The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC861-SW-42

VxWorks Device Driver

4 Channel Isolated Serial Interface (RS422/RS485)

Version 4.0.x

User Manual

Issue 4.0.0

March 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

TPMC861-SW-42

VxWorks Device Driver

4 Chan. Isolated Serial Interface (RS422/RS485)

Supported Modules:
TPMC861

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2001-2012 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	July 7, 2001
1.1	Overrun Error added	February 18, 2001
1.2	Mark/Space Parity added	June 26, 2003
2.0.0	New File List, tpmc861Drv() and tpmc861DevCreate() have changed, Description of tpmc861PciInit() added, Advanced description of the ioctl() function codes	August 14, 2006
2.0.1	New Address TEWS LLC	October 10, 2006
2.0.2	Description of BSP dependencies	February 12, 2007
2.1.0	Description of default configuration, Description how to include device driver into VxWorks projects modified, Address TEWS LLC removed	July 6, 2009
3.0.0	New version of driver, Legacy and VxBus-Support	August 30, 2010
3.1.0	File list modified, Document layout revision.	September 6, 2011
4.0.0	New ioctl function FIO_EXAR16XXX_CHANNEL_INFO, new chapter Configuration of FIFO-Trigger-Levels	March 1, 2012

Table of Contents

1 Introduction

1.1 Device Driver

The TPMC861-SW-42 VxWorks device driver software allows the operation of the supported modules conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with *open*(), *close(), read(), write(),* and *ioctl()* functions and a buffered I/O interface (*fopen()*, *fclose()*, *fprintf()*, *fscanf()*, ...).

Special I/O operation that do not fit to the standard I/O calls will be performed by calling the ioctl() function with a specific function code and an optional function dependent argument.

The TPMC861-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.

The TPMC861 driver includes the following functions supported by the *VxWorks tty driver support library for pre-VxBus systems or the sio driver library for VxBus compatible systems.*

- ring buffering of input and output
- raw mode
- optional line mode with backspace and line-delete functions
- optional processing of X-on/X-off
- optional RETURN/LINEFEED conversion
- optional echoing of input characters
- optional stripping of the parity bit from 8 bit input
- optional special characters for shell abort and system restart

Additionally the following optional functions:

- select FIFO triggering point
- use 5...8 bit data words
- use 1, 1.5 or 2 stop bits
- optional even or odd parity
- changing baudrates
- reading board information and PCI location

The TPMC861-SW-42 supports the modules listed below:

TPMC861-10	4 Channel Isolated Serial Interface (RS422/RS485)	(PMC)

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

TPMC861 User Manual
TPMC861 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC861-SW-42':

TPMC861-SW-42-4.0.0.pdf	PDF copy of this manual
TPMC861-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TPMC861-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TPMC861-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tpmc861':

tpmc861drv.c	TPMC861 device driver source (TPMC861 specific)
tpmc861def.h	TPMC861 driver include file
tpmc861defaults.h	TPMC861 device default configuration
tpmc861.h	TPMC861 include file for driver and application
exar16xxxDrv.c	device driver source (controller specific)
exar16xxxDef.h	driver include file (controller specific)
exar16xxx.h	include file for driver and application (controller specific)
Makefile	Driver Makefile
40tpmc861.cdf	Component description file for VxWorks development tools
tpmc861.dc	Configuration stub file for direct BSP builds
tpmc861.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tpmc861exa.c	Example application

The archive TPMC861-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tpmc861':

tpmc861drv.c	TPMC861 device driver source
tpmc861def.h	TPMC861 driver include file
tpmc861defaults.h	TPMC861 device default configuration
tpmc861.h	TPMC861 include file for driver and application
tpmc861pci.c	TPMC861 device driver source for x86 based systems
exar16xxxDrv.c	device driver source (controller specific)
exar16xxxDef.h	driver include file (controller specific)
exar16xxx.h	include file for driver and application (controller specific)
tpmc861exa.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) ▪ 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 TPMC861-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 TPMC861 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tpmc861.*

At this point the TPMC861 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 processer (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/tpmc861

(3) Invoke the build command for the required processor and build tool with optional VXBUILD argument
make CPU=cpuName TOOL=tool [VXBUILD=xxx]

For Windows hosts this may look like this:

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc861
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 TOOL=gnu CPU=CORE VXBUILD=LP64
```

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

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

To integrate the TPMC861 driver with the VxWorks development tools (Workbench), the component configuration file *40tpmc861.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc861
C:> copy 40tpmc861.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 TPMC861 driver can be included in VxWorks projects by selecting the *"TEWS TPMC861 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 TPMC861 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\tpmc861
C:> copy tpmc861.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tpmc861.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
```

2.2.2 Modification of the 'Number of serial ports'

The new number of serial ports must be specified in the configuration tool. By default only the number of onboard serial ports is specified and TPMC861 will not be set up. To support the TPMC861 ports the value of *'/hardware/peripherals/serial/SIO/number of serial ports'* (*NUM_TTY*) must be set (at least) to the total number of installed serial ports. For example, if there are two onboard ports and one TPMC861 with 4 ports should be supported, the value must be set to a value of 6 at least.

2.3 Legacy Driver Installation

2.3.1 Include Device Driver in VxWorks Projects

For including the TPMC861-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 TPMC861-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 C files in the tpmc861 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 TPMC861 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 TPMC861 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tpmc861pci.c** contains the function *tpmc861PciInit()*. This routine finds out all TPMC861 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 *tpmc861PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window).

Be sure that the function is called prior to MMU initialization otherwise the TPMC861 PCI spaces remains unmapped and an access fault occurs during driver initialization.

Please insert the following call at a suitable place in **sysLib.c**:

```
tpmc861PciInit();
```

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)	---

(*1) For legacy drivers only one semaphore is used,
for VxBus-driver one semaphore is used per installed board

The specified requirements are specific to the driver. The VxWorks terminal manager will require extra resources for each device.

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.

2.5 Default Configuration

The driver will create the port with the default configuration specified in tpmc861defaults.h. All channels will be set up with the same default configuration. If a different configuration is necessary, the configuration can be changed by modifying the file. The assigned values must be compatible to the values allowed for the corresponding ioctl() function.

The following defines are made for configuration:

TPMC861_DEFAULT_BAUD

Specifies the default baudrate.
(Default value: 9600)

TPMC861_DEFAULT_OPTIONS

Specifies the default terminal settings.
(Default value: OPT_RAW)

TPMC861_DEFAULT_RXFIFOTRIG

Specifies the default receive FIFO trigger level.
(See also 2.6 Configuration of FIFO-Trigger-Levels)
(Default value: 20)

TPMC861_DEFAULT_TXFIFOTRIG

Specifies the default transmit FIFO trigger level.
(See also 2.6 Configuration of FIFO-Trigger-Levels)
(Default value: 90)

TPMC861_DEFAULT_DATABITS

Specifies the default length of the data word.
(Default value: EXAR16XXX_DB_8)

TPMC861_DEFAULT_STOPBITS

Specifies the default length of the stop bit.
(Default value: EXAR16XXX_SB_10)

TPMC861_DEFAULT_PARITY

Specifies the default parity mode.
(Default value: EXAR16XXX_NOP)

If an illegal value is specified in the file the default value (in the delivered file) will be used.

2.6 Configuration of FIFO-Trigger-Levels

The FIFO trigger-levels may influence the behavior of the target system. A modification of the FIFO-trigger-levels also means changing the duration of a single interrupt and the number of interrupts that will be generated.

Increasing the receive FIFO-trigger-level will lower the number of generated interrupts, but it will also increase the execution time of a single interrupt function and it may increase the risk of loosing data by FIFO overrun.

Increasing the transmit FIFO-trigger-level will increase the number of generated interrupts, but it will also lower the execution time of a singe interrupt function and decrease the chance of gaps in the transmission stream..

Known issue with interrupt execution time

In newer systems (VxWorks 6.x) a long interrupt execution time may lead into work queue overflow, which may result in system crash or error state. If such a situation occurs while a data transfer is working there are two ways to solve the problem: first the FIFO-trigger-levels can be adapted to decrease the interrupt execution time, and secondly the Work Queue Size can be increased (value of WIND_JOBS_MAX). Please refer to the VxWorks documentation for description of project configuration.

3 VxBus Driver Support

The TPMC861 will be fully integrated to the VxWorks system and the devices will be automatically created when booting VxWorks.

3.1 Assignment of Port Names

The port names are assigned automatically when the ports are created. The assigned port name will be '/tyCo/<*n*>" where <n> specifies the port number. Generally the first two port numbers ('/tyCo/0', '/tyCo/1') are assigned to system ports and the additional ports on the TPMC861 supported boards will start with port number 2. For example a system with one TPMC861 (4 channels) will assign the following device names:

/tyCo/0	1st system port
/tyCo/1	2nd system port
/tyCo/2	1st channel of TPMC861
/tyCo/3	2nd channel of TPMC861
/tyCo/4	3rd channel of TPMC861
/tyCo/5	4th channel of TPMC861

If there is more than one supported TPMC861 board installed, the assignment of the channel numbers to the boards depends on the search order of the system, but all the channels of one board will follow up in a row.

After booting the available devices can be checked with *devs()*. This function will return a list of all created devices. If fewer devices have been created, please first check the defined maximum number of serial devices. (See *2.2.2 Modification of the 'Number of serial ports'*)

3.2 VxBus Error Codes

There will be just system generated return codes for the 'Basic I/O Functions'. The TPMC861 specific 'Error Codes' described with the functions are not valid for VxBus devices.

3.3 Compatibility to pre-VxBus Applications

The VxBus driver is compatible to the legacy version of this driver. The only point which must be guaranteed is, that the driver initialization is made via tpmc861Init() and not with tpmc861Drv() and tpmc861DevCreate().

Legacy compatible initialization function

```
STATUS tpmc861Init
(
    int         *firstChanNo,
    int         *lastChanNo
)
```

This routine just returns the number of the first (*firstChanNo*) and last (*lastChanNo*) port number assigned to the TPMC861 driver. The devices will be named '/tyCo/<*firstChanNo*>' up to '/tyCo/<*lastChanNo*>'

This function has been created for compatibility to the legacy driver. It allows usage of the same example for bath, legacy and VxBus systems. It is not necessary to call this function in custom application.

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 TPMC861 driver. For the VxBus-enabled TPMC861 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.

4.1 tpmc861Drv

NAME

tpmc861Drv - installs the TPMC861 driver in the I/O system

This function is not implemented for systems supporting VxBus.

SYNOPSIS

```
#include “tpmc861.h”

STATUS tpmc861drv
(
        void
)
```

DESCRIPTION

This function searches for devices on the PCI bus, installs the TPMC861 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 "tpmc861.h"

STATUS            result;

/*-------------------
  Initialize Driver
  -------------------*/
result = tpmc861Drv();
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 TPMC861 found

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.2 tpmc861DevCreate

NAME

tpmc861DevCreate – Add a TPMC861 device to the VxWorks system

SYNOPSIS

#include “tpmc861.h”

```
STATUS tpmc861DevCreate
(
    char        *name,
    int         glbChanNo,
    int         rdBufSize,
    int         wrtBufSize,
    void        *devConf
)
```

DESCRIPTION

This routine creates a device on a specified serial channel that will be serviced by the TPMC861 driver.

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

This function is not implemented for systems supporting VxBus.

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.

If more than one modules are installed the channel numbers will be assigned in the order the VxWorks *pciFindDevice()* function will find the devices.

rdBufSize

This value specifies the size of the receive software FIFO.

wrtBufSize

This value specifies the size of the transmit software FIFO.

devConf

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

EXAMPLE

```
#include "tpmc861.h”

STATUS            result;

/*------------------------------------------------------
  Create the device "/tyCo/2" for the first device
     1KB transmit and receive FIFO
  ------------------------------------------------------*/
result = tpmc861DevCreate(   "/tyCo/2",
                             0,
                             1024,
                             1024,
                             NULL);
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_iosLib_DEVICE_NOT_FOUND	Driver has not been started, or the specified channel has not been detected, or channel structure has not been allocated

SEE ALSO

VxWorks Programmer’s Guide: I/O System

4.3 tpmc861PciInit

NAME

tpmc861PciInit – Generic PCI device initialization

SYNOPSIS

void tpmc861PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC861 PCI spaces (base address register) and to enable the TPMC861 device for access.

The global variable *tpmc861Status* 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 *tpmc861Status* is equal to the number of mapped PCI spaces
0	No TPMC861 device found
< 0	Initialization failed. The value of (*tpmc861Status* & 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 tpmc861PciInit();

tpmc861PciInit();
```

4.4 tpmc861Init

NAME

tpmc861Init – initialize TPMC861 driver and devices and return the assigned channel numbers

SYNOPSIS

```
#include “tpmc861.h”

STATUS tpmc861Init
(
    int         *firstDevIdx,
    int         *lastDevIdx
)
```

DESCRIPTION

This function is used by the TPMC861 example application to install the driver, to add all available devices to the VxWorks system and to determine the assigned port names.

All software FIFOs (Receive / Transmit) will be configured with a size of 1KB.

The function calls tpmc861Drv() and tpmc861DevCreate(). The devices will be named with ‘/tyCo/<*n*>’ where <*n*> specifies the channel. Because the default serial devices are named in the same kind, the driver searches for the first free number and will name the TPMC861 starting with this number in a row.

For example already two local serial devices are created and one TPMC861 is installed, the names ‘/tyCo/0’ and ‘/tyCo/1’ are assigned to the local channels, ‘/tyCo/2’ up to ‘/tyCo/5’ will be assigned to the 4 TPMC861 channels. In this example the function will set a 2 for the first and a 5 for the last assigned device.

After calling this function, it is not necessary to call tpmc861Drv() or tpmc861DevCreate() explicitly.

PARAMETER

firstDevIdx

Pointer where the lowest assigned device number for TPMC861 devices will be returned.

lastDevIdx

Pointer where the highest assigned device number for TPMC861 devices will be returned.

EXAMPLE

```
#include "tpmc861.h”

STATUS    result;
int       firstNo;
int       lastNo;
char      devName[20];
int       chanNo;


result = tpmc861Init(&firstNo, &lastNo);

if (result == ERROR)
{
     /* Error handling */
}
else
{
     for (chanNo = firstNo; chanNo <= lastNo; chanNo++)
     {
          sprintf(devName, “\tyCo\%d”, chanNo);
          fd = open(devName, …);
          …
     }
}
```

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.

5 Basic I/O Functions

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 TPMC861 device, a file descriptor must be opened by invoking the basic I/O function *open().*

PARAMETER

name

Specifies the device which shall be opened.
For the legacy driver version, the name specified for the device (e.g. by *tpmc861DevCreate()*) must be used.
For the VxBus driver version the system assigned device name (‘/tyCo/<n>’) must be used.

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*----------------------------------------
  Open the device named "/tyCo/2" for I/O
  ----------------------------------------*/
fd = open("/tyCo/2", 0, 0);
if (fd == ERROR)
{
     /* error handling */
}
```

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)
{
     /* error handling */
}
```

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 read

NAME

read – read data from a specified device.

SYNOPSIS

```
int read
(
        int             fd,
        char            *buffer,
        size_t          maxbytes
)
```

DESCRIPTION

This function can be used to read data from the device.

PARAMETER

fd

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

buffer

This argument points to a user supplied buffer. The returned data will be filled into this buffer.

maxbytes

This parameter specifies the maximum number of read bytes (buffer size).

EXAMPLE

```
#define   BUFSIZE   100

int           fd;
char          buffer[BUFSIZE];
int           retval;

/*-----------------------------
  Read data from TPMC861 device
  -----------------------------*/
retval = read(fd, buffer, BUFSIZE);
if (retval != ERROR)
{
     printf("%d bytes read\n", retval);
}
else
{
     /* handle the read error */
}
```

RETURNS

Number of bytes read 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 - read()

5.4 write

NAME

write – write data from a buffer to a specified device.

SYNOPSIS

```
int write
(
        int         fd,
        char        *buffer,
        size_t      nbytes
)
```

DESCRIPTION

This function can be used to write data to the device.

PARAMETER

fd

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

buffer

This argument points to a user supplied buffer. The data of the buffer will be written to the device.

nbytes

This parameter specifies the number of bytes to be written.

EXAMPLE

```
int           fd;
char          buffer[] = “Hello World”;
int           retval;

/*------------------------------
  Write data to a TPMC861 device
  ------------------------------*/
retval = write(fd, buffer, strlen(buffer));
if (retval != ERROR)
{
     printf(“%d bytes written\n”, retval);
}
else
{
     /* handle the write error */
}
```

RETURNS

Number of bytes written 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 - write()

5.5 ioctl

NAME

ioctl - performs an I/O control function.

SYNOPSIS

#include “tpmc861.h”

```
int ioctl
(
        int                             fd,
        int                             request,
        EXAR16XXX_IOCTL_ARG_T           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. The TPMC861 device driver uses the standard *tty driver support library tyLib*. For details of supported *ioctl* functions see *VxWorks Reference Manual*: tyLib and *VxWorks Programmer's Guide*: I/O System. Following additional functions are defined:

Function	Description
FIO_EXAR16XXX_DATABITS	Set length of data word
FIO_EXAR16XXX_STOPBITS	Set length of the stop bit
FIO_EXAR16XXX_PARITY	Set parity checking mode
FIO_EXAR16XXX_SETBREAK	Set Break signal
FIO_EXAR16XXX_CLEARBREAK	Release Break signal
FIO_EXAR16XXX_CHECKBREAK	Check if a Break signal has been detected
FIO_EXAR16XXX_CHECKERRORS	Get error state of the device
FIO_EXAR16XXX_RECONFIGURE	Reconfigure device with the default parameters
FIO_EXAR16XXX_FIFO	Configure use of FIFO and set trigger levels

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()*.

For TPMC861 legacy driver version: 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.

For TPMC861 VxBus driver version: The error code is always a standard error code set by the I/O system. There are no driver specific error codes.

SEE ALSO

ioLib, basic I/O routine - ioctl()

5.5.1 FIOBAUDRATE

This I/O control function configures the baudrate for the specified device. It is basically a standard function with a few points to pay attention to. The function specific control parameter arg passes the selected baudrate to the device driver.

The selected baud rate is always set to the nearest selectable value.

How to calculate baudrates, please refer to the TPMC861 User Manual.

Examples:

Required Baud Rate	Selected Baud Rate
9600	9600
100000	115200
115200	115200

Higher baud rates shall be used with enabled FIFO, this will avoid loosing data.

EXAMPLE

```
#include “tpmc861.h”

int       fd;
int       result;

/*-------------------------
  Set baud rate to 9600
  -------------------------*/
result = ioctl(fd, FIOBAUDRATE, 9600);
if (result == OK)
{
     /* Success */
}
else
{
     /* Function failed */
}
```

ERROR CODES

Error Code	Description
EINVAL	Baudrate out of range

5.5.2 FIO_EXAR16XXX_DATABITS

This I/O control function selects the number of data bits in one word for the specific device.

The function specific control parameter arg passes the selected value to the device driver. The following values are possible:

Value	Description
EXAR16XXX_DB_5	use 5 data bits
EXAR16XXX_DB_6	use 6 data bits
EXAR16XXX_DB_7	use 7 data bits
EXAR16XXX_DB_8	use 8 data bits

EXAMPLE

```
#include "tpmc861.h"

int      fd;
int      result;

/*-------------------------------------
  Set channel to a word length of 7 bit
  -------------------------------------*/
result = ioctl(fd, FIO_EXAR16XXX_DATABITS, EXAR16XXX_DB_7);
if (result == OK)
{
     /* Success */
}
else
{
     /* Function failed */
}
```

ERROR CODES

Error Code	Description
EINVAL	Invalid number of data bits specified

5.5.3 FIO_EXAR16XXX_STOPBITS

This I/O control function selects the number of stop bits used for the specific device.

The function specific control parameter arg passes the selected value to the device driver. The following values are possible:

Value	Description
EXAR16XXX_SB_10	use 1 stop bit
EXAR16XXX_SB_15	use 1.5 stop bits
EXAR16XXX_SB_20	use 2 stop bits

EXAMPLE

```
#include "tpmc861.h"

int       fd;
int       result;

/*----------------------------------------
  Set channel to a stop bit length of 1 bit
  ----------------------------------------*/
result = ioctl (fd, FIO_EXAR16XXX_STOPBITS, EXAR16XXX_SB_10);
if (result == OK)
{
     /* Success */
}
else
{
     /* Function failed */
}
```

ERROR CODES

Error Code	Description
EINVAL	Invalid number of stop bits specified

5.5.4 FIO_EXAR16XXX_PARITY

This I/O control function selects parity checking mode for the specific device.

The function specific control parameter arg passes the selected value to the device driver. The following values are possible:

Value	Description
EXAR16XXX_NOP	do not use parity
EXAR16XXX_EVP	use EVEN parity
EXAR16XXX_ODP	use ODD parity
EXAR16XXX_SPP	use SPACE parity
EXAR16XXX_MAP	use MARK parity

EXAMPLE

```
#include "tpmc861.h"

int       fd;
int       result;

/*------------------------------
  Configure channel no parity
  ------------------------------*/
result = ioctl(fd, FIO_EXAR16XXX_PARITY, EXAR16XXX_NOP);
if (result == OK)
{
     /* Success */
}
else
{
     /* Function failed */
}
```

ERROR CODES

Error Code	Description
EINVAL	Invalid parity mode specified

5.5.5 FIO_EXAR16XXX_SETBREAK

This I/O control function sets break state on transmit line. The function specific control parameter arg is unused and will be ignored.

EXAMPLE

```
#include “tpmc861.h”

int                fd;
int                retval;

/*-----------------------
  Set break on Tx line(s)
  -----------------------*/
retval = ioctl(fd, FIO_EXAR16XXX_SETBREAK, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

5.5.6 FIO_EXAR16XXX_CLEARBREAK

This I/O control function resets break state on transmit line. The function specific control parameter arg is unused and will be ignored.

EXAMPLE

```
#include “tpmc861.h”

int                fd;
int                retval;

/*-----------------------
  Clear break on Tx line(s)
  -----------------------*/
retval = ioctl(fd, FIO_EXAR16XXX_CLEARBREAK, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

5.5.7 FIO_EXAR16XXX_CHECKBREAK

This I/O control function returns if a break event on the receive line has been detected since the last call of the function. The function specific control parameter arg passes a pointer (int*) where the return value will be stored. A return value TRUE indicates that a break event has been detected, the value FALSE indicates that no break event has been detected.

EXAMPLE

```
#include “tpmc861.h”

int                fd;
int                retval;
int                breakDetect;

/*-----------------------
  Check break
  -----------------------*/
retval = ioctl(fd, FIO_EXAR16XXX_CHECKBREAK,
               (EXAR16XXX_IOCTL_ARG_T)&breakDetect);
if (retval != ERROR)
{
     /* function succeeded */
     if (breakDetect)
     {
          /* A break has been detected */
     }
}
else
{
     /* handle the error */
}
```

5.5.8 FIO_EXAR16XXX_CHECKERRORS

This I/O control function returns the error state of the device. The function specific control parameter arg points to a buffer (unsigned int) the status will be returned. The returned status is an OR'ed value of the following flags:

Value	Description
EXAR16XXX_FRAMING_ERR	This bit is set if a framing error has been detected since the last call.
EXAR16XXX_PARITY_ERR	This bit is set if a parity error has been detected since the last call.
EXAR16XXX_OVERRUN_ERR	This bit is set if an overrun error has been detected since the last call.

EXAMPLE

```
#include "tpmc861.h"

int                fd;
int                retval;
unsigned long      errStat;

/*------------------
  Get receive status
  ------------------*/
retval = ioctl(fd, FIO_EXAR16XXX_CHECKERRORS,
               ((EXAR16XXX_IOCTL_ARG_T))&errStat);
if (retval != ERROR)
{
     /* function succeeded */
     if (errStat & EXAR16XXX_FRAMING_ERR)
     {
          /* Framing error occurred */
     }
}
else
{
     /* handle the error */
}
```

5.5.9 FIO_EXAR16XXX_RECONFIGURE

This I/O control function resets the device to the default configuration. The function specific control parameter arg is not used for this function.

EXAMPLE

```
#include “tpmc861.h”

int                fd;
int                retval;

/*--------------------------
  Reconfigure serial channel
  --------------------------*/
retval = ioctl(fd, FIO_EXAR16XXX_RECONFIGURE, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

5.5.10 FIO_EXAR16XXX_FIFO

This I/O control function specifies if FIFOs shall be enabled and which trigger levels should be used for interrupt generation. The function specific control parameter arg passes a pointer to the FIFO setting structure (EXAR16XXX_FIFO_STRUCT).

```
typedef struct
{
        int             rxFifoTrigger;
        int             txFifoTrigger;
} EXAR16XXX_FIFO_STRUCT;
```

rxFifoTrigger

Specifies the receive FIFO trigger level. Allowed values are:

1…127	FIFOs enabled, value specifies receive FIFO trigger level
EXAR16XXX_F_NO	FIFOs disabled, only valid if transmit FIFO will also be disabled.

txFifoTrigger

Specifies the transmit FIFO trigger level. Allowed values are:

1…127	FIFOs enabled, value specifies transmit FIFO trigger level
EXAR16XXX_F_NO	FIFOs disabled, only valid if receive FIFO will also be disabled.

Changing the FIFO-fifo-trigger levels may influence the behavior of your target system, therefore please refer to chapter 2.6 Configuration of FIFO-Trigger-Levels.

EXAMPLE

```
#include “tpmc861.h”

int                     fd;
int                     result;
EXAR16XXX_FIFO_STRUCT   fifoSet;

/*---------------------------
  Enable FIFO with
     - receive trigger at 85
     - transmit trigger at 15
  ---------------------------*/
fifoSet.rxFifoTrigger = 85;
fifoSet.txFifoTrigger = 15
result = ioctl(fd, FIO_EXAR16XXX_FIFO, (EXAR16XXX_IOCTL_ARG_T)&fifoSet);
if (result == OK)
{
     /* Success */
}
else
{
     /* Function failed */
}
```

ERROR CODES

Error Code	Description
EINVAL	Invalid Trigger Level specified or the combination of trigger levels is not allowed.

5.5.11 FIO_EXAR16XXX_CHANNEL_INFO

This I/O control function returns information regarding the specified channel. The returned information contains information about the board where the channel is located. The function will also return information about the PCI-bus location where the controller of the channel can be found. This information may be helpful to find a special channel in the system and to assign a physical channel to a logical device.

The function specific control parameter arg passes a pointer to an information structure (EXAR16XXX_CHANNEL_INFO_STRUCT) where the information will be filled in.

```
typedef struct
{
    struct exar16xx_board_info_struct       board;
    struct exar16xx_controller_info_struct  controller;
} EXAR16XXX_CHANNEL_INFO_STRUCT;
```

board

This structure (struct exar16xx_board_info_struct) contains board information that belongs to a specified channel.

```
struct exar16xx_board_info_struct
{
    int             channelNo;
    unsigned int    boardId;
    unsigned int    boardVariant;
    int             boardIndex;
};
```

channelNo

This value returns the channel number of the board where the channel is located. The returned number will match the channel number assigned in the User Manual.

boardId

This value returns a unique ID, which identifies the used board type. This information may be of interest if other serial boards are used. The driver will always return TPMC861_MODULE_ID identifying the TPMC861.

boardVariant

This value returns the board variant. The returned number specified the *xx* in the board name TPMC861-*xx*.

boardIndex

This value returns the index of the specified board. If just one TPMC861 is used, this index will always be 0, but if more than a single TPMC861 is installed, the index value returned is the index for PCI-search (The index is depends on the search order of the BSP).

controller

This structure (struct exar16xx_controller_info_struct) contains information that belongs to the controller and the specified channel which describes the location of the controller and channel on PCI-bus.

```
struct exar16xx_controller_info_struct
{
    int             pciBusNo;
    int             pciDeviceNo;
    int             pciFunctionNo;
    int             controllerPort;
};
```

pciBusNo

This PCI bus number the channels controller is located at.

pciDeviceNo

This PCI device number the channels controller is located at.

pciFunctionNo

This PCI function number the channels controller is located at. The TPMC861 is not a multifunction device, therefore the function number is always 0.

controllerPort

This value specifies the channel index within the controller, as assigned in the documentation of the controller chip.

EXAMPLE

```
#include “tpmc861.h”

int                             fd;
int                             retval;
EXAR16XXX_CHANNEL_INFO_STRUCT   channelInfo;

…
```

```
…

/*-----------------------------
  Get Channel Board Information
  -----------------------------*/
result = ioctl(fd, FIO_EXAR16XXX_CHANNEL_INFO,
               (EXAR16XXX_IOCTL_ARG_T)&channelInfo);
if (result == OK)
{
     printf("Get Channel Board Information successfully executed\n");
     printf("Board: TPMC%d-%02d - Board Index: %d\n",
          channelInfo.board.boardId,
          channelInfo.board.boardVariant,
          channelInfo.board.channelNo);
     printf("    Channel number on board: %d\n",
          channelInfo.board.channelNo);

     printf("Controller: PCI-Location: [%d/%d/%d]\n",
          channelInfo.controller.pciBusNo,
          channelInfo.controller.pciDeviceNo,
          channelInfo.controller.pciFunctionNo);
     printf("    Local channel number on controller: %d\n",
          channelInfo.controller.controllerPort);
}
else
{
     /* handle the error */
}
```