The Embedded I/O Company

TPMC550-SW-42

VxWorks Device Driver

8/4 Channel 12 Bit D/A

Version 3.0.x

User Manual

Issue 3.0.0

August 2012

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

TPMC550-SW-42

VxWorks Device Driver

8/4 Channel 12 Bit D/A

Supported Modules:
TPMC550

Issue	Description	Date
1.0	First Issue	May 2001
1.1	General Revision	November 2003
1.1.1	File-list changed	August 8, 2005
2.0.0	API description added, VxBus support added, Legacy-driver: I/O interface updated, file list modified, General revision	February 2, 2011
3.0.0	API revised and documented, Basic I/O functions removed	August 28, 2012

Table of Contents

1 INTRODUCTION ... 4

2 INSTALLATION ... 5

2.1 Legacy vs. VxBus Driver ... 6

2.2 VxBus Driver Installation ... 7

2.2.1 Direct BSP Builds ... 8

2.3 Legacy Driver Installation ... 9

2.3.1 Include Device Driver in VxWorks Projects ... 9

2.3.2 Special Installation for Intel x86 based Targets ... 9

2.3.3 BSP Dependent Adjustments ... 10

2.4 System Resource Requirement ... 11

3 API DOCUMENTATION ... 12

3.1 General Functions ... 12

3.1.1 tpmc550Open ... 12

3.1.2 tpmc550Close ... 14

3.1.3 tpmc550GetModuleInfo ... 16

3.2 DAC Output Functions ... 18

3.2.1 tpmc550DacWrite ... 18

3.2.2 tpmc550DacWriteMulti ... 20

3.3 Sequencer Functions ... 22

3.3.1 tpmc550SeqSetup ... 22

3.3.2 tpmc550SeqStart ... 25

3.3.3 tpmc550SeqStop ... 27

3.3.4 tpmc550SeqWrite ... 29

3.3.5 tpmc550SeqFlush ... 32

3.3.6 tpmc550SeqStatus ... 34

4 LEGACY I/O SYSTEM FUNCTIONS ... 37

4.1 tpmc550PciInit ... 37

5 DEBUGGING AND DIAGNOSTIC ... 38

1 Introduction

The TPMC550-SW-42 VxWorks device driver software allows the operation of the supported PMC conforming to the VxWorks I/O system specification.

The TPMC550-SW-42-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 TPMC550-SW-42 device driver supports the following features:

- Setting DAC output value
- Configure, start, and stop DAC-sequencer
- Write data for sequencer cycle
- Use of data correction for simple conversion and in sequencer mode
- Use of latched writes for synchronous output
- Reading TPMC550 configuration (number of channels and uni-/bipolar output)

The TPMC550-SW-42 supports the modules listed below:

TPMC550-x0	8 channel 12-bit D/A	(PMC)
TPMC550-x1	4 channel 12-bit D/A	(PMC)

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

TPMC550 User Manual
TPMC550 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC550-SW-42':

TPMC550-SW-42-3.0.0.pdf	PDF copy of this manual
TPMC550-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TPMC550-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TPMC550-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tpmc550':

tpmc550drv.c	TPMC550 device driver source
tpmc550def.h	TPMC550 driver include file
tpmc550.h	TPMC550 include file for driver and application
tpmc550api.c	TPMC550 API file
tpmc550api.h	TPMC550 API include file
Makefile	Driver Makefile
40tpmc550.cdf	Component description file for VxWorks development tools
tpmc550.dc	Configuration stub file for direct BSP builds
tpmc550.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tpmc550exa.c	Example application

The archive TPMC550-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tpmc550':

tpmc550drv.c	TPMC550 device driver source
tpmc550def.h	TPMC550 driver include file
tpmc550.h	TPMC550 include file for driver and application
tpmc550pci.c	TPMC550 device driver source for x86 based systems
tpmc550api.c	TPMC550 API file
tpmc550api.h	TPMC550 API include file
tpmc550exa.c	Example application
tpmc550init.c	Legacy driver initialization
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 ▪ VxWorks 6.5 and earlier releases ▪ VxWorks 6.x releases without VxBus PCI bus support	▪ VxWorks 6.6 and later releases with VxBus PCI bus ▪ SMP systems (only the VxBus driver is SMP safe!) ▪ 64-bit systems (only the VxBus driver is 64-bit compatible)

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 TPMC550-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 TPMC550 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tpmc550.*

At this point the TPMC550 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/tpmc550

(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\tpmc550
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 build 64-bit libraries, the argument VXBUILD=LP64 must be added to the command line

```
> make CPU=CORE TOOL=gnu VXBUILD=LP64
```

For 64-bit SMP-enabled libraries a build command may look like this

```
> make CPU=CORE TOOL=gnu VXBUILD="LP64 SMP"
```

To integrate the TPMC550 driver with the VxWorks development tools (Workbench), the component configuration file *40tpmc550.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc550
C:> copy 40tpmc550.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 TPMC550 driver and API can be included in VxWorks projects by selecting the *"TEWS TPMC550 Driver" and "TEWS TPMC550 API"* components 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 TPMC550 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\tpmc550
C:> copy tpmc550.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tpmc550.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 TPMC550-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 TPMC550-SW-42-LEGACY.zip to your project directory.

(2) Add the device drivers 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 tpmc550 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 TPMC550 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 a MMU mapping entry has to be added for the required TPMC550 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tpmc550pci.c** contains the function *tpmc550PciInit().* This routine finds out all TPMC550 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 *tpmc550PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window):

```
tpmc550PciInit();
```

Be sure that the function is called prior to MMU initialization otherwise the TPMC550 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 way 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	---	1

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 tpmc550Open

NAME

tpmc550Open – opens a device.

SYNOPSIS

```
TPMC550_HANDLE tpmc550Open
(
        char            *DeviceName
)
```

DESCRIPTION

Before I/O can be performed to a device, a device handle must be opened by a call to this function. If the legacy TPMC550 driver is used, this function will also install the legacy driver and create devices with the first call. The VxBus TPMC550 driver will be installed automatically by the VxBus system.

The tpmc550Open function can be called multiple times (e.g. in different tasks).

PARAMETERS

DeviceName

This parameter points to a null-terminated string that specifies the name of the device. The following device naming must be used:

Device Number	Device Name
1	/tpmc550/0
2	/tpmc550/1

EXAMPLE

```
#include “tpmc550api.h”

TPMC550_HANDLE     hdl;

/*
** open the specified device
*/
hdl = tpmc550Open(“/tpmc550/0”);
if (hdl == NULL)
{
     /* handle open error */
}
```

RETURNS

A device handle, 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 tpmc550Close

NAME

tpmc550Close – closes a device.

SYNOPSIS

```
TPMC550_STATUS tpmc550Close
(
    TPMC550_HANDLE        hdl
)
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tpmc550api.h”

TPMC550_HANDLE     hdl;
TPMC550_STATUS     result;

/*
** close the device
*/
result = tpmc550Close(hdl);
if (result != TPMC550_OK)
{
     /* handle close error */
}
```

RETURNS

On success, TPMC550_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC550_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.3 tpmc550GetModuleInfo

NAME

tpmc550GetModuleInfo – Get module information

SYNOPSIS

```
TPMC550_STATUS tpmc550GetModuleInfo
(
        TPMC550_HANDLE          hdl,
        int                     *NumChan,
        int                     bipolar[TPMC550_MAX_CHAN],
        int                     OffsCorr[TPMC550_MAX_CHAN],
        int                     GainCorr[TPMC550_MAX_CHAN]
);
```

DESCRIPTION

This function reads module information data from the specified device.

PARAMETERS

hdl

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

NumChan

This argument is a pointer to an int variable where the number of available DAC channels is returned.

bipolar

This argument is a pointer to an int array where the configured voltage range of each DAC channel is returned as boolean value. The array element bipolar[0] contains the range stetting for DAC channel 1, bipolar[1] for DAC channel 2 and so forth. If the corresponding value is TRUE then the voltage range of the channel is configured to +/- 10V output (bipolar); otherwise it is configured to 0…10V output voltage range.

OffsCorr

This argument is a pointer to an int array where the factory programmed offset correction data is returned. OffsCorr[0] contains correction data for DAC channel 1, OffsCorr[1] for DAC channel 2 and so forth.

GainCorr

This argument is a pointer to an int array where the factory programmed gain correction data are returned. GainCorr[0] contains correction data for DAC channel 1, GainCorr[1] for DAC channel 2 and so forth.

EXAMPLE

```
#include “tpmc550api.h”

TPMC550_HANDLE   hdl;
TPMC550_STATUS   result;
int              NumChan;
int              bipolar[TPMC550_MAX_CHAN];
int              OffsCorr[TPMC550_MAX_CHAN];
int              GainCorr[TPMC550_MAX_CHAN];

/* Get module information data */

result = tpmc550GetModuleInfo(hdl, &NumChan, bipolar, OffsCorr, GainCorr);

if (result != TPMC550_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC550_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC550_ERR_INVALID_HANDLE	The specified TPMC550_HANDLE is invalid.

3.2 DAC Output Functions

3.2.1 tpmc550DacWrite

NAME

tpmc550DacWrite – write D/A value to specified channel

SYNOPSIS

```
TPMC550_STATUS tpmc550DacWrite
(
        TPMC550_HANDLE          hdl,
        int                     channel,
        unsigned int            flags,
        short                   value
);
```

DESCRIPTION

This function writes a new value to a specific channel and starts D/A conversion immediately in transparent mode

PARAMETERS

hdl

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

channel

This argument specifies the DAC channel which shall be updated. Possible values are 1 up to the number of available DAC channels of the specific module.

flags

This argument specifies a set of bit flags that control the D/A conversion:

Value	Description
TPMC550_CORR	Perform an offset and gain correction with factory calibration data stored in the TPMC550 EEPROM.

value

This argument specifies the new 12-bit D/A value. Valid data range depends on the voltage range of the specified channel (0…4095 for 0...10V voltage range and -2048…2047 for +/-10V voltage range).

EXAMPLE

```
#include “tpmc550api.h”

TPMC550_HANDLE  hdl;
TPMC550_STATUS  result;

result = tpmc550DacWrite(hdl, 1, TPMC550_CORR, 1234);

if (result != TPMC550_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC550_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC550_ERR_INVALID_HANDLE	The specified TPMC550_HANDLE is invalid.
TPMC550_ERR_RANGE	Invalid channel number
TPMC550_ERR_TIMEOUT	Timeout during D/A conversion
TPMC550_ERR_BUSY	This error occurs if the sequencer is still running. Please stop the sequencer before executing this function.

3.2.2 tpmc550DacWriteMulti

NAME

tpmc550DacWriteMulti – write D/A value to multiple channels

SYNOPSIS

```
TPMC550_STATUS tpmc550DacWriteMulti
(
        TPMC550_HANDLE        hdl,
        unsigned int          ChannelMask,
        unsigned int          flags,
        short                 values[TPMC550_MAX_CHAN]
);
```

DESCRIPTION

This function writes new values to specified channels and starts D/A conversion immediately (transparent mode) or simultaneously (latched mode).

PARAMETERS

hdl

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

ChannelMask

This argument selects DAC channels which shall be updated. A set (1) bit specifies that the corresponding channel shall be updated. Bit 0 corresponds to the first DAC channel, bit 1 corresponds to the second DAC channel and so on.

flags

This argument specifies a set of bit flags that control the D/A conversion:

Value	Description
TPMC550_CORR	Perform an offset and gain correction with factory calibration data stored in the TPMC550 EEPROM for all selected channels.
TPMC550_SIMCONV	Start conversion of selected channels in latched mode and update analog outputs simultaneously.

values

This array contains the new 12-bit D/A values. Valid data range depends on the voltage range of the specified channel (0…4095 for 0...10V voltage range and -2048…2047 for +/-10V voltage range).
Array index 0 corresponds to the first DAC channel, array index 1 corresponds to the second DAC channel and so on. Only channels selected for update (*ChannelMask)* will be modified.

EXAMPLE

```
#include “tpmc550api.h”

TPMC550_HANDLE  hdl;
TPMC550_STATUS  result;
unsigned int    ChannelMask;
unsigned int    flags;
short           values[TPMC550_MAX_CHAN];

/* Update channel 1, 4 and 8 simultaneously with corrected D/A values */
ChannelMask = (1<<0) | (1<<3) | (1<<7);
flags = TPMC550_CORR | TPMC550_SIMCONV;
value[0] = 111;    /* channel 1 */
value[3] = 444;    /* channel 4 */
value[7] = 888;    /* channel 8 */

result = tpmc550DacWriteMulti(hdl, ChannelMask, flags, values);

if (result != TPMC550_OK)
{
    /* handle error */
}
```

RETURNS

On success, TPMC550_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC550_ERR_INVALID_HANDLE	The specified TPMC550_HANDLE is invalid.
TPMC550_ERR_RANGE	Invalid channel number
TPMC550_ERR_TIMEOUT	Timeout during D/A conversion
TPMC550_ERR_BUSY	This error occurs if the sequencer is still running. Please stop the sequencer before executing this function.

3.3 Sequencer Functions

3.3.1 tpmc550SeqSetup

NAME

tpmc550SeqSetup – Setup sequencer facility

SYNOPSIS

```
TPMC550_STATUS tpmc550SeqSetup
(
        TPMC550_HANDLE      hdl,
        int                 CycleTime,
        int                 NumActiveChannels,
        int                 NumBufTuples,
        int                 ChannelAllocation[TPMC550_MAX_CHAN],
        unsigned int        flags
);
```

DESCRIPTION

This function configures the sequencer facility and allocates memory for the sequencer software ring buffer. The behaviour of the sequencer facility is controlled by a set of bit flags which are described below.

Basically the sequencer will perform a D/A conversion on active channels in a deterministic time period controlled by a cycle timer or the duration of the conversion itself. To be sure that D/A data will be available for the next cycle just in-time, data for the sequencer will be provided by a configurable ring buffer. The ring buffer can be asynchronously filled by the application program.

The sequencer facility provides two operating modes. In loop mode (TPMC550_LOOP) the buffer will be filled completely with new data (e.g. wave form). The contents of the buffer will be output continuously in a loop. In normal mode (TPMC550_LOOP is not set) the application program must provide new data for every cycle. If the buffer is empty then the sequencer will stop and it holds the last output value until new data arrives.

PARAMETERS

hdl

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

CycleTime

This argument specifies the sequencer cycle time in steps of 100 µs. This argument is only relevant if the flag TPMC550_TIMERMODE is set.

NumActiveChannels

This argument specifies the number of active channels. Valid range is 1 up to the number of available channels (4 or 8).

NumBufTuples

This argument specifies the size of the sequencer software ring buffer. In this case size is not the number of bytes to allocate but rather the number of tuples (data for all active channels per cycle).

ChannelAllocation

This argument specifies the channel number of active channels and their enumeration inside a tuple. The function tpmc550SeqWrite awaits new data for active channels in this order. The first array element contains the channel number (1...n) of the first active channel. The second array element the channel number of the second active channel and so forth. Unused array elements can be left undefined.

flags

This argument specifies a set of bit flags that control the sequencer operation:

Value	**Description**
TPMC550_TIMERMODE	If set, the cycle of D/A conversions will be controlled by the sequencer timer in steps of 100 microseconds; otherwise the sequencer will run in continuous mode as fast as possible (based on the conversion time).
TPMC550_LOOP	If this flag is set (loop mode) the ring buffer never becomes empty. Once completely filled the sequencer will continuously get data out of the buffer for the next conversion. If this flag is not set (normal mode) and the buffer becomes empty then the sequencer will stop and it holds the last output value until new data arrives.
TPMC550_CORR	Perform an offset and gain correction with factory calibration data stored in the TPMC550 EEPROM for all selected channels.
TPMC550_SIMCONV	Start conversion of active channels in latched mode and update analog outputs simultaneously.

EXAMPLE

```
#include “tpmc550api.h”

TPMC550_HANDLE   hdl;
TPMC550_STATUS   result;
int              ChannelAllocation[TPMC550_MAX_CHAN];
unsigned int     flags;

/* Setup the sequencer with 2 active channels (1 and 4) in timer mode */
/* with 1 ms cycle time. The sequencer buffer shall store data tuples */
/* for up to 100 cycles. */

ChannelAllocation[0] = 1;
ChannelAllocation[1] = 4;
flags = TPMC550_TIMERMODE | TPMC550_CORR | TPMC550_SIMCONV;

result = tpmc550SeqSetup(hdl, 10, 2, 100, ChannelAllocation, flags);

if (result != TPMC550_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC550_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC550_ERR_INVALID_HANDLE	The specified TPMC550_HANDLE is invalid.
TPMC550_ERR_RANGE	Invalid channel number or invalid number of channels.
TPMC550_ERR_NOMEM	Unable to allocate memory for the ring buffer.
TPMC550_ERR_BUSY	This error occurs if the sequencer is still running. Please stop the sequencer before executing this function.

3.3.2 tpmc550SeqStart

NAME

tpmc550SeqStart – start sequencer facility

SYNOPSIS

```
TPMC550_STATUS tpmc550SeqStart
(
    TPMC550_HANDLE      hdl
);
```

DESCRIPTION

This function starts the sequencer facility. Before calling this function the sequencer must be setup with tpmc550SeqSetup und the ring buffer must be filled with tpmc550SeqWrite.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tpmc550api.h”

TPMC550_HANDLE  hdl;
TPMC550_STATUS  result;

/* start the sequencer */
result = tpmc550SeqStart(hdl);

if (result != TPMC550_OK)
{
    /* handle error */
}
```

RETURNS

On success, TPMC550_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC550_ERR_INVALID_HANDLE	The specified TPMC550_HANDLE is invalid.
TPMC550_ERR_NOT_READY	The sequencer facility was not properly configured. Execute the function tpmc550SeqSetup first.
TPMC550_ERR_NODATA	No data is available in the ring buffer to start the sequencer facility. Use the function tpmc550SeqWrite to write at least one data tuple before starting the sequencer.

3.3.3 tpmc550SeqStop

NAME

tpmc550SeqStop – stop the sequencer facility

SYNOPSIS

```
TPMC550_STATUS tpmc550SeqStop
(
        TPMC550_HANDLE        hdl
);
```

DESCRIPTION

This function stops the sequencer facility. All allocated resources (e.g. ring buffer memory) will be freed.

PARAMETERS

hdl

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

EXAMPLE

```
#include “tpmc550api.h”

TPMC550_HANDLE  hdl;
TPMC550_STATUS  result;

/* stop the sequencer */
result = tpmc550SeqStop(hdl);

if (result != TPMC550_OK)
{
     /* handle error */
}
```

RETURNS

On success, TPMC550_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC550_ERR_INVALID_HANDLE	The specified TPMC550_HANDLE is invalid.

3.3.4 tpmc550SeqWrite

NAME

tpmc550SeqWrite – write new sequencer data

SYNOPSIS

```
TPMC550_STATUS tpmc550SeqWrite
(
        TPMC550_HANDLE        hdl,
        int                   size,
        short                 *values,
        int                   *WrittenSize
);
```

DESCRIPTION

This function writes new data to the sequencers data buffer. The provided data buffer must always contain new data for all active channels (tuple). The number of tuples per write must be at least one up to “unlimited”. This function will always write as many tuples as possible. If the buffer becomes full the function will return immediately with the error TPMC550_ERR_BUF_FULL. The number of written bytes will be returned in a variable pointed to by WrittenSize.

PARAMETERS

hdl

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

size

This argument specifies the size (in bytes) of the data buffer to write.

values

This argument is a pointer to an array of short variables that contains data for all active channels for at least one sequencer cycle (tuple). Despite of the declaration as simple short pointer this array is treated as a two-dimensional array with variable dimensions. The rows of the array represent the number of tuples and the columns the number of active channels. A declaration of this array will look like this: *data[tuples][channels]*.

WrittenSize

This argument is a pointer to an int variable where the number of written bytes is returned. In case of the error TPMC550_ERR_BUF_FULL this value can be used to adjust the buffer start pointer for subsequent writes.

EXAMPLE

```
#include “tpmc550api.h”

TPMC550_HANDLE  hdl;
TPMC550_STATUS  result;
int             WrittenSize;
short           ForOneCycle[4];
short           ForHundredCycles[100][4];

/* Fill new data into the data buffers */
ForHundredCycles[0][0] = 1;      /* first cycle, first channel */
ForHundredCycles[0][1] = 2;      /* first cycle, second channel */
...
ForHundredCycles[1][0] = 11;     /* second cycle, first channel */
...
ForHundredCycles[99][3] = 1234;  /* 100th cycle, last channel */

/* Write new data for 100 cycles and 4 active channels (100 * 4 values) */

result = tpmc550SeqWrite(
              hdl,
              sizeof(ForHundredCycles),
              (short*)ForHundredCycles,
              &WrittenSize
              );

if (result != TPMC550_OK)
{
     /* handle error */
     if (result == TPMC550_ERR_BUF_FULL)
     {
          /* send remaining data later */
     }
}

/* Write new data for 1 cycle and 4 active channels (4 values) */

result = tpmc550SeqWrite(
              hdl,
              sizeof(ForOneCycle),
              ForOneCycle,
              &WrittenSize
              );
```

RETURNS

On success, TPMC550_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	**Description**
TPMC550_ERR_INVALID_HANDLE	The specified TPMC550_HANDLE is invalid.
TPMC550_ERR_NOT_READY	The sequencer is not running
TPMC550_ERR_BUF_TOO_SMALL	The buffer does not contain enough data for all active channels.
TPMC550_ERR_NOMEM	The passed data buffer does not fit into the configured sequencer buffer. This error is only relevant in loop mode (TPMC550_LOOP)
TPMC550_ERR_BUF_FULL	The sequencer buffer is full. Not all data was written to the buffer. Use the contents of WrittenSize to adjust the data pointer to write the remaining data tuples.

3.3.5 tpmc550SeqFlush

NAME

tpmc550SeqFlush – flush the sequencer ring buffer

SYNOPSIS

```
TPMC550_STATUS tpmc550SeqFlush
(
        TPMC550_HANDLE        hdl
);
```

DESCRIPTION

This function flushes the ring buffer of the sequencer facility. The analog output of active channels will hold the last converted data until new data is written with the tpmc550SeqWrite function.

PARAMETERS

hdl

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

EXAMPLE

```
#include "tpmc550api.h"

TPMC550_HANDLE  hdl;
TPMC550_STATUS  result;

/* flush the sequencer ring buffer */
result = tpmc550SeqFlush(hdl);

if (result != TPMC550_OK)
{
    /* handle error */
}
```

RETURNS

On success, TPMC550_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC550_ERR_INVALID_HANDLE	The specified TPMC550_HANDLE is invalid.

3.3.6 tpmc550SeqStatus

NAME

tpmc550SeqStatus – get sequencer status and statistic information

SYNOPSIS

```
TPMC550_STATUS tpmc550SeqStatus
(
        TPMC550_HANDLE        hdl,
        int                   *OperatingState,
        int                   *status,
        int                   *CycleCount,
        int                   *UnderflowCount,
        int                   *EmptyCount
);
```

DESCRIPTION

This function reads sequencer status and statistic information from the specified device.

PARAMETERS

hdl

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

OperatingState

This argument is a pointer to an int variable where the current operating state of the sequencer is returned. Possible operating states are:

Value	**Description**
TPMC550_STOPPED	The sequencer is stopped.
TPMC550_READY	The sequencer facility is configured and ready to start.
TPMC550_RUNNING	The sequencer is running.

status

This argument is a pointer to an int variable where current error/status of the sequencer is returned. After calling this function the error/status code will be set to TPMC550_SEQ_OK. Possible error/status codes are:

Value	Description
TPMC550_SEQ_OK	Sequencer is working fine. No errors detected.
TPMC550_SEQ_UNDERFLOW	The sequencer hardware has detected a data underflow condition. The driver was not able to provide new data within a sequencer timer cycle.
TPMC550_SEQ_NODATA	No data available in the ring buffer for output.

CycleCount

This argument is a pointer to an int variable where the total number of sequencer cycles since sequencer start is returned.

UnderflowCount

This argument is a pointer to an int variable where the total number of sequencer underflows since sequencer start is returned.

EmptyCount

This argument is a pointer to an int variable where the total number of empty buffer cycles since sequencer start is returned.

EXAMPLE

```
#include “tpmc550api.h”

TPMC550_HANDLE   hdl;
TPMC550_STATUS   result;
int              OperatingState;
int              status;
int              CycleCount;
int              UnderflowCount;
int              EmptyCount;

/* Read sequencer status and statistic information */

result = tpmc550SeqStatus(hdl, &OperatingState, &status, &CycleCount,
                          &UnderflowCount, &EmptyCount);

if (result != TPMC550_OK)
{
    /* handle error */
}
```

RETURNS

On success, TPMC550_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TPMC550_ERR_INVALID_HANDLE	The specified TPMC550_HANDLE is invalid.

4 Legacy I/O System Functions

This chapter describes functions which are relevant only for the legacy TPMC550 driver.

4.1 tpmc550PciInit

NAME

tpmc550PciInit() – Generic PCI device initialization

SYNOPSIS

void tpmc550PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC550 PCI spaces (base address register) and to enable the TPMC550 device for access.

The global variable *tpmc550Status* 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 tpmc550Status is equal to the number of mapped PCI spaces
0	No TPMC550 device found
< 0	Initialization failed. The value of (tpmc550Status & 0xFF) is equal to the number of mapped spaces until the error occurs. Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[]. Remedy: Add dummy entries as necessary (syslib.c).

EXAMPLE

```
extern void tpmc550PciInit();

tpmc550PciInit();
```

5 Debugging and Diagnostic

The TPMC550 device driver provides a function and debug statements to display versatile information of the driver installation and status on the debugging console.

If the VxBus driver is used, the TPMC550 show routine is included in the driver by default and can be called from the VxWorks shell. If this function is not needed or program space is rare the function can be removed from the code by un-defining the macro INCLUDE_TPMC550_SHOW in tpmc550drv.c

The tpmc550Show function (only if VxBus is used) displays detailed information about probed modules, assignment of devices respective device names to probed TPMC550 modules and device statistics.

If TPMC550 modules were probed but no devices were created it may helpful to enable debugging code inside the driver code by defining the macro TPMC550_DEBUG in tpmc501drv.c.

In contrast to VxBus TPMC550 devices, legacy TPMC550 devices must be created "manually". This will be done with the first call to the tpmc550Open API function.

```
-> tpmc550Show
Probed Modules:
    [0] TPMC550: Bus=4, Dev=1, DevId=0x9050, VenId=0x10b5, Init=OK, vxDev=0x5380

Associated Devices:
    [0] TPMC550:     /tpmc550/0

Device Statistics:
    /tpmc550/0:
        Open Count            = 0
        Sequencer Cycle Count = 0
        Channels Output Range and Correction-Data (Offset/Gain):
            #1 [-10V... +10V] -   -1/5
            #2 [-10V... +10V] -    0/3
            #3 [-10V... +10V] -   -2/2
            #4 [-10V... +10V] -   -3/8
            #5 [  0V... +10V] -   -9/9
            #6 [  0V... +10V] -   -2/13
            #7 [  0V... +10V] -   -5/7
            #8 [  0V... +10V] -   -3/9
```