The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV010-SW-82

Linux Device Driver

Isolated 2x CAN Bus

Version 1.0.x

User Manual

Issue 1.0.1

March 2009

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

TEWS TECHNOLOGIES LLC
9190 Double Diamond Parkway,
Suite 127, Reno, NV 89521, USA
www.tews.com
Phone: +1 (775) 850 5830
Fax: +1 (775) 201 0347
e-mail: usasales@tews.com

TDRV010-SW-82

Linux Device Driver

Isolated 2x CAN Bus

Supported Modules:
TPMC310
TPMC810

Issue	Description	Date
1.0.0	First Issue	May 9, 2007
1.0.1	Bitrate values corrected	March 9, 2009

Table of Contents

1 INTRODUCTION 4

2 INSTALLATION 5

2.1 Build and install the device driver 5
2.2 Uninstall the device driver 6
2.3 Install the device driver in the running kernel 6
2.4 Remove device driver from the running kernel 6
2.5 Change Major Device Number 7
2.6 Receive Queue Configuration 7

3 DEVICE INPUT/OUTPUT FUNCTIONS 8

3.1 open() 8
3.2 close() 10
3.3 ioctl() 11
3.3.1 TDRV010_IOCXREAD 13
3.3.2 TDRV010_IOCSWRITE 16
3.3.3 TDRV010_IOCSBITTIMING 19
3.3.4 TDRV010_IOCSSETFILTER 21
3.3.5 TDRV010_IOCBUSON 23
3.3.6 TDRV010_IOCBUSOFF 24
3.3.7 TDRV010_IOCFLUSH 25
3.3.8 TDRV010_IOCGCANSTATUS 26
3.3.9 TDRV010_IOCENABLE_SELFTEST 28
3.3.10 TDRV010_IOCDISABLE_SELFTEST 29
3.3.11 TDRV010_IOCENABLE_LISTENONLY 30
3.3.12 TDRV010_IOCDISABLE_LISTENONLY 31
3.3.13 TDRV010_IOCSSETLIMIT 32
3.3.14 TDRV010_IOCTCANRESET 33
3.3.15 TDRV010_IOCTCANSEL 35
3.3.16 TDRV010_IOCTCANINT 37
3.4 Step by Step Driver Initialization 39

4 DIAGNOSTIC 40

1 Introduction

The TDRV010-SW-82 Linux device driver allows the operation of the TDRV010 2x CAN PMC devices conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with open(), close() 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.

Supported features:

- Transmission and reception of Standard and Extended Identifiers
- Standard bit rates from 20 kbit up to 1 Mbit and user defined bit rates
- Message acceptance filtering
- Single-Shot transmission
- Listen only mode
- Message self reception
- Programmable error warning limit
- Creates devices with dynamically allocated or fixed major device numbers
- DEVFS and SYSFS (UDEV) support for automatic device node creation

The TDRV010-SW-82 device driver supports the modules listed below:

TPMC310	Isolated 2 x CAN Bus	(PMC, Conduction Cooled)
TPMC810	Isolated 2 x CAN Bus	(PMC)

In this document all supported modules and devices will be called TDRV010. Specials for certain devices will be advised.

To get more information about the features and use of TDRV010 devices it is recommended to read the manuals listed below.

TPMC310, TPMC810 User manual

TPMC310, TPMC810 Engineering Manual

SJA1000 CAN Controller Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV010-SW-82':

TDRV010-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
TDRV010-SW-82-1.0.1.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

For installation the files have to be copied to the desired target directory.

The GZIP compressed archive TDRV010-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tdrv010/':

tdrv010.c	Driver source code
tdrv010def.h	Driver include file
tdrv010.h	Driver include file for application program
sja1000.h	Driver include file (CAN Controller Spec.)
include/tpxxxhwdep.c	Hardware dependent library
include/tpxxxhwdep.h	Hardware dependent library header file
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
include/config.h	Driver independent library header file
Makefile	Device driver make file
makenode	Script to create device nodes in the file system
example/tdrv010exa.c	Example application
example/Makefile	Example application make file

In order to perform an installation, extract all files of the archive TDRV010-SW-82-SRC.tar.gz to the desired target directory.

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>* enter:

  # make install

- Only after the first build we have to execute *depmod* to create a new dependency description for loadable kernel modules. This dependency file is later used by *modprobe* to automatically load dependent kernel modules.

  # 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

- Update kernel module dependency description file

  # depmod –aq

2.3 Install the device driver in the running kernel

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

  # **modprobe tdrv010drv**

- 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 dynamic 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 TDRV010 CAN Channel found. The first TDRV010 CAN Channel can be accessed with device node /dev/tdrv010_0, the second with /dev/tdrv010_1, the third with /dev/tdrv010_2 and so on.

The assignment of device nodes to physical TDRV010 modules depends on the search order of the PCI bus driver. For more details on channel assignment see # *cat /proc/tews-tdrv010*.

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 tdrv010drv –r

If your kernel has enabled a dynamic device file system like devfs or sysfs (udev), all /dev/tdrv010_x 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 "*tdrv010drv: 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

The TDRV010 driver uses dynamic allocation of major device numbers by default. If this isn't suitable for the application it is possible to define a major number for the driver. If the kernel has enabled devfs the driver will not use the symbol TDRV010_MAJOR.

To change the major number edit the file *tdrv010def.h*, change the following symbol to an appropriate value and enter **make install** to create a new driver.

TDRV010_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TDRV010_MAJOR       122
```

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

2.6 Receive Queue Configuration

Received CAN messages will be stored in a FIFO buffer. The depth of the FIFO can be adapted by changing the following symbol in *tdrv010def.h*.

TDRV010_RX_FIFO_SIZE	Defines the depth of the message FIFO buffer (default = 100). Valid numbers are in range between 1 and MAXINT.

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/tdrv010_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

E_NODEV	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

E_NODEV	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 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 tdrv010.h:

Function	Description
TDRV010_IOCXREAD	Receive a CAN message
TDRV010_IOCSWRITE	Send a CAN message
TDRV010_IOCSBITTIMING	Setup a new bit timing
TDRV010_IOCSSETFILTER	Setup acceptance filter
TDRV010_IOCBUSON	Enter the bus on state
TDRV010_IOCBUSOFF	Enter the bus off state
TDRV010_IOCFLUSH	Flush one or all receive queues
TDRV010_IOCGCANSTATUS	Returns CAN controller status information
TDRV010_IOCENABLE_SELFTEST	Enable self test mode
TDRV010_IOCDISABLE_SELFTEST	Disable self test mode
TDRV010_IOCENABLE_LISTENONLY	Enable listen only mode
TDRV010_IOCDISABLE_LISTENONLY	Disable listen only mode
TDRV010_IOCSSETLIMIT	Set new error warning limit
TDRV010_IOCTCANRESET	Set reset/operating mode (TPMC310 only)
TDRV010_IOCTCANSEL	Set silent/operating mode (TPMC310 only)
TDRV010_IOCTCANINT	Enable/disable interrupts (TPMC310 only)

See behind for more detailed information on each control code.

To use these TDRV010 specific control codes the header file tdrv010.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*.

Other function dependant error codes will be described for each ioctl code separately. Note, the TDRV010 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TDRV010_IOCXREAD

NAME

TDRV010_IOCXREAD – Read a CAN message

DESCRIPTION

This ioctl function reads a CAN message from the driver's receive queue. A pointer to the caller's message buffer (*TDRV010_MSG_BUF*) is passed by the parameter *argp* to the driver.

```
typedef struct
{
        unsigned long     Identifier;
        unsigned char     IOFlags;
        unsigned char     MsgLen;
        unsigned char     Data[8];
        long              Timeout;
        unsigned char     Status;
} TDRV010_MSG_BUF, *PTDRV010_MSG_BUF;
```

Identifier

Receives the message identifier of the read CAN message.

IOFlags

Receives CAN message attributes as a set of bit flags. The following attribute flags are possible:

Value	Description
TDRV010_EXTENDED	Set if the received message is an extended message frame. Reset for standard message frames.
TDRV010_REMOTE_FRAME	Set if the received message is a remote transmission request (RTR) frame.

MsgLen

Receives the number of message data bytes (0...8).

Data

This buffer receives up to 8 data bytes. Data[0] receives message data 0, Data[1] receives message data 1 and so on.

Timeout

Specifies the amount of time (in system ticks) the caller is willing to wait for execution of this function. A value of 0 means wait indefinitely.

Status

This parameter receives status information about overrun conditions either in the CAN controller or intermediate software FIFO.

Value	Description
TDRV010_SUCCESS	No messages lost
TDRV010_FIFO_OVERRUN	At least one message was overwritten in the receive queue FIFO. This problem occurs if the FIFO is too small for the application read interval.
TDRV010_MSGOBJ_OVERRUN	At least one message was overwritten in the CAN controller's message object because the interrupt latency is too large. Reduce the CAN bit rate or upgrade the system speed.

EXAMPLE

```
#include “tdrv010.h

int                fd;
int                result;
TDRV010_MSG_BUF    msgBuf;

msgBuf.Timeout = 200;

result = ioctl(fd, TDRV010_IOCXREAD, &msgBuf);

if (result < 0) {
     /* read operation failed. */
} else {
  /* process received CAN message */
}
```

RETURNS

On success this function returns the size of structure TDRV010_MSG_BUF. 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 size of the message buffer is too small.
EFAULT	Invalid pointer to the message buffer.
ECONNREFUSED	The controller is in bus off state and no message is available in the driver receive queue. Note, as long as CAN messages are available in the receive queue FIFO, bus off conditions are not reported by a read function. This means you can read all CAN messages out of the receive queue FIFO during bus off state without an error result.
EAGAIN	Resource temporarily unavailable; the call might work if you try again later. This error occurs only if the device is opened with the flag O_NONBLOCK set.
ETIME	The allowed time to finish the read request has elapsed.
EINTR	Interrupted function call; an asynchronous signal occurred and prevented completion of the call. When this happens, you should try the call again.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3.2 TDRV010_IOCSWRITE

NAME

TDRV010_IOCSWRITE – Write a CAN message

DESCRIPTION

This ioctl function writes a CAN message to the device specified by *filedes*. A pointer to the caller's message buffer (*TDRV010_MSG_BUF*) is passed by the parameter *argp* to the driver.

```
typedef struct
{
        unsigned long     Identifier;
        unsigned char     IOFlags;
        unsigned char     MsgLen;
        unsigned char     Data[8];
        long              Timeout;
        unsigned char     Status;
} TDRV010_MSG_BUF, *PTDRV010_MSG_BUF;
```

Identifier

Contains the message identifier of the CAN message to write.

IOFLags

Contains a set of bit flags, which define message attributes and controls the write operation. To set more than one bit flag the predefined macros may be binary ORed.

Value	Description
TDRV010_EXTENDED	Transmit an extended message frame. If this macro isn't set or the "dummy" macro TDRV010_STANDARD is set a standard frame will be transmitted.
TDRV010_REMOTE_FRAME	A remote transmission request (RTR bit is set) will be transmitted.
TDRV010_SINGLE_SHOT	No re-transmission will be performed if an error occurred or the arbitration will be lost during transmission (single-shot transmission).
TDRV010_SELF_RECEPTION	The message will be transmitted and simultaneously received if the acceptance filter is set to the corresponding identifier.

MsgLen

Contains the number of message data bytes (0..8).

Data

This buffer contains up to 8 data bytes. Data[0] contains message data 0, Data[1] contains message data 1 and so on.

Timeout

Specifies the amount of time (in system ticks) the caller is willing to wait for execution of this function.

Status

This parameter is unused for this control function.

EXAMPLE

```
#include “tdrv010.h”

...
int                fd;
int                result;
TDRV010_MSG_BUF    msgBuf;

...

/*
**   Write two data bytes with extended identifier 1234 to
**   the CANbus and wait max. 200 ticks for execution.
**   The transmitted frame will be received simultaneously.
*/
msgBuf.Identifier  = 1234;
msgBuf.Timeout     = 200;
msgBuf.IOFlags     = TDRV010_EXTENDED | TDRV010_SELF_RECEPTION;
msgBuf.MsgLen      = 2;
msgBuf.Data[0]     = 0xaa;
msgBuf.Data[1]     = 0x55;

result = ioctl(fd, TDRV010_IOCSWRITE, &msgBuf);

if (result < 0) {
  printf( "\nWrite failed --> Error = %d.\n", errno );
}
```

RETURNS

On success this function returns the size of structure TDRV010_MSG_BUF. 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 size of the message buffer is too small.
EFAULT	Invalid pointer to the message buffer.
ECONNREFUSED	The controller is in bus off state and unable to transmit messages.
EAGAIN	Resource temporarily unavailable; the call might work if you try again later. This error occurs only if the device is opened with the flag O_NONBLOCK set.
ETIME	The allowed time to finish the write request is elapsed. This occurs if currently no message object is available or if the CAN bus is overloaded and the priority of the message identifier is too low.
EINTR	Interrupted function call; an asynchronous signal occurred and prevented completion of the call. When this happens, you should try the call again.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3.3 TDRV010_IOCSBITTIMING

NAME

TDRV010_IOCSBITTIMING - Setup new bit timing

DESCRIPTION

This ioctl function modifies the bit timing register of the CAN controller to setup a new CAN bus transfer speed. A pointer to the caller's parameter buffer (*TDRV010_TIMING*) is passed by the argument pointer *argp* to the driver.

Keep in mind to setup a valid bit timing value before changing into the Bus On state.

```
typedef struct
{
        unsigned short      TimingValue;
        unsigned short      ThreeSamples;
}TDRV010_TIMING, *PTDRV010_TIMING;
```

TimingValue

This parameter holds the new value for the bit timing register 0 (bit 0...7) and for the bit timing register 1 (bit 8...15). Possible transfer rates are between 5 Kbit per second and 1 Mbit per second. The include file 'tdrv010.h' contains predefined transfer rate symbols (TDRV010_5KBIT ... TDRV010_1MBIT).
For other transfer rates please follow the instructions of the *SJA1000 Product Specification*, which is also part of the TDRV010 engineering kits.

ThreeSamples

If this parameter is TRUE (1) the CAN bus is sampled three times per bit time instead of one.

Use one sample point for faster bit rates and three sample points for slower bit rate to make the CAN bus more resistant against noise spikes.

EXAMPLE

```
#include “tdrv010.h”

int                fd;
int                result;
TDRV010_TIMING     BitTimingParam;

...

BitTimingParam.TimingValue   = TDRV010_100KBIT;
BitTimingParam.ThreeSamples  = 0;      /* FALSE */

result = ioctl(fd, TDRV010_IOCSBITTIMING, (char*)&BitTimingParam);

if (result < 0) {
  /* handle ioctl error */
}
...
```

SEE ALSO

tdrv010.h for predefined bus timing constants

SJA1000 Product Specification Manual – 6.5.1/2 BUS TIMING REGISTER

3.3.4 TDRV010_IOCSSETFILTER

NAME

TDRV010_IOCSSETFILTER - Setup acceptance filter

DESCRIPTION

This ioctl function modifies the acceptance filter of the specified CAN controller device.

The acceptance filter compares the received identifier with the acceptance filter and decides whether a message should be accepted or not. If a message passes the acceptance filter it is stored in the RXFIFO.

The acceptance filter is defined by the acceptance code registers and the acceptance mask registers. The bit patterns of messages to be received are defined in the acceptance code register.

The corresponding acceptance mask registers allow defining certain bit positions to be "don't care" (a 1 at a bit position means "don't care").

A pointer to the caller's parameter buffer (*TDRV010_FILTER*) is passed by the parameter pointer *argp* to the driver.

```
typedef struct
{
        int             SingleFilter;
        unsigned long   AcceptanceCode;
        unsigned long   AcceptanceMask;
} TDRV010_FILTER, *PTDRV010_FILTER;
```

SingleFilter

Set TRUE (1) for single filter mode. Set FALSE (0) for dual filter mode.

AcceptanceCode

The content of this parameter will be written to acceptance code register of the controller.

AcceptanceMask

The content of this parameter will be written to the acceptance mask register of the controller.

A detailed description of the acceptance filter and possible filter modes can be found in the SJA1000 Product Specification Manual.

EXAMPLE

```
#include “tdrv010.h”

int                fd;
int                result;
TDRV010_FILTER     AcceptFilter;

...

/* Mark all bit position don't care */
AcceptFilter.AcceptanceMask = 0xffffffff;

/* Not relevant because all bits are "don't care" */
AcceptFilter.AcceptanceCode = 0x0;

/* Single Filter Mode */
AcceptFilter.SingleFilter = 1;   /* TRUE */

result = ioctl(fd, TDRV010_IOCSSETFILTER, (char*)&AcceptFilter);

if (result < 0) {
  /* handle ioctl error */
}
...
```

SEE ALSO

SJA1000 Product Specification Manual – *6.4.15 ACCEPTANCE FILTER*

3.3.5 TDRV010_IOCBUSON

NAME

TDRV010_IOCBUSON - Enter the bus on state

DESCRIPTION

This ioctl function sets the specified CAN controller into the Bus On state.

After an abnormal rate of occurrences of errors on the CAN bus or after driver startup, the CAN controller enters the Bus Off state. This control function resets the "reset mode" bit in the mode register. The CAN controller begins the busoff recovery sequence and resets the transmit and receive error counters. If the CAN controller counts 128 packets of 11 consecutive recessive bits on the CAN bus, the Bus Off state is exited.

The optional argument can be omitted for this ioctl function.

Before the driver is able to communicate over the CAN bus after driver startup, this control function must be executed.

EXAMPLE

```
#include “tdrv010.h”

int fd;
int result;

...

result = ioctl(fd, TDRV010_IOCBUSON);

if (result < 0) {
  /* handle ioctl error */
}
...
```

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD)*

3.3.6 TDRV010_IOCBUSOFF

NAME

TDRV010_IOCBUSOFF - Enter the bus off state

DESCRIPTION

This ioctl function sets the specified CAN controller into the Bus Off state.

After execution of this control function the CAN controller is completely removed from the CAN bus and cannot communicate until the control function TDRV010_IOCBUSON is executed. Note: During a pending write of another concurrent thread it is not possible to set the device bus off.

The optional argument pointer can be omitted for this ioctl function.

EXAMPLE

```
#include “tdrv010.h”

int fd;
int result;

...

result = ioctl(fd, TDRV010_IOCBUSOFF);

if (result < 0) {
  /* handle ioctl error */
}
...
```

ERRORS

EBUSY	Another concurrent thread is writing to the device. Try it again later.
EIO	Unable to enter the BUSOFF mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD)*

3.3.7 TDRV010_IOCFLUSH

NAME

TDRV010_IOCFLUSH - Flush the received message FIFO

DESCRIPTION

This ioctl function flushes the FIFO buffer of received messages.

The optional argument pointer can be omitted for this ioctl function.

EXAMPLE

```
#include “tdrv010.h”

int fd;
int result;

...

result = ioctl(fd, TDRV010_IOCFLUSH);

if (result < 0) {
  /* handle ioctl error */
}
...
```

3.3.8 TDRV010_IOCGCANSTATUS

NAME

TDRV010_IOCGCANSTATUS - Returns CAN controller status information

DESCRIPTION

This ioctl function returns the actual contents of several CAN controller registers for diagnostic purposes. A pointer to the caller's status buffer (*TDRV010_STATUS*) is passed by the parameter *argp*.

```
typedef struct
{
    unsigned char    ArbitrationLostCapture;
    unsigned char    ErrorCodeCapture;
    unsigned char    TxErrorCounter;
    unsigned char    RxErrorCounter;
    unsigned char    ErrorWarningLimit;
    unsigned char    StatusRegister;
    unsigned char    ModeRegister;
    unsigned char    RxMessageCounterMax;
    int              ModuleType;
} TDRV010_STATUS, *PTDRV010_STATUS;
```

ArbitrationLostCapture

This parameter receives content of the arbitration lost capture register. This register contains information about the bit position of losing arbitration.

ErrorCodeCapture

This parameter receives content of the error code capture register. This register contains information about the type and location of errors on the bus.

TxErrorCounter

This parameter receives content of the TX error counter register. This register contains the current value of the transmit error counter.

RxErrorCounter

This parameter receives content of the RX error counter register. This register contains the current value of the receive error counter.

ErrorWarningLimit

This parameter receives content of the error warning limit register.

StatusRegister

This parameter receives content of the status register.

ModeRegister

This parameter receives the content of the mode register.

RxMessageCounterMax

Contains the peak value of messages in the RXFIFO. This internal counter value will be reset to 0 after reading.

ModuleType

This parameter returns “310” for TPMC310 and “810” for TPMC810 CAN controller carrier boards. For detailed channel location information see /proc/tews-tdrv010 file system entry, which is part of the driver diagnostic.

EXAMPLE

```
#include “tdrv010.h”

int                fd;
int                result;
TDRV010_STATUS     CanStatus;
...

result = ioctl(fd, TDRV010_IOCGCANSTATUS, (char*)&CanStatus);

if (result < 0) {
  /* handle ioctl error */
}
...
```

SEE ALSO

SJA1000 Product Specification Manual

3.3.9 TDRV010_IOCENABLE_SELFTEST

NAME

TDRV010_IOCENABLE_SELFTEST - Enable self test mode

DESCRIPTION

This ioctl function enables the self test facility of the SJA1000 CAN controller.

In this mode a full node test is possible without any other active node on the bus using the self reception facility. The CAN controller will perform a successful transmission even if there is no acknowledge received.

Also in self test mode the normal functionality is given, that means the CAN controller is able to receive messages from other nodes and can transmit message to other nodes if any connected.

The optional argument pointer can be omitted for this ioctl function.

This ioctl command will be accepted only in reset mode (BUSOFF). Enter TDRV010_IOCBUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include “tdrv010.h”

int fd;
int result;

result = ioctl(fd, TDRV010_IOCENABLE_SELFTEST);

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

ERRORS

EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD)*

3.3.10 TDRV010_IOCDISABLE_SELFTEST

NAME

TDRV010_IOCDISABLE_SELFTEST - Disable self test mode

DESCRIPTION

This ioctl function disables the self test facility of the SJA1000 CAN controller, which was before enabled with the ioctl command TDRV010_IOCENABLE_SELFTEST.

The optional argument pointer can be omitted for this function.

This ioctl command will be accepted only in reset mode (BUSOFF). Enter TDRV010_IOCBUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include “tdrv010.h”

int fd;
int result;

...
result = ioctl(fd, TDRV010_IOCDISABLE_SELFTEST);

if (result < 0) {
  /* handle ioctl error */
}
...
```

ERRORS

EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD)*

3.3.11 TDRV010_IOCENABLE_LISTENONLY

NAME

TDRV010_IOCENABLE_LISTENONLY - Enable listen only mode

DESCRIPTION

This ioctl function enables the listen only facility of the SJA1000 CAN controller.

In this mode the CAN controller would give no acknowledge to the CAN-bus, even if a message is received successfully. Message transmission is not possible. All other functions can be used like in normal mode.

This mode can be used for software driver bit rate detection and 'hot-plugging'.

The optional argument pointer can be omitted for this ioctl function.

This ioctl command will be accepted only in reset mode (BUSOFF). Enter TDRV010_IOCBUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include “tdrv010.h”

int fd;
int result;

result = ioctl(fd, TDRV010_IOCENABLE_LISTENONLY);

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

ERRORS

EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD)*

3.3.12 TDRV010_IOCDISABLE_LISTENONLY

NAME

TDRV010_IOCDISABLE_LISTENONLY - Disable listen only mode

DESCRIPTION

This ioctl function disables the listen only facility of the SJA1000 CAN controller, which was enabled before with the ioctl command TDRV010_IOCENABLE_ LISTENONLY.

The optional argument pointer can be omitted in this ioctl function.

This ioctl command will be accepted only in reset mode (BUSOFF). Enter TDRV010_IOCBUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include “tdrv010.h”

int fd;
int result;

...

result = ioctl(fd, TDRV010_DISABLE_LISTENONLY);

if (result < 0) {
  /* handle ioctl error */
}
...
```

ERRORS

EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD)*

3.3.13 TDRV010_IOCSSETLIMIT

NAME

TDRV010_IOCSSETLIMIT - Set new error warning limit

DESCRIPTION

This ioctl function sets a new error warning limit in the corresponding CAN controller register. The default value (after hardware reset) is 96.

The new error warning limit will be set in an unsigned char variable. A pointer to this variable is passed by the argument pointer *argp* to the driver.

This ioctl command will be accepted only in reset mode (BUSOFF). Enter TDRV010_IOCBUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include “tdrv010.h”

int           fd;
int           result;
unsigned char limit;

...
limit = 200;
result = ioctl(fd, TDRV010_IOCSSETLIMIT, (char*)&limit);

if (result < 0) {
  /* handle ioctl error */
}
...
```

ERRORS

EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.3 MODE REGISTER (MOD)*

3.3.14 TDRV010_IOCTCANRESET

NAME

TDRV010_IOCTCANRESET – Set reset/operating mode (TPMC310 only)

DESCRIPTION

This ioctl function sets the certain CAN controller in reset or operating mode. The function specific control parameter *argp* specifies the new configuration. This function is only available for TPMC310 devices.

argp

0 to set the certain CAN channel in reset mode
1 to set the certain CAN channel in operating mode

EXAMPLE

```
#include “tdrv010.h”
…
int  fd;
int  result;
…
/*----------------------------------
  Execute ioctl() function
  Set the controller in reset mode
  --------------------------------*/
result = ioctl(fd, TDRV010_IOCTCANRESET, (char *)0);
/* for operating mode*/
/* result = ioctl(fd, TDRV010_IOCTCANRESET, (char *)1); */
if (result >= 0)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERRORS

EINVAL	Unsupported ioctl command. This ioctl command is for TPMC310 devices only.
EIO	Unable to enter the BUSOFF mode during initialization.

SEE ALSO

TPMC310 User Manual

3.3.15 TDRV010_IOCTCANSEL

NAME

TDRV010_IOCTCANSEL – Set silent/operating mode (TPMC310 only)

DESCRIPTION

This ioctl function sets the certain CAN controller in silent or operating mode. The function specific control parameter *argp* specifies the new configuration. This function is only available for TPMC310 devices.

argp

0 to set the certain CAN channel in silent mode
1 to set the certain CAN channel in operating mode

EXAMPLE

```
#include “tdrv010.h”

int  fd;
int result;

/*-----------------------------------------------
  Execute ioctl() function
  Set the certain CAN controller in silent mode
  -----------------------------------------------*/
result = ioctl(fd, TDRV010_IOCTCANSEL, (char *)0);
/* for operating mode */
/* result = ioctl(fd, TDRV010_IOCTCANSEL, (char *)1); */
if (result >= 0)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERRORS

EINVAL	Unsupported ioctl command. This ioctl command is for TPMC310 devices only.

SEE ALSO

TPMC310 User Manual

3.3.16 TDRV010_IOCTCANINT

NAME

TDRV010_IOCTCANINT – Enable/disable interrupts (TPMC310 only)

DESCRIPTION

This I/O control function enables or disables the certain CAN controller interrupts. The function specific control parameter *argp* specifies the new configuration. This function is only available for TPMC310 devices.

argp

0 to disable the certain CAN channel interrupt
1 to enable the certain CAN channel interrupt

EXAMPLE

```
#include “tdrv010.h”

int  fd;
int  result;

/*-----------------------------------------------------
  Execute ioctl() function
  Disable the interrupts of the certain CAN controller
  -----------------------------------------------------*/

result = ioctl(fd, TDRV010_IOCTCANINT, (char *)0);
/* to enable interrupts */
/* result = ioctl(fd, TDRV010_IOCTCANINT, (char *)1); */
if (result >= 0)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

ERRORS

EINVAL	Unsupported ioctl command. This ioctl command is for TPMC310 devices only.

SEE ALSO

TPMC310 User Manual

3.4 Step by Step Driver Initialization

The following code example illustrates all necessary steps to initialize a CAN device for communication.

```
/*
**  ( 1.) Setup CAN bus bit timing
*/
BitTimingParam.TimingValue   = TDRV010_100KBIT;
BitTimingParam.ThreeSamples  = 0;      /* FALSE */

result = ioctl(fd, TDRV010_IOCSBITTIMING, (char*)&BitTimingParam);



/*
**  ( 2.) Setup acceptance filter masks
*/
AcceptFilter.AcceptanceCode  = 0x0;
AcceptFilter.AcceptanceMask  = 0xFFFFFFFF;
AcceptFilter.SingleFilter    = 1;

result = ioctl(fd, TDRV010_IOCSSETFILTER, (char*)&AcceptFilter);



/*
**  ( 3.) Enter Bus On State
*/

result = ioctl(fd, TDRV010_IOCBUSON);
```

Now you should be able to send and receive CAN messages with appropriate calls to TDRV010_IOCSWRITE and TDRV010_IOCXREAD ioctl functions.

4 Diagnostic

If the TDRV010 does not work properly it is helpful to get some status information from the driver respective kernel. To get debug output from the driver enable the following symbols in '*tdrv010.c*' by replacing "#undef" with "#define":

```
#define DEBUG_TDRV010
#define DEBUG_TDRV010_INTR
```

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 TDRV010 driver (see also the proc man pages).

```
# tail –f /var/log/messages /* before modprobing the TDRV010 driver */


May  9 09:03:30 linuxsmp2 kernel: TEWS TECHNOLOGIES - TDRV010 Isolated 2x
CAN Bus -  version 1.0.x (<Release Date>)
May  9 09:03:30 linuxsmp2 kernel: TDRV010:  Probe new device
(vendor=0x1498, device=0x0136, type=310)
May  9 09:03:30 linuxsmp2 kernel: TDRV010:  Probe new device
(vendor=0x1498, device=0x032A, type=810)
/* if SYSFS + UDEV is present */
May  9 09:03:30 linuxsmp2 udev[3674]: creating device node '/dev/tdrv010_0'
May  9 09:03:30 linuxsmp2 udev[3676]: creating device node '/dev/tdrv010_1'
May  9 09:03:30 linuxsmp2 udev[3688]: creating device node '/dev/tdrv010_2'
May  9 09:03:30 linuxsmp2 udev[3689]: creating device node '/dev/tdrv010_3'
...

/* after modprobing the TDRV010 driver */

# cat /proc/tews-tdrv010 /* advanced CAN channel status information */
TEWS TECHNOLOGIES - TDRV010 Isolated 2x CAN Bus -  version 1.0.0 (2007-05-
09)
Supported modules: TPMC310, TPMC810

Registered SJA1000 CAN controller channels:
/dev/tdrv010_0 (phy: TPMC310 #0, mod:01 stat:3C rec:00 tec:00 alc:00 ecc:00
ewl:60, RxFIFO[rd:0,wr:0,pk#0:])
/dev/tdrv010_1 (phy: TPMC310 #1, mod:01 stat:3C rec:00 tec:00 alc:00 ecc:00
ewl:60, RxFIFO[rd:0,wr:0,pk#0:])
/dev/tdrv010_2 (phy: TPMC810 #0, mod:01 stat:3C rec:00 tec:00 alc:00 ecc:00
ewl:60, RxFIFO[rd:0,wr:0,pk#0:])
/dev/tdrv010_3 (phy: TPMC810 #1, mod:01 stat:3C rec:00 tec:00 alc:00 ecc:00
ewl:60, RxFIFO[rd:0,wr:0,pk#0:])

/*
     phy = carrier + #channel
     mod = mode register
```

```
     stat = status register
     rec = receive error counter
     tec = transmit error counter
     alc = arbitration lost capture
     ecc = error code capture
     ewl = actual error warning limit
     RxFIFO
          rd = FIFO read pointer
          wr = FIFO write pointer
          pk = FIFO message counter peak value
*/

# cat /proc/pci
.../* TPMC310 */
  Bus  2, device   8, function  0:
    Class 0280: PCI device 1498:0136 (rev 0).
      IRQ 177.
      Non-prefetchable 32 bit memory at 0xff5fe400 [0xff5fe47f].
      I/O at 0xa800 [0xa87f].
      Non-prefetchable 32 bit memory at 0xff5fe000 [0xff5fe00f].
      Non-prefetchable 32 bit memory at 0xff5fdc00 [0xff5fddff].
.../* TPMC810 */
  Bus  2, device   9, function  0:
    Class 0280: PCI device 1498:032a (rev 0).
      IRQ 169.
      Non-prefetchable 32 bit memory at 0xff5fec00 [0xff5fec7f].
      I/O at 0xa880 [0xa8ff].
      Non-prefetchable 32 bit memory at 0xff5fe800 [0xff5fe9ff].
```

```
# cat /proc/interrupts
           CPU0       CPU1
  0:    5860733    5901379  IO-APIC-edge  timer
  1:       2099       1872  IO-APIC-edge  i8042
  2:          0          0  XT-PIC  cascade
  8:          0          1  IO-APIC-edge  rtc
  9:          2          0  IO-APIC-level  acpi
 12:      50793      50084  IO-APIC-edge  i8042
 14:     155677     148926  IO-APIC-edge  ide0
169:     712307     709746  IO-APIC-level  radeon@PCI:1:0:0, TDRV010
177:          0          2  IO-APIC-level  uhci_hcd, AMD AMD8111, TDRV010
185:      25775         31  IO-APIC-level  uhci_hcd, eth0
193:          0          1  IO-APIC-level  libata, ehci_hcd, ..., TDRV010
NMI:          0          0
LOC:   11763048   11763049
ERR:          0
MIS:          0

# cat /proc/iomem
...
  /* TPMC310 */
  ff5fdc00-ff5fddff : 0000:02:08.0
    ff5fdc00-ff5fddff : TDRV010CAN
  ff5fe000-ff5fe00f : 0000:02:08.0
    ff5fe000-ff5fe00f : TDRV010PLD
  ff5fe400-ff5fe47f : 0000:02:08.0
  /* TPMC810 */
  ff5fe800-ff5fe9ff : 0000:02:09.0
    ff5fe800-ff5fe9ff : TDRV010CAN
...
```