

# TDRV011-SW-82

## Linux Device Driver

Extended CAN Bus

Version 1.0.x

## User Manual

Issue 1.0.1

March 2008

---

**TEWS TECHNOLOGIES GmbH**

Am Bahnhof 7  
25469 Halstenbek, Germany  
[www.tews.com](http://www.tews.com)

Phone: +49-(0)4101-4058-0  
Fax: +49-(0)4101-4058-19  
e-mail: [info@tews.com](mailto:info@tews.com)

**TEWS TECHNOLOGIES LLC**

9190 Double Diamond Parkway,  
Suite 127, Reno, NV 89521, USA  
[www.tews.com](http://www.tews.com)

Phone: +1 (775) 850 5830  
Fax: +1 (775) 201 0347  
e-mail: [usasales@tews.com](mailto:usasales@tews.com)

**TDRV011-SW-82**

Linux Device Driver

Extended CAN Bus

Supported Modules:

TPMC316

TPMC816

TPMC901

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2007-2008 by TEWS TECHNOLOGIES GmbH

<b>Issue</b>	<b>Description</b>	<b>Date</b>
1.0.0	First Issue	May 3, 2007
1.0.1	Filelist and installation section modified	March 4, 2008

# Table of Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>4</b>
<b>2</b>	<b>INSTALLATION.....</b>	<b>5</b>
	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
<b>3</b>	<b>DEVICE INPUT/OUTPUT FUNCTIONS .....</b>	<b>8</b>
	3.1 open() .....	8
	3.2 close().....	10
	3.3 ioctl() .....	11
	3.3.1 TDRV011_IOCXREAD.....	13
	3.3.2 TDRV011_IOCWRITE .....	16
	3.3.3 TDRV011_IOCSEBTIMING .....	19
	3.3.4 TDRV011_IOCSEBTFILTER .....	21
	3.3.5 TDRV011_IOCGETFILTER .....	23
	3.3.6 TDRV011_IOCSEBON .....	25
	3.3.7 TDRV011_IOCSEBOFF .....	26
	3.3.8 TDRV011_IOCSEBFLUSH .....	27
	3.3.9 TDRV011_IOCSEBSTATUS .....	28
	3.3.10 TDRV011_IOCSEBFRXBUF.....	29
	3.3.11 TDRV011_IOCSEBFRMTBUF .....	31
	3.3.12 TDRV011_IOCSEBUPDATEBUF .....	33
	3.3.13 TDRV011_IOCSEBRELESEBUF .....	35
<b>4</b>	<b>DIAGNOSTIC.....</b>	<b>37</b>

# 1 Introduction

The TDRV011-SW-82 Linux device driver allows the operation of the TDRV011 Extended 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.

The TDRV011-SW-82 device driver supports the following features:

- Transmission and reception of Standard and Extended CAN Messages
- Up to 15 receive message queues with user defined size
- Variable allocation of receive message objects to receive queues
- Separate task queues for each receive queue and transmission buffer message object
- Standard bit rates from 5 kbit up to 1.0 Mbit and user defined bit rates
- Message acceptance filtering
- Definition of receive and remote buffer message objects
- Designed as Linux kernel module with dynamically loading
- Creates devices with dynamically allocated or fixed major device numbers
- DEVFS and SYSFS (UDEV) support for automatic device node creation

The TDRV011-SW-82 device driver supports the modules listed below:

TPMC316	CAN Bus	PMC Conduction Cooled
TPMC816	Two Independent Channels Extended CAN Bus	PMC
TPMC901	6/4/2 Channels Extended CAN Bus	PMC

**In this document all supported modules and devices will be called TDRV011. Specials for a certain device will be advised.**

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

- TPMC316, TPMC816 and TPMC901 Hardware User manual
- TPMC316, TPMC816 and TPMC901 Engineering Manual

## 2 Installation

The directory TDRV011-SW-82 on the distribution media contains the following files:

TDRV011-SW-82-1.0.1.pdf	This manual in PDF format
TDRV011-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

The GZIP compressed archive TDRV011-SW-82-SRC.tar.gz contains the following files and directories:

Directory path './tdrv011/':

tdrv011.c	Driver source code
tdrv011def.h	Driver include file
tdrv011.h	Driver include file for application program
l82527.h	Driver include file (CAN Controller Spec.)
Makefile	Device driver make file
makenode	Script to create device nodes in the file system
include/config.h	Driver independent library header file
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
example/tdrv011exa.c	Example application
example/Makefile	Example application make file

In order to perform an installation, extract all files of the archive TDRV011-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzf TDRV011-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy *tdrv011.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>* 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 tdrv011drv**
- 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 TDRV011 CAN Channel found. The first TDRV011 CAN Channel can be accessed with device node */dev/tdrv011\_0*, the second with */dev/tdrv011\_1*, the third with */dev/tdrv011\_2* and so on.

The assignment of device nodes to physical TDRV011 modules depends on the search order of the PCI bus driver. For more details on channel assignment see **# cat /proc/tews-tdrv011**.

## 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 tdrv011drv -r**

If your kernel has enabled devfs or sysfs (udev), all */dev/tdrv011\_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 "*tdrv011drv: 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 TDRV011 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 TDRV011\_MAJOR.

To change the major number edit the file *tdrv011def.h*, change the following symbol to appropriate value and enter **make install** to create a new driver.

TDRV011\_MAJOR            Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TDRV011_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 receive queues. Each receive queue contains a FIFO and a separate task wait queue. The number of receive queues and the depth of the FIFO can be adapted by changing the following symbols in *tdrv011def.h*.

NUM\_RX\_QUEUES        Defines the number of receive queues for each device (default = 3). Valid numbers are in range between 1 and 15.

RX\_FIFO\_SIZE         Defines the depth of the message FIFO inside each receive queue (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 used for communication over the CAN Bus.

### 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/tdrv011_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 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 *tdrv011.h*:

Function	Description
TDRV011_IOCXREAD	Receive a CAN message
TDRV011_IOCWRITE	Send a CAN message
TDRV011_IOCSEBTIMING	Setup new bit timing
TDRV011_IOCSETFILTER	Setup acceptance filter masks
TDRV011_IOCGETFILTER	Get the current acceptance filter masks
TDRV011_IOCBUSON	Enter the bus on state
TDRV011_IOCBUSOFF	Enter the bus off state
TDRV011_IOCFLUSH	Flush one or all receive queues
TDRV011_IOCSCANSTATUS	Returns the contents of the CAN controller status register
TDRV011_IOCSEDEFRXBUF	Define a receive buffer message object
TDRV011_IOCSEDEFRTMBUF	Define a remote transmit buffer message object
TDRV011_IOCUPDATEBUF	Update a remote or receive buffer message object
TDRV011_IOCRELEASEBUF	Release an allocated message buffer object

See behind for more detailed information on each control code.

**To use these TDRV011 specific control codes the header file *tdrv011.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 TDRV011 driver always returns standard Linux error codes.

## SEE ALSO

GNU C Library description – Low-Level Input/Output, ioctl man pages

### 3.3.1 TDRV011\_IOCTLXREAD

#### NAME

TDRV011\_IOCTLXREAD – Receive a CAN message

#### DESCRIPTION

This ioctl function reads a CAN message from the specified receive queue. A pointer to the caller's message buffer (*TDRV011\_MSG\_BUF*) is passed by the parameter *argp* to the driver.

typedef struct

```
{
    unsigned long    identifier;
    long            timeout;
    unsigned char    rx_queue_num;
    unsigned char    extended;
    unsigned char    status;
    unsigned char    msg_len;
    unsigned char    data[8];
} TDRV011_MSG_BUF, *PTDRV011_MSG_BUF;
```

*identifier*

Receives the message identifier of the read CAN message.

*timeout*

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

*rx\_queue\_num*

Specifies the receive queue number from which the data will be read. Valid receive queue numbers are in range between 1 and *n*, in which *n* depends on the definition of *NUM\_RX\_QUEUES* (see also 2.6).

*extended*

Receives TRUE for extended CAN messages.

*status*

Receives status information about overrun conditions either in the CAN controller or intermediate software FIFO's.

<b>Value</b>	<b>Description</b>
TDRV011_SUCCESS	No messages lost
TDRV011_FIFO_OVERRUN	One or more messages was overwritten in the receive queue FIFO. This problem occurs if the FIFO is too small for the application read interval.
TDRV011_MSGOBJ_OVERRUN	One or more messages were overwritten in the CAN controller message object because the interrupt latency is too large. Keep in mind Linux isn't a real-time operating system. Use message object 15 (buffered) to receive this time critical CAN messages, reduce the CAN bit rate or upgrade the system speed.
TDRV011_RAW_FIFO_OVERRUN	One or more messages was overwritten in the FIFO between the interrupt service routine and post-processing in the driver (bottom half).

*msg\_len*

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.

**EXAMPLE**

```
#include "tdrv011.h"

int          fd;
ssize_t      NumBytes;
TDRV011_MSG_BUF MsgBuf;

MsgBuf.rx_queue_num = 1;
MsgBuf.timeout      = 200;

NumBytes = ioctl(fd, TDRV011_IOCXREAD, &MsgBuf);

if (NumBytes > 0) {
    /* process received CAN message */
}
```

## RETURNS

On success this function returns the size of structure `TDRV011_MSG_BUF`. In case of an error, a value of `-1` is returned. The global variable `errno` contains the detailed error code.

## ERRORS

<code>EINVAL</code>	Invalid argument. This error code is returned if the size of the message buffer is too small.
<code>ECHRNG</code>	The specified receive queue number is out of range.
<code>EFAULT</code>	Invalid pointer to the message buffer.
<code>ECONNREFUSED</code>	The controller is in bus off state and no message is available in the specified receive queue. Note, as long as CAN messages are available in the receive queue FIFO, bus off conditions were 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.
<code>EAGAIN</code>	Resource temporarily unavailable; the call might work if you try again later. This error occurs only if the device is opened with the flag <code>O_NONBLOCK</code> set.
<code>ETIME</code>	The allowed time to finish the read request has elapsed.
<code>EINTR</code>	Interrupted function call; an asynchronous signal occurred and prevented completion of the call. When this happens, you should try the call again.

### 3.3.2 TDRV011\_IOCWRITE

#### NAME

TDRV011\_IOCWRITE – Send a CAN message

#### DESCRIPTION

This ioctl function writes a CAN message to the specified CAN device. A pointer to the caller's message buffer (*TDRV011\_MSG\_BUF*) is passed by the parameter *argp* to the driver.

This ioctl function dynamically allocates a free message object for this transmit operation. The search begins at message object 1 and ends at message object 14. The first free message object found is used. If currently no message object is available the write operation is blocked until any message object becomes free or a timeout occurs.

typedef struct

```
{
    unsigned long    identifier;
    long            timeout;
    unsigned char    rx_queue_num;
    unsigned char    extended;
    unsigned char    status;
    unsigned char    msg_len;
    unsigned char    data[8];
} TDRV011_MSG_BUF, *PTDRV011_MSG_BUF;
```

*identifier*

Contains the message identifier of the CAN message to write.

*timeout*

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

*rx\_queue\_num*

Unused for this control function.

*extended*

Contains TRUE (1) for extended CAN messages.

*status*

Unused for this control function.

*msg\_len*

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.

**EXAMPLE**

```
#include "tdrv011.h"

int          fd;
ssize_t     NumBytes;
TDRV011_MSG_BUF MsgBuf;

MsgBuf.identifier = 1234;
MsgBuf.timeout    = 200;
MsgBuf.extended   = TRUE;
MsgBuf.msg_len    = 2;
MsgBuf.data[0]    = 0xaa;
MsgBuf.data[1]    = 0x55;

NumBytes = ioctl(fd, TDRV011_IOCWRITE, &MsgBuf);

if (NumBytes > 0) {
    /* CAN message successfully transmitted */
}
```

## RETURNS

On success this function returns the size of structure `TDRV011_MSG_BUF`. In case of an error, a value of `-1` is returned by `ioctl()`. The global variable `errno` contains the detailed error code.

## ERRORS

<code>EINVAL</code>	Invalid argument. This error code is returned if the size of the message buffer is too small.
<code>EFAULT</code>	Invalid pointer to the message buffer.
<code>ECONNREFUSED</code>	The controller is in bus off state and unable to transmit messages.
<code>EAGAIN</code>	Resource temporarily unavailable; the call might work if you try again later. This error occurs only if the device is opened with the flag <code>O_NONBLOCK</code> set.
<code>ETIME</code>	The allowed time to finish the write request has 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.
<code>EINTR</code>	Interrupted function call; an asynchronous signal occurred and prevented completion of the call. When this happens, you should try the call again.

### 3.3.3 TDRV011\_IOCSEBITTIMING

#### NAME

TDRV011\_IOCSEBITTIMING - 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 (*TDRV011\_BITTIMING*) is passed by the argument *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    timing_value;
    unsigned short    three_samples;
} TDRV011_BITTIMING, *PTDRV011_BITTIMING;
```

*timing\_value*

This parameter holds the new values 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 1.0 MBit per second. The include file *tdrv011.h* contains predefined transfer rate symbols (TDRV011\_5KBIT ... TDRV011\_1\_0MBIT).

For other transfer rates please follow the instructions of the Intel 82527 Architectural Overview, which is also part of the engineering documentation.

*three\_samples*

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 rates to make the CAN bus more resistant against noise spikes.**

## EXAMPLE

```
#include "tdrv011.h"

int          fd;
int          result;
TDRV011_BITTIMING BitTimingParam;

BitTimingParam.timing_value  = TDRV011_100KBIT;
BitTimingParam.three_samples = FALSE;
result = ioctl(fd, TDRV011_IOCSBITTIMING, &BitTimingParam);

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

## ERRORS

EFAULT

Invalid pointer to the parameter buffer. Please check the argument *argp*.

## SEE ALSO

tdrv011.h for predefined bus timing constants

Intel 82527 Architectural Overview - 4.13 *Bit Timing Overview*

### 3.3.4 TDRV011\_IOCSETFILTER

#### NAME

TDRV011\_IOCSETFILTER - Setup acceptance filter masks

#### DESCRIPTION

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

The acceptance masks allow message objects to receive messages with a larger range of message identifiers instead of just a single message identifier. A "0" value means "don't care", or accept a "0" or "1" for that bit position. A "1" value means that the incoming bit value "must-match" identically to the corresponding bit in the message identifier.

A pointer to the caller's parameter buffer (*TDRV011\_ACCEPT\_MASKS*) is passed by the parameter *argp* to the driver.

```
typedef struct
{
    unsigned long    message_15_mask;
    unsigned long    global_mask_extended;
    unsigned short   global_mask_standard;
} TDRV011_ACCEPT_MASKS, *PTDRV011_ACCEPT_MASKS;
```

#### *message\_15\_mask*

This parameter specifies the value for the Message 15 Mask Register. The Message 15 Mask Register is a local mask for message object 15. This 29 bit identifier mask appears in bit 3...31 of this parameter. The Message 15 Mask is "ANDed" with the Global Mask. This means that any bit defined as "don't care" in the Global Mask will automatically be a "don't care" bit for message 15. (See also Intel 82527 Architectural Overview).

#### *global\_mask\_extended*

This parameter specifies the value for the Global Mask-Extended Register. The Global Mask-Extended Register applies only to messages using the extended CAN identifier. This 29 bit identifier mask appears in bit 3...31 of this parameter.

#### *global\_mask\_standard*

This parameter specifies the value for the Global Mask-Standard Register. The Global Mask-Standard Register applies only to messages using the standard CAN identifier. The 11 bit identifier mask appears in bit 5...15 of this parameter.

**The TDRV011 device driver copies the parameter directly into the corresponding registers of the CAN controller, without shifting any bit positions. For more information see the Intel 82527 Architectural Overview - 4.7...4.10**

## EXAMPLE

```
#include "tdrv011.h"

int          fd;
int          result;
TDRV011_ACCEPT_MASKS AcceptMasksParam;

/* Standard identifier bits 0..3 don't care */
AcceptMasksParam.global_mask_standard = 0xfe00;

/* extended identifier bits 0..3 don't care */
AcceptMasksParam.global_mask_extended = 0xffffffff80;

/* Message object 15 identifier bits 0..7 don't care */
AcceptMasksParam.message_15_mask = 0xffffffff800;

result = ioctl(fd, TDRV011_IOCSETFILTER, &AcceptMasksParam);

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

## ERRORS

EFAULT	Invalid pointer to the parameter buffer. Please check the argument <i>argp</i> .
--------	--

## SEE ALSO

Intel 82527 Architectural Overview - 4.9 Acceptance Filtering

### 3.3.5 TDRV011\_IOCTLGETFILTER

#### NAME

TDRV011\_IOCTLGETFILTER - Get the current acceptance filter masks

#### DESCRIPTION

This ioctl function returns the current acceptance filter masks of the specified CAN Controller.

A pointer to the caller's parameter buffer (*TDRV011\_ACCEPT\_MASKS*) is passed by the parameter *argp* to the driver.

```
typedef struct
{
    unsigned long    message_15_mask;
    unsigned long    global_mask_extended;
    unsigned short   global_mask_standard;
} TDRV011_ACCEPT_MASKS, *PTDRV011_ACCEPT_MASKS;
```

#### *message\_15\_mask*

This parameter receives the value for the Message 15 Mask Register. The Message 15 Mask Register is a local mask for message object 15. This 29 bit identifier mask appears in bit 3...31 of this parameter.

#### *global\_mask\_extended*

This parameter receives the value for the Global Mask-Extended Register. The Global Mask-Extended Register applies only to messages using the extended CAN identifier. This 29 bit identifier mask appears in bit 3...31 of this parameter.

#### *global\_mask\_standard*

This parameter receives the value for the Global Mask-Standard Register. The Global Mask-Standard Register applies only to messages using the standard CAN identifier. The 11 bit identifier mask appears in bit 5...15 of this parameter.

**The TDRV011 device driver copies the masks directly from the corresponding registers of the CAN controller into the parameter buffer, without shifting any bit positions. For more information see the Intel 82527 Architectural Overview - 4.7...4.10**

## EXAMPLE

```
#include "tdrv011.h"

int          fd;
int          result;
TDRV011_ACCEPT_MASKS AcceptMasksParam;

result = ioctl(fd, TDRV011_IOCTLGETFILTER, &AcceptMasksParam);

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

## ERRORS

EFAULT	Invalid pointer to the parameter buffer. Please check the argument <i>argp</i> .
--------	--

## SEE ALSO

Intel 82527 Architectural Overview - *4.9 Acceptance Filtering*

### 3.3.6 TDRV011\_IOCBUSON

#### NAME

TDRV011\_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 init bit in the control 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 "tdrv011.h"

int fd;
int result;

result = ioctl(fd, TDRV011_IOCBUSON);

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

#### SEE ALSO

Intel 82527 Architectural Overview - 3.2 *Software Initialization*

### 3.3.7 TDRV011\_IOCBUSOFF

#### NAME

TDRV011\_IOCBUSOFF - Enter the bus off state

#### DESCRIPTION

This ioctl function sets the specified CAN controller into the Bus Off state. After a successful execution of this control function the CAN controller is completely removed from the CAN bus and cannot communicate until the control function TDRV011\_IOCBUSON is executed. It is not possible to set the device bus off during a write operation of another concurrent process.

The optional argument can be omitted for this ioctl function.

**Execute this control function before the last close to the CAN controller channel.**

#### EXAMPLE

```
#include "tdrv011.h"

int fd;
int result;

result = ioctl(fd, TDRV011_IOCBUSOFF);

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

#### ERRORS

EBUSY

Device busy. Another concurrent process is writing to the device at the moment. Try it again later.

#### SEE ALSO

Intel 82527 Architectural Overview - 3.2 *Software Initialization*

### 3.3.8 TDRV011\_IOCFLUSH

#### NAME

TDRV011\_IOCFLUSH - Flush one or all receive queues

#### DESCRIPTION

This ioctl function flushes the message FIFO of the specified receive queue(s).

The optional argument *argp* passes the receive queue number to the device driver on which the FIFO is to be flushed. If this parameter is 0 the FIFOs of all receive queues of the device will be flushed, otherwise only the FIFO of the specified receive queue will be flushed.

#### EXAMPLE

```
#include "tdrv011.h"

int fd;
int result;

/* flush all receive queues */

result = ioctl(fd, TDRV011_IOCFLUSH, (int)0);

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

#### ERRORS

EINVAL

Invalid argument. This error code is returned if the specified receive queue is out of range.

### 3.3.9 TDRV011\_IOCPCANSTATUS

#### NAME

TDRV011\_IOCPCANSTATUS - Returns the contents of the CAN status register

#### DESCRIPTION

This ioctl function returns the current content of the CAN controller status register for diagnostic purposes.

The content of the controller status register is received in an unsigned char variable. A pointer to this variable is passed by the argument *argp* to the driver.

#### EXAMPLE

```
#include "tdrv011.h"

int          fd;
int          result;
unsigned char CanStatus;

result = ioctl(fd, TDRV011_IOCPCANSTATUS, &CanStatus);

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

#### ERRORS

EFAULT

Invalid pointer to the unsigned char variable which receives the contents of the CAN status register. Please check the argument *argp*.

#### SEE ALSO

Intel 82527 Architectural Overview - 4.3 status Register (01H)

### 3.3.10 TDRV011\_IOCSEDFRXBUF

#### NAME

TDRV011\_IOCSEDFRXBUF - Define a receive buffer message object

#### DESCRIPTION

This ioctl function defines a CAN message object to receive a single message identifier or a range of message identifiers (see also Acceptance Mask). All CAN messages received by this message object are directed to the associated receive queue and can be read with the standard read function (see also 3.3.1).

Before the driver can receive CAN messages it is necessary to define at least one receive message object. If only one receive message object is defined at all preferably message object 15 should be used because this message object is buffered.

A pointer to the caller's message description (*TDRV011\_BUF\_DESC*) is passed by the argument *argp* to the driver.

```
typedef struct
{
    unsigned long    identifier;
    unsigned char    msg_obj_num;
    unsigned char    rx_queue_num;
    unsigned char    extended;
    unsigned char    msg_len;
    unsigned char    data[8];
} TDRV011_BUF_DESC, *PTDRV011_BUF_DESC;
```

#### *identifier*

Specifies the message identifier for the message object to be defined.

#### *msg\_obj\_num*

Specifies the number of the message object to be defined. Valid object numbers are in range between 1 and 15.

#### *rx\_queue\_num*

Specifies the associated receive queue for this message object. All CAN messages received by this object are directed to this receive queue. The receive queue numbers are in range between 1 and *n*, in which *n* depends on the definition of *NUM\_RX\_QUEUES* (see also 2.6).

#### *extended*

Set to TRUE for extended CAN messages.

#### *msg\_len*

Unused for this control function. Set to 0.

*data*

Unused for this control function.

**It is possible to assign more than one receive message object to one receive queue.**

## EXAMPLE

```
#include "tdrv011.h"

int fd;
int result;
TDRV011_BUF_DESC BufDesc;

BufDesc.msg_obj_num = 15;
BufDesc.rx_queue_num = 1;
BufDesc.identifier = 1234;
BufDesc.extended = TRUE;

/* Define message object 15 to receive the extended      */
/* message identifier 1234 and store received messages    */
/* in receive queue 1                                     */

result = ioctl(fd, TDRV011_IOCDEFRXBUF, &BufDesc);

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

## ERRORS

EFAULT	Invalid pointer to the parameter buffer. Please check the argument <i>argp</i> .
EINVAL	Invalid argument. This error code is returned if either the message object number, or the specified receive queue is out of range.
EADDRINUSE	The requested message object is already occupied.

## SEE ALSO

Intel 82527 Architectural Overview - 4.18 82527 Message Objects

### 3.3.11 TDRV011\_IOCSEDEFRTMBUF

#### NAME

TDRV011\_IOCSEDEFRTMBUF - Define a remote transmit buffer message object

#### DESCRIPTION

This ioctl function defines a remote transmission CAN message buffer object. A remote transmission object is similar to normal transmission objects with exception that the CAN message is transmitted only after receiving a remote frame with the same identifier.

This type of message object can be used to make process data available for other nodes which can be polled around the CAN bus without any action of the provider node.

The message data remains available for other CAN nodes until this message object is updated with the control function *TDRV011\_IOCUPDATEBUF* or cancelled with *TDRV011\_I OCTRELEASEBUF*.

A pointer to the caller's message description (*TDRV011\_BUF\_DESC*) is passed by the argument *argp* to the driver.

```
typedef struct
{
    unsigned long    identifier;
    unsigned char    msg_obj_num;
    unsigned char    rx_queue_num;
    unsigned char    extended;
    unsigned char    msg_len;
    unsigned char    data[8];
} TDRV011_BUF_DESC, *PTDRV011_BUF_DESC;
```

#### *identifier*

Specifies the message identifier for the message object to be defined.

#### *msg\_obj\_num*

Specifies the number of the message object to be defined. Valid object numbers are in range between 1 and 14.

Keep in mind that message object 15 is only available for receive message objects.

#### *rx\_queue\_num*

Unused for remote transmission message objects. Set to 0.

#### *extended*

Set to TRUE for extended CAN messages.

#### *msg\_len*

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.

## EXAMPLE

```
#include "tdrv011.h"

int          fd;
int          result;
TDRV011_BUF_DESC  BufDesc;

BufDesc.msg_obj_num   = 10;
BufDesc.identifier    = 777;
BufDesc.extended      = TRUE;
BufDesc.msg_len       = 1;
BufDesc.data[0]       = 123;

/* Define message object 10 to transmit the extended */
/* message identifier 777 after receiving of a remote */
/* frame with the same identifier                      */

result = ioctl(fd, TDRV011_IOC_SDEFRMTBUF, &BufDesc);

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

## ERRORS

EFAULT	Invalid pointer to the parameter buffer. Please check the argument <i>argp</i> .
EINVAL	Invalid argument. This error code is returned if the message object number is out of range.
EADDRINUSE	The requested message object is already occupied.
EMSGSIZE	Invalid message size. msg_len must be in range between 0 and 8.

## SEE ALSO

Intel 82527 Architectural Overview - 4.18 82527 Message Objects

### 3.3.12 TDRV011\_IOCUPDATEBUF

#### NAME

TDRV011\_IOCUPDATEBUF - Update a remote or receive buffer message object

#### DESCRIPTION

This ioctl function updates a previously defined receive or remote transmission message buffer object.

To update a receive message object a remote frame is transmitted over the CAN bus to request new data from a corresponding remote transmission message object on other nodes.

To update a remote transmission object only the message data and message length of the specified message object is changed. No transmission is initiated by this control function.

A pointer to the caller's message description (*TDRV011\_BUF\_DESC*) is passed by the argument *argp* to the driver.

```
typedef struct
{
    unsigned long    identifier;
    unsigned char    msg_obj_num;
    unsigned char    rx_queue_num;
    unsigned char    extended;
    unsigned char    msg_len;
    unsigned char    data[8];
} TDRV011_BUF_DESC, *PTDRV011_BUF_DESC;
```

#### *identifier*

Unused for this control function. Set to 0.

#### *msg\_obj\_num*

Specifies the number of the message object to be updated. Valid object numbers are in range between 1 and 14.

Keep in mind that message object 15 is available only for receive message objects.

#### *rx\_queue\_num*

Unused. Set to 0.

#### *extended*

Set to TRUE for extended CAN messages.

#### *msg\_len*

Contains the number of message data bytes (0..8). This parameter is used only for remote transmission object updates.

**data**

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

This parameter is used only for remote transmission object updates.

**EXAMPLE**

```
#include "tdrv011.h"

int          fd;
int          result;
TDRV011_BUF_DESC  BufDesc;

/* Update a receive message object */
BufDesc.msg_obj_num  = 14;

result = ioctl(fd, TDRV011_IOCSUPDATEBUF, &BufDesc);

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

/* Update a remote message object */
BufDesc.msg_obj_num  = 10;
BufDesc.msg_len      = 1;
BufDesc.data[0]      = 124;

result = ioctl(fd, TDRV011_IOCSUPDATEBUF, &BufDesc);

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

**ERRORS**

EFAULT	Invalid pointer to the parameter buffer. Please check the argument <i>argp</i> .
EINVAL	Invalid argument. This error code is returned if either the message object number is out of range or the requested message object is not defined.
EMSGSIZE	Invalid message size. <i>msg_len</i> must be in range between 0 and 8.

**SEE ALSO**

Intel 82527 Architectural Overview - 4.18 82527 Message Objects

### 3.3.13 TDRV011\_IOCTLRELEASEBUF

#### NAME

TDRV011\_IOCTLRELEASEBUF - Release an allocated message buffer object

#### DESCRIPTION

This control function releases a previously defined CAN message object. Any CAN bus transactions of the specified message object will be disabled. After releasing the message object can be defined again with *TDRV011\_IOCSDFRXBUF* and *TDRV011\_IOCSDFRMTBUF* control functions.

A pointer to the caller's message description (*TDRV011\_BUF\_DESC*) is passed by the argument *argp* to the driver.

```
typedef struct
{
    unsigned long    identifier;
    unsigned char    msg_obj_num;
    unsigned char    rx_queue_num;
    unsigned char    extended;
    unsigned char    msg_len;
    unsigned char    data[8];
} TDRV011_BUF_DESC, *PTDRV011_BUF_DESC;
```

#### *msg\_obj\_num*

Specifies the number of the message object to be released. Valid object numbers are in range between 1 and 15.

All other parameters are not used and should be set to 0.

## EXAMPLE

```
#include "tdrv011.h"

int          fd;
int          result;
TDRV011_BUF_DESC  BufDesc;

BufDesc.msg_obj_num  = 14;

result = ioctl(fd, TDRV011_IOCTLRELEASEBUF, &BufDesc);

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

## ERRORS

EFAULT	Invalid pointer to the parameter buffer. Please check the argument <i>argp</i> .
EINVAL	Invalid argument. This error code is returned if the message object number is out of range.
EBADMSG	The requested message object is not defined.
EBUSY	The message object is currently busy transmitting data.

## 4 Diagnostic

If the TDRV011 driver 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 "tdrv011.c" by replacing "#undef" with "#define" and reinstall the driver:

```
#define DEBUG_TDRV011
#define DEBUG_TDRV011_INTR
```

The Linux */proc* file system provides additional information about kernel, resources, drivers, devices and so on. The following screen dumps display information of a correct running TDRV011 driver (see also the proc man pages).

```
# tail -f /var/log/messages /* before modprobing the TDRV011 driver */

May  3 11:10:06 linuxsmp2 kernel: TEWS TECHNOLOGIES - TDRV011 6,4,2 and 1
Channel Extended CAN Bus - version 1.0.1 (2008-03-04)
May  3 11:10:06 linuxsmp2 kernel: TDRV011:  Probe new device
(vendor=0x10B5, device=0x9050, type=816)
May  3 11:10:06 linuxsmp2 kernel: TDRV011: 1x I82527 CAN controller
May  3 11:10:06 linuxsmp2 kernel:
May  3 11:10:06 linuxsmp2 kernel: TDRV011: Add tdrv011 node into the list
of kown major devices
May  3 11:10:06 linuxsmp2 kernel: TDRV011:  Probe new device
(vendor=0x10B5, device=0x9050, type=901)
May  3 11:10:06 linuxsmp2 kernel: TDRV011: 6x I82527 CAN controller
May  3 11:10:06 linuxsmp2 kernel:
May  3 11:10:06 linuxsmp2 kernel: TDRV011: Add tdrv011 node into the list
of kown major devices
May  3 11:10:06 linuxsmp2 kernel: TDRV011:  Probe new device
(vendor=0x1498, device=0x013C, type=316)
May  3 11:10:06 linuxsmp2 kernel: TDRV011: 2x I82527 CAN controller
May  3 11:10:06 linuxsmp2 kernel:
May  3 11:10:06 linuxsmp2 kernel: TDRV011: Add tdrv011 node into the list
of kown major devices

/* after modprobing the driver */
```

```
# cat /proc/tews-tdrv011
/* after driver start, reset condition */
TEWS TECHNOLOGIES - TDRV011 6,4,2 and 1 Channel Extended CAN Bus - version
1.0.1 (2008-03-04)
Supported modules: TPMC316, TPMC816, TPMC901
```

```
Registered Intel 82527 CAN controller channels:
/dev/tdrv011_0 (Car: TPMC816 #0, stat: 0x07 ctrl: 0x49, RxQ1[0,0],
RxQ2[0,0], RxQ3[0,0])
/dev/tdrv011_1 (Car: TPMC901 #0, stat: 0x07 ctrl: 0x49, RxQ1[0,0],
RxQ2[0,0], RxQ3[0,0])
/dev/tdrv011_2 (Car: TPMC901 #1, stat: 0x07 ctrl: 0x49, RxQ1[0,0],
RxQ2[0,0], RxQ3[0,0])
/dev/tdrv011_3 (Car: TPMC901 #2, stat: 0x07 ctrl: 0x49, RxQ1[0,0],
RxQ2[0,0], RxQ3[0,0])
/dev/tdrv011_4 (Car: TPMC901 #3, stat: 0x07 ctrl: 0x49, RxQ1[0,0],
RxQ2[0,0], RxQ3[0,0])
/dev/tdrv011_5 (Car: TPMC901 #4, stat: 0x07 ctrl: 0x49, RxQ1[0,0],
RxQ2[0,0], RxQ3[0,0])
/dev/tdrv011_6 (Car: TPMC901 #5, stat: 0x07 ctrl: 0x49, RxQ1[0,0],
RxQ2[0,0], RxQ3[0,0])
/dev/tdrv011_7 (Car: TPMC316 #0, stat: 0x07 ctrl: 0x49, RxQ1[0,0],
RxQ2[0,0], RxQ3[0,0])
/dev/tdrv011_8 (Car: TPMC316 #1, stat: 0x07 ctrl: 0x49, RxQ1[0,0],
RxQ2[0,0], RxQ3[0,0])...
```

```
# cat /proc/devices
Character devices:
 1 mem
 4 /dev/vc/0
 4 tty
...
180 usb
226 drm
254 tdrv011drv
```

---

```
# 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, TDRV011
177:          0          2 IO-APIC-level uhci_hcd, AMD AMD8111, TDRV011
185:     25775         31 IO-APIC-level uhci_hcd, eth0
193:          0          1 IO-APIC-level libata, ehci_hcd, ..., TDRV011
NMI:          0          0
LOC:   11763048   11763049
ERR:          0
MIS:          0

```

```
# lspci -v
```

```
/* TPMC816-XX */
```

```
02:07.0 Network controller: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 0a)
```

```

Subsystem: TEWS Datentechnik GmbH: Unknown device 0330
Flags: medium devsel, IRQ 193
Memory at ff5fe000 (32-bit, non-prefetchable)
I/O ports at a400 [size=128]
Memory at ff5fd400 (32-bit, non-prefetchable) [size=256]

```

```
/* TPMC901-XX */
```

```
02:08.0 Network controller: PLX Technology, Inc. PCI <-> IOBus Bridge (rev 01)
```

```

Subsystem: TEWS Datentechnik GmbH: Unknown device 0385
Flags: medium devsel, IRQ 177
Memory at ff5fe400 (32-bit, non-prefetchable)
I/O ports at a800 [size=128]
Memory at ff5fd800 (32-bit, non-prefetchable) [size=2K]
I/O ports at a480 [size=4]

```

```
/* TPMC316-XX */
```

```
02:09.0 Network controller: TEWS Datentechnik GmbH: Unknown device 013c (rev 0a)
```

```

Subsystem: TEWS Datentechnik GmbH: Unknown device 000a
Flags: medium devsel, IRQ 169
Memory at ff5fec00 (32-bit, non-prefetchable)
I/O ports at a880 [size=128]
Memory at ff5fe800 (32-bit, non-prefetchable) [size=512]

```