

The Embedded I/O Company



TPMC816-SW-95

QNX6 - Neutrino Device Driver

Two Channel Extended CAN Bus

Version 1.1.x

User Manual

Issue 1.1.0

October 2005

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7
Phone: +49-(0)4101-4058-0
e-mail: info@tews.com

25469 Halstenbek / Germany
Fax: +49-(0)4101-4058-19
www.tews.com

TEWS TECHNOLOGIES LLC

1 E. Liberty Street, Sixth Floor
Phone: +1 (775) 686 6077
e-mail: usasales@tews.com

Reno, Nevada 89504 / USA
Fax: +1 (775) 686 6024
www.tews.com

TPMC816-SW-95

Two Channel Extended CAN Bus

QNX6-Neutrino Device Driver

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

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

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

©2005 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	June 1, 2001
1.1.0	General Revision	October 19, 2005

Table of Contents

1	INTRODUCTION.....	4
2	INSTALLATION.....	5
	2.1 Build the device driver	5
	2.2 Build the example application	5
	2.3 Start the driver process.....	5
	2.4 Receive Queue Configuration.....	6
3	DEVICE INPUT/OUTPUT FUNCTIONS	7
	3.1 open()	7
	3.2 close().....	8
	3.3 devctl()	9
	3.3.1 DCMD_TPMC816_READ(_NOWAIT)	11
	3.3.2 DCMD_TPMC816_WRITE.....	14
	3.3.3 DCMD_TPMC816_BITTIMING	16
	3.3.4 DCMD_TPMC816_SETFILTER	18
	3.3.5 DCMD_TPMC816_BUSON	20
	3.3.6 DCMD_TPMC816_BUSOFF	21
	3.3.7 DCMD_TPMC816_FLUSH	22
	3.3.8 DCMD_TPMC816_CANSTATUS	23
	3.3.9 DCMD_TPMC816_DEFRXBUF	24
	3.3.10 DCMD_TPMC816_DEFRMTBUF	26
	3.3.11 DCMD_TPMC816_UPDATEBUF	28
	3.3.12 DCMD_TPMC816_RELEASEBUF	31
	3.3.13 DCMD_TPMC816_SETTXOBJ	33
	3.4 Step by Step Driver Initialization	34

1 Introduction

The TPMC816-SW-95 QNX-Neutrino device driver allows the operation of TPMC816 - Two Channel Extended CAN Bus PMCs on QNX-Neutrino operating systems.

The TPMC816 device driver is basically implemented as a user installable Resource Manager. The standard file (I/O) functions (open, close and devctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

The TPMC816 device driver includes the following functions:

- Transmission and receive of Standard and Extended Identifiers
- 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 1.6 Mbit and user defined bit rates
- Message acceptance filtering
- Definition of receive and remote buffer message objects
- Transmission and receive of Standard and Extended Identifiers
- Supports shared IRQ's.
- Automatic configuration on startup, i.e. driver scans the entire PCI bus and initializes all found TPMC816 modules

To understand all features of this device driver, it is very important to read the Architectural Overview of the Intel 82527 CAN controller, which is part of the engineering kit TPMC816-EK.

2 Installation

The driver software on the distribution media is stored in a tar-archive named *TPMC816-SW-95-SRC.tar*. This manual is located on the media too.

Following files are stored in the tar-archive:

<code>/driver/tpmc816.c</code>	Driver source code
<code>/driver/tpmc816.h</code>	Driver interface definitions and data structures
<code>/driver/tpmc816def.h</code>	Device driver include
<code>/driver/i82527.h</code>	Intel 82527 CAN controller definitions
<code>/driver/node.h</code>	Queue management definitions
<code>/driver/node.c</code>	Queue management source code
<code>/example/tpmc816exa.c</code>	Example application

Additional distribution media files:

<code>TPMC816-SW-95-1.1.0.pdf</code>	This Manual in PDF format
<code>Release.txt</code>	Release information

For installation copy the tar-archive into the `/usr/src` directory and unpack it (e.g. `tar -xvf TPMC816-SW-95-SRC.tar`). After that the necessary directory structure for the automatic build and the source files are available underneath the new directory called *tpmc816*.

It is absolutely important to extract the TPMC816 tar archive in the /usr/src directory. Otherwise the automatic build with make will fail.

2.1 Build the device driver

Change to the `/usr/src/tpmc816/driver` directory

Execute the Makefile:

```
# make install
```

After successful completion the driver binary (*tpmc816*) will be installed in the `/bin` directory.

2.2 Build the example application

Change to the `/usr/src/tpmc816/example` directory

Execute the Makefile:

```
# make install
```

After successful completion the example binary (*tpmc816exa*) will be installed in the `/bin` directory.

2.3 Start the driver process

To start the TPMC816 device driver respective you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tpmc816 [-v] &
```

The TPMC816 Resource Manager registers created devices in the Neutrinos pathname space under following names.

```
/dev/tpmc816_0  
/dev/tpmc816_1  
...  
/dev/tpmc816_x
```

This pathname must be used in the application program to open a path to the desired TPMC816 device.

```
fd = open("/dev/tpmc816_0", O_RDWR);
```

For debugging you can start the TPMC816 Resource Manager with the `-v` option. Now the Resource Manager will print versatile information about TPMC816 configuration and command execution on the terminal window.

```
tpmc816 -v &
```

2.4 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 `tpmc816def.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.

3.1 open()

NAME

open() - open a file descriptor

SYNOPSIS

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
int open (const char *pathname, int flags)
```

DESCRIPTION

The *open* function creates and returns a new file descriptor for the TPMC816 named by pathname. The flags argument controls how the file is to be opened. TPMC816 devices must be opened O_RDWR.

EXAMPLE

```
int fd;

fd = open("/dev/tpmc816_0", O_RDWR);
```

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

Returns only Neutrino specific error codes, see Neutrino Library Reference.

SEE ALSO

Library Reference - open()

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

Returns only Neutrino specific error code, see Neutrino Library Reference.

SEE ALSO

Library Reference - close()

3.3 devctl()

NAME

devctl() – device control functions

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>
#include <devctl.h>
```

```
int devctl
(
    int          filedes,
    int          dcmd,
    void         *data_ptr,
    size_t       n_bytes,
    int          *dev_info_ptr
)
```

DESCRIPTION

The devctl function sends a control code directly to a device, specified by *filedes*, causing the corresponding device to perform the requested operation.

The argument *dcmd* specifies the control code for the operation.

The arguments *data_ptr* and *n_bytes* depends on the command and will be described for each command in detail later in this chapter. Usually *data_ptr* points to a buffer that passes data between the user task and the driver and *n_bytes* defines the size of this buffer.

The argument *dev_info_ptr* is unused for the TPMC816 driver and should be set to NULL.

The following devctl command codes are defined in *tpmc816.h*:

Value	Description
<i>DCMD_TPMC816_READ</i>	Read a CAN message from the specified queue
<i>DCMD_TPMC816_READ_NOWAIT</i>	Read CAN message from the specified queue and return immediately if queue is empty
<i>DCMD_TPMC816_WRITE</i>	Write message to the CAN bus
<i>DCMD_TPMC816_BITTIMING</i>	Setup a new bit timing
<i>DCMD_TPMC816_SETFILTER</i>	Setup acceptance filter
<i>DCMD_TPMC816_BUSON</i>	Enter the bus on state
<i>DCMD_TPMC816_BUSOFF</i>	Enter the bus off state
<i>DCMD_TPMC816_FLUSH</i>	Flush one or all receive queues
<i>DCMD_TPMC816_CANSTATUS</i>	Returns CAN controller status information
<i>DCMD_TPMC816_DEFRXBUF</i>	Define a receive buffer message object

<i>DCMD_TPMC816_DEFRMTBUF</i>	Define a remote transmit buffer message object
<i>DCMD_TPMC816_UPDATEBUF</i>	Update a remote or receive buffer message object
<i>DCMD_TPMC816_RELEASEBUF</i>	Release an allocated message buffer object
<i>DCMD_TPMC816_SETTXOBJ</i>	Define default transmit object

See behind for more detailed information on each control code.

To use these TPMC816 specific control codes the header file TPMC816.h must be included in the application.

RETURNS

On success, EOK is returned. In the case of an error, the appropriate error code is returned by the function (not in errno!).

ERRORS

Returns only Neutrino specific error codes, see Neutrino Library Reference.

Other function dependent error codes will be described for each devctl code separately. Note, the TPMC816 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_TPMC816_READ(_NOWAIT)

NAME

DCMD_TPMC816_READ(_NOWAIT) – Read a CAN message

DESCRIPTION

This devctl function reads a CAN message from the specified receive queue. A pointer to the callers message buffer (TPMC816_MSG_BUF) and the size of this structure is passed by the parameters `data_ptr` and `n_bytes` to the device.

If currently no message is available the caller will be blocked until a new CAN message is received and this request is in front of the receive queue, or a timeout occur.

To avoid blocking the DCMD_TPMC816_READ_NOWAIT command should be used instead. This devctl command will return immediately with the error ENODATA if no CAN message is available.

The TPMC816_MSG_BUF structure has the following layout:

```
typedef struct {
    unsigned long    Identifier;
    long            Timeout;
    unsigned char    RxQueueNum;
    unsigned char    Extended;
    unsigned char    Status;
    unsigned char    MsgLen;
    unsigned char    Data[8];
} TPMC816_MSG_BUF, *PTPMC816_MSG_BUF;
```

Identifier

Receives the message identifier of the read CAN message .

Timeout

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

RxQueueNum

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 **Fehler! Verweisquelle konnte nicht gefunden werden.**).

Extended

Receives TRUE for extended CAN messages.

Status

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

TPMC816_SUCCESS	No messages lost
TPMC816_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.
TPMC816_MSGOBJ_OVERRUN	One or more messages were overwritten in the CAN controller message FIFO because the interrupt latency is too large. Use message object 15 (buffered) to receive this time critical CAN messages, reduce the CAN bit rate or upgrade the system speed.

MsgLen

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

Data[8]

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

EXAMPLE

```
{  
  
    int fd;  
    int result;  
    TPMC816_MSG_BUF  MsgBuf;  
  
    . . .  
  
    MsgBuf.RxQueueNum = 1;  
    MsgBuf.Timeout = 10;    /* seconds */  
  
    result = devctl(fd,  
        DCMD_TPMC816_READ,  
        &MsgBuf,  
        sizeof(MsgBuf),  
        NULL);  
  
    if (result == EOK) {  
        /* process received CAN message */  
    }  
  
    . . .  
}
```

ERRORS

<i>EINVAL</i>	Invalid argument. This error code is returned if the size of the message buffer is too small, or the specified receive queue is out of range.
<i>ENOMEM</i>	No memory available to allocated resources to handle the read command.
<i>ECONNREFUSED</i>	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.
<i>ENODATA</i>	Currently no CAN message available for read (only DCMD_TPMC816_READ_NOWAIT).
<i>ETIMEDOUT</i>	The allowed time to finish the read request is elapsed.

SEE ALSO

Library Reference - devctl()

3.3.2 DCMD_TPMC816_WRITE

NAME

DCMD_TPMC816_WRITE - Write a CAN message

DESCRIPTION

This devctl function writes a message to the CAN bus. A pointer to the callers message buffer (TPMC816_MSG_BUF) and the size of this structure is passed by the parameters `data_ptr` and `n_bytes` to the device. If the CAN controller is busy transmitting an other message the caller becomes blocked until all previous pending requests are serviced or a timeout occurs. Keep in mind to configure the default transmit message object with the control function DCMD_TPMC816_SETTXOBJ before (see also 3.3.13).

The TPMC816_MSG_BUF structure has the following layout:

```
typedef struct {
    unsigned long    Identifier;
    long            Timeout;
    unsigned char    RxQueueNum;
    unsigned char    Extended;
    unsigned char    Satus;
    unsigned char    MsgLen;
    unsigned char    Data[8];
} TPMC816_MSG_BUF, *PTPMC816_MSG_BUF;
```

Identifier

Contains the message identifier of the CAN message to write.

Timeout

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

RxQueueNum

Unused for this control function. Can be 0.

Extended

Contains TRUE (1) for extended CAN messages..

Status

Unused for this control function. Can be 0.

MsgLen

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

Data[8]

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

EXAMPLE

```
{
    int fd;
    int result;
    TPMC816_MSG_BUF MsgBuf;
    . . .
    MsgBuf.Identifier = 1234;
    MsgBuf.Timeout = 2;      /* 2 sec*/
    MsgBuf.Extended = TRUE;
    MsgBuf.MsgLen = 2;
    MsgBuf.Data[0] = 0xaa;
    MsgBuf.Data[1] = 0x55;

    result = devctl(fd,
        DCMD_TPMC816_WRITE,
        &MsgBuf,
        sizeof(MsgBuf),
        NULL);
    if (result == EOK) {
        /* CAN message successful transmitted */
    }
    . . .
}
```

ERRORS

<i>EINVAL</i>	Invalid argument. This error code is returned if the size of the message buffer is too small.
<i>ENOMEM</i>	No memory available to allocated resources to handle the read command.
<i>ECONNREFUSED</i>	The controller is in bus OFF state and unable to transmit messages.
<i>EMSGSIZE</i>	Invalid message size. <i>MsgBuf.MsgLen</i> must be in range between 0 and 8.
<i>ETIMEDOUT</i>	The allowed time to finish the write request is elapsed. This occur if currently no message object is available or if the CAN bus is overloaded and the priority of the message identifier is too low.

SEE ALSO

Library Reference - devctl()

3.3.3 DCMD_TPMC816_BITTIMING

NAME

DCMD_TPMC816_BITTIMING – Setup new bit timing

DESCRIPTION

This devctl function modifies the bit timing register of the CAN controller to setup a new CAN bus transfer speed. A pointer to the callers parameter buffer (*TPMC816_TIMING*) and the size of this structure are passed by the parameters *data_ptr* and *n_bytes* to the device.

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

The *TPMC816_TIMING* structure has the following layout:

```
typedef struct {
    unsigned short    TimingValue;
    unsigned short    ThreeSamples;
}TPMC816_BITTIMING, *PTPMC816_BITTIMING;
```

unsigned short 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 1.6 Mbit per second. The include file 'tpmc816.h' contains predefined transfer rate symbols (TPMC816_5KBIT ... TPMC816_1_6MBIT).

For other transfer rates please follow the instructions of the *Intel 82527 Product Specification*, which is also part of the engineering kit TPMC816-EK.

unsigned short 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 immune against noise spikes.

This devctl command will be accepted only in reset mode (BUSOFF). Enter DCMD_TPMC816_BUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
{
    int fd;
    int result;
    TPMC816_BITTIMING BitTimingParam;

    BitTimingParam.TimingValue = TPMC816_100KBIT;
    BitTimingParam.ThreeSamples = FALSE;

    result = devctl(    fd,
                      DCMD_TPMC816_BITTIMING,
                      &BitTimingParam,
                      sizeof(BitTimingParam),
                      NULL);

    if (result != EOK) {
        /* process devctl() error */
    }
}
```

ERRORS

EACCES

Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the bit timing.

SEE ALSO

Library Reference - devctl().

tpmc816.h for predefined bus timing constants.

Intel 82527 Architectural Overview - 4.13 Bit Timing Overview

3.3.4 DCMD_TPMC816_SETFILTER

NAME

DCMD_TPMC816_SETFILTER - Setup acceptance filter

DESCRIPTION

This devctl 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 callers parameter buffer (*TPMC816_ACCEPT_MASKS*) is passed by the parameter *data_ptr* to the driver.

The *TPMC816_ACCEPT_MASKS* structure has the following layout:

```
typedef struct {
    unsigned long      Message15Mask;
    unsigned long      GlobalMaskExtended;
    unsigned short     GlobalMaskStandard;
} TPMC816_ACCEPT_MASKS, *PTPMC816_ACCEPT_MASKS;
```

Message15Mask

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).

GlobalMaskExtended

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.

GlobalMaskStandard

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.

Note: The TPMC816 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

```
{
    int fd;
    int result;
    TPMC816_ACCEPT_MASKS AcceptMasksParam;

    . . .

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

    /* Extended identifier bits 0..3 don't care */
    AcceptMasksParam.GlobalMaskExtended = 0xffffffff80;

    /* Message object 15 identifier bits 0..7 don't care */
    AcceptMasksParam.Message15Mask = 0xffffffff800;

    result = devctl(fd,
        DCMD_TPMC816_SETFILTER,
        & AcceptMasksParam,
        sizeof(AcceptMasksParam),
        NULL);

    if (result != EOK) {
        /* handle devctl error */
    }

    . . .
}
```

ERRORS

This *devctl* function returns no function specific error codes.

SEE ALSO

Library Reference - *devctl*().

Intel 82527 Architectural Overview - 4.9 Acceptance Filtering

3.3.5 DCMD_TPMC816_BUSON

NAME

DCMD_TPMC816_BUSON - Enter the bus on state

DESCRIPTION

This devctl 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 arguments *data_ptr* and *n_bytes* are unused and should be set to *NULL* respective 0.

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

EXAMPLE

```
{
    int fd;
    int result;
    . . .
    result = devctl(fd, DCMD_TPMC816_BUSON, NULL, 0, NULL);

    if (result != EOK) {
        /* handle devctl error */
    }
    . . .
}
```

ERRORS

This devctl function returns no function specific error codes.

SEE ALSO

Library Reference - devctl()

Intel 82527 Architectural Overview - 3.2 *Software Initialization*

3.3.6 DCMD_TPMC816_BUSOFF

NAME

DCMD_TPMC816_BUSOFF - Enter the bus off state

DESCRIPTION

This devctl 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 *DCMD_TPMC816_BUSON* is executed.

The arguments *data_ptr* and *n_bytes* are unused and should be set to *NULL* respective 0.

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

EXAMPLE

```
{  
  
    int fd;  
    int result;  
  
    . . .  
    result = devctl(fd, DCMD_TPMC816_BUSOFF, NULL, 0, NULL);  
  
    if (result != EOK) {  
        /* handle devctl error */  
    }  
    . . .  
}
```

ERRORS

This devctl function returns no function specific error codes.

SEE ALSO

Library Reference - devctl()

Intel 82527 Architectural Overview - 3.2 *Software Initialization*

3.3.7 DCMD_TPMC816_FLUSH

NAME

DCMD_TPMC816_FLUSH - Flush one or all receive queues

DESCRIPTION

This devctl function flushes the message FIFO of the specified receive queue(s). The argument *data_ptr* points to an *int* variable, which contains the receive queue number. The argument *n_bytes* specifies the size of this variable (size of *int*). If the receive queue number is 0 the FIFO's of all receive queues will be flushed, otherwise only the FIFO of the specified receive queue will be flushed.

EXAMPLE

```
{
    int fd;
    int result;
    int RxQueueNum;
    . . .
    /* flush all receive queues */
    RxQueueNum = 0;

    result = devctl(fd,
        DCMD_TPMC816_FLUSH,
        &RxQueueNum,
        sizeof(RxQueueNum),
        NULL);

    if (result != 0) {
        /* handle devctl error */
    }
    . . .
}
```

ERRORS

EINVAL

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

SEE ALSO

Library Reference - devctl()

3.3.8 DCMD_TPMC816_CANSTATUS

NAME

DCMD_TPMC816_CANSTATUS - Returns the contents of the CAN status register

DESCRIPTION

This devctl function returns the actual contents of the CAN controller status register for diagnostic purposes.

The contents of the controller status register is received in an *int* variable. A pointer to this variable is passed by the argument *data_ptr* to the driver. The argument *n_bytes* specifies the size of this variable (size of *int*).

EXAMPLE

```
{
    int fd;
    int result;
    int CanStatus;

    . . .

    result = devctl(fd,
        DCMD_TPMC816_CANSTATUS,
        &CanStatus,
        sizeof(CanStatus),
        NULL);

    if (result != EOK) {
        /* handle devctl error */
    }

    . . .
}
```

ERRORS

This devctl function returns no function specific error codes.

SEE ALSO

Library Reference - devctl()

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

3.3.9 DCMD_TPMC816_DEFRXBUF

NAME

DCMD_TPMC816_DEFRXBUF - Define a receive buffer message object

DESCRIPTION

This devctl 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 DCMD_TPMC816_READ command. Before the driver can receive CAN messages it's 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 double-buffered. A pointer to the callers message description (*TPMC816_BUF_DESC*) is passed by the argument *data_ptr* to the driver. The argument *n_bytes* specifies the size of the message description structure.

The *TPMC816_BUF_DESC* structure has the following layout:

```
typedef struct {
    unsigned long      Identifier;
    unsigned char      MsgObjNum;
    unsigned char      RxQueueNum;
    unsigned char      Extended;
    unsigned char      MsgLen;
    unsigned char      Data[8];
} TPMC816_BUF_DESC, *PTPMC816_BUF_DESC;
```

Identifier

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

MsgObjNum

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

RxQueueNum

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 number is one based, valid numbers are in range between 1 and n. In which n depends on the definition of NUM_RX_QUEUES (see also **Fehler! Verweisquelle konnte nicht gefunden werden.**).

Note. It's possible to assign more than one receive message object to the same receive queue.

Extended

Set to TRUE for extended CAN messages.

MsgLen

Unused for this control function. Set to 0.

Data[8]

Unused for this control function.

EXAMPLE

```
{
    int fd;
    int result;
    TPMC816_BUF_DESC BufDesc;
    . . .

    BufDesc.MsgObjNum = 15;
    BufDesc.RxQueueNum = 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 = devctl(fd,
        DCMD_TPMC816_DEFRXBUF,
        &BufDesc,
        sizeof(BufDesc),
        NULL);

    if (result != EOK) {
        /* handle devctl error */
    }
    . . .
}
```

ERRORS

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

SEE ALSO

Library Reference - devctl()

Intel 82527 Architectural Overview - 4.18 82527 Message Objects

3.3.10 DCMD_TPMC816_DEFRMTBUF

NAME

DCMD_TPMC816_DEFRMTBUF - Define a remote transmit buffer message object

DESCRIPTION

This devctl function defines a remote transmission CAN message buffer object. A remote transmission object is similar to normal transmission object with exception that the CAN message is transmitted only after receiving of 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 remain available for other CAN nodes until this message object is updated with the control function *DCMD_TPMC816_UPDATEBUF* or cancelled with *DCMD_TPMC816_RELEASEBUF*. A pointer to the callers message description (*TPMC816_BUF_DESC*) is passed by the argument *data_ptr* to the driver. The argument *n_bytes* specifies the size of the message description structure.

The *TPMC816_BUF_DESC* structure has the following layout:

```
typedef struct {
    unsigned long      Identifier;
    unsigned char      MsgObjNum;
    unsigned char      RxQueueNum;
    unsigned char      Extended;
    unsigned char      MsgLen;
    unsigned char      Data[8];
} TPMC816_BUF_DESC, *PTPMC816_BUF_DESC;
```

Identifier

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

MsgObjNum

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.

RxQueueNum

Unused for remote transmission message objects. Set to 0.

Extended

Set to TRUE for extended CAN messages.

MsgLen

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

Data[8]

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

EXAMPLE

```
{
    int fd;
    int result;
    TPMC816_BUF_DESC BufDesc;
    . . .
    BufDesc.MsgObjNum = 10;
    BufDesc.Identifier = 777;
    BufDesc.Extended = TRUE;
    BufDesc.MsgLen = 1;
    BufDesc.Data[0] = 123;

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

    result = devctl(fd,
        DCMD_TPMC816_DEFRMTBUF,
        &BufDesc,
        sizeof(BufDesc),
        NULL);

    if (result != EOK) {
        /* handle devctl error */
    }
    . . .
}
```

ERRORS

<i>EINVAL</i>	Invalid argument. This error code is returned if the message object number is out of range.
<i>EADDRINUSE</i>	The requested message object is already occupied.
<i>EMSGSIZE</i>	Invalid message size. BufDesc.MsgLen must be in range between 0 and 8.

SEE ALSO

Library Reference - devctl()

Intel 82527 Architectural Overview - 4.18 82527 Message Objects

3.3.11 DCMD_TPMC816_UPDATEBUF

NAME

DCMD_TPMC816_UPDATEBUF - Update a remote or receive buffer message object

DESCRIPTION

This devctl function updates a previous 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. Received message by this message object will be stored in the associated receive queue.

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 callers message description (*TPMC816_BUF_DESC*) is passed by the argument *data_ptr* to the driver. The argument *n_bytes* specifies the size of the message description structure.

The *TPMC816_BUF_DESC* structure has the following layout:

```
typedef struct {
    unsigned long    Identifier;
    unsigned char    MsgObjNum;
    unsigned char    RxQueueNum;
    unsigned char    Extended;
    unsigned char    MsgLen;
    unsigned char    Data[8];
} TPMC816_BUF_DESC, *PTPMC816_BUF_DESC;
```

Identifier

Unused for this control function. Set to 0.

MsgObjNum

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.

RxQueueNum

Unused for this control function. Set to 0.

Extended

Set to TRUE for extended CAN messages.

MsgLen

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

Data[8]

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

```
{
    int fd;
    int result;
    TPMC816_BUF_DESC  BufDesc;

    . . .

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

    result = devctl(fd,
        DCMD_TPMC816_UPDATEBUF,
        &BufDesc,
        sizeof(BufDesc),
        NULL);

    if (result != EOK) {
        /* handle devctl error */
    }

    /* Update a remote message object */
    BufDesc.MsgObjNum = 10;
    BufDesc.MsgLen     = 1;
    BufDesc.Data[0]    = 124;

    result = devctl(fd,
        DCMD_TPMC816_UPDATEBUF,
        &BufDesc,
        sizeof(BufDesc),
        NULL);

    if (result != EOK) {
        /* handle devctl error */
    }

    . . .
}
```

ERRORS

<i>EINVAL</i>	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.
<i>EADDRINUSE</i>	The requested message object is already occupied.
<i>EMSGSIZE</i>	Invalid message size. BufDesc.MsgLen must be in range between 0 and 8.

SEE ALSO

Library Reference - devctl()

Intel 82527 Architectural Overview - *4.18 82527 Message Objects*

3.3.12 DCMD_TPMC816_RELEASEBUF

NAME

DCMD_TPMC816_RELEASEBUF - Release an allocated message buffer object

DESCRIPTION

This devctl function releases a previous defined CAN message object. Any CAN bus transactions of the specified message object become disabled. After releasing the message object can be defined again with *DCMD_TPMC816_DEFRXBUF* and *DCMD_TPMC816_DEFRMTBUF* control functions.

A pointer to the callers message description (*TPMC816_BUF_DESC*) is passed by the argument *data_ptr* to the driver. The argument *n_bytes* specifies the size of the message description structure.

The *TPMC816_BUF_DESC* structure has the following layout:

```
typedef struct {
    unsigned long      Identifier;
    unsigned char      MsgObjNum;
    unsigned char      RxQueueNum;
    unsigned char      Extended;
    unsigned char      MsgLen;
    unsigned char      Data[8];
} TPMC816_BUF_DESC, *PTPMC816_BUF_DESC;
```

MsgObjNum

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

Note: All other parameter are not used and could be left blank.

EXAMPLE

```
{
    int fd;
    int result;
    TPMC816_BUF_DESC BufDesc;

    . . .

    BufDesc.MsgObjNum = 14;

    result = devctl(fd,
        DCMD_TPMC816_RELEASEBUF,
        &BufDesc,
        sizeof(BufDesc),
        NULL);

    if (result != EOK) {
        /* handle devctl error */
    }

    . . .
}
```

ERRORS

<i>EINVAL</i>	Invalid argument. This error code is returned if the message object number is out of range.
<i>EBADMSG</i>	The requested message object is not defined.
<i>EBUSY</i>	The message object is currently busy transmitting data.

SEE ALSO

Library Reference - devctl()

3.3.13 DCMD_TPMC816_SETTXOBJ

NAME

DCMD_TPMC816_SETTXOBJ - Define a default transmit message object

DESCRIPTION

This devctl function defines the message object, which is used to transmit CAN messages with the *DCMD_TPMC816_WRITE* control function. Valid transmission object numbers are in range between 1..14. The argument *data_ptr* points to an *int* variable, which contains the message object number. The argument *n_bytes* specifies the size of this variable (size of *int*).

Note: As long as the default transmission object is unknown no messages can be send with the *DCMD_TPMC816_WRITE* control function.

EXAMPLE

```
{
    int fd;
    int result;
    int TxObjectNum;
    ...
    /* flush all receive queues */
    TxObjectNum = 0;

    result = devctl(fd, DCMD_TPMC816_SETTXOBJ, &TxObjectNum,
        sizeof(TxObjectNum), NULL);
    if (result != 0) {
        /* handle devctl error */
    }
    ...
}
```

ERRORS

<i>EINVAL</i>	Invalid argument. This error code is returned if the specified message object number is out of range or the desired message object is already occupied.
---------------	---

SEE ALSO

Library Reference - devctl()

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 = TPMC816_100KBIT;
BitTimingParam.ThreeSamples = FALSE;

result = devctl( fd,
    DCMD_TPMC816_BITTIMING,
    &BitTimingParam,
    sizeof(BitTimingParam),
    NULL);

/*
** ( 2.) Setup acceptance filter masks
*/
AcceptMasksParam.GlobalMaskStandard = 0xFFFF;
AcceptMasksParam.GlobalMaskExtended = 0xFFFFFFFF;
AcceptMasksParam.Message15Mask = 0;

result = devctl( fd,
    DCMD_TPMC816_SETFILTER,
    &AcceptMasksParam,
    sizeof(AcceptMasksParam),
    NULL);

/*
** ( 3.) Define message object 15 for reception ( receive
** all identifiers )
*/
BufDesc.MsgObjNum = 15;
BufDesc.RxQueueNum = 1;
BufDesc.Identifier = 0;
BufDesc.Extended = TRUE;

result = devctl( fd,
    DCMD_TPMC816_DEFRXBUF,
    &BufDesc,
    sizeof(BufDesc),
```

```
        NULL);

/*
** ( 4.) Define default transmit message object
*/
TxObjectNum = 1;

result = devctl( fd,
                DCMD_TPMC816_SETTXOBJ,
                &TxObjectNum,
                sizeof(TxObjectNum),
                NULL);

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

result = devctl(fd, DCMD_TPMC816_BUSON, NULL, 0, NULL);
```