The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV002-SW-82

Linux Device Driver

Multi-Channel Serial Interface

Version 1.8.x

User Manual

Issue 1.8.0

March 2013

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7 25469 Halstenbek, Germany

Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19

e-mail: info@tews.com www.tews.com

TDRV002-SW-82

Linux Device Driver

Multi-Channel Serial Interface

Supported Modules:

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

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_IOCT_SPEED added	January 10, 2007
1.4.0	New IOCTL command TDRV002_IOCQ_GET_SPEED, New IOCTL command 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
1.4.4	Chapter Installation modified	January 24, 2011
1.4.5	New supported boards added to list	August 1, 2011
1.5.0	New file list, diagnostic chapter modified	December 20, 2011
1.6.0	New IOCTL command TDRV002_IOCT_SET_FIFOTRIG	January 20, 2012
1.7.0	IOCTL command TDRV002_IOCQ_GET_INFO extended	September 20, 2012
1.8.0	New board type TXMC375 added, Description of allowed values changed in TDRV002_IOCT_SET_FIFOTRIG, Chapter “Special Baud Rates set via termios” removed, Description ”Setting up Baud Rates” modified	March 26, 2013

Table of Contents

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 is based on the standard Linux serial device driver and supports all standard terminal functions (TERMIOS).

Supported features:

- Extended baud rates up to 5.5296 Mbaud.
- Depending on the board, 64 Byte or 256 Byte transmit and receive hardware FIFO per channel
- Programmable trigger level for transmit and receive FIFO.
- Hardware (RTS/CTS) and software flow control (XON/XOFF) directly 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 dynamic loading.
- Supports shared IRQ's.
- Built on new style PCI driver layout
- Creates a TTY device ttyTDRV002 and dial out 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
TPMC377	4 Channel Isolated 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
TPMC470	4 Channel Isolated 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
TCP469	8 Channel Isolated Serial Interface	CompactPCI
TCP470	4 Channel Isolated Serial Interface	CompactPCI
TXMC375	8 Channel Serial Interface	XMC Conduction Cooled

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.

Corresponding Modules Hardware User Manual
Corresponding Modules Engineering Manual
Exar XR17D15x PCI UART or Exar XR17D35x PCIexpress UART User Manual

2 Installation

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

TDRV002-SW-82-1.8.0.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 (for Kernels 2.6.x and newer)
serial/Makefile	Serial driver makefile
serial/tdrv002serial.c	Serial driver source file
serial /tdrv002serialdef.h	Serial driver private header file
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/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
Makefile	Top-level Makefile

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 tdrv002 target directory
- To create and install the HAL driver and SERIAL driver in the module directory */lib/modules/<version>/misc* enter:

  # make install

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

  # **depmod -aq**

2.2 Uninstall the Device Driver

- Login as *root*
- Change to the tdrv002 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 uses 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 *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 dial out 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 Setting up Baud Rates

The driver allows setting all baud rates supported by the channel. Not only standard baud rates are supported, also special baud rates are supported. The driver will always try to set the best matching baud rate.

There are two possibilities setting up baud rates:

The first is used to setup predefined baud rates, this is the standard way by using the termios structure (e.g. using ssty).

The second way allows the selection of all baud rates the module can support. This way uses the ioctl function *TDRV002_IOCT_SPEED* (please refer to the description of the ioctl function).

3.2 ioctl

NAME

ioctl() device control functions

SYNOPSIS

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

int ioctl(*int filedes, int request [, void *argp])*

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *filedes*, 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_IOCT_CONF_TRANS	Configure transceiver (physical interface)
TDRV002_IOCT_SPEED	Setup user defined baud rates
TDRV002_IOCQ_GET_SPEED	Returns the current configured baud rate
TDRV002_IOCQ_GET_INFO	Reads out hardware information of a channel
TDRV002_IOCT_SET_FIFOTRIG	Configure FIFO trigger levels

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

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

Other function dependent 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

3.2.1 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 an 8 channel UART, so all line arrays are index with [7:0]. For the two and four channel UARTs, the line arrays should be indexed with [1:0] or [3:0].

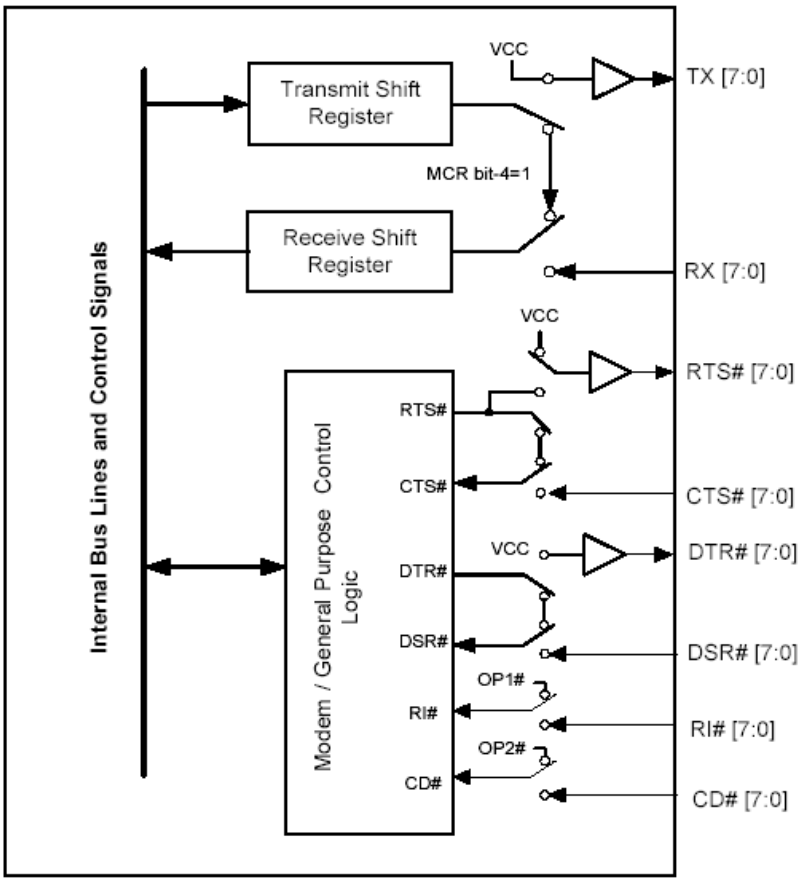

Fig. 3-1

The line test contains a test of all modem lines (RTS/CTS, DTR/DSR, OP1/RI, OP2/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 baud rate 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 < 0) {
   printf("ERRNO %d - %s\n", errno, strerror(errno));
}
else if (result > 0) {
   printf("BIST detected a line error!\n");
   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 successfully tested.\n");
}
```

RETURNS

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

Return Code	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

Error Code	Description
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.

3.2.2 TDRV002_IOCT_CONF_TRANS

NAME

TDRV002_IOCT_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:

Value	Meaning
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 whole 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:

Value	Meaning
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_IOCT_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_IOCT_CONF_TRANS, config);

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

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

result = ioctl(tty2, TDRV002_IOCT_CONF_TRANS, config);

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

ERRORS

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

3.2.3 TDRV002_IOCT_SPEED

NAME

TDRV002_IOCT_SPEED – Setup user defined baud rates

DESCRIPTION

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

The new baud rate is passed by value by the parameter ***arg*** to the driver. The baud rate limits are device and configuration dependent, so please refer to the suitable manual.

The function tries to set the baud rate in "X16-mode", if the nearest configurable baud rate has a difference greater than 3% to the chosen one, the driver will setup the baud rate in "X8-mode" or in"X4-mode", if supported by hardware.

If a user defined baud rate is set, standard tools (like *stty*) will return invalid information about the selected baud rate.

EXAMPLE

```
#include <tdrv002user.h>

int result, tty1;

/* Setup 14400 Baud */
result = ioctl(tty1, TDRV002_IOCT_SPEED, 14400);

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

3.2.4 TDRV002_IOCQ_GET_SPEED

NAME

TDRV002_IOCQ_GET_SPEED – Read the current configured baud rate

DESCRIPTION

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

The current baud rate 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);
}
```

3.2.5 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];
        int             intfProgrammable;
        unsigned char   intfConfig;
} 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 of 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.

deviceId

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

intfProgrammable

Returns TRUE (1) if the specific channel offers a programmable interface.

intfConfig

Returns the current transceiver interface configuration. For a description of this value refer to function TDRV002_IOCT_CONF_TRANS.

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);
  if (infoBuf.intfProgrammable)
  {
    printf("Interface configuration : %02Xh\n", infoBuf.intfConfig);
  } else {
    printf("Interface is not configurable.\n");
  }
}
```

3.2.6 TDRV002_IOCT_SET_FIFOTRIG

NAME

TDRV002_IOCT_SET_FIFOTRIG – Configure FIFO trigger levels

DESCRIPTION

This ioctl function configures the FIFO trigger levels for hardware receive and transmit FIFO. This allows optimizing interrupt load or data loss protection.

The new FIFO trigger level must be specified in structure (TDRV002_SET_FIFO_STRUCT). The pointer of the structure must be passed by the parameter ***arg*** to the driver.

```
typedef struct
{
        unsigned int        txFifoTrig;
        unsigned int        rxFifoTrig;
} TDRV002_SET_FIFO_STRUCT;
```

txFifoTrig

This value specifies the new FIFO trigger level, which specifies the number of characters left in the transmit FIFO when the controller will generate an interrupt announcing that there is space in the transmit FIFO to be filled with more data ready to be transmitted. Allowed values are 0...64 (for all TPMC and TCP boards) and 0…255 (for TXMC boards), but 0 specifies to use the default value of 16.

Increasing the value will increase interrupt load but the possibility of gaps in data transmission (while data is ready to send) will be decreased. Decreasing the value will decrease interrupt load but increase the risk of transmission gaps.

rxFifoTrig

This value specifies the new FIFO trigger level, which specifies the number of characters in the receive FIFO when the controller will generate an interrupt announcing that data should be read. Allowed values are 0...64 (for all TPMC and TCP boards) and 0…255 (for TXMC boards), but 0 specifies to use the default value of 48.

Increasing this value will decrease interrupt load, but will increase the risk of data loss, if hardware handshake is not used. Decreasing the value will increase interrupt load, but decrease the risk of data loss.

EXAMPLE

```
#include <tdrv002user.h>

int                          result, tty1;
TDRV002_SET_FIFO_STRUCT      fifoBuf;

/* Rx-FIFO-trigger: 32 */
/* Tx-FIFO-trigger: 40 */
fifoBuf.rxFifoTrig = 32;
fifoBuf.txFifoTrig = 40;
result = ioctl(tty1, TDRV002_IOCT_SET_FIFOTRIG, &fifoBuf);

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

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_IOCT_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 display information of a correct running TDRV002 driver (see also the proc man pages).

(The example output below has been created with kernel 3.1.5-2.fc16.x86_64 and an installed TPMC461.)

```
# cat /proc/tty/driver/tdrv002serial
serinfo:1.0 driver revision:
0: uart:XR17D15X mmio:0xFEB9F000 irq:16 tx:8192 rx:8192
1: uart:XR17D15X mmio:0xFEB9F200 irq:16 tx:1024 rx:1024 CTS
2: uart:XR17D15X mmio:0xFEB9F400 irq:16 tx:1024 rx:1024 CTS
3: uart:XR17D15X mmio:0xFEB9F600 irq:16 tx:1024 rx:1024
4: uart:XR17D15X mmio:0xFEB9F800 irq:16 tx:1024 rx:1024
5: uart:XR17D15X mmio:0xFEB9FA00 irq:16 tx:1024 rx:1024
6: uart:XR17D15X mmio:0xFEB9FC00 irq:16 tx:1024 rx:1024
7: uart:XR17D15X mmio:0xFEB9FE00 irq:16 tx:1024 rx:1024
…

# cat /proc/tty/drivers
/dev/tty         /dev/tty            5     0     system:/dev/tty
/dev/console     /dev/console        5     1     system:console
/dev/ptmx        /dev/ptmx           5     2     system
/dev/vc/0        /dev/vc/0           4     0     system:vtmaster
tdrv002serial    /dev/ttySTDRV002_ 250 0-127     serial
usbserial        /dev/ttyUSB       188 0-253     serial
serial           /dev/ttyS           4 64-95     serial
pty_slave        /dev/pts          136 0-1048575 pty:slave
pty_master       /dev/ptm          128 0-1048575 pty:master
unknown          /dev/tty            4 1-63      console

# lspci –v
…
04:01.0 Serial controller: TEWS Technologies GmbH Device 01cd (rev 02)
  (prog-if 02 [16550])
        Subsystem: TEWS Technologies GmbH Device 000c
        Flags: fast devsel, IRQ 16
        Memory at feb9f000 (32-bit, non-prefetchable) [size=4K]
        Kernel driver in use: TEWS TECHNOLOGIES - TDRV002HAL Driver
        Kernel modules: tdrv002haldrv
…
```