled.

*buf_stat*

This value specifies the current state and errors will be shown in this argument. The state is an ORed value of the following flags:

| Flags | Description |
|---|---|
| TPMC500_SEQ_BUF_OVERRUN | The user supplied FIFO is full and new data cannot be stored |
| TPMC500_SEQ_DATA_OVERFLOW | Old data not read by the software when new values are ready |
| TPMC500_SEQ_TIMER_ERR | The specified cycle time is not long enough to convert the specified channels |
| TPMC500_SEQ_INST_RAM_ERR | The sequencer is started, but no channel has been selected. |

*putIdx*

This parameter holds the current put index, which specifies the position in the FIFO where the next data will be written to. This index should just be read for information but never be changed by the application.

*getIdx*

This parameter holds the current get index, which specifies the position, where the application shall read the next value from FIFO. This index must be modified by the application after reading a value from the FIFO. This index is not changed by the driver.

*buffer*

This parameter points to the user supplied memory area where the sequencer input data will be stored in.

## EXAMPLE

```
#include “tpmc500.h”

#define   SBUF_SIZE     0x200

int                fd;
int                retval;
TPMC500_SEQ_BUF    seq_buf;
TPMC500_IO_BUFFER  seq_rw_par[2];
long               seq_data_buf[SBUF_SIZE];

…
```

…

```
/*-----------------------------------------------
  Start sequencer using channel 1 and 3
    Channel 1:
        differential, data correction on, gain = 1
    Channel 3:
        single-ended, data correction off, gain = 5
  -----------------------------------------------*/
seq_buf.cycletime       = 10000;           /* Cycletime 1s */
seq_buf.buffer          = seq_data_buf;
seq_buf.act_channels    = 2;
seq_buf.buf_size        = SBUF_SIZE / seq_buf.act_channels;
seq_buf.chan_setup      = seq_rw_par;

seq_rw_par[0].Channel   = 1;
seq_rw_par[0].Gain      = 1;
seq_rw_par[0].flags     = TPMC500_CORRENA | TPMC500_DIFFMODE;
seq_rw_par[1].Channel   = 3;
seq_rw_par[1].Gain      = 5;
seq_rw_par[1].Mode      = TPMC500_CORRDIS | TPMC500_SNGLMODE;

retval = ioctl(fd, FIO_TPMC500STARTSEQ, (int)&seq_buf);
if (retval != ERROR)
{
     /* sequencer started */
     /* reading data from buffer, see 3.2.6 tpmc500StartSequencer */
}
else
{
     /* handle the error */
}
```

## ERROR CODES

| Error code | Description |
|---|---|
| S_tpmc500Drv_MODBUSY | The device is currently in use |

## 5.3.4 FIO_TPMC500STOPSEQ

This I/O control function stops the sequencer. The function specific control parameter **arg** is not used for this function.

### EXAMPLE

```
#include “tpmc500.h”

int                fd;
int                retval;

/*------------------------
  Stop Sequencer
  ------------------------*/
retval = ioctl(fd, FIO_TPMC500STOPSEQ, 0);
if (retval != ERROR)
{
     /* Sequencer is stopped */
}
else
{
     /* handle the error */
}
```

### ERROR CODES

| Error code | Description |
|---|---|
| S_tpmc500Drv_MODBUSY | The device is currently in use |