

TDRV002-SW-82

Linux Device Driver

Multi-Channel Serial Interface

Version 1.4.x

User Manual

Issue 1.4.3

April 2010

TDRV002-SW-82

Linux Device Driver

Multi-Channel Serial Interface

Supported Modules:

TPMC371
 TPMC372
 TPMC375
 TPMC376
 TPMC460
 TPMC461
 TPMC462
 TPMC463
 TPMC465
 TPMC466
 TPMC467
 TCP460
 TCP461
 TCP462
 TCP463
 TCP465
 TCP466
 TCP467

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.

©2005-2010 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0.0	First Issue	February 21, 2005
1.1.0	Built-In-Self-Test (BIST) added	March 11, 2005
1.1.1	depmod for driver installation added	October 13, 2005
1.2.0	New module support and transceiver programming IOCTL added. ChangeLog.txt release history file added, file list modified.	July 26, 2006
1.2.1	New Address TEWS LLC	November 7, 2006
1.3.0	New IOCTL command TDRV002_IOCTL_SPEED added	January 10, 2007
1.4.0	New IOCTL commands TDRV002_IOCQ_GET_SPEED, New IOCTL commands TDRV002_IOCQ_GET_INFO, example file added to file list	March 01, 2007
1.4.1	"TDRV002" device naming note added Source file archive extraction command line added	June 20, 2007
1.4.2	File list changed, include path moved	September 26, 2007
1.4.3	Address TEWS LLC removed, general Revision	April,27, 2010

Table of Contents

1	INTRODUCTION.....	4
2	INSTALLATION.....	6
	2.1 Build and install the device driver.....	7
	2.2 Uninstall the device driver	7
	2.3 Install device driver into the running kernel	7
	2.4 Remove device driver from the running kernel	8
	2.5 Change Major Device Number	8
3	DEVICE DRIVER PROGRAMMING	9
	3.1 Simple Programming example	9
	3.2 Using special baudrates.....	10
	3.2.1 Special baudrates set via termios	10
	3.3 ioctl()	11
	3.3.1 TDRV002_IOCQ_BIST	13
	3.3.2 TDRV002_IOCTL_CONF_TRANS	16
	3.3.3 TDRV002_IOCTL_SPEED.....	18
	3.3.4 TDRV002_IOCQ_GET_SPEED	19
	3.3.5 TDRV002_IOCQ_GET_INFO	20
4	TDRV002CONFIG – COMMAND LINE TOOL	22
5	DIAGNOSTIC.....	23

1 Introduction

The TDRV002-SW-82 Linux device driver is a full-duplex serial driver which allows the operation of a supported serial PMC on Linux operating systems.

The TDRV002-SW-82 device driver based on the standard Linux serial device driver and supports all standard terminal functions (TERMIOS).

Supported features:

- Extended baudrates up to 5.5296 Mbaud.
- Each channel has a 64 Byte transmit and receive hardware FIFO
- Programmable trigger level for transmit and receive FIFO.
- Hardware (RTS/CTS) and software flow control (XON/XOFF) direct controlled by the serial controller. The advantage of this feature is that the transmission of characters will immediately stop as soon as a complete character is transmitted and not when the transmit FIFO is empty for handshake under software control. This will greatly improve flow control reliability.
- Direct support of different physical interfaces (e.g. RS-232, RS-422).
- Designed as Linux kernel module with dynamically loading.
- Supports shared IRQ's.
- Build on new style PCI driver layout
- Creates a TTY device ttyTDRV002 and dialout device cuaTDRV002 (Kernel 2.4.x) with dynamically allocated or fixed major device numbers.
- DEVFS and UDEV support for automatic device node creation

The TDRV002-SW-82 device driver supports the modules listed below:

TPMC371	8 Channel Serial Interface	PMC Conduction Cooled
TPMC372	4 Channel Serial Interface	PMC Conduction Cooled
TPMC375	8 Channel Serial Interface	PMC Conduction Cooled
TPMC376	4 Channel Serial Interface	PMC Conduction Cooled
TPMC460	16 Channel Serial Interface	PMC
TPMC461	8 Channel Serial Interface	PMC
TPMC462	4 Channel Serial Interface	PMC
TPMC463	4 Channel Serial Interface	PMC
TPMC465	8 Channel Serial Interface	PMC
TPMC466	4 Channel Serial Interface	PMC
TPMC467	4 Channel Serial Interface	PMC
TCP460	16 Channel Serial Interface	CompactPCI
TCP461	8 Channel Serial Interface	CompactPCI
TCP462	4 Channel Serial Interface	CompactPCI
TCP463	4 Channel Serial Interface	CompactPCI
TCP465	8 Channel Serial Interface	CompactPCI
TCP466	4 Channel Serial Interface	CompactPCI
TCP467	4 Channel Serial Interface	CompactPCI

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 the supported devices it is recommended to read the manuals listed below.

TPMC37x, TPMC46x and TCP46x Hardware User manual

TPMC37x, TPMC46x and TCP46x Engineering Manual

Exar XR17D15x PCI UART User Manual

2 Installation

The directory TDRV002-SW-82 on the distribution media contains the following files:

TDRV002-SW-82-1.4.3.pdf	This manual in PDF format
TDRV002-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

The GZIP compressed archive TDRV002-SW-82-SRC.tar.gz contains the following files and directories:

hal/	Hardware abstraction layer driver needed for all kernel versions
hal/Makefile	HAL driver makefile
hal/tdrv002hal.c	HAL driver source file
hal/tdrv002haldef.h	HAL driver private header file
serial/	UART driver directory
serial/2.4.x	Kernel 2.4.x sources directory
serial/2.4.x/Makefile	Serial driver makefile
serial/2.4.x/tdrv002serial.c	Serial driver source file
serial/2.4.x/tdrv002serialdef.h	Serial driver private header file
serial/2.6.x	Kernel 2.6.x sources directory
serial/2.6.x/Makefile	Serial driver makefile
serial/2.6.x/tdrv002serial.c	Serial driver source file
serial/2.6.x/tdrv002serialdef.h	Serial driver private header file
serial/makenode	Shell script to create devices nodes without a device FS
serial/makenodeFM24	Same as makenode with additional support for CUA devices
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
include/config.h	Driver independent library header file
include/tpxxxhwdep.c	HAL low level WINNT style hardware access functions source file
include/tpxxxhwdep.h	Access functions header file
example/Makefile	Example application makefile
example/tdrv002example.c	Send and receive example application
example/tdrv002setspeed.c	Speed configuration example application
example/tdrv002bist.c	Example for using Built-In-Self-Test
example/tdrv002config.c	Command-Line Tool for transceiver programming
example/tdrv002readinfo.c	Example displays hardware information of a channel
tdrv002.h	Driver header file
tdrv002user.h	User application header file

In order to perform an installation, extract all files of the archive TDRV002-SW-82-SRC.tar.gz to the desired target directory. (Note: to extract the archive file use # `tar -xvzf TDRV002-SW-82-SRC.tar.gz`)

- Login as *root* and change to the target directory
- Copy `tdrv002user.h` to `/usr/include`

2.1 Build and install the device driver

- Login as *root*
- Change to the *hal/* target directory
- To create and install the HAL driver in the module directory */lib/modules/<version>/misc* enter:

```
# make install
```

- Change to the *serial/<version>* target directory
- To create and install the SERIAL driver in the module directory */lib/modules/<version>/misc* enter:

```
# make install
```

For Linux kernel 2.6.x, there may be compiler warnings claiming some undefined `tdrv002_hal_*` symbols. These warnings are caused by the HAL driver, which is unknown during compilation of this SERIAL driver. The warnings can be ignored.

- To update the device driver's module dependencies, enter:

```
# depmod -aq
```

2.2 Uninstall the device driver

- Login as *root*
- Change to the target directory
- To remove the driver from the module directory */lib/modules/<version>/misc* enter:

```
# make uninstall
```

2.3 Install device driver into the running kernel

- To load the device driver into the running kernel, login as *root* and execute the following commands:

```
# modprobe tdrv002serialdrv
```

- After the first build or if you are using dynamic major device allocation it's necessary to create new device nodes on the file system. Please execute the script file *makenode*, which resides in *serial/* directory, to do this. If your kernel has enabled the device file system (*devfs*, *udev*, ...) then skip running the *makenode* script. Instead of creating device nodes from the script the driver itself takes creating and destroying of device nodes in its responsibility.

```
# sh makenode
```

On success the device driver will create a minor device for each compatible channel found. The first channel of the first PMC module can be accessed with device node */dev/ttySTDRV002_0*, the second channel with device node */dev/ttySTDRV002_1* and so on. The assignment of device nodes to physical PMC modules depends on the search order of the PCI bus driver.

2.4 Remove device driver from the running kernel

- To remove the device driver from the running kernel login as root and execute the following command:

```
# modprobe -r tdrv002serialdrv
```

If your kernel has enabled a device file system (devfs, udev, ...), all `/dev/ttySTDRV002_*` nodes will be automatically removed from your file system after this.

Be sure that the driver isn't opened by any application program. If opened you will get the response "*tdrv002serialdrv: Device or resource busy*" and the driver will still remain in the system until you close all opened files and execute *modprobe -r* again.

2.5 Change Major Device Number

This paragraph is only for Linux kernels without a device file system (devfs, udev, ...) installed.

The released TDRV002 driver use dynamic allocation of major device numbers. If this isn't suitable for the application it's possible to define a major number separately for the *TTY* and *CUA* driver.

To change the major number edit the file `serial/<version>/tdrv002serial.c`, change the following symbols to appropriate values and enter *make install* to create a new driver.

TDRV002_TTY_MAJOR	Defines the value for the terminal device. Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.
TDRV002_CUA_MAJOR	Defines the value for the dialout device. Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TDRV002_TTY_MAJOR      122
#define TDRV002_CUA_MAJOR     123
```

Be sure that the desired major number isn't used by other drivers. Please check `/proc/devices` to see which numbers are free.

Keep in mind that's necessary to create new device nodes if the major number for the TDRV002 driver has changed and the `makenode` script isn't used.

3 Device Driver Programming

The TDRV002-SW-82 driver loosely bases on the standard Linux terminal driver. Due to this way of implementation the driver interface and functionality is compatible to the standard Linux terminal driver.

Please refer to the TERMIOS man page and driver programming related man pages for more information about serial driver programming.

3.1 Simple Programming example

This example program opens the first serial channel of a TDRV002 compatible device for read/write. After the device is open it writes a "Hello World" string to the device and receives up to 80 bytes from the serial channel.

```
int main()
{
    int fd;
    int count;
    char buffer[81];

    /* open the desired PMC device channel*/
    fd = open( "/dev/ttySTDRV002_0", O_RDWR | O_NOCTTY);

    if (fd < 0) exit(-1);

    /* write data to the certain channel */
    count = write(fd, "Hello World\n", 12);
    printf("%d bytes written\n", count);

    /* read up to 80 bytes from the device */
    count = read(fd, buffer, 80)
    if (count < 0) {
        printf("read error\n");
    }
    else {
        buffer[count] = 0;
        printf("%d bytes read <%s>\n", count, buffer);
    }

    close(fd);
}
```

The source files *tdrv002example.c* and *tdrv002setspeed.c* contains additional programming examples.

3.2 Using special baudrates

There are two possibilities setting up special baudrates. The first is used to setup some predefined baudrates, this is the standard way by using the termios structure. The second way allows the selection of all baud rates the module can support. This way uses the ioctl function `TDRV002_IOCTL_SPEED`. (Please refer to the description of the ioctl function).

3.2.1 Special baudrates set via termios

Some of the supported modules allow very high non standard baud rates. To use these baudrates with the standard Linux IOCTL functions regard the following section.

The file “tdrv002user.h” defines some special baudrates.

Define	Value	Description
<code>TDRV002_BUSER</code>	<code>B1000000</code>	Default value for baudrate definition
<code>TDRV002_B1382400</code>	<code>TDRV002_BUSER</code>	Select 1382400 Baud
<code>TDRV002_B2764800</code>	<code>(TDRV002_BUSER+1)</code>	Select 2764800 Baud
<code>TDRV002_B5529600</code>	<code>(TDRV002_BUSER+2)</code>	Select 5529600 Baud

By overriding some standard baudrates, you can easily use it with a termios structure. To setup speed e.g. to 5529600 BAUD use the following code:

```
newtermios.c_cflag = (oldtermios.c_cflag & ~CBAUD) | TDRV002_B5529600;
tcsetattr(<your-device-handle>, TCSANOW, &newtermios);
```

You can also use the following, it does the same:

```
newtermios.c_cflag = (oldtermios.c_cflag & ~CBAUD) | B1500000;
tcsetattr(<your-device-handle>, TCSANOW, &newtermios);
```

But it is recommended to use the first solution for a better readability.

3.3 ioctl()

NAME

ioctl() device control functions

SYNOPSIS

```
#include <sys/ioctl.h>
#include <tdrv002.h>
#include <tdrv002user.h>
```

```
int ioctl(int fildes, int request [, void *argp])
```

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *fildes*, causing the corresponding device to perform the requested operation. The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *tdrv002user.h*:

Value	Meaning
TDRV002_IOCQ_BIST	Start Built-In-Self-Test
TDRV002_IOCTL_CONF_TRANS	Configure transceiver (physical interface)
TDRV002_IOCTL_SPEED	Setup user defined baudrates
TDRV002_IOCQ_GET_SPEED	Returns the current configured baudrate
TDRV002_IOCQ_GET_INFO	Reads out hardware information of a channel

See below for more detailed information on each control code.

To use these TDRV002 specific control codes the header file *tdrv002user.h* must be included in the application.

RETURNS

On success, zero is returned. In case of an error, a value of -1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EINVAL Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument *request*.

Other function dependant error codes will be described for each ioctl code separately. Note, the TDRV002 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TDRV002_IOCQ_BIST

NAME

TDRV002_IOCQ_BIST – Start Built-In-Self-Test

DESCRIPTION

The TDRV002 driver (version 1.1.0 and higher) supports a special IOCTL function for testing module hardware and for system diagnostic. The optional argument can be omitted for this ioctl function.

The functionality is called Built-In-Self-Test or BIST. With BIST you can test each channel of all your modules separately. There are three different test classes. First is a line test, second an interrupt test and the last a data integrity test. All tests run with local channel loopback enabled, so you don't need an external cable connection. The Fig. 3-1 describes the loop back configuration of a XR17D158 8 channel UART, so all line arrays are index with [7:0]. For the two and four channel UARTs XR17D152 and XR17D154 line arrays should be indexed with [1:0] or [3:0].

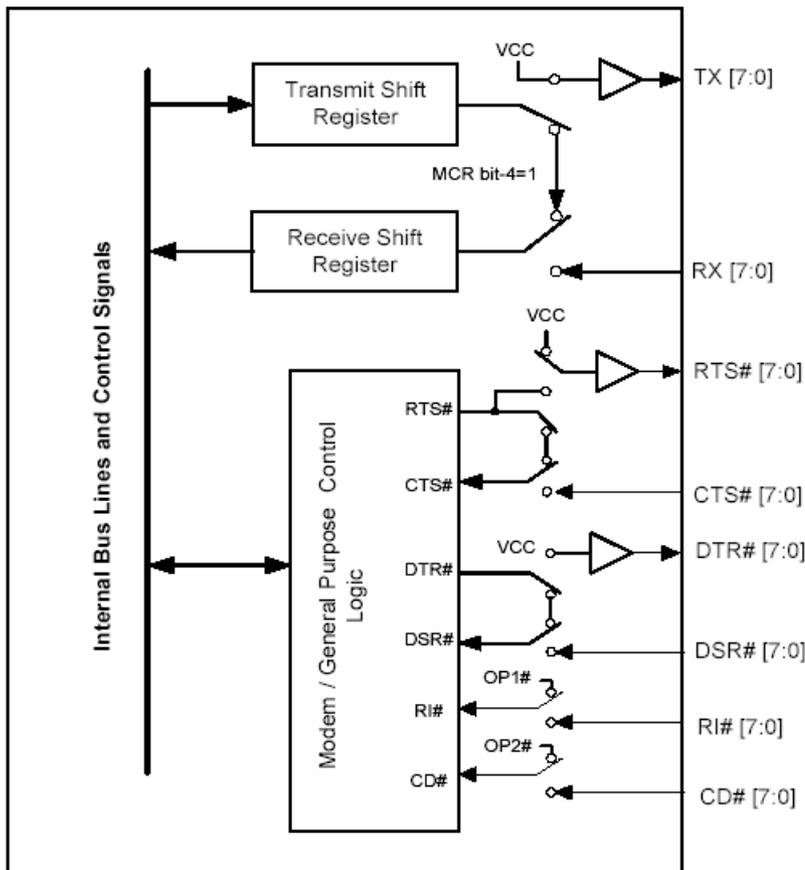


Fig. 3-1

The line test contains a test of all modem lines, as you can see RTS and CTS, DTR and DSR, OP1 and RI and finally OP2 and CD. Only the static states for both electrical levels are tested on each sender – receiver line pair.

For testing interrupts the BIST transmits a test buffer with known data and size. All data should be received on same channel during internal loopback. If not, there is an interrupt error. The buffer size is 1024 BYTE. The baudrate has to be set through the standard terminal IOCTL functions.

The last test verifies received data to assert data integrity.

EXAMPLE

```
#include <tdrv002user.h>

int result, tty1;

/* Start Built-In Selftest, */
result = ioctl(tty1, TDRV002_IOCQ_BIST, NULL);

if(result) {
    printf("Error during Built-In Selftest <%d, 0x%08X>!\n", result, result);
}
if (result < 0) {
    printf("ERRNO %d - %s\n", errno, strerror(errno));
}
else if (result > 0) {
    if (result & TDRV002_ERTSCTS)
        printf("RTS/CTS line broken!\n");
    if (result & TDRV002_EDTRDSR)
        printf("DTR/DSR line broken!\n");
    if (result & TDRV002_ERI)
        printf("OP1/RI line broken!\n");
    if (result & TDRV002_ECD)
        printf("OP2/DCD line broken!\n");
    if (result & TDRV002_EDATA)
        printf("Data integrity test failed!\n");
}
else {
    printf("INFO: Port %s successfully tested.\n", DevName);
}
```

RETURNS

If return value is > 0 one of three tests failed. Use the following flags to get a detailed error description.

TDRV002_ERTSCTS	If set RTS/CTS line broken.
TDRV002_EDTRDSR	If set DTR/DSR line broken.
TDRV002_ERI	If set OP1/RI line broken.
TDRV002_ECD	If set OP2/CD line broken.
TDRV002_EDATA	Data integrity test failed. No correct transmission possible.

ERRORS

ETIME	A timeout occurred during wait, interrupts do not work correctly.
EAGAIN	Your task should never been blocked. Change it to use the Built-In-Self-Test.
ERESTARTSYS	Interrupted by external signal.

3.3.2 TDRV002_IOCTL_CONF_TRANS

NAME

TDRV002_IOCTL_CONF_TRANS – Configure transceiver

DESCRIPTION

This ioctl function configures the transceiver circuit of all TDRV002 modules with a programmable physical interface.

The configuration is passed by value by the parameter **arg** to the driver.

The flags below are available and should be ORed to build a configuration value:

TDRV002_CFG_RS485_RS232	Set to enable RS485 interface, clear to enable RS232 interface.
TDRV002_CFG_HDPLX	Set to enable half-duplex interface, clear to enable full-duplex interface.
TDRV002_CFG_RENA	Set to enable “auto RS485 receiver enable” feature, clear to disable it.
TDRV002_CFG_RTERM	Set to enable receiver termination, clear to disable it.
TDRV002_CFG_TTERM	Set to enable transmitter termination, clear to disable it.
TDRV002_CFG_SLEWLIMIT	Set to enable slew limit mode, clear to disable it.
TDRV002_CFG_SHDN	Set to shutdown the hole transceiver circuit, clear to enable the transceiver.
TDRV002_CFG_AUTO_RS485	Set to enable “UART Auto RS485 Mode”, clear to disable it. (See UART XR17D15x Hardware User Manual)

Beside the certain flags the tdrv002user.h header file offers a set of standard configurations that could be used alternatively. The following predefined macros could be used:

TDRV002_INTF_OFF	Shutdown mode / disable interface
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 <tdrv002user.h>

unsigned long config;
int result;
int tty1, tty2; /* device handles of modules with programmable
                 transceivers */

/* Setup channel as RS485 Full Duplex (Master)*/
config = TDRV002_CFG_RS485_RS232 |
        TDRV002_CFG_RTERM |
        TDRV002_CFG_TTERM;

result = ioctl(tty1, TDRV002_IOCTL_CONF_TRANS, config);

if (result < 0) {
    /* handle errors */
}

/* Setup channel as RS485 Full Duplex (Master) (alternative way) */
config = TDRV002_INTF_RS485FDM;

result = ioctl(tty1, TDRV002_IOCTL_CONF_TRANS, config);

if (result < 0) {
    /* handle errors */
}

/* Shutdown the physical interface of certain channel */
config = TDRV002_INTF_OFF;

result = ioctl(tty2, TDRV002_IOCTL_CONF_TRANS, config);

if (result < 0) {
    /* handle errors */
}
```

ERRORS

ENODEV

The selected device has no programmable physical interface. See Hardware User Manual for detailed information about programmable transceivers.

3.3.3 TDRV002_IOCTL_SPEED

NAME

TDRV002_IOCTL_SPEED – Setup user defined baudrates

DESCRIPTION

This ioctl function sets up a user defined baudrate. This allows using the TDRV002 device with every adjustable baudrate.

The new baudrate is passed by value by the parameter **arg** to the driver. The baudrate limits are device and configuration dependent therefore refer to the suitable manual.

The function tries to set the baudrate in “X16-mode”, if the nearest configurable baudrate has a difference greater than 3% to the chosen one, the driver will setup the baudrate in “X8-mode”.

If a user defined baudrate is defined, standard tools will return invalid information about the selected baudrate.

EXAMPLE

```
#include <tdrv002user.h>

int result, tty1;

/* Setup 14400 Baud */
result = ioctl(tty1, TDRV002_IOCTL_SPEED, 14400);

if (result < 0) {
    /* handle errors */
}
```

3.3.4 TDRV002_IOCQ_GET_SPEED

NAME

TDRV002_IOCQ_GET_SPEED – Read the current configured baudrate

DESCRIPTION

This ioctl function returns the currently configured baudrate of the specified channel. This allows checking if a baudrate can be configured correctly or if it is substituted by the nearest configurable baudrate.

The current baudrate is returned in the integer argument the parameter **arg** points on.

EXAMPLE

```
#include <tdrv002user.h>

int result, tty1, baudrate;

result = ioctl(tty1, TDRV002_IOCQ_GET_SPEED, &baudrate);

if (result < 0) {
    /* handle errors */
}
else {
    printf("Current Baudrate: %d\n", baudrate);
}
```

3.3.5 TDRV002_IOCQ_GET_INFO

NAME

TDRV002_IOCQ_GET_INFO – Reads information about the position and type of a channel

DESCRIPTION

This ioctl function reads the module position, module ID and the local channel number of a specified channel. This information may allow an easier module identification and configuration checking in the system.

A pointer to the information buffer (TDRV002_GET_INFO_STRUCT) is passed by the parameter *arg* to the driver

```
typedef struct
{
    int    pciBusNo;
    int    pciParentBusNo;
    int    pciDeviceNo;
    int    localChannelNo;
    int    deviceId;
    int    subSystemId;
    char   typeStr[20];
} TDRV002_GET_INFO_STRUCT;
```

pciBusNo

Returns the PCI bus number the UART is mounted. Some TDRV002 supported modules have their own PCI bridge in this case the value is the number of the local PCI bus on the module.

pciParentBusNo

Returns the PCI bus number of the parent PCI bus. This value may be interesting if a module type with an own PCI bridge is used. If there is no parent PCI bus, the value will be -1.

pciDeviceNo

Returns the PCI device number the UART controller. This specifies a fix place on the PCI bus and may be used to identify a special module. The value returns the PCI device number of the UART not that one of the TDRV002 supported module.

localChannelNo

Returns the local channel number of the specified device. The first channel on a module starts with 0, the second is 1 and so on.

deviceld

Returns the PCI device ID, this identifies the model type.

subSystemId

Returns the PCI Subsystem ID, this identifies the model variant.

typeStr[]

Returns a string with the product name, e.g. TPMC461-12 or TCP462-10

EXAMPLE

```
#include <tdrv002user.h>

int result, tty1;
TDRV002_GET_INFO_STRUCT infoBuf;

/* Display channel position and Moduletype */
result = ioctl(tty1, TDRV002_IOCQ_GET_INFO, &infoBuf);

if (result < 0) {
    printf("Device: %d/%d/%d: %s\n",
        infoBuf.pciBusNo,
        infoBuf.pciDeviceNo,
        infoBuf.localChannelNo,
        infoBuf.typeStr);
}
```

4 tdrv002config – Command Line Tool

To setup the physical interface of a certain channel you can use `example/tdrv002config` for programming of the transceiver circuit.

format : `tdrv002config <minor1> <options>`

example: `tdrv002config 0 crs485 crterm`

configures `/dev/ttySTDRV002_0` to RS422 full duplex master

List of all options:

- `crs485`
- `chdplx`
- `crena`
- `crterm`
- `ctterm`
- `cslewlimit`
- `cshdn`
- `cautors485`

For detailed configuration options information see `TDRV002_IOCTL_CONF_TRANS` ioctl function description.

5 Diagnostic

If the TDRV002 driver does not work properly it is helpful to get some status information from the driver respective kernel.

The Linux `/proc` file system provides information about kernel, resources, driver, devices and so on. The following screen dumps displays information of a correct running TDRV002 driver (see also the `proc` man pages).

```
# cat /proc/tty/driver/tdrv002serial
TEWS TECHNOLOGIES - TDRV002 Generic UART driver (Kernel 2.4.x): <Version>
revision: <Revsision Date>
0: uart:XR17D15X port:D08E8000 irq:10 tx:0 rx:0
1: uart:XR17D15X port:D08E8200 irq:10 tx:0 rx:0
2: uart:XR17D15X port:D08E8400 irq:10 tx:0 rx:0
3: uart:XR17D15X port:D08E8600 irq:10 tx:0 rx:0
4: uart:XR17D15X port:D08E8800 irq:10 tx:0 rx:0
5: uart:XR17D15X port:D08E8A00 irq:10 tx:0 rx:0
6: uart:XR17D15X port:D08E8C00 irq:10 tx:0 rx:0
7: uart:XR17D15X port:D08E8E00 irq:10 tx:0 rx:0
8: uart:XR17D15X port:D08EA000 irq:11 tx:0 rx:0
9: uart:XR17D15X port:D08EA200 irq:11 tx:0 rx:0
10: uart:XR17D15X port:D08EA400 irq:11 tx:0 rx:0
11: uart:XR17D15X port:D08EA600 irq:11 tx:0 rx:0
...
```

```
# cat /proc/tty/drivers
tdrv002serial      /dev/cua/cuaTDRV002_ 253  0-127 serial:callout
tdrv002serial      /dev/tts/ttySTDRV002_ 254  0-127 serial
serial             /dev/cua/%d          5   64-127 serial:callout
serial             /dev/tts/%d          4   64-127 serial
pty_slave         /dev/pts/%d         143  0-255 pty:slave
pty_master        /dev/ptm            135  0-255 pty:master
pty_slave         /dev/pts/%d         142  0-255 pty:slave
pty_master        /dev/ptm            134  0-255 pty:master
pty_slave         /dev/pts/%d         141  0-255 pty:slave
pty_master        /dev/ptm            133  0-255 pty:master
pty_slave         /dev/pts/%d         140  0-255 pty:slave
pty_master        /dev/ptm            132  0-255 pty:master
pty_slave         /dev/pts/%d         139  0-255 pty:slave
pty_master        /dev/ptm            131  0-255 pty:master
pty_slave         /dev/pts/%d         138  0-255 pty:slave
pty_master        /dev/ptm            130  0-255 pty:master
pty_slave         /dev/pts/%d         137  0-255 pty:slave
pty_master        /dev/ptm            129  0-255 pty:master
pty_slave         /dev/pts/%d         136  0-255 pty:slave
pty_master        /dev/ptm            128  0-255 pty:master
```

```

pty_slave          /dev/pty/s%d      3    0-255 pty:slave
pty_master         /dev/pty/m%d      2    0-255 pty:master
/dev/vc/0          /dev/vc/0         4      0 system:vtmaster
/dev/ptmx          /dev/ptmx         5      2 system
/dev/console       /dev/console      5      1 system:console
/dev/tty           /dev/tty          5      0 system:/dev/tty
unknown           /dev/vc/%d       4    1-63 console

```

```
# cat /proc/interrupts
```

```

          CPU0
 0:      60160          XT-PIC  timer
 1:         6          XT-PIC  keyboard
 2:         0          XT-PIC  cascade
 8:         1          XT-PIC  rtc
10:         0          XT-PIC  ehci-hcd, tdrv002serial
11:      2635          XT-PIC  usb-uhci, usb-uhci, eth0, tdrv002serial
12:        49          XT-PIC  PS/2 Mouse
14:      7783          XT-PIC  ide0
NMI:         0
ERR:         0

```

```
# lspci -v
```

```
...
```

```
00:0f.0 Serial controller: TEWS Datentechnik GmBH: Unknown device 01cd (rev 02) (prog-if 02 [16550])
```

```
Subsystem: TEWS Datentechnik GmBH: Unknown device 000a
```

```
Flags: fast devsel, IRQ 10
```

```
Memory at cfffc000 (32-bit, non-prefetchable) [size=4K]
```

```
00:11.0 Serial controller: TEWS Datentechnik GmBH: Unknown device 01cf (rev 01) (prog-if 02 [16550])
```

```
Subsystem: TEWS Datentechnik GmBH: Unknown device 000a
```

```
Flags: fast devsel, IRQ 11
```

```
Memory at cfffd000 (32-bit, non-prefetchable) [size=2K]
```

```
...
```