The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV002-SW-42

VxWorks Device Driver

Multiple Channel Serial Interface

Version 3.1.x

User Manual

Issue 3.1.0

February 2021

powerBridge Computer

Ehlbeek 15a
30938 Burgwedel
fon 05139-9980-0
fax 05139-9980-49

www.powerbridge.de
info@powerbridge.de

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7 25469 Halstenbek, Germany
49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
ail: info@tews.com www.tews.com

TDRV002-SW-42

VxWorks Device Driver

Multiple Channel Serial Interface

Supported Modules:

TPMC37x
TPMC46x
TPMC47x
TXMC37x
TXMC46x
TCP46x
TCP47x

Issue	Description	Date
1.0	First Issue (TPMC461-SW-42)	July 14, 2004
1.1.0	Driver Name changed to TDRV002	January 18, 2005
1.2.0	Local Loopback Mode and Selftest added	February 28, 2005
1.2.1	File list changed	April 1, 2005
1.3.0	Support for programmable interfaces / modules added, tdrv002DevCreate() parameters changed, new ioctl() function FIOSETINTERFACE	September 28, 2005
1.4.0	TPMC467/TCP467 Support added, correction in description of tdrv002DevCreate() Distribution file list changed Missing description of error codes added Introduction / Installation updated	July 20, 2006
1.4.1	New Address TEWS LLC	October 9, 2006
1.4.2	Address TEWS LLC removed	November 18, 2009
2.0.0	VxBus Driver Support added	January 19, 2010
2.0.1	Configuration Hint added, additional information for Selftest execution	February 18, 2010
2.0.2	Legacy vs. VxBus Driver modified	March 25, 2010
2.1.0	New debug function tdrv002Show for VxBus support	April 26, 2010
2.2.0	Description and schematic for local loopback added, Support of TPMC377, TPMC470, TCP469, TCP470 added	November 5, 2010
2.2.1	tdrv002Show function modified	February 3. 2011
2.3.0	Function parameters modified for 64-bit compatibility	January 10, 2012
2.4.0	Functions for handshake and modem line support added	February 17, 2012
2.5.0	Function channel identification added, Description of WorkQueue issue	March 8, 2012
2.6.0	New ioctl() function FIOWAITMODEMSTATE, Description of Legacy Driver Include modified	April 17, 2012
2.7.0	Support of TXMC375 added, New ioctl() function FIOSETFIFOTRIGGER	February 26, 2013
2.8.0	Support of TPMC378 added	February 21, 2014
2.9.0	Support of TXMC376, TXMC465 and TXMC466 added Description of WorkQueue issue removed	February 6, 2015
2.9.1	Support of TCP468, TXMC463 added Table of Supported Modules renewed	January 26, 2016
3.0.0	VxWorks 7 support added Legacy I/O Functions removed from VxBus support New function tdrv002GetDriverType() New function tdrv002GetDeviceNamePrefix() (for VxBus) New function tdrv002GetValidDeviceNumbers() (for VxBus) Chapter VxBus Driver Support modified Chapter additional error codes removed	January 10, 2017
3.0.1	Description of boardId in FIOCHANNELINFO modified	January 10, 2017
3.0.2	No change of content, PDF recreated	November 26, 2020
3.1.0	ioctl for RTP-Support	February 24, 2021

Table of Contents

1 Introduction

The TDRV002-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 TDRV002-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 64-bit and SMP systems.

The TDRV002 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 functions are supported (if the channel supports this function):

- 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)
- control full modem lines
- check and wait for control and modem line states
- changing Baud Rates
- enabling/disabling local loopback mode
- local selftest
- changing I/O interface (only programmable interfaces)

The TDRV002-SW-42 supports the modules listed below:

Module	Serial Interfaces	Programmable Interfaces	FIFO-Size (Bytes)	Isolated	Form Factor	Conduction Cooled
TPMC371	8		64		PMC	●
TPMC372	4		64		PMC	●
TPMC375	8	●	64		PMC	●
TPMC376	4	●	64		PMC	●

Module	Serial Interfaces	Programmable Interfaces	FIFO-Size (Bytes)	Isolated	Form Factor	Conduction Cooled
TPMC377	4	●	64	●	PMC	●
TPMC378	8		64	●	PMC	●
TPMC460	16		64		PMC	
TPMC461	8		64		PMC	
TPMC462	4		64		PMC	
TPMC463	4		64		PMC	
TPMC465	8	●	64		PMC	
TPMC466	4	●	64		PMC	
TPMC467	4	●	64		PMC	
TPMC470	4	●	64	●	PMC	
TXMC375	8	●	256		XMC	●
TXMC376	4	●	256		XMC	●
TXMC463	4		256		XMC	
TXMC465	8	●	256		XMC	
TXMC466	4	●	256		XMC	
TCP460	16		64		cPCI	
TCP461	8		64		cPCI	
TCP462	4		64		cPCI	
TCP463	4		64		cPCI	
TCP465	8	●	64		cPCI	
TCP466	4	●	64		cPCI	
TCP467	4	●	64		cPCI	
TCP468	4		64		cPCI	
TCP469	8	●	64	●	cPCI	
TCP470	4	●	64	●	cPCI	

In this document all supported modules and devices will be called TDRV002. Specials for certain devices will be advised.

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

TEWS TECHNOLOGIES VxWorks Device Drivers - Installation Guide
User Manual of the used module
Programmer's Guide: I/O System – Serial I/O devices
Kernel Programmer's Guide: I/O System – Serial I/O devices

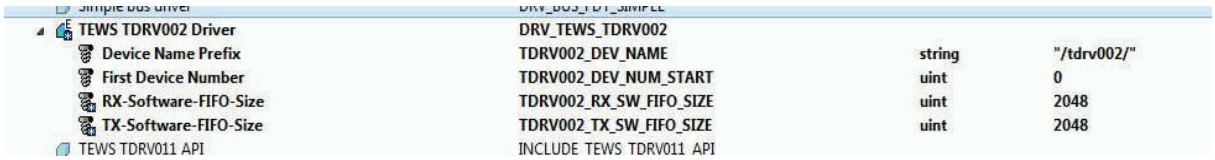
2 VxBus Driver Support

The TDRV002 will be fully integrated to the VxWorks system and the devices will be automatically created when booting VxWorks.

2.1 Device Driver Configuration Parameters

There are parameters to configure the names of the devices and to configure the size of the Software-FIFOs allocated for the devices.

The TDRV002 parameters can be modified in the image project configuration. The parameter list can be found in a folder below of the TDRV002 driver include.



2.1.1 Assignment of Port Names

The port names are assigned automatically when the ports are created during start-up. The assigned port names are defined by configuration parameters which may be adapted before creating the final project image.

The parameter TDRV002_DEV_NAME specifies the prefix of the devices.

The parameter TDRV002_DEV_NUM_START specifies the first assigned device number.

The device names will be built as <TDRV002_DEV_NAME><(TDRV002_DEV_NUM_START + n)>.

It is necessary, that the parameters TDRV002_DEV_NAME and TDRV002_DEV_NUM_START are chosen that there is a unique naming for all devices, otherwise there may undesirable effects. Please consider this especially if the TDRV002 naming should look like the naming of local serial ports ("/tyCo/<n>").

For example a system with one TPMC462 (4 channels) will assign the following device names, if the default parameters (shown above) are used:

/tdrv002/0	1st channel of TPMC462
/tdrv002/1	2nd channel of TPMC462
/tdrv002/2	3rd channel of TPMC462
/tdrv002/3	4th channel of TPMC462

If the parameters are modified, e.g. to use the naming of the local serial ports (e.g. 2 local serial ports) (TDRV002_DEV_NAME = "/tyCo/" and TDRV002_DEV_NUM_START = 2) the following device names will be assigned to the TDRV002 devices:

/tyCo/0	1st local port
/tyCo/1	2nd local port
/tyCo/2	1st channel of TPMC462
/tyCo/3	2nd channel of TPMC462
/tyCo/4	3rd channel of TPMC462
/tyCo/5	4th channel of TPMC462

If there is more than one TDRV002 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 variant will follow up in a row. For example a system with one TPMC462 (4 channels) and one TPMC372 (4 channels) may assign the following two device names table. (default settings)

	(TPMC462 found first)	(TPMC372 found first)
/tdrv002/0	1st channel of TPMC462	1st channel of TPMC372
/tdrv002/1	2nd channel of TPMC462	2nd channel of TPMC372
/tdrv002/2	3rd channel of TPMC462	3rd channel of TPMC372
/tdrv002/3	4th channel of TPMC462	4th channel of TPMC372
/tdrv002/4	1st channel of TPMC372	1st channel of TPMC462
/tdrv002/5	2nd channel of TPMC372	2nd channel of TPMC462
/tdrv002/6	3rd channel of TPMC372	3rd channel of TPMC462
/tdrv002/7	4th channel of TPMC372	4th channel of TPMC462

After booting the available devices can be checked with *devs()*. This function will return a list of all created devices.

2.1.2 SW-FIFO Configuration

The parameters TDRV002_RX_SW_FIFO_SIZE and TDRV002_RX_SW_FIFO_SIZE specify the size of receive and transmit software FIFO in Bytes. Depending on the application it might be necessary to increase the size, for example if the application collects data over some time or if large "packets" shall be send or received.

The default value is 2048 Byte for both FIFOs.

2.2 Default Port Configuration

The driver will create the port with the following default configuration:

- 9600 Baud
- 8 Data- and 1 Stopbit
- FIFO enabled (Triggerlevels: Rx = 56 – Tx = 8)

Ports supporting a programmable interface (e.g. TPMC465) will start-up with a disabled interface. Before using the port it must be configure with the corresponding ioctl-function (*FIOSETINTERFACE*).

For further information of setting the FIFO-trigger-levels, please refer to 4.2 Configuration of FIFO-Trigger-Levels.

2.3 Enable RTP-Support

Using TDRV002 devices tunneled from RTPs is implemented. For this the “TEWS TDRV002 IOCTL command validation” must be enabled in system configuration.

If “tdrv002.h” is included into the sources of RTP-Projects the definition of TVXB_RTP_CONTEXT must be added to the project. (Find more detailed information in “TEWS TECHNOLOGIES VxWorks Device Drivers - Installation Guide”).

All legacy functions, functions for version compatibility and debugging functions are not usable from RTPs.

Legacy I/O System Functions

This chapter describes the 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 necessary for the legacy TDRV002 driver. For the VxBus-enabled TDRV002 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules. Therefore these functions are not implemented in the VxBus-enabled TDRV002 driver.

2.4 tdrv002Drv

NAME

tdrv002Drv() - installs the TDRV002 driver in the I/O system.

SYNOPSIS

```
#include “tdrv002.h”

STATUS tdrv002Drv
(
        void
)
```

DESCRIPTION

This function searches for devices on the PCI bus and installs the TDRV002 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   "tdrv002.h"
STATUS     result;

/*----------------------
  Initialize Driver
  ----------------------*/
result = tdrv002Drv();
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
S_tdrv002Drv_NOMEM	Driver cannot allocate memory
S_tdrv002Drv_NXIO	No device found

SEE ALSO

VxWorks Programmer's Guide: I/O System

2.5 tdrv002DevCreate

NAME

tdrv002DevCreate() – Adds TDRV002 device to the system and initializes the device hardware with the specified configuration

SYNOPSIS

#include “tdrv002.h”

```
STATUS tdrv002DevCreate
(
    char                    *name,
    int                     glbChanNo,
    int                     rdBufSize,
    int                     wrtBufSize,
    TDRV002_CHANCONFIG      *devConf
)
```

DESCRIPTION

This routine creates a device on a specified serial channel that will be serviced by the TDRV002 driver.

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

PARAMETER

name

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

glbChanNo

This index number specifies the device to add to the system.
The index number depends on the search priority of the modules. The modules will be searched in the following order:

TPMC371-10, -11, -12,	TPMC372-xx,	TPMC375-xx,
TPMC376-xx,	TPMC377-xx,	
TPMC460-xx,	TPMC461-xx,	TPMC462-xx,
TPMC463-xx,	TPMC465-xx,	TPMC466-xx,
TPMC467-xx,	TPMC470-xx,	
TCP460-xx,	TCP461-x,	TCP462-xx,
TCP463-xx,	TCP465-xx,	TCP466-xx,
TCP467-xx,	TCP469-xx,	TCP470-xx,
TXMC375-xx		

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

Example: (A system with 2x TPMC461-10, 1x TPMC372-10, 1x TPMC372-11) will assign the following device indices:

Module	**Device Index**
TPMC372-10	0 ... 3
TPMC372-11	4 … 7
TPMC461-10 (1st)	8 … 15
TPMC461-10 (2nd)	16 … 23

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 points to a structure (*TDRV002_CHANCONFIG*) containing the default configuration of the channel. (This function will be used for reconfigurations).

```
typedef struct
{
        unsigned int        baudrate;
        unsigned int        comPara;
        unsigned char       rxFSize;
        unsigned char       txFSize;
        int                 options;
} TDRV002_CHANCONFIG;
```

baudrate

Selects the initial baud rate of the channel. (Allowed values depend on hardware)

comPara

This value is a field of ORed definitions, specifying the channel setup. One value of every group must be ORed into the value.

Number of data bits:

Word-Length	Description
TDRV002_DATABIT_5	word length = 5 bit
TDRV002_DATABIT_6	word length = 6 bit
TDRV002_DATABIT_7	word length = 7 bit
TDRV002_DATABIT_8	word length = 8 bit

Length of stop bit:

Bit-Length	Description
TDRV002_STOPBIT_1	stop bit length = 1 bit
TDRV002_STOPBIT_1_5	stop bit length = 1.5 bit, (only data length 5)
TDRV002_STOPBIT_2	stop bit length = 2 bit, (only data length 6,7,8)

Parity mode:

Mode	Description
TDRV002_PARITY_NO	parity is disabled
TDRV002_PARITY_ODD	odd parity is used
TDRV002_PARITY_EVEN	even parity is used
TDRV002_PARITY_MARK	a mark parity bit is used
TDRV002_PARITY_SPACE	a space parity bit is used

Hardware handshake:

Mode	Description
TDRV002_HWHS_DISABLE	hardware handshake is disabled
TDRV002_HWHS_ENABLE	hardware handshake is enabled (only if FIFO is enabled)

FIFO mode:

Mode	Description
TDRV002_FIFO_DISABLE	Hardware FIFO is disabled
TDRV002_FIFO_ENABLE	Hardware FIFO is enabled. Receiver and transmitter trigger level must be set in rxFSize and txFSize.

Local loopback mode:

Mode	Description
TDRV002_LOCALLOOP_DISABLE	Disable local loopback mode
TDRV002_LOCALLOOP_ENABLE	Enabled local loopback mode

Interface configuration (only valid for programmable I/O interfaces):
(A combination of the flags below must be specified to configure the interface)

Flag	Description
TDRV002_TRANS_RS485_RS232_SEL	RS485/RS232# configuration pin
TDRV002_TRANS_HDPLX_SEL	HDPLX configuration pin
TDRV002_TRANS_RENA_SEL	RENA configuration pin
TDRV002_TRANS_RTERM_SEL	RTERM configuration pin
TDRV002_TRANS_TTERM_SEL	TTERM configuration pin
TDRV002_TRANS_SLEWLIMIT_SEL	SLEWLIMIT configuration pin
TDRV002_TRANS_SHDN_SEL	SHDN configuration pin
TDRV002_AUTO_RS485_SEL_ENABLE	enable Auto RS485 Operation mode of XR17D15x

The function of the interface configuration pins can be found in the corresponding hardware User Manual.

There are predefined values of the interface configuration described in the hardware manual, you can just OR the predefined value instead of a list of configuration flags. Below is a list of the values:

Value	Interface
TDRV002_INTF_OFF	interface disabled
TDRV002_INTF_OFF	interface disabled
TDRV002_INTF_RS232	RS232
TDRV002_INTF_RS422	RS422 (Multidrop / Full duplex)
TDRV002_INTF_RS485FDM	RS485 (Full duplex master)
TDRV002_INTF_RS485FDS	RS485 (Full duplex slave)
TDRV002_INTF_RS485HD	RS485 (Half duplex)

rxFSize

Specifies the HW receiver trigger level if the HW FIFO is enabled. Allowed values depend on the HW FIFO size. (1..64 for TPMCxxx and TCPxxx, or 1..256 for TXMCxxx)
(For further information of setting the FIFO-trigger-levels, please refer to 4.2 Configuration of FIFO-Trigger-Levels.)

txFSize

Specifies the HW transmitter trigger level if the HW FIFO is enabled. Allowed values depend on the HW FIFO size. (1..64 for TPMCxxx and TCPxxx, or 1..256 for TXMCxxx)
(For further information of setting the FIFO-trigger-levels, please refer to 4.2 Configuration of FIFO-Trigger-Levels.)

options

Selects the initial VxWorks driver options. (Please refer to VxWorks manuals)

EXAMPLE

```
#include   "tdrv002.h"

STATUS               result;
TDRV002_CHANCONFIG   tdrv002conf;

/*-------------------------------------------------------
  Create the device "/tdrv002/0" on channel 0
     read and write buffer sizes of 1024 byte.
     Baudrate:       115200Baud
     Databits:       8
     Stopbits:       1
     Parity:         off
     Handshake: off
     FIFOs:          enabled
     Rx Trigger:     more than 30 characters in FIFO
     Tx Trigger:     less than 10 characters in FIFO
     Local Loop:     off
     Options:        raw mode
     I/O interface:  RS232
  -------------------------------------------------------*/
tdrv002conf.baudrate =    115200;
tdrv002conf.comPara =     TDRV002_DATABIT_8    |
                          TDRV002_STOPBIT_1    |
                          TDRV002_PARITY_NO    |
                          TDRV002_HWHS_DISABLE |
                          TDRV002_FIFO_ENABLE  |
                          TDRV002_LOCALLOOP_DISABLE |
                          TDRV002_INTF_RS232;
tdrv002conf.rxFSize =     30;
tdrv002conf.txFSize =     10;
tdrv002conf.options =     OPT_RAW;

result = tdrv002DevCreate ("/tdrv002/0", 0, 1024, 1024, &tdrv002conf);
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_tdrv002Drv_NODRV	The TDRV002 driver is not installed
S_tdrv002Drv_NODEV	Specified device not found
S_tdrv002Drv_EXISTS	The specified device has already been created
S_tdrv002Drv_ILLINTF	Illegal interface specified
S_tdrv002Drv_ILLBAUD	Illegal default baud rate specified
S_tdrv002Drv_ILLPARAM	Illegal parameter specified
S_tdrv002Drv_MODENOTSUPP	Unsupported default mode specified
S_tdrv002Drv_CONFERR	Configuration error (specified flags exclude each other)

SEE ALSO

VxWorks Programmer's Guide: I/O System

2.6 tdrv002PciInit

NAME

tdrv002PciInit() – Generic PCI device initialization

SYNOPSIS

```
void tdrv002PciInit
(
        void
)
```

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TDRV002 PCI spaces (base address register) and to enable the TDRV002 device for access.

The global variable *tdrv002Status* 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 tdrv002Status is equal to the number of mapped PCI spaces
0	No TDRV002 device found
< 0	Initialization failed. The value of (tdrv002Status & 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 tdrv002PciInit();

tdrv002PciInit();
```

3 Basic I/O Functions

3.1 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 TDRV002 device, a file descriptor must be opened by invoking the basic I/O function *open()*.

PARAMETER

name

Specifies the name of the device which shall be opened.

flags

Not used

mode

Not used

EXAMPLE

```
int        fd;

/*-----------------------------------------
  Open the device named "/ tdrv002/2" for I/O
  -----------------------------------------*/
fd = open("/tdrv002/2", 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()*

3.2 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()

3.3 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 TDRV002 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()

3.4 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 TDRV002 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()

3.5 ioctl

NAME

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

SYNOPSIS

#include “tdrv002.h”

```
int ioctl
(
        int                     fd,
        int                     request,
        TDRV002_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 TDRV002 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
FIODATABITS	Set length of data word
FIOSTOPBITS	Set length of the stop bit
FIOPARITY	Set parity checking mode
FIOHWHS	Enable/Disable hardware handshake mode
FIOSETBREAK	Set/Release Break
FIOSETMODEM	Sets specified modem control lines
FIOSETCLEARMODEM	Clears specified modem control lines
FIOGETMODEM	Returns the state of the modem control lines
FIOWAITMODEMSTATE	Wait for a specified state on modem or control line
…	

…	
FIORECONFIGURE	Reconfigure device with the default parameters
FIOSTATUS	Get state of the device
FIOLOCALLOOP	Enable/Disable local loopback mode
FIOLOCALSELFTEST	Execute a local selftest
FIOSETINTERFACE	Change the programmable I/O interface
FIOSETFIFOTRIGGER	Set FIFO Trigger levels
FIOCHANNELINFO	Returns information regarding the specified channel

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()*.

SEE ALSO

ioLib, basic I/O routine - ioctl()

3.5.1 FIOBAUDRATE

This I/O control function sets up a new baudrate. The function specific control parameter arg specifies the new baudrate.

EXAMPLE

```
#include "tdrv002.h"

int                  fd;
int                  retval;


/*----------------------
  Set baud rate to 9600
  ----------------------*/
retval = ioctl(fd, FIOBAUDRATE, 9600);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_ILLBAUD	Illegal baud rate specified

3.5.2 FIODATABITS

This I/O control function sets the data word length. The function specific control parameter arg specifies the length of the data word. The following values are defined:

Value	Description
TDRV002_DATABIT_5	word length = 5 bit
TDRV002_DATABIT_6	word length = 6 bit
TDRV002_DATABIT_7	word length = 7 bit
TDRV002_DATABIT_8	word length = 8 bit

EXAMPLE

```
#include "tdrv002.h"

int                 fd;
int                 retval;


/*-------------------------------------
  Set channel to a word length of 7 bit
  -------------------------------------*/
retval = ioctl(fd, FIODATABITS, TDRV002_DATABIT_7);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_ILLPARAM	Illegal data word length specified

3.5.3 FIOSTOPBITS

This I/O control function sets the length of the stop bit. The function specific control parameter arg specifies the length of the stop bit word. The following values are defined:

Value	Description
TDRV002_STOPBIT_1	stop bit length = 1 bit
TDRV002_STOPBIT_1_5	stop bit length = 1.5 bit, (only data length 5)
TDRV002_STOPBIT_2	stop bit length = 2 bit, (only data length 6, 7, 8)

EXAMPLE

```
#include "tdrv002.h"

int                 fd;
int                 retval;


/*----------------------------------------
  Set channel to a stop bit length of 1 bit
  ----------------------------------------*/
retval = ioctl(fd, FIOSTOPBITS, TDRV002_STOPBIT_1);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_ILLPARAM	Illegal data stop bit length specified

3.5.4 FIOPARITY

This I/O control function sets the parity mode. The function specific control parameter arg specifies the new parity mode. The following values are defined:

Value	Description
TDRV002_PARITY_NO	parity is disabled
TDRV002_PARITY_ODD	odd parity is used
TDRV002_PARITY_EVEN	even parity is used
TDRV002_PARITY_MARK	a mark parity bit is used
TDRV002_PARITY_SPACE	a space parity bit is used

EXAMPLE

```
#include "tdrv002.h"

int                 fd;
int                 retval;


/*----------------------------
  Configure channel no parity
  ----------------------------*/
retval = ioctl(fd, FIOPARITY, TDRV002_PARITY_NO);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_ILLPARAM	Illegal parity mode specified

3.5.5 FIOHWHS

This I/O control function enables or disables the hardware handshake. The function specific control parameter arg specifies if the hardware handshake shall be enabled or disabled. The following values are defined:

Value	Description
TDRV002_HWHS_DISABLE	hardware handshake is disabled
TDRV002_HWHS_ENABLE	hardware handshake is enabled (only if FIFO is enabled)

EXAMPLE

```
#include "tdrv002.h"

int                 fd;
int                 retval;


/*--------------------------
  Disable hardware handshake
  --------------------------*/
retval = ioctl(fd, FIOHWHS, TDRV002_HWHS_DISABLE);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_ILLPARAM	Illegal handshake mode specified
S_tdrv002Drv_MODENOTSUPP	The hardware does not support the specified mode

3.5.6 FIOSETBREAK

This I/O control function sets or resets break state on transmit line. The function specific control parameter arg specifies the state on transmit line. The following values are defined:

Value	Description
TDRV002_BREAK_SET	Set break on transmit line(s)
TDRV002_BREAK_RESET	Reset break on transmit line(s)

EXAMPLE

```
#include "tdrv002.h"

int                 fd;
int                 retval;


/*-----------------------
  Set break on Tx line(s)
  -----------------------*/
retval = ioctl(fd, FIOSETBREAK, TDRV002_BREAK_SET);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_ILLPARAM	Illegal parameter specified

3.5.7 FIOSETMODEM

This I/O control function sets the specified modem and handshake lines into active state. The function specific control parameter arg is an ORed value and specifies the lines that shall be set. The following values are defined:

Value	Description
TDRV002_MCTL_RTS	Set RTS line into active state
TDRV002_MCTL_DTR	Set DTR line into active state

The supported handshake and modem lines depend on the used TDRV002 board, port number and interface. Which handshake and modem lines are supported can be found in the User Manual of the TDRV002 supported module.

EXAMPLE

```
#include "tdrv002.h"

int                 fd;
int                 retval;


/*--------------
  Set DTR active
  --------------*/
retval = ioctl(fd, FIOSETMODEM, TDRV002_MCTL_DTR);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_MODENOTSUPP	The specified handshake or modem line is not supported

3.5.8 FIOCLEARMODEM

This I/O control function sets the specified modem and handshake lines into passive state. The function specific control parameter arg is an ORed value and specifies the lines that shall be reset. The following values are defined:

Value	Description
TDRV002_MCTL_RTS	Set RTS line into passive state
TDRV002_MCTL_DTR	Set DTR line into passive state

The supported handshake and modem lines depend on the used TDRV002 board, port number and interface. Which handshake and modem lines are supported can be found in the User Manual of the TDRV002 supported module.

EXAMPLE

```
#include "tdrv002.h"

int                 fd;
int                 retval;


/*----------------------
  Set RTS and DTR active
  ----------------------*/
retval = ioctl(fd, FIOCLEARMODEM, (TDRV002_MCTL_RTS | TDRV002_MCTL_DTR));
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_MODENOTSUPP	The specified handshake or modem line is not supported

3.5.9 FIOGETMODEM

This I/O control function returns the state of the modem and control lines 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
TDRV002_MCTL_RTS	If set RTS is in active state
TDRV002_MCTL_DTR	If set DTR is in active state
TDRV002_MCTL_CTS	If set CTS is in active state
TDRV002_MCTL_DSR	If set DSR is in active state
TDRV002_MCTL_RI	If set RI is in active state
TDRV002_MCTL_CD	If set CD is in active state

The supported handshake and modem lines depend on the used TDRV002 board, port number and interface. Which handshake and modem lines are supported can be found in the User Manual of the TDRV002 supported module.

This function always returns the state of all handshake and modem lines as they are shown by the controller even if they are not available for the specified port.

EXAMPLE

```
#include "tdrv002.h"

int                 fd;
int                 retval;
unsigned int        modStat;

/*----------------------
  Get modem line status
  ----------------------*/
retval = ioctl(fd, FIOGETMODEM, (TDRV002_IOCTL_ARG_T)&modStat);
if (retval != ERROR)
{     /* check CTS state */
      if (modStat & TDRV002_MCTL_CTS)
      {
            /* CTS is active */
      }
}
else
{
      /* handle the error */
}
```

3.5.10 FIOWAITMODEMSTATE

This I/O control function returns the state of the modem and control lines of the device if one of the specified states occurs, it will return immediately if a state already matches. The function specific control parameter arg points to a supplied buffer (TDRV002_WAITMODEM_BUFFER).

The supported handshake and modem lines depend on the used TDRV002 board, port number and interface. Which handshake and modem lines are supported can be found in the User Manual of the TDRV002 supported module.

This function always returns the state of all handshake and modem lines as they are shown by the controller even if they are not available for the specified port.

The function can wait for states on all the handshake and modem lines even if they are not available for the specified port. The value will not change by external changes, but it may be used if local loopback is enabled.

```
typedef struct
{
        unsigned int        lineMask;
        unsigned int        lineTargetState;
        int                 timeout;
        unsigned int        modemState;
} TDRV002_WAITMODEM_BUFFER;
```

lineMask

This parameter specifies the observed modem and status lines. If one of the specified lines has the specified state or it changes into it the function will return. The mask is an OR'ed value of the following flags:

Value	Description
TDRV002_MCTL_CTS	If set observe CTS state
TDRV002_MCTL_DSR	If set observe DSR state
TDRV002_MCTL_RI	If set observe RI state
TDRV002_MCTL_CD	If set observe CD state

lineTargetState

This parameter specifies the desired states of the modem and status lines. This value is an OR'ed value of the following flags:

Value	Description
TDRV002_MCTL_CTS	If set an active CTS line will be signaled, if not set an passive CTS line will be signaled.
TDRV002_MCTL_DSR	If set an active DSR line will be signaled, if not set an passive DSR line will be signaled.
TDRV002_MCTL_RI	If set an active RI line will be signaled, if not set an passive RI line will be signaled.
TDRV002_MCTL_CD	If set an active CD line will be signaled, if not set an passive CD line will be signaled.

timeout

This value specifies the time (in unit ticks) the function is willing to wait for a specified state before it returns with an error. A value of WAIT_FOREVER means wait for ever.

modemState

The returned value shows the current state of the modem and status lines. The returned status is an OR'ed value of the following flags:

Value	Description
TDRV002_MCTL_RTS	If set RTS is in active state
TDRV002_MCTL_DTR	If set DTR is in active state
TDRV002_MCTL_CTS	If set CTS is in active state
TDRV002_MCTL_DSR	If set DSR is in active state
TDRV002_MCTL_RI	If set RI is in active state
TDRV002_MCTL_CD	If set CD is in active state

EXAMPLE

```
#include "tdrv002.h"

int                             fd;
int                             retval;
TDRV002_WAITMODEM_BUFFER   waitBuf;

/*-------------------------------------------------
  Wait for an active DTR line or a passive CTS line
  -------------------------------------------------*/
waitBuf.lineMask          = TDRV002_MCTL_DTR | TDRV002_MCTL_CTS;
waitBuf.lineTargetState   = TDRV002_MCTL_DTR;
waitBuf.timeout      = 1000;
retval = ioctl(fd, FIOWAITMODEMSTATE, (TDRV002_IOCTL_ARG_T)&waitBuf);
if (retval != ERROR)
{     /* check DTR state */
      if (waitBuf.modemState & TDRV002_MCTL_DTR)
      {
            /* DTR is active */
      }
      if (waitBuf.modemState & TDRV002_MCTL_CTS)
      {
            /* CTS is active */
      }
}
else
{
      /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_ILLPARAM	A specified parameter is not allowed (lineMask == 0?)
S_tdrv002Drv_BUSY	An other task is already waiting for modem line status

3.5.11 FIORECONFIGURE

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 "tdrv002.h"

int                  fd;
int                  retval;


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

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device

3.5.12 FIOSTATUS

This I/O control function returns the 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
TDRV002_STATUS_FRAMINGERR	This bit is set if a framing error has been detected since the last call.
TDRV002_STATUS_PARITYERR	This bit is set if a parity error has been detected since the last call.
TDRV002_STATUS_OVERRUNERR	This bit is set if an overrun error has been detected since the last call.
TDRV002_STATUS_PENDBREAK	This bit is set if a break signal has been detected since the last call.

EXAMPLE

```
#include "tdrv002.h"

int                   fd;
int                   retval;
unsigned int          inStat;


/*------------------
  Get receive status
  ------------------*/
retval = ioctl(fd, FIOSTATUS, (TDRV002_IOCTL_ARG_T)&inStat);
if (retval != ERROR)
{
      /* function succeeded */
      if (inStat & TDRV002_STATUS_FRAMINGERR)
      {
            /* Framing error occurred */
      }
}
else
{
      /* handle the error */
}
```

3.5.13 FIOLOCALLOOP

This I/O control function enables or disables the local loop back mode. For a description of the local loopback wiring, refer to 4.3 Internal Loopback.

The function specific control parameter arg specifies if the local loop back shall be enabled or disabled. The following values are defined:

Value	Description
TDRV002_LOCALLOOP_DISABLE	Local loopback mode is disabled
TDRV002_LOCALLOOP_ENABLE	Local loopback mode is enabled

EXAMPLE

```
#include "tdrv002.h"

int                 fd;
int                 retval;


/*---------------------------
  Disable local loopback mode
  ---------------------------*/
retval = ioctl(fd, FIOLOCALLOOP, TDRV002_LOCALLOOP_DISABLE);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_ILLPARAM	Illegal parameter specified

3.5.14 FIOLOCALSELFTEST

This I/O control function executes a local selftest of the specified device. The local loopback mode will be used to check the communication and all local I/O signals (RxD/TxD/RTS/CTS/DTR/DSR/RI/CD) are used. For a description of the local loopback wiring, refer to 4.3 Internal Loopback.

This function is not allowed to be used in RTP-Context.

The function can be executed with application supplied Rx/Tx buffers or with driver allocated buffers. The function will return a status if the test has been executed.

The function specific control parameter arg points to a supplied buffer (TDRV002_LOCALSELFTEST_BUFFER). The following values are defined:

```
typedef struct
{
        char            *transmitBuffer;
        int             transmitSize;
        char            *receiveBuffer;
        int             receiveSize;
        int             receiveCount;
        unsigned int    status;
} TDRV002_LOCALSELFTEST_BUFFER;
```

transmitBuffer

This is a pointer to a buffer with data that should be transmitted with the local loopback test. This allows the application to check the transmitted data and select the content and size of the test data.

transmitSize

This argument specifies the size of the transmitBuffer. If this argument is set to 0 or to a negative value the driver will allocate a buffer and create test data automatically. If automatically allocated buffers are used, the parameters transmitBuffer, receiveBuffer, receiveSize and receiveCount will be ignored.

receiveBuffer

This is a pointer to a buffer that will return the locally transmitted data. This buffer can be used to compare received and transmitted data.

receiveSize

This argument specifies the size of the receive buffer. The size of the receive buffer must be at least as big as the transmit buffer.

receiveCount

This is the count of characters that have been received during the selftest. The returned value should be the same as transmitSize.

status

This is a bit-field specifying the found problems. The status is an OR'ed value of the following flags:

Status Flag	Description
TDRV002_STATUS_LS_TXRX	Problem with TxD/RxD communication
TDRV002_STATUS_LS_RTSCTS	Problem with RTS/CTS connection
TDRV002_STATUS_LS_DTRDSR	Problem with DTR/DTS connection
TDRV002_STATUS_LS_RI	Problem with RI states
TDRV002_STATUS_LS_CD	Problem with CD states

EXAMPLE

```
#include "tdrv002.h"

int                                 fd;
int                                 retval;
TDRV002_LOCALSELFTEST_BUFFER        selftestBuf;
char                                txBuf[128] = …;
char                                rxBuf[150];


/*-------------------------------------------------------
  Execute local selftest with application supplied buffers
  -------------------------------------------------------*/
selftestBuf.transmitBuffer     = txBuf;
selftestBuf.transmitSize   = 128;
selftestBuf.receiveBuffer  = rxBuf;
selftestBuf.receiveSize    = 150;
selftestBuf.receiveCount   = 0;
retval = ioctl(fd, FIOLOCALSELFTEST, (TDRV002_IOCTL_ARG_T)&selftestBuf);
if (retval != ERROR)
{
     /* function succeeded */
     if (selftestBuf.status)
     {
          /* Check status flags */
     }
     else
     {
          /* No problems found */
     }
}
…
```

```
…

else
{
     /* handle the error */
}

…

/*-----------------------------------------
  Execute local Selftest with internal buffers
  -----------------------------------------*/
selftestBuf.transmitSize = 0;
retval = ioctl(fd, FIOLOCALSELFTEST, (TDRV002_IOCTL_ARG_T)&selftestBuf);
if (retval != ERROR)
{
     /* function succeeded */
     if (selftestBuf.status)
     {
          /* Check status flags */
     }
     else
     {
          /* No problems found */
     }
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_NOMATCHBUF	Receive buffer is smaller than the transmit buffer

3.5.15 FIOSETINTERFACE

This I/O control function sets a new I/O interface configuration. This function is only usable for devices supporting a programmable I/O interface. The function specific control parameter arg specifies the new configuration of the programmable transceivers. (Only allowed for channels supporting a programmable I/O interface) A combination of the flags below must be specified to configure the interface.

Value	Description
TDRV002_TRANS_RS485_RS232_SEL	RS485/RS232# configuration pin
TDRV002_TRANS_HDPLX_SEL	HDPLX configuration pin
TDRV002_TRANS_RENA_SEL	RENA configuration pin
TDRV002_TRANS_RTERM_SEL	RTERM configuration pin
TDRV002_TRANS_TTERM_SEL	TTERM configuration pin
TDRV002_TRANS_SLEWLIMIT_SEL	SLEWLIMIT configuration pin
TDRV002_TRANS_SHDN_SEL	SHDN configuration pin
TDRV002_AUTO_RS485_SEL_ENABLE	enable Auto RS485 Operation mode of XR17D15x

The function of the interface configuration pins can be found in the corresponding hardware User Manual.

There are predefined values of the interface configuration described in the hardware manual, you can just OR the predefined value instead of a list of configuration flags. Below is a list of the values:

Value	Description
TDRV002_INTF_OFF	interface disabled
TDRV002_INTF_RS232	RS232
TDRV002_INTF_RS422	RS422 (Multidrop / Full duplex)
TDRV002_INTF_RS485FDM	RS485 (Full duplex master)
TDRV002_INTF_RS485FDS	RS485 (Full duplex slave)
TDRV002_INTF_RS485HD	RS485 (Half duplex)

EXAMPLE

```
#include "tdrv002.h"

int                   fd;
int                   retval;


/*--------------------------------------
  Set I/O interface for RS485 half duplex
  ---------------------------------------*/
retval = ioctl(fd, FIOSETINTERFACE, TDRV002_INTF_RS485HD);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_NOTSUPP	The device has no programmable I/O interface
S_tdrv002Drv_ILLINTF	The specified interface type is not supported by the device
S_tdrv002Drv_ILLBAUD	The specified interface type cannot support the current baud rate
S_tdrv002Drv_MODENOTSUPP	Handshake mode is not supported for the specified interface type

3.5.16 FIOSETFIFOTRIGGER

This I/O control function sets up the FIFO trigger levels, which shall be used for the specified channel. For a description of the controller's FIFO depth and possible values for the trigger level, please refer to the corresponding hardware user manual.

The function specific control parameter arg points to a supplied buffer (TDRV002_FIFOCONFIG).

```
typedef struct
{
        unsigned char       rxFSize;
        unsigned char       txFSize;
        int                 fifoEnable;
} TDRV002_FIFOCONFIG;
```

rxFSize

This value specifies the trigger level of the receive FIFO. The controller will generate receive interrupts if the specified level is reached in the hardware receive FIFO. Depending on the used UART controller's FIFO size, the value can be set to values between 0 and 64, or between 0 and 256.

txFSize

This value specifies the trigger level of the transmit FIFO. The controller will generate transmit interrupts if the number of data in the hardware transmit FIFO falls below specified level. Depending on the used UART controller's FIFO size, the value can be set to values between 0 and 64, or between 0 and 256.

fifoEnable

This argument specifies if the FIFO shall be enabled or disabled. If the FIFO is disabled the previous parameters are not used and there will be no FIFO for the specified channel. Because of the risk of data loss, if FIFO is disabled, it is recommended to disable the FIFO only in special and well checked cases. The values below must be used:

Value	Description
TDRV002_FIFO_DISABLE	Disable FIFO functionality
TDRV002_FIFO_ENABLE	Enable FIFO functionality

EXAMPLE

```
#include "tdrv002.h"

int                  fd;
int                  retval;
TDRV002_FIFOCONFIG   fifoConf;

fifoConf.rxFSize          = 32;
fifoConf.txFSize          = 24;
fifoConf.fifoEnable  = TDRV002_FIFO_ENABLE;

/*--------------------------------------
  Set FIFO trigger levels
     Receive = 32, Transmit = 24
  --------------------------------------*/
retval = ioctl(fd, FIOSETFIFOTRIGGER, (TDRV002_IOCTL_ARG_T)&fifoConf);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error Code	Description
S_tdrv002Drv_SELFTESTBUSY	Selftest mode is active for the device
S_tdrv002Drv_ILLPARAM	Illegal parameter specified

3.5.17 FIOCHANNELINFO

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 (TDRV002_CHANNEL_INFO_BUFFER) where the information will be filled in.

```
typedef struct
{
    int                         channelNo
    struct tdrv002_board_info   board;
    struct tdrv002_controller_info  controller;
} TDRV002_CHANNEL_INFO_BUFFER;
```

channelNo

This value returns the serial channel number. For example: the function will return 7 for “/tdrv002/7”.
The legacy driver will return the number at the end of the device name as described above, but if there is no number at the end of the device (e.g. “/ser/A”), the function will return a driver internal channel count.

board

This structure (struct tdrv002_board_info) contains board information that belongs to a specified channel.

```
struct tdrv002_board_info
{
    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. The define of the boardId consists of TDRV002_CHANINFO_ and xxx where xxx is replaced by the name of the board.

The list below shows some examples of the returned board IDs and board types, defined in tdrv002.h:

Board Id	Board Type
TDRV002_CHANINFO_TPMC371	TPMC371-xx
TDRV002_CHANINFO_TPMC372	TPMC372-xx
TDRV002_CHANINFO_TPMC375	TPMC375-xx
…	…
TDRV002_CHANINFO_TCP460	TCP460-xx
TDRV002_CHANINFO_TCP461	TCP461-xx
…	…
TDRV002_CHANINFO_TXMC375	TXMC375-xx
…	…

boardVariant

This value returns the board variant. The returned number specified the *xx* in the board name, e.g. TPMC461-*xx*.

boardIndex

This value returns the index of the specified board. If just one TDRV002 board is used, this index will always be 0, but if more than one TDRV002 board with the same boardId 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 tdrv002_controller_info) contains information that belongs to the controller and the specified channel which describes the location of the controller and channel on PCI-bus.

```
struct tdrv002_controller_info
{
    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.

controllerPort

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

EXAMPLE

```
#include "tdrv002.h"

int                             fd;
int                             retval;
TDRV002_CHANNEL_INFO_BUFFER     channelInfo

/*----------------------
  Read board information
  ----------------------*/
retval = ioctl(fd, FIOCHANNELINFO, (TDRV002_IOCTL_ARG_T)&channelInfo);
if (result != ERROR)
{
     printf("Get Channel Board Information successfully executed\n");
     printf("    Channel-Name: %s%d\n", TDRV002_DEVICENAME,
          channelInfo.channelNo);

     printf("    Board: %s%d-%02d - Board Index: %d\n",
          boardTypeList[(channelInfo.board.boardId >> 12) & 0xF],
          channelInfo.board.boardId & 0xFFF,
          channelInfo.board.boardVariant,
          channelInfo.board.boardIndex);
     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 */
}
```

4 Appendix

4.1 Driver Property Functions (global)

There are two functions implemented to allow applications being compatible to different VxWorks Version and different Project settings. These functions are global and may be used by the application, except in RTPs, where the symbols are not available.

4.1.1 tdrv002GetDriverType

NAME

tdrv002GetDriverType() – get VxWorks driver type

SYNOPSIS

```
int tdrv002GetDriverType
(
        void
)
```

DESCRIPTION

This function returns the type of the device driver.

EXAMPLE

```
include <tdrv002.h>

switch (tdrv002GetDriverType())
{
case TDRV002_VXBUS_DRV:
     /* Handle stuff, necessary for VxBus enabled driver */
     break;
case TDRV002_LEGACY_DRV:
     /* Handle stuff, necessary for Legacy driver */
     /* e.g. calling tdrv002() and tdrv002DevCreate() for driver start */
     break;
default:
     /* unknown driver type */
     break;
}
```

RETURNS

The driver type will be returned.

Driver Type	Description
TDRV002_VXBUS_DRV	VxBus driver is used The driver will be started and devices will be created automatically on system start-up.
TDRV002_LEGACY_DRV	Legacy driver is used The driver must be started manually and devices must be created by the application.

4.1.2 tdrv002GetDeviceNamePrefix

NAME

tdrv002GetDeviceNamePrefix() – get the device name prefix

SYNOPSIS

```
STATUS tdrv002GetDeviceNamePrefix
(
    char*       name,
    int         len
)
```

DESCRIPTION

This function returns the device name prefix for all devices of created by the TDRV002. The returned name matches the definition TDRV002_DEV_NAME parameter. (See also 2.1.1Assignment of Port Names.)

PARAMETER

name

This argument points to a character array, where the device name prefix will be stored to. A null-terminated string will be returned.

len

This parameter specifies the length of the name array. If len is smaller than the buffer length, the function will return the begin of the device name prefix.

EXAMPLE

```
include <tdrv002.h>

char       deviceNamePrefix[30];
char       deviceName[50];
int        devHandle[2];

if (tdrv002GetDeviceNamePrefix(deviceNamePrefix, 30) == ERROR)
{
     /* deviceNamePrefix must be known and set by application */
     …
}

/* open first and second device */
for (i=0, i<2; i++)
{
     sprintf(deviceName, "%s%d", deviceNamePrefix. i);
     devHandle[i] = open(deviceName,0,0);
     …
}
```

RETURNS

OK if a string is returned or ERROR.

4.1.3 tdrv002GetValidDeviceNumbers

NAME

tdrv002GetValidDeviceNumbers() – get the range of valid device numbers

SYNOPSIS

```
STATUS tdrv002GetValidDeviceNumbers
(
    int         *first,
    int         *last
)
```

DESCRIPTION

This function returns the first and the last device number of the device name. The returned values can be used to open just valid device. (TDRV002 device names are build liken this: <device prefix><device number>)

PARAMETER

first

This argument returns the number of the first device.

last

This argument returns the number of the last device.

EXAMPLE

```
include <tdrv002.h>

int         firstDevNo;
int         lastDevNo;

if (tdrv002GetValidDeviceNumbers(&firstDevNo, &lastDevNo) == ERROR)
{
     /* no valid devices */
     …
}
```

RETURNS

OK if valid device exist or ERROR if there is no valid device.

4.2 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.

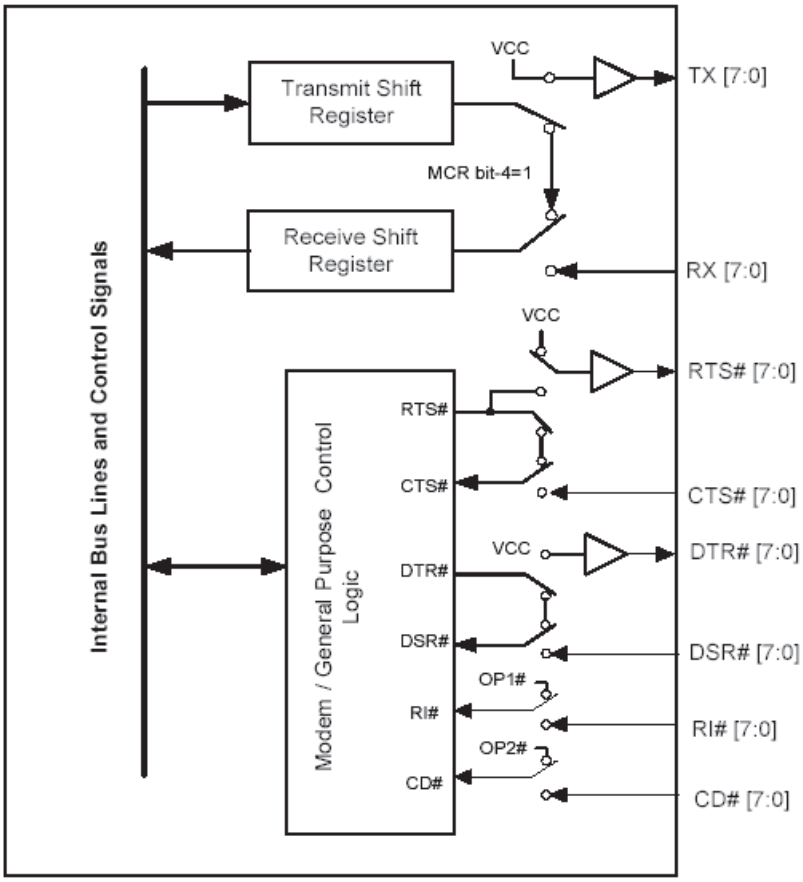
Increasing the receive FIFO-trigger-level will lower the number of generated interrupts, but it will also increase the execution time of a single interrupt function and it may increase the risk of losing data by FIFO overrun.

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

4.3 Internal Loopback

The internal loopback mode connects output lines with input lines of the corresponding channel. This allows testing the software and general board access without any external wiring.

If internal loopback is enabled, all I/O lines can be used regardless if they are supported by board I/O or not.



4.4 Debugging Driver and Devices

Driver start-up of the TDRV002 is mainly executed at a time which does not allow the output debug messages. Therefore we have implemented a function that displays some information about driver and devices.

```
void tdrv002Show
(
      void
)
```

The function is only available if the VxBus driver is used.

The function will display the following information:
- start up errors (if occurred)
- list of boards which has been probed by the driver and position of the boards on PCI bus
- list of created TDRV002 devices and assignment to the boards
- some statistics to the TDRV002 devices and their configuration

For example, the function can be called from the VxWorks shell to display the information.

Below is an example output for installed TXMC375-10 and TPMC461-12:

```
-> tdrv002Show
Probed Modules:
   [0] TXMC375: Bus=3, Dev=0, DevId=0x9177, VenId=0x1498, Init=OK, vxDev=0x80040df0
   [1] TPMC461: Bus=4, Dev=2, DevId=0x01cd, VenId=0x1498, Init=OK, vxDev=0x800fd6d0

Associated Devices:
    [0] TXMC375:
        Channel 0:    /tdrv002/0
        Channel 1:    /tdrv002/1
        Channel 2:    /tdrv002/2
        Channel 3:    /tdrv002/3
        Channel 4:    /tdrv002/4
        Channel 5:    /tdrv002/5
        Channel 6:    /tdrv002/6
        Channel 7:    /tdrv002/7
    [1] TPMC461:
        Channel 0:    /tdrv002/8
        Channel 1:    /tdrv002/9
        Channel 2:    /tdrv002/10
        Channel 3:    /tdrv002/11
        Channel 4:    /tdrv002/12
        Channel 5:    /tdrv002/13
        Channel 6:    /tdrv002/14
        Channel 7:    /tdrv002/15
```

```
Device Statistics:
    /tdrv002/0:
        Interrupt Count     = 0
        Transmitted Chars   = 0
        Received Chars      = 0
            (Receive-FIFO max. use: 0 (of 256))
        Error Count         = 0
        Configuration:
            Interface       = OFF (programmable)
    /tdrv002/1:
        Interrupt Count     = 0
        Transmitted Chars   = 0
        Received Chars      = 0
            (Receive-FIFO max. use: 0 (of 256))
        Error Count         = 0
        Configuration:
            Interface       = OFF (programmable)

…     (continues for /tdrv002/2 to /tdrv002/7)

    /tdrv002/8:
        Interrupt Count     = 0
        Transmitted Chars   = 0
        Received Chars      = 0
            (Receive-FIFO max. use: 0 (of 64))
        Error Count         = 0
        Configuration:
            Interface       = RS232
            max. Baudrate   = 1000000
            Baudrate        = 9600
            Data/Stopbits   = 8/1
            Parity          = None
            Local-Loopback  = disabled
    /tdrv002/9:
        Interrupt Count     = 0
        Transmitted Chars   = 0
        Received Chars      = 0
            (Receive-FIFO max. use: 0 (of 64))
        Error Count         = 0
        Configuration:
            Interface       = RS232
            max. Baudrate   = 1000000
            Baudrate        = 9600
            Data/Stopbits   = 8/1
            Parity          = None
            Local-Loopback  = disabled

…     (continues for /tdrv002/10 to /tdrv002/11)
```

```
/tdrv002/12:
    Interrupt Count     = 0
    Transmitted Chars   = 0
    Received Chars      = 0
        (Receive-FIFO max. use: 0 (of 64))
    Error Count         = 0
    Configuration:
        Interface       = RS422
        max. Baudrate   = 10000000
        Baudrate        = 9600
        Data/Stopbits   = 8/1
        Parity          = None
        Local-Loopback  = disabled

…    (continues for /tdrv002/13 to /tdrv002/15)
```