The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV016-SW-42

VxWorks Device Driver

32 / 16 Channels of 16 bit D/A

Version 1.0.x

User Manual

Issue 1.0.0

November 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

TDRV016-SW-42

VxWorks Device Driver

32 / 16 Channels of 16 bit D/A

Supported Modules:
TPMC553
TPMC554

Issue	Description	Date
1.0.0	First Issue	November 12, 2010

Table of Contents

1 Introduction

1.1 Device Driver

The TDRV016-SW-42 VxWorks device driver software allows the operation of the supported PMCs conforming to the VxWorks I/O system specification.

The TDRV016-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 TDRV016-SW-42 device driver supports the following features:

- Configuration of DAC channel voltage ranges
- Configuration of Q-DAC operation modes (I/M/T- or F-Mode)
- Write analog output values in I- and M-Mode
- Use analog sequencer modes (T- or F-Mode)
- SMP support (VxBus driver only)

The TDRV016-SW-42 supports the modules listed below:

TPMC553	32 / 16 Channels of 16 bit D/A	(PMC)
TPMC554	32 / 16 Channels of 16 bit D/A with memory	(PMC)

In this document all supported modules and devices will be called TDRV016. Specials for certain devices will be advised.

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

TPMC553/554 User Manual
TPMC553/554 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV016-SW-42':

TDRV016-SW-42-1.0.0.pdf	PDF copy of this manual
TDRV016-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TDRV016-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TDRV000-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tdrv016':

tdrv016drv.c	TDRV016 device driver source
tdrv016def.h	TDRV016 driver include file
tdrv016.h	TDRV016 include file for driver and application
tdrv016api.c	TDRV016 API file
Makefile	Driver Makefile
40tdrv016.cdf	Component description file for VxWorks development tools
tdrv016.dc	Configuration stub file for direct BSP builds
tdrv016.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tdrv016exa.c	Example application

The archive TDRV016-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tdrv016':

tdrv016drv.c	TDRV016 device driver source
tdrv016def.h	TDRV016 driver include file
tdrv016.h	TDRV016 include file for driver and application
tdrv016pci.c	TDRV016 device driver source for x86 based systems
tdrv016api.c	TDRV016 API file
tdrv016exa.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 ▪ 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!)

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 TDRV016-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 TDRV016 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tdrv016.*

At this point the TDRV016 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/tdrv016

(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\tdrv016
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 TDRV016 driver with the VxWorks development tools (Workbench), the component configuration file *40tdrv016.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv016
C:> copy 40tdrv016.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 TDRV016 driver and API can be included in VxWorks projects by selecting the *"TEWS TDRV016 Driver" and "TEWS TDRV016 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 TDRV016 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\tdrv016
C:> copy tdrv016.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tdrv016.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 TDRV016-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 TDRV016-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 tdrv016 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 TDRV016 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 TDRV016 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tdrv016pci.c** contains the function *tdrv016PciInit().* This routine finds out all TDRV016 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 *tdrv016PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window):

```
tdrv016PciInit();
```

Be sure that the function is called prior to MMU initialization otherwise the TDRV016 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.

2.3.4 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	0	40

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 tdrv016Open()

Name

tdrv016Open() – opens a device.

Synopsis

```
TDRV016_DEV tdrv016Open
(
      char          *DeviceName
)
```

Description

Before I/O can be performed to a device, a device 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 TDRV016 device is named “/tdrv016/0”, the second device is named “/tdrv016/1” and so on.

Example

```
#include “tdrv016.h”

TDRV016_DEV   pDev;

/*
** open the specified device
*/
pDev = tdrv016Open(“/tdrv016/0”);
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 tdrv016Close()

Name

tdrv016Close() – closes a device.

Synopsis

```
int tdrv016Close
(
      TDRV016_DEV    pDev
)
```

Description

This function closes previously opened devices.

Parameters

pDev

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

Example

```
#include “tdrv016.h”

TDRV016_DEV    pDev;
int            result;

/*
** close the device
*/
result = tdrv016Close(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 tdrv016SetVoltageRange

Name

tdrv016SetVoltageRange – set voltage range

Synopsis

```
STATUS tdrv016SetVoltageRange
(
    TDRV016_DEV   pDev,
    int           DacChannel,
    int           VoltageRange,
    int           Polarity
)
```

Description

This function configures the voltage range of a specific D/A channel.

Parameters

pDev

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

DacChannel

This argument specifies the DAC channel number. Possible values are 1 up to the available number of channels for the specific module.

VoltageRange

This argument specifies the desired voltage range for the selected DAC channel. Possible values are:

Value	Description
TDRV016_VOLTRANGE_5V	Vmax = 5V
TDRV016_VOLTRANGE_10V	Vmax = 10V
TDRV016_VOLTRANGE_10P8V	Vmax = 10.8V

Polarity

This argument specifies the desired polarity for the selected DAC channel. Possible values are:

Value	Description
TDRV016_POLARITY_UNIPOL	0V .. +Vmax
TDRV016_POLARITY_BIPOL	-Vmax .. +Vmax

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;

/*
** Configure DAC channel 3 to -10V .. +10V
*/
result = tdrv016SetVoltageRange( pDev,
                                 3,
                                 TDRV016_VOLTRANGE_10V,
                                 TDRV016_POLARITY_BIPOL);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

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()*.

Error code	Description
EINVAL	Invalid channel or parameter specified.
EIO	Error during hardware configuration.
EBADF	The device handle is invalid

3.2.2 tdrv016GetVoltageRange

Name

tdrv016GetVoltageRange – get current voltage range

Synopsis

```
STATUS tdrv016GetVoltageRange
(
        TDRV016_DEV     pDev,
        int             DacChannel,
        int             *pVoltageRange,
        int             *pPolarity
)
```

Description

This function reads the currently configured voltage range of a specific D/A channel.

Parameters

pDev

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

DacChannel

This argument specifies the DAC channel number. Possible values are 1 up to the available number of channels for the specific module.

pVoltageRange

This argument is a pointer to an *int* value where the configured voltage range for the selected DAC channel is returned. Possible values are:

Value	Description
TDRV016_VOLTRANGE_5V	Vmax = 5V
TDRV016_VOLTRANGE_10V	Vmax = 10V
TDRV016_VOLTRANGE_10P8V	Vmax = 10.8V

pPolarity

This argument is a pointer to an *int* value where the configured polarity for the selected DAC channel is returned. Possible values are:

Value	Description
TDRV016_POLARITY_UNIPOL	0V .. +Vmax
TDRV016_POLARITY_BIPOL	-Vmax .. +Vmax

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;
int                VoltageRange, Polarity

/*
** Read current voltage range configuration of DAC channel 3
*/
result = tdrv016GetVoltageRange( pDev,
                                 3,
                                 &VoltageRange,
                                 &Polarity);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

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()*.

Error code	Description
EINVAL	Invalid channel specified.
EBADF	The device handle is invalid

3.2.3 tdrv016GetModuleInfo

Name

tdrv016GetModuleInfo – get module information

Synopsis

```
STATUS tdrv016GetModuleInfo
(
    TDRV016_DEV   pDev,
    int           *pModuleType,
    int           *pModuleVariant,
    int           *pPciBusNo,
    int           *pPciDevNo
)
```

Description

This function reads the currently configured voltage range of a specific D/A channel.

Parameters

pDev

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

pModuleType

This argument is a pointer to an *int* value where the Module Type is returned. Possible values are:

Value	Description
TDRV016_MODTYPE_TPMC553	TPMC553
TDRV016_MODTYPE_TPMC554	TPMC554

pModuleVariant

This argument is a pointer to an *int* value where the Module Variant is returned. Possible values are:

Value	Description
10	-10 (32 D/A channels)
11	-11 (16 D/A channels)

pPciBusNo

This argument is a pointer to an *int* value where the PCI Bus number of the module is returned.

pPciDevNo

This argument is a pointer to an *int* value where the PCI Device number of the module is returned.

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;
int                ModuleType;
int                ModuleVariant;
int                PciBusNo;
int                PciDevNo;

/*
** Read module information
*/
result = tdrv016GetModuleInfo(   pDev,
                                 &ModuleType,
                                 &ModuleVariant,
                                 &PciBusNo,
                                 &PciDevNo);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
     printf(“Module Type   : %d\n”, ModuleType);
     printf(“Module Variant: %d\n”, ModuleVariant);
     printf(“Localization  : Bus %d / Device %d\n”, PciBusNo, PciDevNo);
}
```

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()*.

Error code	Description
EBADF	The device handle is invalid

3.2.4 tdrv016QDacConfig

Name

tdrv016QDacConfig – configure Q-DAC mode

Synopsis

```
STATUS tdrv016QDacConfig
(
        TDRV016_DEV     pDev,
        int             QDacNumber,
        int             QDacMode,
        int             GlobalLoadMode
)
```

Description

This function configures the operation mode of a specific Q-DAC, which serves 4 single D/A channels.

Parameters

pDev

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

QDacNumber

This argument specifies the Q-DAC number. Possible values are 1 up to the available number of Q-DACs for the specific module. Q-DAC 1 serves D/A channels 1 to 4, Q-DAC 2 serves D/A channels 5 to 8 and so on.

QDacMode

This argument specifies the desired operation mode for the selected Q-DAC. Possible values are:

Value	Description
TDRV016_QDACMODE_INSTANT	Instant Mode. DAC values are written immediately.
TDRV016_QDACMODE_MANUAL	Manual mode. DAC values are written after manual load operation.
TDRV016_QDACMODE_TIMER	Timer mode. DAC values are written in sequencer mode.

GlobalLoadMode

This argument specifies if the Q-DAC should synchronize to other Q-DACs. If TRUE, all selected Q-DACs are updated simultaneously. This parameter is only relevant for Manual mode.

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;

/*
** Configure Q-DAC 1 (D/A channel 1 to 4)
** - use Manual Mode without global synchronization
*/
result = tdrv016QDacConfig(  pDev,
                             1,
                             TDRV016_QDACMODE_MANUAL,
                             FALSE);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

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()*.

Error code	Description
EINVAL	Invalid channel or parameter specified.
EIO	Error during hardware configuration.
EBADF	The device handle is invalid

3.2.5 tdrv016DacWrite

Name

tdrv016DacWrite – Write one DAC value to a specific DAC channel

Synopsis

```
STATUS tdrv016DacWrite
(
        TDRV016_DEV     pDev,
        int             DacChannel,
        int             DacValue,
        int             Flags
)
```

Description

This function writes one DAC value to a specific DAC channel. This function is supported for channels configured to Instant or Manual Mode.

Parameters

pDev

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

DacChannel

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.

DacValue

This argument specifies the new DAC value for the specified channel.

Flags

This argument specifies additional options for this DAC update. Possible OR'ed flags are:

Value	Description
TDRV016_CORR	Use data correction for this conversion.
TDRV016_LOAD	Perform Load Operation for corresponding Q-DAC (only if Q-DAC is in Manual Mode).

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;

/*
** Write new DAC value to channel 1, use data correction.
*/
result = tdrv016DacWrite(    pDev,
                             1,
                             0x1000,
                             TDRV016_CORR);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

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()*.

Error code	Description
EINVAL	Invalid DAC channel specified.
ENOTSUP	The specified channel is not in I- or M-Mode.
EBADF	The device handle is invalid

3.2.6 tdrv016DacWriteMulti

Name

tdrv016DacWriteMulti – Write DAC values to multiple DAC channels

Synopsis

```
STATUS tdrv016DacWriteMulti
(
        TDRV016_DEV     pDev,
        UINT32          DacChannelMask,
        UINT32          CorrectionMask,
        int             PerformLoad,
        unsigned short  DacData[32]
)
```

Description

This function writes different DAC value to specified DAC channels. This function is supported for channels configured to Instant or Manual Mode.

Parameters

pDev

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

DacChannelMask

This argument specifies 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.

CorrectionMask

This argument specifies if data correction shall be used for specific DAC channels. A set (1) bit enables data correction for the corresponding channel. Bit 0 corresponds to the first DAC channel, bit 1 corresponds to the second DAC channel and so on.

PerformLoad

This argument specifies if the corresponding Q-DACs shall be updated. If TRUE, all affected Q-DACs are updated using the Load Operation. If this parameter is FALSE, all Q-DACs configured to Manual Mode will not be updated.

DacData

This argument specifies the new DAC data. Array index 0 corresponds to the first DAC channel, array index 1 corresponds to the second DAC channel and so on. Only channels marked for update using parameter *DacChannelMask* will be modified.

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;
unsigned short     DacData[32];

/*
** Write new DAC values to channel 1, 2 and 32.
** Use data correction only for channel 1.
** Update all channels which are in M-Mode.
*/
DacData[0]    = 0x1000;
DacData[1]    = 0x2000;
DacData[31]   = 0x0000;

result = tdrv016DacWriteMulti(   pDev,
                              ((1 << 31) | (1 << 1) | (1 << 0)),
                              (1 << 0),
                              TRUE,
                              DacData);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

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()*.

Error code	Description
EINVAL	Invalid Q-DAC specified.
ENOTSUP	At least one of the specified channels is not in I- or M-Mode.
EBADF	The device handle is invalid

3.2.7 tdrv016QDacLoad

Name

tdrv016QDacLoad – Perform Load Operation for specified Q-DACs

Synopsis

```
STATUS tdrv016QDacLoad
(
    TDRV016_DEV   pDev,
    UINT32        QDacMask
)
```

Description

This function performs the Load Operation for specified Q-DACs, to achieve simultaneous update of multiple DAC channels. This function is supported for Q-DACs configured to Manual Mode.

Parameters

pDev

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

QDacMask

This argument specifies the Q-DACs which shall be loaded. A set (1) bit specifies that the corresponding Q-DAC shall be loaded. Bit 0 corresponds to the first Q-DAC (DAC channels 1 to 4), bit 1 corresponds to the second Q-DAC (DAC channels 5 to 8) and so on.

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;

/*
** Load Q-DACs 1 and 8 simultaneously.
*/
result = tdrv016QDacLoad(    pDev,
                             ((1 << 7) | (1 << 0)));
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

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()*.

Error code	Description
EINVAL	Invalid Q-DAC specified.
EACCES	At least one of the specified Q-DACs is not in M-Mode.
EBADF	The device handle is invalid

3.2.8 tdrv016SequencerConfig

Name

tdrv016SequencerConfig – configure sequencer cycle time

Synopsis

```
STATUS tdrv016SequencerConfig
(
    TDRV016_DEV    pDev,
    int            QDacNumber,
    UINT32         CycleTime
)
```

Description

This function configures the sequencer cycle time of a specific Q-DAC, which serves 4 single D/A channels. The configured sequencer cycle time is used in both Timer and FIFO mode.

Parameters

pDev

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

QDacNumber

This argument specifies the Q-DAC number. Possible values are 1 up to the available number of Q-DACs for the specific module. Q-DAC 1 serves D/A channels 1 to 4, Q-DAC 2 serves D/A channels 5 to 8 and so on.

CycleTime

This argument specifies the sequencer cycle time. The sequencer timer is configurable in steps of 10µs. Possible values are 0 to the maximum value specified in the corresponding module hardware user manual.

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;

/*
** Configure Sequencer Timer of Q-DAC 1 (D/A channel 1 to 4)
** Use 1ms cycle time.
*/
result = tdrv016SequencerConfig( pDev,
                                 1,
                                 99);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

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()*.

Error code	Description
EINVAL	Invalid Q-DAC specified.
EBADF	The device handle is invalid

3.2.9 tdrv016SequencerStart

Name

tdrv016SequencerStart – start sequencer timer

Synopsis

```
STATUS tdrv016SequencerStart
(
    TDRV016_DEV   pDev,
    UINT32        SequencerMask
)
```

Description

This function starts the sequencer timer of specified Q-DACs. This function starts the sequencer operation of both Timer and FIFO mode.

Parameters

pDev

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

SequencerMask

This argument specifies the Q-DACs which shall be started in sequencer mode. A set (1) bit specifies that the corresponding Q-DAC shall be started. Bit 0 corresponds to the first Q-DAC (DAC channels 1 to 4), bit 1 corresponds to the second Q-DAC (DAC channels 5 to 8) and so on.

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;

/*
** Start Sequencer Timer of Q-DAC 1 and 2
*/
result = tdrv016SequencerStart(  pDev, (1 << 1) | (1 << 0));

if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

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()*.

Error code	Description
EINVAL	At least one specified Q-DAC is not in Timer or FIFO mode.
EBADF	The device handle is invalid

3.2.10 tdrv016SequencerStop

Name

tdrv016SequencerStop – stop sequencer timer

Synopsis

```
STATUS tdrv016SequencerStop
(
    TDRV016_DEV   pDev,
    UINT32        SequencerMask
)
```

Description

This function stops the sequencer timer of specified Q-DACs. This function stops the sequencer operation of both Timer and FIFO mode.

Parameters

pDev

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

SequencerMask

This argument specifies the Q-DACs which shall be stopped. A set (1) bit specifies that the corresponding Q-DAC shall be stopped. Bit 0 corresponds to the first Q-DAC (DAC channels 1 to 4), bit 1 corresponds to the second Q-DAC (DAC channels 5 to 8) and so on.

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;

/*
** Stop Sequencer Timer of Q-DAC 2
*/
result = tdrv016SequencerStop(   pDev, (1 << 1));

if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

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()*.

Error code	Description
EBADF	The device handle is invalid

3.2.11 tdrv016SequencerWrite

Name

tdrv016SequencerWrite – Write DAC data in Timer mode

Synopsis

```
STATUS tdrv016SequencerWrite
(
    TDRV016_DEV   pDev,
    int           QDacNumber,
    UINT32        UseCorrection,
    int           DacValue[4],
    int           timeout
)
```

Description

This function writes new DAC data to a specific Q-DAC, which serves 4 single D/A channels. All four channels of the Q-DAC are affected. This function is only supported in Timer Mode. This function might block until the next sequencer interrupt allows transferrin the DAC data, or the timeout expires.

Parameters

pDev

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

QDacNumber

This argument specifies the Q-DAC number. Possible values are 1 up to the available number of Q-DACs for the specific module. Q-DAC 1 serves D/A channels 1 to 4, Q-DAC 2 serves D/A channels 5 to 8 and so on.

UseCorrection

This argument specifies if data correction shall be used for specific DAC channels. A set (1) bit enables data correction for the corresponding channel. Bit 0 corresponds to the first DAC channel of the Q-DAC, bit 1 corresponds to the second DAC channel of the Q-DAC and so on.

DacValue

This argument specifies the new DAC data. Array index 0 corresponds to the first DAC channel of the Q-DAC, array index 1 corresponds to the second DAC channel of the Q-DAC and so on.

timeout

This parameter specifies the time the function will block until the data is transferred into the DAC channels, which is done using interrupts. The interrupts are raised based upon the configured sequencer cycle time, so this timeout value must be chosen according to the configured cycle time. This timeout value is specified in milliseconds. The resulting time depends on the system tick granularity. To wait indefinitely, specify -1.

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;
int                DacData[4];

/*
** Write new data to Q-DAC 2 without data correction.
** Use 500ms for timeout.
*/
result = tdrv016SequencerWrite(  pDev,
                                 2,
                                 0,
                                 DacData,
                                 500);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

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()*.

Error code	Description
EINVAL	Invalid Q-DAC specified or Q-DAC not in Timer mode.
EBADF	The device handle is invalid

3.2.12 tdrv016FifoConfig

Name

tdrv016FifoConfig – configure FIFO mode (TPMC554 only)

Synopsis

```
STATUS tdrv016FifoConfig
(
    TDRV016_DEV   pDev,
    UINT32        DacChannelMask,
    UINT32        ContinuousModeMask,
    int           Size[32],
    int           Limit[32]
)
```

Description

This function configures the FIFO mode of specified DAC channels. All four channels of one affected Q-DAC are used in FIFO mode. All channels which were previously configured to FIFO mode and are not again configured with this function are configured to Instant mode without changing the DAC value.

Parameters

pDev

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

DacChannelMask

This argument specifies the DAC channels which shall be used in FIFO mode. All four DAC channels of one affected Q-DAC must be configured for FIFO mode. A set (1) bit specifies that the corresponding channel shall be configured. Bit 0 corresponds to the first DAC channel, bit 1 corresponds to the second DAC channel and so on.

ContinuousModeMask

This argument specifies if the corresponding DAC channel FIFO shall be used in continuous mode. A set (1) configures the corresponding channel to repeat its FIFO data. An unset (0) bit configures the channel to stop the data output if the FIFO runs empty. Bit 0 corresponds to the first DAC channel, bit 1 corresponds to the second DAC channel and so on.

Size

This argument specifies the size of the FIFO in number of values. Array index 0 corresponds to the first DAC channel, array index 1 corresponds to the second DAC channel and so on.

Limit

This argument specifies the FIFO trigger limit where an interrupt is raised. The limit is specified as 2^{Limit}. Array index 0 corresponds to the first DAC channel, array index 1 corresponds to the second DAC channel and so on.

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;
int                Size[32];
int                Limit[32];


/*
** Configure FIFO mode for channel 1 to 8 (Q-DAC 1 and 2)
** - Use DAC 1 and 4 in Continuous Mode
*/
Size[0]  = 100;
Limit[0] = 5;      /* Limit at 32 values */
...

result = tdrv016FifoConfig(  pDev,
                             0x000000ff,
                             (1 << 3) | (1 << 0),
                             Size,
                             Limit);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

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()*.

Error code	Description
ENOTSUP	FIFO mode is not supported by TPMC553.
EINVAL	Not all DAC channels of a Q-DAC specified, invalid Size or Limit.
EBADF	The device handle is invalid

3.2.13 tdrv016FifoWrite

Name

tdrv016FifoWrite – Write DAC data in FIFO mode (TPMC554 only)

Synopsis

```
STATUS tdrv016FifoWrite
(
    TDRV016_DEV     pDev,
    int             DacChannel,
    UINT32          Flags,
    int             NumValues,
    unsigned short  *pDacData,
    int             timeout
)
```

Description

This function writes new DAC data of a specific DAC channel into the FIFO. This function is only supported in FIFO Mode. The function blocks until all data is written into the FIFO, or the timeout expires.

Parameters

pDev

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

DacChannel

This argument specifies the DAC channel number. Possible values are 1 up to the available number of DACs for the specific module.

Flags

This argument specifies additional options for this DAC update. Possible value:

Value	Description
TDRV016_CORR	Use data correction for all values.

NumValues

This argument specifies the number of DAC data values which shall be written into the FIFO.

pDacData

This parameter points to the DAC data section where the specified number of 16bit values is stored.

timeout

This parameter specifies the time the function will block until the data is transferred into the FIFO, which might be done using interrupts. The interrupts are raised based upon the configured sequencer cycle time, so this timeout value must be chosen according to the configured cycle time. This timeout value is specified in milliseconds. The resulting time depends on the system tick granularity. To wait indefinitely, specify -1.

The timeout value specifies the time to wait for the next interrupt. Multiple interrupts might be required to transfer the complete amount of specified data into the FIFO.

Example

```
#include “tdrv016.h”

TDRV016_DEV        pDev;
STATUS             result;
unsigned short     DacData[100];

/*
** Write 100 DAC data values to FIFO 2 with data correction.
** Use 500ms for timeout.
*/
DacData[0] = 0x1234;
DacData[1] = 0x5678;
DacData[2] = 0x9ABC;
...

result = tdrv016FifoWrite(   pDev,
                             2,
                             TDRV016_CORR,
                             100,
                             DacData,
                             500);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* successful */
}
```

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()*.

Error code	Description
ENOTSUP	FIFO mode is not supported by TPMC553.
EINVAL	Invalid DAC channel specified.
EACCES	Channel not in FIFO mode.
EBUSY	This DAC channel is already busy transferring data into the FIFO.
EBADF	The device handle is invalid

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 TDRV016 driver. For the VxBus-enabled TDRV016 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.

4.1 tdrv016Drv()

NAME

tdrv016Drv() - installs the TDRV016 driver in the I/O system

SYNOPSIS

#include “tdrv016.h”

STATUS tdrv016Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus, installs the TDRV016 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 "tdrv016.h”

STATUS            result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tdrv016Drv();
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
ENOBUFS	Not enough system resources.

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.2 tdrv016DevCreate()

NAME

tdrv016DevCreate() – Add a TDRV016 device to the VxWorks system

SYNOPSIS

```
#include “tdrv016.h”

STATUS tdrv016DevCreate
(
        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 index number depends on the search priority of the modules. The modules will be searched in the following order:

- TPMC553-xx
- TPMC554-xx

If modules of the same type are installed the channel numbers will be assigned in the order the VxWorks *pciFindDevice()* function will find the devices.

Example: (A system with 1x TPMC553-xx and 2x TPMC554-xx) will assign the following device indices:

Module	Device Index
TPMC553-xx	0
TPMC554-xx (1st)	1
TPMC554-xx (2nd)	2

funcType

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

EXAMPLE

```
#include "tdrv016.h”

STATUS            result;

/*-------------------------------------------------------
  Create the device "/tdrv016/0" for the first device
  -------------------------------------------------------*/
result = tdrv016DevCreate(   "/tdrv016/0",
                             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
ENXIO	Device driver not started or device not found.

SEE ALSO

VxWorks Programmer’s Guide: I/O System

4.3 tdrv016PciInit()

NAME

tdrv016PciInit() – Generic PCI device initialization

SYNOPSIS

void tdrv016PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TDRV016 PCI spaces (base address register) and to enable the TDRV016 device for access.

The global variable *tdrv016Status* 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 tdrv016Status is equal to the number of mapped PCI spaces
0	No TDRV016 device found
< 0	Initialization failed. The value of (tdrv016Status & 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 tdrv016PciInit();

tdrv016PciInit();
```

4.4 tdrv016Init()

NAME

tdrv016Init() – initialize TDRV016 driver and devices

SYNOPSIS

#include “tdrv016.h”

STATUS tdrv016Init(void)

DESCRIPTION

This function is used by the TDRV016 example application to install the driver and to add all available devices to the VxWorks system.

See also 3.1.1 tdrv016Open()for the device naming convention for legacy devices.

After calling this function it is not necessary to call tdrv016Drv() and tdrv016DevCreate() explicitly.

EXAMPLE

```
#include "tdrv016.h”

STATUS    result;


result = tdrv016Init();

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 and 4.2 for a description of possible error codes.