The Embedded I/O Company

TEWS TECHNOLOGIES

TIP810-SW-42

VxWorks Device Driver

CAN Bus IP

Version 3.0.x

User Manual

Issue 3.0.0

August 2010

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7 25469 Halstenbek, Germany

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

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

TIP810-SW-42

VxWorks Device Driver

CAN Bus IP

Supported Modules:
TIP810-10

Issue	Description	Date
1.0	First Issue	January 1995
1.1	General Revision	September 2003
1.2	Added support for PeliCAN mode	July 2004
2.0.0	TEWS Carrier Support added, new address TEWS LLC	November 20, 2008
3.0.0	SMP Support, IPAC carrier interface functions removed	August 17, 2010

Table of Contents

1 Introduction

1.1 Device Driver

The TIP810-SW-42 VxWorks device driver allows the operation of the TIP810 CAN Bus IP conforming to the VxWorks I/O system specification. This includes a device-independent interface with *open*, *read*, *write* and *ioctl* functions.

After installation of the device driver in the I/O system messages can be transmitted to and received from the CAN bus by calling the *write()* or *read()* functions.

Special I/O operation that does 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.

This driver invokes a mutual exclusion and queuing mechanism to prevent simultaneous requests by multiple users from interfering with each other.

The TIP810-SW-42 device driver supports the following features:

- Transmission and receive of Standard and Extended CAN Frames
- Standard bit rates from 20 Kbit/s up to 1.0 Mbit/s and user defined bit rates
- Message acceptance filtering
- Single-Shot transmission
- Listen only mode
- Message self reception
- Programmable error warning limit

The TIP810-SW-42 supports the modules listed below:

TIP810-10	CAN Bus	IndustryPack® compatible

To get more information about the features and use of TIP810 devices it is recommended to read the manuals listed below.

TIP810 User manual
TIP810 Engineering Manual
CARRIER-SW-42 IPAC Carrier User Manual
SJA1000 PeliCAN controller User Manual

1.2 IPAC Carrier Driver

IndustryPack (IPAC) carrier boards have different implementations of the system to IndustryPack bus bridge logic, different implementations of interrupt and error handling and so on. Also the different byte ordering (big-endian versus little-endian) of CPU boards will cause problems on accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which work with any supported carrier board, TEWS TECHNOLOGIES has designed a so called Carrier Driver that hides all differences of different carrier boards under a well defined interface.

The TEWS TECHNOLOGIES IPAC Carrier Driver CARRIER-SW-42 is part of this TIP810-SW-42 distribution. It is located in directory CARRIER-SW-42 on the corresponding distribution media.

This IPAC Device Driver requires a properly installed IPAC Carrier Driver. Due to the design of the Carrier Driver, it is sufficient to install the IPAC Carrier Driver once, even if multiple IPAC Device Drivers are used.

Please refer to the CARRIER-SW-42 User Manual for a detailed description how to install and setup the CARRIER-SW-42 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

Following files are located on the distribution media:

Directory path 'TIP810-SW-42':

tip810drv.c	TIP810 Driver source
tip810.h	TIP810 Driver include file for driver and application
tip810def.h	TIP810 Driver private include file
tip810exa.c	TIP810 Driver usage example program
sja1000.h	Header file for the SJA1000 CAN Controller (TIP810 V2.0+)
pca82c200.h	Header file for the PCA82C200 CAN Controller (TIP810 V1.0)
include/ipac_carrier.h	Carrier driver interface definitions
TIP810-SW-42-3.0.0.pdf	PDF copy of this manual
Release.txt	Release information
ChangeLog.txt	Release history

2.1 Include device driver in VxWorks project

For including the TIP810-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

- Copy the files from the distribution media into a subdirectory in your project path. (For example: ./TIP810)
- Add the device drivers C-files to your project.
- Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)

2.2 System resource requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 1 KB
Stack	< 1 KB	< 1 KB
Semaphores	0	4

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

3 I/O system functions

This chapter describes the driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

3.1 t810Drv()

NAME

t810Drv() - installs the TIP810 driver in the I/O system

SYNOPSIS

#include “tip810.h”

STATUS t810Drv(void)

DESCRIPTION

This function installs the TIP810 driver in the I/O system.

A call to this function is the first thing the user has to do before adding any device to the system or performing any I/O request.

EXAMPLE

```
#include "tip810.h”


STATUS      result;

/*-------------------
  Initialize Driver
  -------------------*/
result = t810Drv();
if (result == ERROR)
{
  /* Error handling */
}
```

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 t810DevCreate()

NAME

t810DevCreate() - add a TIP810 device to the system and initialize device hardware

SYNOPSIS

```
#include “tip810.h”

STATUS t810DevCreate
(
        char        *name,
        int         devIdx,
        int         funcType,
        void        *pParam
)
```

DESCRIPTION

This routine adds the selected device to the VxWorks system. The device hardware will be setup and prepared for use.

This function must be called before performing any I/O request to this device.

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

devIdx

This index number specifies the desired device instance beginning by 0. This parameter is 0 for the first TIP810 in the system, 1 for the second TIP810 and so forth. The order of TIP810 modules depends on the search order of the IPAC carrier driver.

funcType

This parameter is unused and should be set to *0*.

pParam

This parameter is unused and should be set to NULL.

EXAMPLE

```
#include “tip810.h”

STATUS               result;
TIP810_DEVCONFIG     tip810Conf;

/*-------------------------------------------------------
  Create the device "/t810/0" for the first CAN device
  o reserve a read buffer for 1000 messages
  o transfer rate = 100kbit/s, one sample per bit
  o only messages with identifiers 128…135 passes the acceptance filter
  -------------------------------------------------------*/
tip810Conf.filter.acceptance_code   = 128 << 3;
tip810Conf.filter.acceptance_mask   = (8 << 3) – 1;
tip810Conf.filter.single_filter     = 1;
tip810Conf.timing.timing_value      = T810_100KBIT;
tip810Conf.timing.three_samples     = 0;

result = t810DevCreate( "/t810/0",
                        0,
                        0,
                        (void*)&tip810Conf);
if (result == ERROR)
{
  /* Error occurred when creating the device */
}
```

RETURNS

OK, or ERROR if no device was found or an error occurred during device initialization.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
S_ioLib_NO_DRIVER	The TIP810 driver has not been started
EINVAL	Input parameters are invalid
EISCONN	The device has been created already
S_t810Drv_NO_DEVICE	No or unsupported TIP810 module found

SEE ALSO

VxWorks Programmer's Guide: I/O System

4 I/O interface functions

This chapter describes the interface to the basic I/O system used for communication over the CAN bus.

4.1 open()

NAME

open() - open a device or file.

SYNOPSIS

```
int open
(
        const char  *name,
        int         flags,
        int         mode
)
```

DESCRIPTION

Before I/O can be performed to the TIP810 device, a file descriptor must be opened by invoking the basic I/O function *open()*.

PARAMETER

name

Specifies the device which shall be opened. The name specified in t810DevCreate() must be used.

flags

Not used

mode

Not used

EXAMPLE

```
int    fd;

/*----------------------------------------
  Open the device named "/t810/0" for I/O
  ----------------------------------------*/
fd = open("/t810/0", 0, 0);
if (fd == ERROR)
{
  /* error handling */
}
```

RETURNS

A device descriptor number or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - *open()*

4.2 close()

NAME

close() – close a device or file

SYNOPSIS

STATUS close
(
    int   fd
)

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

This file descriptor specifies the device to be closed. The file descriptor has been returned by the *open()* function.

EXAMPLE

```
int    fd;
STATUS retval;

/*----------------
  close the device
  ----------------*/

retval = close(fd);

if (retval == ERROR)
{
  /* Handle error */
}
```

RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - close()

4.3 read()

NAME

read() - read a message from specified TIP810 device

SYNOPSIS

```
int read (
        int         fd,
        char        *buffer,
        size_t      maxbytes
        )
```

DESCRIPTION

The read function reads a CAN message from the driver receive queue. A pointer to the caller's message buffer (*T810_MSG_BUF*) and the size of this structure are passed by the parameters *buffer* and *maxbytes* to the device driver.

```
typedef struct
{
        unsigned long       identifier;
        unsigned char       io_flags;
        unsigned char       msg_len;
        unsigned char       data[8];
        unsigned char       status;
        int                 timeout;
} T810_MSG_BUF, *PT810_MSG_BUF;
```

identifier

Receives the message identifier of the read CAN message.

io_flags

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

Value	Description
T810_EXTENDED	Set if the received message is an extended message frame. Reset for standard message frames.
T810_REMOTE_FRAME	Set if the received message is a remote transmission request (RTR) frame.

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.

timeout

Specifies the amount of time (in ticks) the caller is willing to wait for execution of read. A timeout value of WAIT_FOREVER means to wait indefinitely; a value of NO_WAIT means do not wait at all.

status

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

Value	Description
T810_SUCCESS	No messages lost
T810_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.
T810_MSGOBJ_OVERRUN	One or more messages were overwritten in the CAN controller message object because the interrupt latency is too large. Reduce the CAN bit rate or upgrade the system speed.

EXAMPLE

```
#include “tip810.h”

T810_MSG_BUF msg_buf;
int          fd;
int          num_bytes;

msg_buf.timeout = 200; /* wait max. 200 ticks */

num_bytes = read(fd, &msg_buf, sizeof(msg_buf));

if (num_bytes != ERROR)
{
  /* process received CAN message */
}
```

RETURNS

ERROR or number of data bytes read [0..8]. If read fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
S_t810Dev_TIMEOUT	The I/O request timed out
S_t810Drv_BUSOFF	The CAN bus controller has entered the BUS OFF state
S_t810Drv_EMPTY	No message in the read buffer (only if NO_WAIT option is selected)
S_t810Drv_ABORT	The I/O request was aborted because the CAN controller has entered the BUS OFF state or at least one of the Error Counters has reached the warning limit

SEE ALSO

ioLib, basic I/O routine - read()

4.4 write()

NAME

write() - write a message to specified TIP810 device

SYNOPSIS

```
int write (
        int             fd,
        char            *buffer,
        size_t          nbytes
        )
```

DESCRIPTION

The write function writes a CAN message to the device with descriptor *fd*. A pointer to the caller's message buffer (*T810_MSG_BUF*) and the size of this structure are passed by the parameters *buffer* and *nbytes* to the device.

```
typedef struct
{
        unsigned long           identifier;
        unsigned char           io_flags;
        unsigned char           msg_len;
        unsigned char           data[8];
        unsigned char           status;
        int                     timeout;
} T810_MSG_BUF, *PT810_MSG_BUF;
```

identifier

Contains the message identifier of the CAN message to write.

io_flags

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

Value	Description
T810_EXTENDED	Transmit an extended message frame. If this macro isn't set or the "dummy" macro T810_STANDARD is set a standard frame will be transmitted.
T810_REMOTE_FRAME	A remote transmission request (RTR bit is set) will be transmitted.

T810_SINGLE_SHOT	No re-transmission will be performed if an error occurred or the arbitration will be lost during transmission (single-shot transmission).
T810_SELF_RECEPTION	The message will be transmitted and simultaneously received if the acceptance filter is set to the corresponding identifier.

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.

timeout

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

status

This parameter is unused for this control function.

EXAMPLE

```
#include “tip810.h”

int          result;
T810_MSG_BUF msg_buf;

/* Write Extended Frame, ID = 1234, Data = {0xaa, 0x55}, Timeout = 200 ticks
** The transmitted frame will be received simultaneously.*/
msg_buf.identifier = 1234;
msg_buf.timeout    = 200;
msg_buf.io_flags   = T810_EXTENDED | T810_SELF_RECEPTION;
msg_buf.msg_len    = 2;
msg_buf.data[0]    = 0xaa;
msg_buf.data[1]    = 0x55;

result = write(fd, &msg_buf, sizeof(msg_buf));

if (result == ERROR) {
   printf( "\nWrite failed --> Error = 0x%08X.\n", errnoGet());
}
```

RETURNS

ERROR or number of data bytes written [0..8]. If write fails an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
S_t810Dev_TIMEOUT	The I/O request times out
S_t810Drv_BUSOFF	The CAN bus controller has entered the BUS OFF state
S_t810Drv_ILLLEN	Invalid message length
S_t810Drv_ABORT	The I/O request was aborted because the CAN controller has entered the BUS OFF state or at least one of the Error Counters has reached the warning limit

SEE ALSO

ioLib, basic I/O routine - write()

4.5 ioctl()

NAME

ioctl() - perform an I/O control function

SYNOPSIS

```
int ioctl (
        int     fd,
        int     function,
        int     arg
        )
```

DESCRIPTION

Special I/O operation that does not fit to the standard basic I/O calls will be performed by calling the *ioctl()* function with a specific function code and an optional function-dependent argument.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

request

This argument specifies the function that shall be executed. Following functions are defined:

Symbol	Meaning
FIO_T810_SET_BUSTIMING	Setup a new bus timing
FIO_T810_SET_FILTER	Setup acceptance filter
FIO_T810_SET_BUSON	Enter the bus on state
FIO_T810_SET_BUSOFF	Enter the bus off state
FIO_T810_GET_CANSTATUS	Returns CAN controller status information
FIO_T810_ENABLE_SELFTEST	Enable self test mode
FIO_T810_DISABLE_SELFTEST	Disable self test mode
FIO_T810_ENABLE_LISTENONLY	Enable listen only mode
FIO_T810_DISABLE_LISTENONLY	Disable listen only mode
FIO_T810_SET_LIMIT	Set new error warning limit
FIOFLUSH	This I/O control function flushes the read ring buffer
FIOCANCEL	Cancel a pending read or write request

arg

This parameter depends on the selected function (request). How to use this parameter is described below with the function.

RETURNS

OK or ERROR. If the function fails an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual). Function specific error codes will be described with the function.

Error code	Description
S_t810Drv_ILLREQUEST	The specified function code is not supported

SEE ALSO

ioLib, basic I/O routine - *ioctl()*

4.5.1 FIO_T810_SET_BUSTIMING

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 (*T810_TIMING*) is passed by the argument pointer ***arg*** 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;
} T810_TIMING, *PT810_TIMING;
```

timing_value

This parameter holds the new value for the bit timing register 0 (Bit 15…8) and for the bit timing register 1 (Bit 7...0). Possible transfer rates are between 20 kbit per second and 1.0 MBit per second. The include file 'tip810.h' contains predefined transfer rate symbols (T810_20KBIT ... T810_1MBIT, …).

For other transfer rates please follow the instructions of the *SJA1000 Product Specification*, which is also part of the engineering kit TIP810-EK.

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 rate to make the CAN bus more resistant against noise spikes.

EXAMPLE

```
#include “tip810.h”

int         fd;
int         result;
T810_TIMING timing;

timing.timing_value  = T810_100KBIT;
timing.three_samples = 0;

result = ioctl(fd, FIO_T810_SET_BUSTIMING, (int)&timing);

if (result == ERROR) {
    /* handle ioctl error */
}
```

ERROR CODES

Error code	Description
S_t810Drv_NOBUSOFF	The CAN bus controller is not in BUS OFF state

SEE ALSO

tip810.h for predefined bus timing constants

SJA1000 Product Specification Manual – 6.5.1/2 BUS TIMING REGISTER

4.5.2 FIO_T810_SET_FILTER

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 (*T810_FILTER*) is passed by the parameter pointer ***arg*** to the driver.

```
typedef struct
{
    int             single_filter;
    unsigned long   acceptance_code;
    unsigned long   acceptance_mask;
}T810_FILTER, *PT810_FILTER;
```

single_filter

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

acceptance_code

The contents of this parameter will be written to acceptance code register of the controller. For TIP810 V1.0 only Bits 7…0 are used.

acceptance_mask

The contents of this parameter will be written to the acceptance mask register of the controller. For TIP810 V1.0 only Bits 7…0 are used.

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

EXAMPLE

```
#include “tip810.h”

int         fd;
int         result;
T810_FILTER filter;

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

/* Mark all bit position don't care */
filter.acceptance_mask = 0xffffffff;

/* Single Filter Mode */
filter.single_filter = 1;    // TRUE

result = ioctl(fd, FIO_T810_SET_FILTER, (int)&filter);

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

ERROR CODES

Error code	Description
S_t810Drv_NOBUSOFF	The CAN bus controller is not in BUS OFF state

SEE ALSO

SJA1000 Product Specification Manual – *6.4.15 ACCEPTANCE FILTER (PeliCAN Mode)*

4.5.3 FIO_T810_SET_BUSON

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 “tip810.h”

int fd;
int result;

result = ioctl(fd, FIO_T810_SET_BUSON, 0);

if (result == ERROR) {
    /* handle ioctl error */
}
```

SEE ALSO

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

4.5.4 FIO_T810_SET_BUSOFF

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 FIO_T810_SET_BUSON is executed. The optional argument pointer is not used by this ioctl function.

EXAMPLE

```
#include “tip810.h”

int fd;
int result;

result = ioctl(fd, FIO_T810_SET_BUSOFF, 0);

if (result == ERROR) {
    /* handle ioctl error */
}
```

ERRORS

S_t810Drv_NOBUSOFF	Unable to enter the BUSOFF mode.

SEE ALSO

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

4.5.5 FIO_T810_GET_CANSTATUS

This ioctl function returns the actual contents of several CAN controller registers for diagnostic purposes. A pointer to the caller's status buffer (*T810_STATUS*) is passed by the parameter ***arg*** to the device driver.

```
typedef struct
{
        unsigned char       arbitration_lost_capture;
        unsigned char       error_code_capture;
        unsigned char       tx_error_counter;
        unsigned char       rx_error_counter;
        unsigned char       error_warning_limit;
        unsigned char       status_register;
        unsigned char       mode_register;
        unsigned char       rx_message_counter_max;
} T810_STATUS, *PT810_STATUS;
```

arbitration_lost_capture

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

error_code_capture

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

tx_error_counter

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

rx_error_counter

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

error_warning_limit

This parameter receives content of the error warning limit register.

status_register

This parameter receives content of the status register.

mode_register

This parameter receives the content of the mode register.

rx_message_counter_max

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

EXAMPLE

```
#include “tip810.h”

int           fd;
int           result;
T810_STATUS   can_status;

result = ioctl(fd, FIO_T810_GET_CANSTATUS, (int)&can_status);

if (result == ERROR) {
    /* handle ioctl error */
}
```

SEE ALSO

SJA1000 Product Specification Manual

4.5.6 FIO_T810_ENABLE_SELFTEST

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 FIO_T810_SET_BUSOFF first otherwise you will get an error (S_t810Drv_NOBUSOFF).

EXAMPLE

```
#include “tip810.h”

int fd;
int result;

result = ioctl(fd, FIO_T810_ENABLE_SELFTEST, 0);

if (result == ERROR) {
    /* handle ioctl error */
}
```

ERRORS

S_t810Drv_NOBUSOFF	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)*

4.5.7 FIO_T810_DISABLE_SELFTEST

This ioctl function disables the self test facility of the SJA1000 CAN controller, which was enabled before with the ioctl command FIO_T810_ENABLE_SELFTEST.

The optional argument pointer can be omitted for this function.

This ioctl command will be accepted only in reset mode (BUSOFF). Enter FIO_T810_SET_BUSOFF first otherwise you will get an error (S_t810Drv_NOBUSOFF).

EXAMPLE

```
#include “tip810.h”

int fd;
int result;

result = ioctl(fd, FIO_T810_DISABLE_SELFTEST, 0);

if (result == ERROR) {
    /* handle ioctl error */
}
```

ERRORS

S_t810Drv_NOBUSOFF	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)*

4.5.8 FIO_T810_ENABLE_LISTENONLY

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 FIO_T810_SET_BUSOFF first otherwise you will get an error (S_t810Drv_NOBUSOFF).

EXAMPLE

```
#include “tip810.h”

int fd;
int result;

result = ioctl(fd, FIO_T810_ENABLE_LISTENONLY, 0);

if (result == ERROR) {
    /* handle ioctl error */
}
```

ERRORS

S_t810Drv_NOBUSOFF	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)*

4.5.9 FIO_T810_DISABLE_LISTENONLY

This ioctl function disables the listen only facility of the SJA1000 CAN controller, which was enables before with the ioctl command FIO_T810_ENABLE_LISTENONLY.

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

This ioctl command will be accepted only in reset mode (BUSOFF). Enter FIO_T810_SET_BUSOFF first otherwise you will get an error (S_t810Drv_NOBUSOFF).

EXAMPLE

```
#include “tip810.h”

int fd;
int result;

result = ioctl(fd, FIO_T810_DISABLE_LISTENONLY, 0);

if (result == ERROR) {
    /* handle ioctl error */
}
```

ERRORS

S_t810Drv_NOBUSOFF	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)*

4.5.10 FIO_T810_SET_LIMIT

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 ***arg*** to the driver.

This ioctl command will be accepted only in reset mode (BUSOFF). Enter FIO_T810_SET_BUSOFF first otherwise you will get an error (S_t810Drv_NOBUSOFF).

EXAMPLE

```
#include “tip810.h”

int           fd;
int           result;
unsigned char limit;

limit = 200;

result = ioctl(fd, FIO_T810_SET_LIMIT, (int)&limit);

if (result == ERROR) {
    /* handle ioctl error */
}
```

ERRORS

S_t810Drv_NOBUSOFF	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)*

4.5.11 FIOFLUSH

This I/O control function clears the read ring buffer.

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

EXAMPLE

```
#include “tip810.h”

int fd;
int result;

result = ioctl(fd, FIOFLUSH, 0);

if (result == ERROR) {
    /* handle ioctl error */
}
```

4.5.12 FIOCANCEL

This I/O control function cancels pending read or write requests. Aborted requests return an ERROR with errno set to S_t810Drv_ABORT.

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

EXAMPLE

```
#include “tip810.h”

int fd;
int result;

result = ioctl(fd, FIOCANCEL, 0);

if (result == ERROR) {
    /* handle ioctl error */
}
```

5 Appendix

5.1 Predefined Symbols

The following symbols are defined in the file *tip810.h.*

Bit Rates (FIO_T810_SET_BUSTIMING)

T810_1MBIT	0x0014	1 Mbit/s	Max. cable length: 40 m
T810_500KBIT	0x001c	500 Kbit/s	Max. cable length: 130 m
T810_250KBIT	0x011c	250 Kbit/s	Max. cable length: 270 m
T810_125KBIT	0x031c	125 Kbit/s	Max. cable length: 530 m
T810_100KBIT	0x432f	100 Kbit/s	Max. cable length: 620 m
T810_50KBIT	0x472f	50 Kbit/s	Max. cable length: 1.3 km
T810_20KBIT	0x532f	20 Kbit/s	Max. cable length: 3.3 km