TPMC866-SW-42

VxWorks Device Driver

8 Channel Serial PMC

Version 3.0.x

User Manual

Issue 3.0.1

November 2017
Supported Modules:
TPMC866

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.

©2000-2017 by TEWS TECHNOLOGIES GmbH

<table>
<thead>
<tr>
<th>Issue</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>First Issue</td>
<td>April 1998</td>
</tr>
<tr>
<td>1.1</td>
<td>Advanced Error Handling</td>
<td>January 1999</td>
</tr>
<tr>
<td>1.2</td>
<td>New Driver Initialization Function</td>
<td>July 1999</td>
</tr>
<tr>
<td>1.3</td>
<td>FIFO Function adapted for TPMC866-12</td>
<td>July 1999</td>
</tr>
<tr>
<td>1.4</td>
<td>Support for Intel x86 based Targets</td>
<td>June 2000</td>
</tr>
<tr>
<td>1.5</td>
<td>Parameter for tp866drv() expanded</td>
<td>February 2001</td>
</tr>
<tr>
<td>1.6</td>
<td>General Revision</td>
<td>January 2004</td>
</tr>
<tr>
<td>2.0.0</td>
<td>Installation section and driver start changed. ioctl() description extended.</td>
<td>June 19, 2006</td>
</tr>
<tr>
<td>2.0.1</td>
<td>General Revision, New Address TEWS LLC</td>
<td>September 17, 2008</td>
</tr>
<tr>
<td>3.0.0</td>
<td>New version of driver, Legacy and VxBus-Support, 64-bit Support</td>
<td>December 15, 2011</td>
</tr>
<tr>
<td>3.0.1</td>
<td>File-List modified</td>
<td>November 14, 2017</td>
</tr>
</tbody>
</table>
# Table of Contents

1 INTRODUCTION ......................................................................................................... 4
   1.1 Device Driver ........................................................................................................ 4  
2 INSTALLATION .......................................................................................................... 5
   2.1 Legacy vs. VxBus Driver ........................................................................................ 6  
   2.2 VxBus Driver Installation ...................................................................................... 6
      2.2.1 Direct BSP Builds ............................................................................................ 8
      2.2.2 Modification of the ‘Number of serial ports’ .................................................... 8
   2.3 Legacy Driver Installation ..................................................................................... 9
      2.3.1 Include Device Driver in VxWorks Projects .................................................... 9
      2.3.2 Special Installation for Intel x86 based Targets .............................................. 9
      2.3.3 BSP dependent Adjustments .......................................................................... 10
   2.4 System Resource Requirement .......................................................................... 11
   2.5 Default Configuration ......................................................................................... 12
3 VXBUS DRIVER SUPPORT ..................................................................................... 13
   3.1 Assignment of Port Names .................................................................................. 13
   3.2 VxBus Error Codes ............................................................................................. 13
   3.3 Compatibility to pre-VxBus Applications ............................................................ 14
4 LEGACY I/O SYSTEM FUNCTIONS ........................................................................ 15
   4.1 tpmc866Drv .......................................................................................................... 15
   4.2 tpmc866DevCreate ............................................................................................... 17
   4.3 tpmc866PciInit ...................................................................................................... 19
   4.4 tpmc866Init ........................................................................................................... 20
5 BASIC I/O FUNCTIONS ........................................................................................... 22
   5.1 open ...................................................................................................................... 22
   5.2 close ...................................................................................................................... 24
   5.3 read ....................................................................................................................... 26
   5.4 write ..................................................................................................................... 28
   5.5 ioctl ....................................................................................................................... 30
      5.5.1 FIOBAUDRATE ................................................................................................. 32
      5.5.2 FIO_EXAR16XXX_DATABITS .......................................................................... 33
      5.5.3 FIO_EXAR16XXX_STOPBITS ......................................................................... 34
      5.5.4 FIO_EXAR16XXX_PARITY ............................................................................... 35
      5.5.5 FIO_EXAR16XXX_ENABLEHWHS ................................................................. 36
      5.5.6 FIO_EXAR16XXX_DISABLEHWHS ............................................................... 37
      5.5.7 FIO_EXAR16XXX_SETBREAK ........................................................................ 38
      5.5.8 FIO_EXAR16XXX_CLEARBREAK ................................................................. 39
      5.5.9 FIO_EXAR16XXX_CHECKBREAK .............................................................. 40
      5.5.10 FIO_EXAR16XXX_CHECKERRORS ............................................................ 41
      5.5.11 FIO_EXAR16XXX_RECONFIGURE ........................................................... 42
      5.5.12 FIO_EXAR16XXX_FIFO .............................................................................. 43
1 Introduction

1.1 Device Driver

The TPMC866-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 TPMC866-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 TPMC866 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
- enable/disable hardware handshake (only in FIFO mode)
- changing baudrates

The TPMC866-SW-42 supports the modules listed below:

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMC866-10</td>
<td>8 channel RS232 serial interface with front I/O and P14 I/O</td>
<td>(PMC)</td>
</tr>
<tr>
<td>TPMC866-11</td>
<td>8 channel RS422 serial interface with front I/O and P14 I/O</td>
<td>(PMC)</td>
</tr>
<tr>
<td>TPMC866-12</td>
<td>8 channel serial interface RS422, RS485 (FD/HD) with front I/O and P14 I/O</td>
<td>(PMC)</td>
</tr>
</tbody>
</table>

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

- TPMC866 User Manual
- ST16C654 and XR16C864 UART Controller Manual
2 Installation

Following files are located on the distribution media:

Directory path 'TPMC866-SW-42':

- TPMC866-SW-42-3.0.1.pdf: PDF copy of this manual
- TPMC866-SW-42-VXBUS.zip: Zip compressed archive with VxBus driver sources
- TPMC866-SW-42-LEGACY.zip: Zip compressed archive with legacy driver sources
- ChangeLog.txt: Release history
- Release.txt: Release information

The archive TPMC866-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tpmc866':

- tpmc866drv.c: TPMC866 device driver source (TPMC866 specific)
- tpmc866def.h: TPMC866 driver include file
- tpmc866defaults.h: TPMC866 device default configuration
- tpmc866.h: TPMC866 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
- 40tpmc866.cdf: Component description file for VxWorks development tools
- tpmc866.dc: Configuration stub file for direct BSP builds
- tpmc866.dr: Configuration stub file for direct BSP builds
- include/tvxbHal.h: Hardware dependent interface functions and definitions
- include/tvxbVersion.h: VxWorks version dependent definitions
- apps/tpmc866exa.c: Example application

The archive TPMC866-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tpmc866':

- tpmc866drv.c: TPMC866 device driver source
- tpmc866def.h: TPMC866 driver include file
- tpmc866defaults.h: TPMC866 device default configuration
- tpmc866.h: TPMC866 include file for driver and application
- tpmc866pci.c: TPMC866 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)
- 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:

<table>
<thead>
<tr>
<th>Legacy Driver</th>
<th>VxBus Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>• VxWorks 5.x releases</td>
<td>• VxWorks 6.6 and later releases with VxBus PCI bus</td>
</tr>
<tr>
<td>• VxWorks 6.5 and earlier releases</td>
<td>• SMP systems (only the VxBus driver is SMP safe)</td>
</tr>
<tr>
<td>• VxWorks 6.x releases without VxBus PCI bus support</td>
<td>• 64-bit systems (only the VxBus driver is 64-bit compatible)</td>
</tr>
</tbody>
</table>

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 TPMC866-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 TPMC866 device driver is located in the vendor and driver-specific directory installDir/vxworks-6.x/target/3rdparty/tews/tpmc866.

At this point the TPMC866 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/tpmc866
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\tesw\tpmc866
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 TPMC866 driver with the VxWorks development tools (Workbench), the component configuration file 40tpmc866.cdf must be copied to the directory

installDir/vxworks-6.x/target/config/comps/VxWorks.

C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tesw\tpmc866
C:> copy 40tpmc866.cdf \Windriver\vxworks-6.7\target\config\comps\vsxWorks

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\vsxWorks
C:> del CxrCat.txt
C:> make

After successful completion of all steps above and restart of the Wind River Workbench, the TPMC866 driver can be included in VxWorks projects by selecting the "TEWS TPMC866 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 TPMC866 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\tpmc866
C:> copy tpmc866.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tpmc866.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 TPMC866 will not be set up. To support the TPMC866 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 TPMC866 with 8 ports should be supported, the value must be set to a value of 10 at least.
2.3 Legacy Driver Installation

2.3.1 Include Device Driver in VxWorks Projects

For including the TPMC866-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 TPMC866-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 tpmc866 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 TPMC866 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 TPMC866 PCI memory spaces prior the MMU initialization (usrMmuInit()) is done.

The C source file tpmc866pci.c contains the function tpmc866PciInit(). This routine finds out all TPMC866 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 tpmc866PciInit() 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 TPMC866 PCI spaces remains unmapped and an access fault occurs during driver initialization.

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

```c
tpmc866PciInit();
```

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>`.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERDEFINED_MEM_OFFSET</td>
<td>The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI memory space access</td>
</tr>
<tr>
<td>USERDEFINED_IO_OFFSET</td>
<td>The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI I/O space access</td>
</tr>
<tr>
<td>USERDEFINED_LEV2VEC</td>
<td>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)</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Driver requirement</th>
<th>Devices requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>&lt; 1 KB</td>
<td>&lt; 1 KB</td>
</tr>
<tr>
<td>Stack</td>
<td>&lt; 1 KB</td>
<td>---</td>
</tr>
<tr>
<td>Semaphores (*1)</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>

(*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.

\[
\text{<total requirement>} = \text{<driver requirement>} + (\text{<number of devices>} \times \text{<device requirement>})
\]

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 tpmc866defaults.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.

For information on FIFO-settings please refer to 5.5.12 FIO_EXAR16XXX_FIFO.

The following defines are made for configuration:

**TPMC866_DEFAULT_BAUD**
Specifies the default baudrate.
(Default value: 9600)

**TPMC866_DEFAULT_OPTIONS**
Specifies the default terminal settings.
(Default value: OPT_RAW)

**TPMC866_DEFAULT_RXFIFOTRIG**
Specifies the default receive FIFO trigger level. (For TPMC866-10/-11)
(Default value: EXAR16XXX_F_R16)

**TPMC866_12_DEFAULT_RXFIFOTRIG**
Specifies the default receive FIFO trigger level. (For TPMC866-12)
(Default value: 16)

**TPMC866_DEFAULT_TXFIFOTRIG**
Specifies the default transmit FIFO trigger level. (For TPMC866-10/-11)
(Default value: EXAR16XXX_F_T8)

**TPMC866_12_DEFAULT_TXFIFOTRIG**
Specifies the default transmit FIFO trigger level. (For TPMC866-12)
(Default value: 100)

**TPMC866_DEFAULT_DATABITS**
Specifies the default length of the data word.
(Default value: EXAR16XXX_DB_8)

**TPMC866_DEFAULT_STOPBITS**
Specifies the default length of the stop bit.
(Default value: EXAR16XXX_SB_10)

**TPMC866_DEFAULT_PARITY**
Specifies the default parity mode.
(Default value: EXAR16XXX_NOP)

**TPMC866_DEFAULT_HWHSENABLE**
Specifies if hardware handshake is enabled or disabled. A value of FALSE will disable hardware handshake, a value of TRUE will enable the hardware handshake.
(Default value: FALSE)

If an illegal value is specified in the file the default value (of the delivered file) will be used.
3 VxBus Driver Support

The TPMC866 will be fully integrated into 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 TPMC866 supported boards will start with port number 2. For example a system with one TPMC866 (8 channels) will assign the following device names:

<table>
<thead>
<tr>
<th>Name</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>/tyCo/0</td>
<td>1st system port</td>
</tr>
<tr>
<td>/tyCo/1</td>
<td>2nd system port</td>
</tr>
<tr>
<td>/tyCo/2</td>
<td>1st channel of TPMC866</td>
</tr>
<tr>
<td>/tyCo/3</td>
<td>2nd channel of TPMC866</td>
</tr>
<tr>
<td>/tyCo/4</td>
<td>3rd channel of TPMC866</td>
</tr>
<tr>
<td>/tyCo/5</td>
<td>4th channel of TPMC866</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>/tyCo/9</td>
<td>8th channel of TPMC866</td>
</tr>
</tbody>
</table>

If there is more than one supported TPMC866 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 only system generated return codes for the ‘Basic I/O Functions’. The TPMC866 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 tpmc866Init() and not with tpmc866Drv() and tpmc866DevCreate().

Legacy compatible initialization function

```c
STATUS tpmc866Init
(
    int *firstChanNo,
    int *lastChanNo
)
```

This routine just returns the number of the first (`firstChanNo`) and last (`lastChanNo`) port number assigned to the TPMC866 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 TPMC866 driver. For the VxBus-enabled TPMC866 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.

4.1 tpmc866Drv

NAME

tpmc866Drv() - installs the TPMC866 driver in the I/O system

This function is not implemented for systems supporting VxBus.

SYNOPSIS

#include “tpmc866.h”

STATUS tpmc866drv
(
    void
)

DESCRIPTION

This function searches for devices on the PCI bus, installs the TPMC866 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 "tpmc866.h"

STATUS result;

/*-------------------
 Initialize Driver
 -------------------*/
result = tpmc866Drv();
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().

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENXIO</td>
<td>No TPMC866 found</td>
</tr>
</tbody>
</table>

SEE ALSO

VxWorks Programmer’s Guide: I/O System
4.2 tpmc866DevCreate

NAME

tpmc866DevCreate() – Add a TPMC866 device to the VxWorks system

SYNOPSIS

#include “tpmc866.h”

STATUS tpmc866DevCreate
(
       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 TPMC866 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

```c
#include "tpmc866.h"

STATUS result;

/*-------------------------------------------------------
Create the device "/tyCo/2" for the first device
1KB transmit and receive FIFO
-------------------------------------------------------*/
result = tpmc866DevCreate(  "/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()`.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_iosLib_DEVICE_NOT_FOUND</td>
<td>Driver has not been started, or the specified channel has not been detected, or channel structure has not been allocated</td>
</tr>
</tbody>
</table>

### SEE ALSO

VxWorks Programmer's Guide: I/O System
4.3 tpmc866PciInit

NAME

tpmc866PciInit() – Generic PCI device initialization

SYNOPSIS

void tpmc866PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC866 PCI spaces (base address register) and to enable the TPMC866 device for access.

The global variable tpmc866Status obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>Initialization successful completed. The value of tpmc866Status is equal to the number of mapped PCI spaces</td>
</tr>
<tr>
<td>0</td>
<td>No TPMC866 device found</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>Initialization failed. The value of (tpmc866Status &amp; 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).</td>
</tr>
</tbody>
</table>

EXAMPLE

extern void tpmc866PciInit();

tpmc866PciInit();
4.4 tpmc866Init

NAME

tpmc866Init() – initialize TPMC866 driver and devices and return the assigned channel numbers

SYNOPSIS

#include “tpmc866.h”

STATUS tpmc866Init
(
    int *firstDevIdx,
    int *lastDevIdx
)

DESCRIPTION

This function is used by the TPMC866 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 tpmc866Drv() and tpmc866DevCreate(). The devices will be named with ‘/tyCo/<n>’ where <n> specifies the channel. Because the default serial devices are named in the same way, the driver searches for the first free number and will name the TPMC866 starting with this number in a row.

For example if already two local serial devices are created and one TPMC866 is installed, the names ‘/tyCo/0’ and ‘/tyCo/1’ are assigned to the local channels, ‘/tyCo/2’ up to ‘/tyCo/9’ will be assigned to the 8 TPMC866 channels. In this example the function will set a 2 for the first and a 9 for the last assigned device.

After calling this function, it is not necessary to call tpmc866Drv() or tpmc866DevCreate() explicitly.

PARAMETER

firstDevIdx

Pointer where the lowest assigned device number for TPMC866 devices will be returned.

lastDevIdx

Pointer where the highest assigned device number for TPMC866 devices will be returned.
EXAMPLE

#include "tpmc866.h"

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

result = tpmc866Init(&firstNo, &lastNo);

if (result == ERROR)
{
    /* Error handling */
}
else
{
    for (chanNo = firstNo; chanNo <= lastNo; chanNo++)
    {
        sprintf(devName, "/tyCo/%d", chanNo);
        fd = open(devName, 0, 0);
        ...
    }
}

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 TPMC866 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 tpm866DevCreate()) 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

```c
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

```c
#define BUFSIZE 100

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

/*----------------------------
  Read data from TPMC866 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

```c
int fd;
char buffer[] = "Hello World";
int retval;

/*-----------------------------
 Write data to a TPMC866 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 “tpmc866.h”

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

DESCRIPTION

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

PARAMETER

fd

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

This argument specifies the function that shall be executed. The TPMC866 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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIO_EXAR16XXX_DATABITS</td>
<td>Set length of data word</td>
</tr>
<tr>
<td>FIO_EXAR16XXX_STOPBITS</td>
<td>Set length of the stop bit</td>
</tr>
<tr>
<td>FIO_EXAR16XXX_PARITY</td>
<td>Set parity checking mode</td>
</tr>
<tr>
<td>FIO_EXAR16XXX_ENABLEHWHS</td>
<td>Enable hardware handshake mode</td>
</tr>
<tr>
<td>FIO_EXAR16XXX_DISABLEHWHS</td>
<td>Disable hardware handshake mode</td>
</tr>
<tr>
<td>FIO_EXAR16XXX_SETBREAK</td>
<td>Set Break signal</td>
</tr>
<tr>
<td>FIO_EXAR16XXX_CLEARBREAK</td>
<td>Release Break signal</td>
</tr>
<tr>
<td>FIO_EXAR16XXX_CHECKBREAK</td>
<td>Check if a Break signal has been detected</td>
</tr>
<tr>
<td>FIO_EXAR16XXX_CHECKERRORS</td>
<td>Get error state of the device</td>
</tr>
<tr>
<td>FIO_EXAR16XXX_RECONFIGURE</td>
<td>Reconfigure device with the default parameters</td>
</tr>
<tr>
<td>FIO_EXAR16XXX_FIFO</td>
<td>Configure use of FIFO and set trigger levels</td>
</tr>
</tbody>
</table>

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 TPMC866 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 TPMC866 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.

For a description of baudrate calculation, please refer to the TPMC866 User Manual.

Examples:

<table>
<thead>
<tr>
<th>Required Baud Rate</th>
<th>Selected Baud Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>9600</td>
<td>9600</td>
</tr>
<tr>
<td>100000</td>
<td>115200</td>
</tr>
<tr>
<td>115200</td>
<td>115200</td>
</tr>
</tbody>
</table>

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

EXAMPLE

```c
#include "tpmc866.h"

int fd;
int result;

/*-------------------------
 * Set baud rate to 460800
 *-------------------------*/
result = ioctl (fd, FIOBAUDRATE, 460800);
if (result == OK)
{
    /* Success */
}
else
{
    /* Function failed */
}
```

ERROR CODES

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL</td>
<td>Baudrate out of range</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAR16XXX_DB_5</td>
<td>use 5 data bits</td>
</tr>
<tr>
<td>EXAR16XXX_DB_6</td>
<td>use 6 data bits</td>
</tr>
<tr>
<td>EXAR16XXX_DB_7</td>
<td>use 7 data bits</td>
</tr>
<tr>
<td>EXAR16XXX_DB_8</td>
<td>use 8 data bits</td>
</tr>
</tbody>
</table>

**EXAMPLE**

```c
#include "tpmc866.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**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL</td>
<td>Invalid number of data bits specified</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAR16XXX_SB_10</td>
<td>use 1 stop bit</td>
</tr>
<tr>
<td>EXAR16XXX_SB_15</td>
<td>use 1.5 stop bits</td>
</tr>
<tr>
<td>EXAR16XXX_SB_20</td>
<td>use 2 stop bits</td>
</tr>
</tbody>
</table>

**EXAMPLE**

```
#include "tpmc866.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**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL</td>
<td>Invalid number of stop bits specified</td>
</tr>
</tbody>
</table>
### 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAR16XXX_NOP</td>
<td>do not use parity</td>
</tr>
<tr>
<td>EXAR16XXX_EVP</td>
<td>use EVEN parity</td>
</tr>
<tr>
<td>EXAR16XXX_ODP</td>
<td>use ODD parity</td>
</tr>
<tr>
<td>EXAR16XXX_SPP</td>
<td>use SPACE parity</td>
</tr>
<tr>
<td>EXAR16XXX_MAP</td>
<td>use MARK parity</td>
</tr>
</tbody>
</table>

**EXAMPLE**

```c
#include "tpmc866.h"

int fd;
int result;

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

**ERROR CODES**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL</td>
<td>Invalid parity mode specified</td>
</tr>
</tbody>
</table>
5.5.5 FIO_EXAR16XXX_ENABLEHWHS

This I/O control function enables RTS/CTS hardware handshake mode. The function specific control parameter arg is unused and will be ignored.

EXAMPLE

#include “tpmc866.h”

int fd;
int retval;

/*--------------------------
   Enable hardware handshake
--------------------------*/
retval = ioctl(fd, FIO_EXAR16XXX_ENABLEHWHS, 0);
if (retval != ERROR)
{
    /* function succeeded */
}
else
{
    /* handle the error */
}

ERROR CODES

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL</td>
<td>The specified device does not support hardware handshake.</td>
</tr>
</tbody>
</table>
5.5.6 FIO_EXAR16XXX_DISABLEHWHS

This I/O control function disables RTS/CTS hardware handshake mode. The function specific control parameter arg is unused and will be ignored.

EXAMPLE

```
#include "tpmc866.h"

int fd;
int retval;

/****************************************************************************
Disable hardware handshake
****************************************************************************/
retval = ioctl(fd, FIO_EXAR16XXX_DISABLEHWHS, 0);
if (retval != ERROR)
{
    /* function succeeded */
}
else
{
    /* handle the error */
}
```

ERROR CODES

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL</td>
<td>The specified device does not support hardware handshake.</td>
</tr>
</tbody>
</table>
5.5.7 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 “tpmc866.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 */
}
5.5.8  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 "tpmc866.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 */
}
5.5.9 **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**

```c
#include “tpmc866.h”

int fd;
int retval;
int breakDetect;

/*-----------------------
 Check break
 -----------------------*/
retval = ioctl(fd, FIO_EXAR16XXX_CHECKBREAK, &breakDetect);
if (retval != ERROR)
{
    /* function succeeded */
    if (breakDetect)
    {
        /* A break has been detected */
    }
} else
{ /* handle the error */
}
```
5.5.10 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) where the status will be returned. The returned status is an OR'ed value of the following flags:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAR16XXX_FRAMING_ERR</td>
<td>This bit is set if a framing error has been detected since the last call.</td>
</tr>
<tr>
<td>EXAR16XXX_PARITY_ERR</td>
<td>This bit is set if a parity error has been detected since the last call.</td>
</tr>
<tr>
<td>EXAR16XXX_OVERRUN_ERR</td>
<td>This bit is set if an overrun error has been detected since the last call.</td>
</tr>
</tbody>
</table>

**EXAMPLE**

```c
#include "tpmc866.h"

int fd;
int retval;
unsigned long errStat;

/**********************************************************************
 Get error status
/**********************************************************************
retval = ioctl(fd, FIO_EXAR16XXX_CHECKERRORS, (int)&errStat);
if (retval != ERROR)
{
    /* function succeeded */
    if (errStat & EXAR16XXX_FRAMING_ERR)
    {
        /* Framing error occurred */
    }
}
else
{
    /* handle the error */
}  
```
5.5.11 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

```c
#include "tpmc866.h"

int fd;
int retval;

/*--------------------------
 Reconfigure serial channel
 --------------------------*/
retval = ioctl(fd, FIO_EXAR16XXX_RECONFIGURE, 0);
if (retval != ERROR)
{
    /* function succeeded */
}
else
{
    /* handle the error */
}
```
5.5.12 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. If a higher value is specified, there will be less interrupts (and overhead), but a single interrupt will run for a longer time.

Allowed values for TPMC866-10/-11 (64-character FIFO) are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAR16XXX_F_R8</td>
<td>FIFOs enabled, receive FIFO trigger level at 8</td>
</tr>
<tr>
<td>EXAR16XXX_F_R16</td>
<td>FIFOs enabled, receive FIFO trigger level at 16</td>
</tr>
<tr>
<td>EXAR16XXX_F_R56</td>
<td>FIFOs enabled, receive FIFO trigger level at 56</td>
</tr>
<tr>
<td>EXAR16XXX_F_R60</td>
<td>FIFOs enabled, receive FIFO trigger level at 60</td>
</tr>
<tr>
<td>EXAR16XXX_F_NO</td>
<td>FIFOs disable, only valid if transmit FIFO will also be disabled.</td>
</tr>
</tbody>
</table>

Allowed values for TPMC866-12 (128-character FIFO) are 1 up to 127.

txFifoTrigger

Specifies the transmit FIFO trigger level. If a lower value is specified, there will be less interrupts (and overhead), but interrupts will run for a longer time.

Allowed values for TPMC866-10/-11 (64-character FIFO) are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXAR16XXX_F_T8</td>
<td>FIFOs enabled, transmit FIFO trigger level at 8</td>
</tr>
<tr>
<td>EXAR16XXX_F_T16</td>
<td>FIFOs enabled, transmit FIFO trigger level at 16</td>
</tr>
<tr>
<td>EXAR16XXX_F_T32</td>
<td>FIFOs enabled, transmit FIFO trigger level at 32</td>
</tr>
<tr>
<td>EXAR16XXX_F_T56</td>
<td>FIFOs enabled, transmit FIFO trigger level at 56</td>
</tr>
<tr>
<td>EXAR16XXX_F_NO</td>
<td>FIFOs disable, only valid if receive FIFO will also be disabled.</td>
</tr>
</tbody>
</table>

Allowed values for TPMC866-12 (128-character FIFO) are 1 up to 127.

An optimal interrupt setting depends on the system and the application. We recommend using the default settings which are configured to keep interrupt execution time quite short.
EXAMPLE

#include "tpmc866.h"

int fd;
int result;
EXAR16XXX_FIFO_STRUCT fifoSet;

/*@-----------------------------
   Enable FIFO with
   - receive trigger at 16
   - transmit trigger at 32
   -----------------------------*/
fifoSet.rxFifoTrigger = EXAR16XXX_F_R16;
fifoSet.txFifoTrigger = EXAR16XXX_F_T32
result = ioctl (fd, FIO_EXAR16XXX_FIFO, &fifoSet);
if (result == OK)
{
   /* Success */
}
else
{
   /* Function failed */
}

ERROR CODES

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EINVAL</td>
<td>Invalid Trigger Level specified or the combination of trigger levels is not allowed.</td>
</tr>
</tbody>
</table>