The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV009-SW-82

Linux Device Driver

High Speed Synch/Asynch Serial Interface

Version 1.0.x

User Manual

Issue 1.0.2

August 2010

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

TDRV009-SW-82

Linux Device Driver

High Speed Synch/Asynch Serial Interface

Supported Modules:
TPMC363
TPMC863
TCP863
TAMC863

Issue	Description	Date
1.0.0	First Issue	April 23, 2007
1.0.1	UDEV description added, syntax errors corrected	August 08, 2007
1.0.2	Address TEWS LLC removed, General Revision	August 19, 2010

Table of Contents

1 Introduction

The TDRV009-SW-82 Linux device driver allows the operation of the TDRV009 compatible devices conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close(), read(), write()* and *ioctl()* functions.

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 TDRV009-SW-82 device driver supports the following features:

- setup and configure serial channels
- send and receive data buffers (character oriented)
- read and write onboard registers directly
- read and write access to onboard EEPROM
- control handshake lines
- wait for interrupts
- Creates devices with dynamically allocated or fixed major device numbers
- DEVFS and SYSFS (UDEV) support for automatic device node creation

The TDRV009-SW-82 supports the modules listed below:

TPMC863	4 Channel High Speed Synch/Asynch Serial Interface	PMC
TPMC363	4 Channel High Speed Synch/Asynch Serial Interface	PMC, Conduction Cooled
TCP863	4 Channel High Speed Synch/Asynch Serial Interface	CompactPCI
TAMC863	4 Channel High Speed Synch/Asynch Serial Interface	Advanced Mezzanine Card

In this document all supported modules and devices will be called TDRV009. Specials for a certain device will be advised.

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

TPMC863 (or compatible) User manual
TPMC863 (or compatible) Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV009-SW-82':

TDRV009-SW-82-1.0.2.pdf	This manual in PDF format
TDRV009-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
ChangeLog.txt	Release history
Release.txt	Release information

The GZIP compressed archive TDRV009-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tdrv009/':

tdrv009.c	TDRV009 device driver source
tdrv009def.h	TDRV009 driver include file
commCtrl.h	Include file for controller chip
tdrv009.h	TDRV009 include file for driver and application
makenode	Script to create device nodes on the file system
Makefile	Device driver make file
example/tdrv009exa.c	Example application
example/Makefile	Example application make file
include/config.h	Driver independent library header file
include/tpmodule.h	Driver and kernel independent library header file
include/tpmodule.c	Driver and kernel independent library source file

In order to perform an installation, extract all files of the archive TDRV009-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TDRV009-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tdrv009.h to */usr/include*

2.1 Build and install the device driver

- Login as *root*
- Change to the target directory
- To create and install the 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 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 tdrv009drv

- After the first build or if you are using dynamic major device allocation it is necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled a device file system (devfs or sysfs with udev) then you have to 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 TDRV009 device can be accessed with device node /dev/tdrv009_0, the second channel with device node /dev/tdrv009_1 and so on. The assignment of device nodes to physical TDRV009 devices 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 tdrv009drv

If your kernel has enabled devfs or sysfs (udev), all /dev/tdrv009_x nodes will be automatically removed from your file system after this.

Be sure that the driver is not opened by any application program. If opened you will get the response ``t*drv009drv: 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 DEVFS installed.

The TDRV009 device driver uses dynamic allocation of major device numbers per default. If this isn't suitable for the application it's possible to define a major number for the driver.

To change the major number edit the file tdrv009def.h, change the following symbol to appropriate value and enter `make install` to create a new driver.

TDRV009_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TDRV009_MAJOR   122
```

Be sure that the desired major number is not used by other drivers. Please check */proc/devices* to see which numbers are free.

Keep in mind that it is necessary to create new device nodes if the major number for the TDRV009 driver has changed and the makenode script is not used.

3 Device Input/Output functions

This chapter describes the interface to the device driver I/O system.

3.1 open()

NAME

open() - open a file descriptor

SYNOPSIS

#include <fcntl.h>

int open
(
    const char *filename,
    int flags
)

DESCRIPTION

The open function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file is to be opened. This is a bit mask; you create the value by the bitwise OR of the appropriate parameters (using the | operator in C).

See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int fd;

fd = open("/dev/tdrv009_0", O_RDWR);
if (fd == -1) {
     /* handle error condition */
}
```

RETURNS

The normal return value from open is a non-negative integer file descriptor. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

ENODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during open. For more information about open error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.2 close()

NAME

close() – close a file descriptor

SYNOPSIS

#include <unistd.h>

int close
(
int filedes
)

DESCRIPTION

The close function closes the file descriptor *filedes*.

EXAMPLE

```
int fd;

if (close(fd) != 0) {
  /* handle close error conditions */
}
```

RETURNS

The normal return value from close is 0. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

ENODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during close. For more information about close error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 read()

NAME

read() – read data from a specified device.

SYNOPSIS

#include <unistd.h>

ssize_t read(int filedes, void *buffer, size_t size)

DESCRIPTION

This function attempts to read an input buffer from a TDRV009 channel associated with the file descriptor *fildes*. The argument *size* specifies the length of the buffer. Available data (up to *size* bytes) is copied into the user's buffer pointed to by *buffer*. The function performs a blocking read operation, i.e. the functions waits until data is available or the specified timeout expires (see *SET_READ_TIMEOUT*).

EXAMPLE

```
int           fd;
char          buffer[100];
int           retval;

/*-----------------------------
  Read data from TDRV009 device
  -----------------------------*/
retval = read(fd, buffer, 100);
if (retval >= 0)
{
     printf("%d bytes read\n", retval);
}
else
{
     /* handle the read error */
}

…
```

RETURNS

On success read returns the number of bytes. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

EFAULT	Error copying data from kernelspace to userspace

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.4 write()

NAME

write() – write to a device

SYNOPSIS

#include <unistd.h>

ssize_t write(int filedes, void *buffer, size_t size)

DESCRIPTION

This function attempts to writes a data buffer to the TDRV009 channel associated with the file descriptor *filedes.* The user specifies a character buffer pointed to by *buffer*. The argument *size* specifies the length of the buffer. The function performs a non-blocking write operation, i.e. the function returns immediately after inserting the data into the transmit-queue.

EXAMPLE

```
int       fd;
ssize_t   num_bytes;
char      buffer[100];

/* send some text */
sprintf( buffer, “Hello World” );
num_bytes = write(fd, buffer, strlen(buffer));
if ( num_bytes != strlen(buffer) )
{
      // process error;
}
```

RETURNS

On success write returns the number of bytes written. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

ENOMEM	Error getting memory for DMA transfer
EFAULT	Error copying data from userspace into kernelspace
EBUSY	No free transmit-descriptor available after timeout.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.5 ioctl()

NAME

ioctl() – device control functions

SYNOPSIS

```
#include <sys/ioctl.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 *tdrv009.h* :

Symbol	Meaning
TDRV009_IOCS_SET_OPERATION_MODE	Configure channel operation mode
TDRV009_IOCG_GET_OPERATION_MODE	Read current channel configuration
TDRV009_IOCT_SET_BAUDRATE	Setup transmission rate
TDRV009_IOCT_SET_RECEIVER_STATE	Configure receiver state
TDRV009_IOC_CLEAR_RX_BUFFER	Clear receive buffer
TDRV009_IOCT_SET_EXT_XTAL	Configure externally supplied clock frequency
TDRV009_IOCT_SET_READ_TIMEOUT	Specify a timeout for read operations
TDRV009_IOCQ_GET_TX_COUNT_ERROR	Read the transmit error counter
TDRV009_IOCQ_GET_TX_COUNT_OK	Read the transmit success counter
TDRV009_IOCS_EEPROMWRITE	Write value to EEPROM
TDRV009_IOCG_EEPROMREAD	Read value from EEPROM
TDRV009_IOCS_REGWRITE	Write value to register space
TDRV009_IOCG_REGREAD	Read value from register space
TDRV009_IOCS_SCCREGWRITE	Read value from SCC register space
TDRV009_IOCG_SCCREGREAD	Write value to SCC register space
TDRV009_IOCX_WAITFORINTERRUPT	Wait for specific channel interrupt

TDRV009_IOC_RTS_SET	Assert RTS handshake line
TDRV009_IOC_RTS_CLEAR	De-Assert RTS handshake line
TDRV009_IOCG_CTS_GET	Read state of CTS
TDRV009_IOC_DTR_SET	Set DTR signal line (only channel 3)
TDRV009_IOC_DTR_CLEAR	Clear DTR signal line (only channel 3)
TDRV009_IOCG_DSR_GET	Read state of DSR (only channel 3)

See behind for more detailed information on each control code.

To use these TDRV009 specific control codes the header file tdrv009.h must be included in the application.

RETURNS

On success, zero is returned. In the 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*
EFAULT	Parameter data can not be copied to the drivers context

Other function dependent error codes will be described for each ioctl code separately. Note, the TDRV009 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.5.1 TDRV009_IOCS_SET_OPERATION_MODE

NAME

TDRV009_IOCS_SET_OPERATION_MODE – Configure channel operation mode

DESCRIPTION

This I/O control function configures the channel's operation mode. The function specific control parameter **argp** is a pointer to a *TDRV009_OPERATION_MODE_STRUCT* structure. It is necessary to completely initialize the structure. This can be done by calling the I/O control function TDRV009_IOCG_GET_OPERATION_MODE described below.

A call to this function must be done prior to any communication operation, because after driver startup, the channel's transceivers are disabled.

```
typedef struct
{
   TDRV009_COMM_TYPE            CommType;
   TDRV009_TRANSCEIVER_MODE     TransceiverMode;
   TDRV009_ENABLE_DISABLE       Oversampling;
   TDRV009_BRGSOURCE            BrgSource;
   TDRV009_TXCSOURCE            TxClkSource;
   unsigned long                TxClkOutput;
   TDRV009_RXCSOURCE            RxClkSource;
   TDRV009_CLKMULTIPLIER        ClockMultiplier;
   unsigned long                Baudrate;
   unsigned char                ClockInversion;
   unsigned char                Encoding;
   TDRV009_PARITY               Parity;
   int                          Stopbits;
   int                          Databits;
   TDRV009_ENABLE_DISABLE       UseTermChar;
   char                         TermChar;
   TDRV009_ENABLE_DISABLE       HwHs;
   TDRV009_CRC                  Crc;
} TDRV009_OPERATION_MODE_STRUCT;
```

CommType

This parameter describes the general communication type for the specific channel. Possible values are:

Value	Description
TDRV009_COMMTYPE_ASYNC	Asynchronous communication
TDRV009_COMMTYPE_HDLC_ADDR0	Standard HDLC communication without address recognition. Used for synchronous communication.
TDRV009_COMMTYPE_HDLC_TRANSP	Extended Transparent mode. No protocol processing, channel works as simple bit collector.

TransceiverMode

This parameter describes the transceiver mode of the programmable multi-protocol transceivers. Possible values are:

Value	Description
TDRV009_TRNSCVR_NOT_USED	Default V.11
TDRV009_TRNSCVR_RS530A	EIA-530A (V.11 / V.10)
TDRV009_TRNSCVR_RS530	EIA-530 (V.11), also suitable for RS422
TDRV009_TRNSCVR_X21	X.21 (V.11)
TDRV009_TRNSCVR_V35	V.35 (V.35 / V.28)
TDRV009_TRNSCVR_RS449	EIA-449 (V.11)
TDRV009_TRNSCVR_V36	V.36 (V.11)
TDRV009_TRNSCVR_RS232	EIA-232 (V.28)
TDRV009_TRNSCVR_V28	V.28 (V.28)
TDRV009_TRNSCVR_NO_CABLE	High impedance

Oversampling

This parameter enables or disables 16times oversampling, used for asynchronous communication. For communication with standard UARTs it is recommended to enable this feature. Valid values are:

Value	Description
TDRV009_DISABLED	The 16 times oversampling is not used.
TDRV009_ENABLED	The 16 times oversampling is used.

BrgSource

This parameter specifies the frequency source used as input to the BRG (Baud Rate Generator). Valid values are:

Value	Description
TDRV009_BRGSRC_XTAL1	XTAL1 oscillator is used for BRG input
TDRV009_BRGSRC_XTAL2	XTAL2 oscillator is used for BRG input
TDRV009_BRGSRC_XTAL3	XTAL3 oscillator is used for BRG input
TDRV009_BRGSRC_RXCEXTERN	External clock at RxC input used for BRG input
TDRV009_BRGSRC_TXCEXTERN	External clock at TxC input used for BRG input

TxClkSource

This parameter specifies the frequency source used as input to the transmit engine. Valid values are:

Value	Description
TDRV009_TXCSRC_BRG	Baud Rate Generator output used for Tx clock
TDRV009_TXCSRC_BRGDIV16	BRG output divided by 16 used for Tx clock
TDRV009_TXCSRC_RXCEXTERN	External clock at RxC input used for Tx clock
TDRV009_TXCSRC_TXCEXTERN	External clock at TxC input used for Tx clock
TDRV009_TXCSRC_DPLL	DPLL output used for Tx clock

TxClkOutput

This parameter specifies which output lines are used to output the transmit clock, e.g. for synchronous communication. The given values can be binary OR'ed. Valid values are:

Value	Description
TDRV009_TXCOUT_TXC	Transmit clock available at TxC output line
TDRV009_TXCOUT_RTS	Transmit clock available at RTS output line

RxClkSource

This parameter specifies the frequency source used as input to the receive engine. Valid values are:

Value	Description
TDRV009_RXCSRC_BRG	Baud Rate Generator output used for Rx clock
TDRV009_RXCSRC_RXCEXTERN	External clock at RxC input used for Rx clock
TDRV009_RXCSRC_DPLL	DPLL output used for Rx clock

ClockMultiplier

This parameter specifies the multiplier used for BRG clock input. Valid values are:

Value	Description
TDRV009_CLKMULT_X1	Clock multiplier disabled
TDRV009_CLKMULT_X4	Selected input clock is multiplied by 4

Baudrate

This parameter specifies the desired frequency to be generated by the Baud Rate Generator (BRG), which can be used as clock input signal. The value is derived from the selected clocksource. Please note that only specific values depending on the selected oscillator are valid. This frequency is internally multiplied by 16, if oversampling shall be used.

ClockInversion

This parameter specifies the inversion of the transmit and/or the receive clock. This value can be binary OR'ed. Possible values are:

Value	Description
TDRV009_CLKINV_NONE	no clock inversion
TDRV009_CLKINV_TXC	transmit clock is inverted
TDRV009_CLKINV_RXC	receive clock is inverted

Encoding

This parameter specifies the data encoding used for communication. Valid values are:

Value	Description
TDRV009_ENC_NRZ	NRZ data encoding
TDRV009_ENC_NRZI	NRZI data encoding
TDRV009_ENC_FM0	FM0 data encoding
TDRV009_ENC_FM1	FM1 data encoding
TDRV009_ENC_MANCHESTER	Manchester data encoding

Parity

This parameter specifies the parity bit generation used for asynchronous communication. Valid values are:

Value	Description
TDRV009_PAR_DISABLED	No parity generation is used.
TDRV009_PAR_EVEN	EVEN parity bit
TDRV009_PAR_ODD	ODD parity bit
TDRV009_PAR_SPACE	SPACE parity bit (always insert ‘0’)
TDRV009_PAR_MARK	MARK parity bit (always insert ‘1’)

Stopbits

This parameter specifies the number of stop bits to use for asynchronous communication. Possible values are 1 or 2.

Databits

This parameter specifies the number of data bits to use for asynchronous communication. Possible values are 5 to 8.

UseTermChar

This parameter enables or disables the usage of a termination character for asynchronous communication. Valid values are:

Value	Description
TDRV009_DISABLED	A termination character is not used.
TDRV009_ENABLED	A termination character is used.

TermChar

This parameter specifies the termination character. After receiving this termination character, the communication controller will forward the received data packet immediately to the host system and use a new data packet for further received data. Any 8bit value may be used for this parameter.

HwHs

This parameter enables or disables the hardware handshaking mechanism using RTS/CTS. Valid values are:

Value	Description
TDRV009_DISABLED	Hardware handshaking is not used.
TDRV009_ENABLED	Hardware handshaking is used.

Crc

This parameter is a structure describing the CRC checking configuration.

```
typedef struct
{
   TDRV009_CRC_TYPE          Type;
   TDRV009_ENABLE_DISABLE    RxChecking;
   TDRV009_ENABLE_DISABLE    TxGeneration;
   TDRV009_CRC_RESET         ResetValue;
} TDRV009_CRC;
```

Type

This parameter describes the CRC type to be used. Possible values are:

Value	Description
TDRV009_CRC_16	16bit CRC algorithm is used for checksum
TDRV009_CRC_32	32bit CRC algorithm is used for checksum

RxChecking

This parameter enables or disables the receive CRC checking. Possible values are:

Value	Description
TDRV009_DISABLED	CRC checking will not be used
TDRV009_ENABLED	CRC checking will be used

TxGeneration

This parameter enables or disables the transmit CRC generation. Possible values are:

Value	Description
TDRV009_DISABLED	A CRC checksum will be generated
TDRV009_ENABLED	A CRC checksum will not be generated

ResetValue

This parameter describes the reset value for the CRC algorithm. Possible values are:

Value	Description
TDRV009_CRC_RST_FFFF	CRC reset value will be 0xFFFF
TDRV009_CRC_RST_0000	CRC reset value will be 0x0000

EXAMPLE

```
#include “tdrv009.h”

int                              fd;
TDRV009_OPERATION_MODE_STRUCT    OperationMode;
int                              retval;

/*-------------------------------------------------
  Configure channel for Async / RS232 / 115200bps
  -------------------------------------------------*/
OperationMode.CommType           = TDRV009_COMMTYPE_ASYNC;
OperationMode.TransceiverMode    = TDRV009_TRNSCVR _RS232;
OperationMode.Oversampling       = TDRV009_ENABLED;
OperationMode.BrgSource          = TDRV009_BRGSRC_XTAL1;
OperationMode.TxClkSource        = TDRV009_TXCSRC_BRG;
OperationMode.TxClkOutput        = 0;
OperationMode.RxClkSource        = TDRV009_RXCSRC_BRG;
OperationMode.ClockMultiplier    = TDRV009_CLKMULT_X1;
OperationMode.Baudrate           = 115200;
OperationMode.ClockInversion     = TDRV009_CLKINV_NONE;
OperationMode.Encoding           = TDRV009_ENC_NRZ;
OperationMode.Parity             = TDRV009_PAR_DISABLED;
OperationMode.Stopbits           = 1;
OperationMode.Databits           = 8;
OperationMode.UseTermChar        = TDRV009_DISABLED;
OperationMode.TermChar           = 0;
OperationMode.HwHs               = TDRV009_DISABLED;
OperationMode.Crc.Type           = TDRV009_CRC_16;
OperationMode.Crc.RxChecking     = TDRV009_DISABLED;
OperationMode.Crc.TxGeneration   = TDRV009_DISABLED;
OperationMode.Crc.ResetValue     = TDRV009_CRC_RST_FFFF;

retval = ioctl(fd, TDRV009_IOCS_SET_OPERATION_MODE, (int)&OperationMode);
if (retval >= 0)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EINVAL	Invalid parameter specified. At least one parameter inside the structure is invalid.

3.5.2 TDRV009_IOCG_GET_OPERATION_MODE

NAME

TDRV009_IOCG_GET_OPERATION_MODE – Read current channel configuration

DESCRIPTION

This I/O control function returns the channel's current operation mode. The function specific control parameter **argp** is a pointer to a *TDRV009_OPERATION_MODE_STRUCT* structure.

```
typedef struct
{
   TDRV009_COMM_TYPE            CommType;
   TDRV009_TRANSCEIVER_MODE     TransceiverMode;
   TDRV009_ENABLE_DISABLE       Oversampling;
   TDRV009_BRGSOURCE            BrgSource;
   TDRV009_TXCSOURCE            TxClkSource;
   unsigned long                TxClkOutput;
   TDRV009_RXCSOURCE            RxClkSource;
   TDRV009_CLKMULTIPLIER        ClockMultiplier;
   unsigned long                Baudrate;
   unsigned char                ClockInversion;
   unsigned char                Encoding;
   TDRV009_PARITY               Parity;
   int                          Stopbits;
   int                          Databits;
   TDRV009_ENABLE_DISABLE       UseTermChar;
   char                         TermChar;
   TDRV009_ENABLE_DISABLE       HwHs;
   TDRV009_CRC                  Crc;
} TDRV009_OPERATION_MODE_STRUCT;
```

CommType

This parameter describes the general communication type for the specific channel. Possible values are:

Value	Description
TDRV009_COMMTYPE_ASYNC	Asynchronous communication
TDRV009_COMMTYPE_HDLC_ADDR0	Standard HDLC communication without address recognition. Used for synchronous communication.
TDRV009_COMMTYPE_HDLC_TRANSP	Extended Transparent mode. No protocol processing, channel works as simple bit collector.

TransceiverMode

This parameter describes the transceiver mode of the programmable multi-protocol transceivers. Possible values are:

Value	Description
TDRV009_TRNSCVR _NOT_USED	Default V.11
TDRV009_TRNSCVR _RS530A	EIA-530A (V.11 / V.10)
TDRV009_TRNSCVR _RS530	EIA-530 (V.11), also suitable for RS422
TDRV009_TRNSCVR _X21	X.21 (V.11)
TDRV009_TRNSCVR _V35	V.35 (V.35 / V.28)
TDRV009_TRNSCVR _RS449	EIA-449 (V.11)
TDRV009_TRNSCVR _V36	V.36 (V.11)
TDRV009_TRNSCVR _RS232	EIA-232 (V.28)
TDRV009_TRNSCVR _V28	V.28 (V.28)
TDRV009_TRNSCVR _NO_CABLE	High impedance

Oversampling

This parameter enables or disables 16times oversampling, used for asynchronous communication. For communication with standard UARTs it is recommended to enable this feature. Valid values are:

Value	Description
TDRV009_DISABLED	The 16 times oversampling is not used.
TDRV009_ENABLED	The 16 times oversampling is used.

BrgSource

This parameter specifies the frequency source used as input to the BRG (Baud Rate Generator). Valid values are:

Value	Description
TDRV009_BRGSRC_XTAL1	XTAL1 oscillator is used for BRG input
TDRV009_BRGSRC_XTAL2	XTAL2 oscillator is used for BRG input
TDRV009_BRGSRC_XTAL3	XTAL3 oscillator is used for BRG input
TDRV009_BRGSRC_RXCEXTERN	External clock at RxC input used for BRG input
TDRV009_BRGSRC_TXCEXTERN	External clock at TxC input used for BRG input

TxClkSource

This parameter specifies the frequency source used as input to the transmit engine. Valid values are:

Value	Description
TDRV009_TXCSRC_BRG	Baud Rate Generator output used for Tx clock
TDRV009_TXCSRC_BRGDIV16	BRG output divided by 16 used for Tx clock
TDRV009_TXCSRC_RXCEXTERN	External clock at RxC input used for Tx clock
TDRV009_TXCSRC_TXCEXTERN	External clock at TxC input used for Tx clock
TDRV009_TXCSRC_DPLL	DPLL output used for Tx clock

TxClkOutput

This parameter specifies which output lines are used to output the transmit clock, e.g. for synchronous communication. The given values can be binary OR'ed. Valid values are:

Value	Description
TDRV009_TXCOUT_TXC	Transmit clock available at TxC output line
TDRV009_TXCOUT_RTS	Transmit clock available at RTS output line

RxClkSource

This parameter specifies the frequency source used as input to the receive engine. Valid values are:

Value	Description
TDRV009_RXCSRC_BRG	Baud Rate Generator output used for Rx clock
TDRV009_RXCSRC_RXCEXTERN	External clock at RxC input used for Rx clock
TDRV009_RXCSRC_DPLL	DPLL output used for Rx clock

ClockMultiplier

This parameter specifies the multiplier used for BRG clock input. Valid values are:

Value	Description
TDRV009_CLKMULT_X1	Clock multiplier disabled
TDRV009_CLKMULT_X4	Selected input clock is multiplied by 4

Baudrate

This parameter specifies the desired frequency to be generated by the BRG (Baud Rate Generator), which can be used as clock input signal. The value is derived from the selected clocksource. Please note that only specific values depending on the selected oscillator are valid. This frequency is internally multiplied by 16, if oversampling shall be used.

ClockInversion

This parameter specifies the inversion of the transmit and/or the receive clock. This value can be binary OR'ed. Possible values are:

Value	Description
TDRV009_CLKINV_NONE	no clock inversion
TDRV009_CLKINV_TXC	transmit clock is inverted
TDRV009_CLKINV_RXC	receive clock is inverted

Encoding

This parameter specifies the data encoding used for communication. Valid values are:

Value	Description
TDRV009_ENC_NRZ	NRZ data encoding
TDRV009_ENC_NRZI	NRZI data encoding
TDRV009_ENC_FM0	FM0 data encoding
TDRV009_ENC_FM1	FM1 data encoding
TDRV009_ENC_MANCHESTER	Manchester data encoding

Parity

This parameter specifies the parity bit generation used for asynchronous communication. Valid values are:

Value	Description
TDRV009_PAR_DISABLED	No parity generation is used.
TDRV009_PAR_EVEN	EVEN parity bit
TDRV009_PAR_ODD	ODD parity bit
TDRV009_PAR_SPACE	SPACE parity bit (always insert '0')
TDRV009_PAR_MARK	MARK parity bit (always insert '1')

Stopbits

This parameter specifies the number of stop bits to use for asynchronous communication. Possible values are 1 or 2.

Databits

This parameter specifies the number of data bits to use for asynchronous communication. Possible values are 5 to 8.

UseTermChar

This parameter enables or disables the usage of a termination character for asynchronous communication. Valid values are:

Value	Description
TDRV009_DISABLED	A termination character is not used.
TDRV009_ENABLED	A termination character is used.

TermChar

This parameter specifies the termination character. After receiving this termination character, the communication controller will forward the received data packet immediately to the host system and use a new data packet for further received data. Any 8bit value may be used for this parameter.

HwHs

This parameter enables or disables the hardware handshaking mechanism using RTS/CTS. Valid values are:

Value	Description
TDRV009_DISABLED	Hardware handshaking is not used.
TDRV009_ENABLED	Hardware handshaking is used.

Crc

This parameter is a structure describing the CRC checking configuration.

```
typedef struct
{
   TDRV009_CRC_TYPE          Type;
   TDRV009_ENABLE_DISABLE    RxChecking;
   TDRV009_ENABLE_DISABLE    TxGeneration;
   TDRV009_CRC_RESET         ResetValue;
} TDRV009_CRC;
```

Type

This parameter describes the CRC type to be used. Possible values are:

Value	Description
TDRV009_CRC_16	16bit CRC algorithm is used for checksum
TDRV009_CRC_32	32bit CRC algorithm is used for checksum

RxChecking

This parameter enables or disables the receive CRC checking. Possible values are:

Value	Description
TDRV009_DISABLED	CRC checking will not be used
TDRV009_ENABLED	CRC checking will be used

TxGeneration

This parameter enables or disables the transmit CRC generation. Possible values are:

Value	Description
TDRV009_DISABLED	A CRC checksum will be generated
TDRV009_ENABLED	A CRC checksum will not be generated

ResetValue

This parameter describes the reset value for the CRC algorithm. Possible values are:

Value	Description
TDRV009_CRC_RST_FFFF	CRC reset value will be 0xFFFF
TDRV009_CRC_RST_0000	CRC reset value will be 0x0000

EXAMPLE

```
#include “tdrv009.h”

…

int                                fd;
TDRV009_OPERATION_MODE_STRUCT      OperationMode;
int                                retval;

…

/*-------------------------------------------------
  Retrieve current configuration
  -------------------------------------------------*/
retval = ioctl(fd, TDRV009_IOCG_GET_OPERATION_MODE, (int)&OperationMode);
if (retval >= 0)
{
     /* function succeeded */
     printf( “Current speed: %d bps\n”, OperationMode.Baudrate );
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EFAULT	Parameter data can not be copied to or from the driver’s context. Please check the argument *argp*.

3.5.3 TDRV009_IOCT_SET_BAUDRATE

NAME

TDRV009_IOCT_SET_BAUDRATE – Setup transmission rate

DESCRIPTION

This I/O control function sets up the transmission rate for the specific channel. This is done without all the other configuration stuff contained in TDRV009_IOCS_SET_OPERATION_MODE. If async oversampling is enabled, the desired baudrate is internally multiplied by 16. It is important that this result can be derived from the selected clocksource. This function specifies the desired frequency which should be generated by the Baud Rate Generator (BRG). The function dependent parameter **argp** passes an *unsigned long* value containing the desired baudrate to the device driver.

EXAMPLE

```
#include “tdrv009.h”

int           fd;
unsigned long Baudrate;
int           retval;

/*-------------------------------------------------
  Set baudrate to 14400bps
  -------------------------------------------------*/
Baudrate = 14400;
retval = ioctl(fd, TDRV009_IOCT_SET_BAUDRATE, (int)Baudrate);
if (retval >= 0)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EINVAL	Invalid parameter specified.

3.5.4 TDRV009_IOCT_SET_RECEIVER_STATE

NAME

TDRV009_IOCT_SET_RECEIVER_STATE – Configure receiver state

DESCRIPTION

This command sets the channel's receiver either to active or inactive.

The parameter **argp** passed to this Ioctl function defines the new state of the receiver. Possible values are:

Value	Description
TDRV009_RCVR_ON	The receiver is enabled.
TDRV009_RCVR_OFF	The receiver is disabled.

Example

```
#include “tdrv009.h”
int  fd;
int  retval;

/*
**  enable the receiver
*/
retval = ioctl( fd, TDRV009_IOCT_SET_RECEIVER_STATE, TDRV009_RCVR_ON);
if (status < 0)
{
     /* process Ioctl error */
}
```

ERROR CODES

Error code	Description
EINVAL	Invalid parameter specified.

3.5.5 TDRV009_IOC_CLEAR_RX_BUFFER

NAME

TDRV006_IOC_ WRITE – Write value to output buffer

DESCRIPTION

This I/O control function removes all received data from the channel's receive buffer, and flushes the hardware FIFO as well. The function dependent argument **argp** is not used for this function.

Example

```
#include “tdrv009.h”

int  fd;
int  retval;


/*------------------------------------------------------------
  Clear receive buffer
  ------------------------------------------------------------*/
retval = ioctl( fd, TDRV009_IOC_CLEAR_RX_BUFFER, 0);
if (retval < 0)
{
     /* process Ioctl error */
}
```

ERROR CODES

Error code	Description
EIO	Error during reset or init of the receiver's hardware DMA engine.

3.5.6 TDRV009_IOCT_SET_EXT_XTAL

NAME

TDRV009_IOCT_SET_EXT_XTAL – Configure externally supplied clock frequency

DESCRIPTION

This I/O control function specifies the frequency of an externally provided clock. This frequency is used for baudrate calculation, and describes the input frequency to the Baud Rate Generator (BRG). The external frequency may be supplied either at input line TxC or RxC. The function dependent argument **argp** passes the clock frequency in Hz.

Example

```
#include “tdrv009.h”

int  fd;
int  retval;


/*------------------------------------------------------------
  Specify 1000000Hz (1MHz) as external clock frequency
  ------------------------------------------------------------*/
retval = ioctl( fd, TDRV009_IOCT_SET_EXT_XTAL, 1000000);
if (retval < 0)
{
     /* process Ioctl error */
}
```

ERROR CODES

Error code	Description
EINVAL	Invalid parameter. The specified frequency must be larger than 0.

3.5.7 TDRV009_IOCT_SET_READ_TIMEOUT

NAME

TDRV009_IOCT_SET_READ_TIMEOUT – Specify a timeout for read operations

DESCRIPTION

This I/O control function specifies the timeout used for read operations. The parameter **argp** passes an unsigned long value containing the new timeout value (in jiffies, i.e. system-ticks) to the device driver.

Example

```
#include “tdrv009.h”

int  fd;
int  retval;


/*------------------------------------------------------------
  specify the read-timeout (100 clock ticks)
  ------------------------------------------------------------*/
retval = ioctl( fd, TDRV009_IOCT_SET_READ_TIMEOUT, 100);
if (retval < 0)
{
     /* process Ioctl error */
}
```

3.5.8 TDRV009_IOCQ_GET_TX_COUNT_ERROR

NAME

TDRV009_IOCQ_GET_TX_COUNT_ERROR – Read the transmit error counter

DESCRIPTION

This I/O control function returns the global transmit error counter for the corresponding channel. The transmit error counter is reset after a call to this function. The parameter **argp** passes a pointer to an unsigned long value to the device driver, where the error count is returned.

Example

```
#include “tdrv009.h”

int  fd;
int  retval;
unsigned long ErrorCount;

/*-------------------------------------------------------------
  Read Transmit Error Counter
  -------------------------------------------------------------*/
retval = ioctl( fd, TDRV009_IOCQ_GET_TX_COUNT_ERROR, &ErrorCount);
if (retval >= 0)
{
     printf(“ErrorCount = %ld\n”, ErrorCount );
} else {
     /* process Ioctl error */
}
```

ERROR CODES

Error code	Description
EFAULT	Parameter data can not be copied to or from the driver’s context. Please check the argument *argp*.

3.5.9 TDRV009_IOCQ_GET_TX_COUNT_OK

NAME

TDRV009_IOCQ_GET_TX_COUNT_OK – Read the transmit success counter

DESCRIPTION

This I/O control function returns the global transmit success counter for the corresponding channel. The transmit success counter is reset after a call to this function. The parameter **argp** passes a pointer to an unsigned long value to the device driver, where the success count is returned.

Example

```
#include “tdrv009.h”

int  fd;
int  retval;
unsigned long OkCount;

/*-------------------------------------------------------------
  Read Transmit Error Counter
  -------------------------------------------------------------*/
retval = ioctl( fd, TDRV009_IOCQ_GET_TX_COUNT_OK, &OkCount);
if (retval >= 0)
{
     printf(“OkCount = %ld\n”, OkCount );
} else {
     /* process Ioctl error */
}
```

ERROR CODES

Error code	Description
EFAULT	Parameter data can not be copied to or from the driver’s context. Please check the argument *argp*.

3.5.10 TDRV009_IOCS_EEPROMWRITE

NAME

TDRV009_IOCS_EEPROMWRITE– Write value to EEPROM

DESCRIPTION

This I/O control function writes one 16bit word into the onboard EEPROM. The first part of the EEPROM is reserved for factory usage, write accesses to this area will result in an error. The function specific control parameter **argp** is a pointer to a *TDRV009_EEPROM_BUFFER* structure.

```
typedef struct
{
   unsigned long    Offset;
   unsigned long    Value;
} TDRV009_EEPROM_BUFFER;
```

Offset

This parameter specifies a 16bit word offset into the EEPROM.
Following offsets are available:

Offset	Access
00h – 5Fh	R
60h – 7Fh	R / W

Value

This parameter specifies the 16bit word to be written into the EEPROM at the given offset.

EXAMPLE

```
#include “tdrv009.h”

…

int                     fd;
TDRV009_EEPROM_BUFFER   EepromBuf;
int                     retval;

…
```

```
…

/*-------------------------------------------------
  Write a 16bit value into the EEPROM, offset 0x61
  -------------------------------------------------*/
EepromBuf.Offset = 0x61;
EepromBuf.Value  = 0x1234;
retval = ioctl(fd, TDRV009_IOCS_EEPROMWRITE, (int)&EepromBuf);
if (retval < 0)
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EFAULT	Parameter data can not be copied to or from the driver's context. Please check the argument *argp*.
EACCES	The specified offset address is invalid, or read-only.

3.5.11 TDRV009_IOCG_EEPROMREAD

NAME

TDRV009_IOCG_EEPROMREAD– Read value from EEPROM

DESCRIPTION

This I/O control function reads one 16bit word from the onboard EEPROM. The function specific control parameter **argp** is a pointer to a *TDRV009_EEPROM_BUFFER* structure.

```
typedef struct
{
   unsigned long    Offset;
   unsigned long    Value;
} TDRV009_EEPROM_BUFFER;
```

Offset

This parameter specifies a 16bit word offset into the EEPROM.
Following offsets are available:

Offset	Access
00h – 5Fh	R
60h – 7Fh	R / W

Value

This parameter returns the 16bit word from the EEPROM at the given offset.

EXAMPLE

```
#include “tdrv009.h”

…

int                     fd;
TDRV009_EEPROM_BUFFER   EepromBuf;
int                     retval;

…

/*-------------------------------------------------
  Read a 16bit value from the EEPROM, offset 0
  -------------------------------------------------*/
EepromBuf.Offset = 0;
retval = ioctl(fd, TDRV009_IOCG_EEPROMREAD, (int)&EepromBuf);
```

```
if (retval >= 0)
{
     printf( “Value = 0x%X\n”, EepromBuf.Value );
} else {
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EFAULT	Parameter data can not be copied to or from the driver’s context. Please check the argument *argp*.
EACCES	Specified offset is invalid, and may not be accessed.

3.5.12 TDRV009_IOCS_REGWRITE

NAME

TDRV009_IOCS_REGWRITE – Write value to register space

DESCRIPTION

This I/O control function writes one 32bit word to the communication controller's register space, relative to the beginning of the register set. The function specific control parameter **argp** is a pointer to a *TDRV009_ADDR_STRUCT* structure.

```
typedef struct
{
   unsigned long    Offset;
   unsigned long    Value;
} TDRV009_ADDR_STRUCT;
```

Offset

This parameter specifies a byte offset into the communication controller's register space. Please refer to the hardware user manual for further information.

Value

This 32bit word will be written to the communication controller's register space.

Modifying register contents may result in communication problems, system crash or other unexpected behavior.

EXAMPLE

```
#include "tdrv009.h"

int                     fd;
TDRV009_ADDR_STRUCT     AddrBuf;
int                     retval;

...
```

```
…

/*-------------------------------------------------
  Write a 32bit value (FIFO Control Register 4)
  -------------------------------------------------*/
AddrBuf.Offset = 0x0034;
AddrBuf.Value  = 0xffffffff;
retval = ioctl(fd, TDRV009_IOCS_REGWRITE, (int)&AddrBuf);
if (retval < 0)
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EFAULT	Parameter data can not be copied to or from the driver's context. Please check the argument *argp*.
EACCES	The specified offset is invalid.

3.5.13 TDRV009_IOCG_REGREAD

NAME

TDRV009_IOCG_REGREAD – Read value from register space

DESCRIPTION

This I/O control function reads one 32bit word from the communication controller's register space, relative to the beginning of the register set. The function specific control parameter **argp** is a pointer to a *TDRV009_ADDR_STRUCT* structure.

```
typedef struct
{
   unsigned long    Offset;
   unsigned long    Value;
} TDRV009_ADDR_STRUCT;
```

Offset

This parameter specifies a byte offset into the communication controller's register space. Please refer to the hardware user manual for further information.

Value

This parameter returns the 32bit word from the communication controller's register space.

EXAMPLE

```
#include "tdrv009.h"

int                     fd;
TDRV009_ADDR_STRUCT     AddrBuf;
int                     retval;

/*-------------------------------------------------
  Read a 32bit value (Version Register)
  -------------------------------------------------*/
AddrBuf.Offset = 0x00F0;
retval = ioctl(fd, TDRV009_IOCG_REGREAD, (int)&AddrBuf);
if (retval >= 0)
{
     printf( "Value = 0x%lX\n", AddrBuf.Value );
} else {
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EFAULT	Parameter data can not be copied to or from the driver's context. Please check the argument *argp*.
EACCES	The specified offset is invalid.

3.5.14 TDRV009_IOCS_SCCREGWRITE

NAME

TDRV009_IOCS_SCCREGWRITE – Write value to SCC register space

DESCRIPTION

This I/O control function writes one 32bit word to the communication controller's register space, relative to the beginning of the specific channel's SCC register set. The function specific control parameter **argp** is a pointer to a *TDRV009_ADDR_STRUCT* structure.

```
typedef struct
{
   unsigned long    Offset;
   unsigned long    Value;
} TDRV009_ADDR_STRUCT;
```

Offset

This parameter specifies a byte offset into the communication controller's channel SCC register space. Please refer to the hardware user manual for further information.

Value

This 32bit word will be written to the communication controller's channel SCC register space.

Modifying register contents may result in communication problems, system crash or other unexpected behavior.

EXAMPLE

```
#include "tdrv009.h"

int                     fd;
TDRV009_ADDR_STRUCT     AddrBuf;
int                     retval;

...
```

```
/*-------------------------------------------------
  Write a 32bit value (Termination Character Register)
  -------------------------------------------------*/
AddrBuf.Offset = 0x0048;
AddrBuf.Value  = (1 << 15) | 0x42;
retval = ioctl(fd, TDRV009_IOCS_SCCREGWRITE, (int)&AddrBuf);
if (retval < 0)
{
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EFAULT	Parameter data can not be copied to or from the driver's context. Please check the argument *argp*.
EACCES	The specified offset is invalid.

3.5.15 TDRV009_IOCG_SCCREGREAD

NAME

TDRV009_IOCG_SCCREGREAD – Read value from SCC register space

DESCRIPTION

This I/O control function reads one 32bit word from the communication controller's register space, relative to the beginning of the specific channel's SCC register set. The function specific control parameter **argp** is a pointer to a *TDRV009_ADDR_STRUCT* structure.

```
typedef struct
{
   unsigned long    Offset;
   unsigned long    Value;
} TDRV009_ADDR_STRUCT;
```

Offset

This parameter specifies a byte offset into the communication controller's channel SCC register space. Please refer to the hardware user manual for further information.

Value

This parameter returns the 32bit word from the communication controller's channel SCC register space.

EXAMPLE

```
#include "tdrv009.h"

int                     fd;
TDRV009_ADDR_STRUCT     AddrBuf;
int                     retval;

/*-------------------------------------------------
  Read a 32bit value (Status Register)
  -------------------------------------------------*/
AddrBuf.Offset = 0x0004;
retval = ioctl(fd, TDRV009_IOCG_SCCREGREAD, (int)&AddrBuf);
if (retval >= 0)
{
     printf( "Value = 0x%lX\n", AddrBuf.Value );
} else {
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EFAULT	Parameter data can not be copied to or from the driver's context. Please check the argument *argp*.
EACCES	The specified offset is invalid.

3.5.16 TDRV009_IOCX_WAITFORINTERRUPT

NAME

TDRV009_IOCX_WAITFORINTERRUPT – Wait for specific channel interrupt

DESCRIPTION

This I/O control function waits until a specified SCC-interrupt or the timeout occurs. The function specific control parameter **argp** is a pointer to a *TDRV009_WAIT_STRUCT* structure.

```
typedef struct
{
   unsigned long    Interrupts;
   int              Timeout;
} TDRV009_WAIT_STRUCT;
```

Interrupts

This parameter specifies specific interrupt bits to wait for. If one interrupt occurs, the value is returned in this parameter. Please refer to the hardware user manual for further information on the possible interrupt bits.

Timeout

This parameter specifies the time (in system ticks) to wait for an interrupt. If 0 is specified, the function will block indefinitely.

EXAMPLE

```
#include “tdrv009.h”

int                     fd;
TDRV009_WAIT_STRUCT     WaitStruct;
int                     retval;

…
```

…

```
/*-------------------------------------------------
  Wait at least 5 seconds (5*HZ) for a
  CTS Staus Change (CSC) interrupt
  -------------------------------------------------*/
WaitStruct.Interrupts = (1 << 14);
WaitSrtuct.Timeout    = 5*HZ;
retval = ioctl(fd, TDRV009_IOCX_WAITFORINTERRUPT, (int)&WaitStruct);
if (retval >= 0)
{
     printf( “Interrupts = 0x%lX\n”, WaitStruct.Interrupts );
} else {
     /* handle the error */
}
```

ERROR CODES

Error code	Description
EFAULT	Parameter data can not be copied to or from the driver’s context. Please check the argument *argp*.
EBUSY	Too many simultaneous wait jobs active.
ETIME	Timeout happened.

3.5.17 TDRV009_IOC_RTS_SET

NAME

TDRV009_IOC_RTS_SET – Assert RTS handshake line

DESCRIPTION

This I/O control function asserts the RTS handshake signal line of the specific channel. This function is not available if the channel is configured for hardware handshaking. The function dependent argument **argp** is not used for this function.

Example

```
#include “tdrv009.h”

int  fd;
int  retval;


/*-------------------------------------------------------------
  Assert RTS
  -------------------------------------------------------------*/
retval = ioctl( fd, TDRV009_IOC_RTS_SET, 0);
if (retval < 0)
{
     /* process Ioctl error */
}
```

ERROR CODES

Error code	Description
EPERM	The channel is in handshake mode, so this function is not allowed.

3.5.18 TDRV009_IOC_RTS_CLEAR

NAME

TDRV009_IOC_RTS_CLEAR – De-Assert RTS handshake line

DESCRIPTION

This I/O control function de-asserts the RTS handshake signal line of the specific channel. This function is not available if the channel is configured for hardware handshaking. The function dependent argument **argp** is not used for this function.

Example

```
#include “tdrv009.h”

int  fd;
int  retval;


/*------------------------------------------------------------
  De-assert RTS
  ------------------------------------------------------------*/
retval = ioctl( fd, TDRV009_IOC_RTS_CLEAR, 0);
if (retval < 0)
{
     /* process Ioctl error */
}
```

ERROR CODES

Error code	Description
EPERM	The channel is in handshake mode, so this function is not allowed.

3.5.19 TDRV009_IOCG_CTS_GET

NAME

TDRV009_IOCG_CTS_GET – Read state of CTS

DESCRIPTION

This I/O control function returns the current state of the CTS handshake signal line of the specific channel. The function dependent argument **argp** passes a pointer to an *unsigned long* value to the driver. Depending on the state of CTS, either 0 (inactive) or 1 (active) is returned.

Example

```
#include “tdrv009.h”

int           fd;
int           retval;
unsigned long CtsState;


/*-------------------------------------------------------------
  Read CTS state
  -------------------------------------------------------------*/
retval = ioctl( fd, TDRV009_IOCG_CTS_GET, (int)&CtsState);
if (retval >= 0)
{
     printf( “CTS = %ld\n”, CtsState );
} else {
     /* process Ioctl error */
}
```

ERROR CODES

Error code	Description
EFAULT	Parameter data can not be copied to or from the driver’s context. Please check the argument *argp*.

3.5.20 TDRV009_IOC_DTR_SET

NAME

TDRV009_IOC_DTR_SET – Set DTR signal line (only channel 3)

DESCRIPTION

This I/O control function sets the DTR signal line to HIGH. This function is only available for local module channel 3. The function dependent argument **argp** is not used for this function.

Example

```
#include “tdrv009.h”

int  fd;
int  retval;


/*------------------------------------------------------------
  Set DTR to HIGH (only valid for channel 3)
  ------------------------------------------------------------*/
retval = ioctl( fd, TDRV009_IOC_DTR_SET, 0);
if (retval < 0)
{
     /* process Ioctl error */
}
```

ERROR CODES

Error code	Description
EACCES	This function is not supported by the specific channel.

3.5.21 TDRV009_IOC_DTR_CLEAR

NAME

TDRV009_IOC_DTR_CLEAR – Clear DTR signal line (only channel 3)

DESCRIPTION

This I/O control function sets the DTR signal line to LOW. This function is only available for local module channel 3. The function dependent argument **argp** is not used for this function.

Example

```
#include “tdrv009.h”

int  fd;
int  retval;


/*-------------------------------------------------------------
  Set DTR to LOW (only valid for channel 3)
  -------------------------------------------------------------*/
retval = ioctl( fd, TDRV009_IOC_DTR_CLEAR, 0);
if (retval < 0)
{
     /* process Ioctl error */
}
```

ERROR CODES

Error code	Description
EACCES	This function is not supported by the specific channel.

3.5.22 TDRV009_IOCG_DSR_GET

NAME

TDRV009_IOCG_DSR_GET – Read state of DSR (only channel 3)

DESCRIPTION

This I/O control function returns the current state of the DSR signal line of the specific channel. This function is only available for local module channel 3. The function dependent argument **argp** passes a pointer to an *unsigned long* value to the driver. Depending on the state of DSR, either 0 (inactive) or 1 (active) is returned.

Example

```
#include “tdrv009.h”

int           fd;
int           retval;
unsigned long DsrState;


/*------------------------------------------------------------
  Read DSR state
  ------------------------------------------------------------*/
retval = ioctl( fd, TDRV009_IOCG_DSR_GET, (int)&DsrState);
if (retval >= 0)
{
     printf( “DSR = %ld\n”, DsrState );
} else {
     /* process Ioctl error */
}
```

ERROR CODES

Error code	Description
EFAULT	Parameter data can not be copied to or from the driver’s context. Please check the argument *argp*.