The Embedded I/O Company

TEWS TECHNOLOGIES

TIP810-SW-82

Linux Device Driver

CAN Bus IP

Version 1.2.x

User Manual

Issue 1.2.3

July 2008

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

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

TIP810-SW-82

Linux Device Driver

CAN Bus IP

Supported Modules:
TIP810

Issue	Description	Date
1.0	First Issue	April 29, 2003
1.1	Support for DEVFS and SMP	March 2, 2004
1.2	Update File List	April 30, 2004
1.2.1	Filelist modified, introduction enhanced, new address TEWS LLC	October 16, 2006
1.2.2	UDEV description added, Filelist corrected	November 26, 2007
1.2.3	Carrier Driver description added	July 7, 2008

Table of Contents

1 Introduction

1.1 Device Driver

The TIP810-SW-82 Linux device driver allows the operation of a TIP810 CAN Bus IP on Linux operating systems.

Because the TIP810 device driver is stacked on the TEWS TECHNOLOGIES IPAC carrier driver, it is necessary to install also the appropriate IPAC carrier driver. Please refer to the IPAC carrier driver user manual for further information.

The TIP810 device driver includes the following functions:

- Transmission and receive of Standard and Extended Identifiers
- Standard bit rates from 5 kbit up to 1 Mbit and user defined bit rates
- Message acceptance filtering
- Single-Shot transmission
- Listen only mode
- Message self reception
- Programmable error warning limit

The TIP810-SW-82 supports the modules listed below:

TIP810-10	1 Channel CAN Bus	(IndustryPack ®)

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

TIP810 User manual

TIP810 Engineering Manual

SJA1000 PeliCAN controller User Manual

CARRIER-SW-82 IPAC Carrier 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-82 is part of this TIP810-SW-82 distribution. It is located in directory CARRIER-SW-82 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-82 User Manual for a detailed description how to install and setup the CARRIER-SW-82 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

The directory TIP810-SW-82 on the distribution media contains the following files:

TIP810-SW-82-1.2.3.pdf	This manual in PDF format
TIP810-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
Release.txt	Release information
ChangeLog.txt	Release history

The GZIP compressed archive TIP810-SW-82-SRC.tar.gz contains the following files and directories:

Directory path ‘./tip810/’:

tip810.c	Driver source code
tip810def.h	Driver include file
tip810.h	Driver include file for application program
sja1000.h	Driver include file (CAN Controller Spec.)
makenode	Script to create device nodes on the file system
Makefile	Device driver make file
example/tip810exa.c	Example application
example/Makefile	Example application make file
include/config.h	Driver independent library header file
include/tpmodule.h	Kernel independent library header file
include/tpmodule.c	Kernel independent library source code file

In order to perform an installation, extract all files of the archive TIP810-SW-82-SRC.tar.gz to the desired target directory. The command ‘tar -xzvf TIP810-SW-82-SRC.tar.gz’ will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tip810.h to */usr/include*

Before building a new device driver, the TEWS TECHNOLOGIES IPAC carrier driver must be installed properly, because this driver includes the header file *ipac_carrier.h*, which is part of the IPAC carrier driver distribution. Please refer to the IPAC carrier driver user manual in the directory path *CARRIER-SW-82* on the separate distribution media.

2.1 Build and install the device driver

- Login as root
- Change to the target directory
- To create and install the driver in the module directory /lib/modules/<version>/misc enter:

  # make install

  For Linux kernel 2.6.x, there may be compiler warnings claiming some undefined ipac_* symbols. These warnings are caused by the IPAC carrier driver, which is unknown during compilation of this TIP driver. The warnings can be ignored.

- Also 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 the correct IPAC carrier driver 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 device driver into the running kernel

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

  # modprobe tip810drv

- 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 TIP810 module found. The first TIP810 can be accessed with device node /dev/tip810_0, the second TIP810 with device node /dev/tip810_1, the third TIP810 with device node /dev/tip810_2 and so on.

The allocation of device nodes to physical TIP810 modules depends on the search order of the IPAC carrier driver. Please refer to the IPAC carrier user manual.

Loading of the TIP810 device driver will only work if kernel KMOD support is installed, necessary carrier board drivers already installed and the kernel dependency file is up to date. If KMOD support isn't available you have to build either a new kernel with KMOD installed or you have to install the IPAC carrier kernel modules manually in the correct order (please refer to the IPAC carrier driver user manual).

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 tip810drv –r

If your kernel has enabled devfs or sysfs (udev), all /dev/tip810_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 "*tip810drv: 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 TIP810 driver uses dynamic allocation of major device numbers per default. If this isn't suitable for the application it's possible to define a major number for the driver.

To change the major number edit the file tip810.c, change the following symbol to appropriate value and enter make install to create a new driver.

TIP810_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TIP810_MAJOR       122
```

2.6 Receive Queue Configuration

Received CAN messages will be stored in a FIFO buffer. The depth of the FIFO can be adapted by changing the following symbol in tip810def.h.

T810_RX_FIFO_SIZE	Defines the depth of the message FIFO buffer (default = 100). Valid numbers are in range between 1 and MAXINT.

3 Device Input/Output functions

This chapter describes the interface to the device driver I/O system 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/tip810_0”, O_RDWR);
if (fd == -1)
{
   /* handle error condition */
}
```

RETURNS

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

ERRORS

ENODEV	The requested minor device does not exist.

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

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.2 close()

NAME

close() – close a file descriptor

SYNOPSIS

#include <unistd.h>

int close (int filedes)

DESCRIPTION

The close function closes the file descriptor *filedes*.

EXAMPLE

```
int fd;

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

RETURNS

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

ERRORS

ENODEV	The requested minor device does not exist.

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

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 read()

NAME

read() – read from a device

SYNOPSIS

#include <unistd.h>

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

DESCRIPTION

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

```
typedef struct
{
    unsigned long       Identifier;
    unsigned char       IOFlags;
    unsigned char       MsgLen;
    unsigned char       Data[8];
    long                Timeout;
    unsigned char       Status;
} T810_MSG_BUF, *PT810_MSG_BUF;
```

Identifier

Receives the message identifier of the read CAN message.

IOFlags

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

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.

MsgLen

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

Data

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

Timeout

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

Status

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

T810_SUCCESS	No messages lost
T810_FIFO_OVERRUN	One or more messages ware 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>

int          fd;
ssize_t      numBytes;
T810_MSG_BUF msgBuf;

msgBuf.Timeout = 200;

numBytes = read(fd, &msgBuf, sizeof(msgBuf));
if (numBytes > 0) {
  /* process received CAN message */
}
```

RETURNS

On success read returns the size of structure T810_MSG_BUF. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

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

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.4 write()

NAME

write() – write to a device

SYNOPSIS

#include <unistd.h>

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

DESCRIPTION

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

```
typedef struct
{
        unsigned long           Identifier;
        unsigned char           IOFlags;
        unsigned char           MsgLen;
        unsigned char           Data[8];
        long                    Timeout;
        unsigned char           Status;
} T810_MSG_BUF, *PT810_MSG_BUF;
```

Identifier

Contains the message identifier of the CAN message to write.

IOFLags

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

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.

MsgLen

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

Data

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

Timeout

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

Status

This parameter is unused for this control function.

EXAMPLE

```
#include <tip810.h>

int          fd;
int          result;
T810_MSG_BUF msgBuf;

...

/*
** Write two data bytes with extended identifier 1234 to
** the CANbus and wait max. 200 ticks on execution.
** The transmitted frame will be received simultaneously.
*/
msgBuf.Identifier = 1234;
msgBuf.Timeout = 200;
msgBuf.IOFlags = T810_EXTENDED | T810_SELF_RECEPTION;
msgBuf.MsgLen = 2;
msgBuf.Data[0] = 0xaa;
msgBuf.Data[1] = 0x55;

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

if( result != sizeof(T810_MSG_BUF)) {
  printf( "\nWrite failed --> Error = %d.\n", errno );
}
```

RETURNS

On success write returns the size of structure T810_MSG_BUF. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

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

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.5 ioctl()

NAME

ioctl() – device control functions

SYNOPSIS

#include <tip810.h>
#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 tip810.h:

Symbol	Meaning
T810_IOCSBITTIMING	Setup a new bit timing
T810_IOCSSETFILTER	Setup acceptance filter
T810_IOCBUSON	Enter the bus on state
T810_IOCBUSOFF	Enter the bus off state
T810_IOCFLUSH	Flush one or all receive queues
T810_IOCGCANSTATUS	Returns CAN controller status information
T810_IOCENABLE_SELFTEST	Enable self test mode
T810_IOCDISABLE_SELFTEST	Disable self test mode
T810_IOCENABLE_LISTENONLY	Enable listen only mode
T810_IOCDISABLE_LISTENONLY	Disable listen only mode
T810_IOCSSETLIMIT	Set new error warning limit

See behind for more detailed information on each control code.

To use these TIP810 specific control codes the header file tip810.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 TIP810 driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.5.1 T810_IOCSBITTIMING

NAME

T810_IOCSBITTIMING - Setup new bit timing

DESCRIPTION

This ioctl function modifies the bit timing register of the CAN controller to setup a new CAN bus transfer speed. A pointer to the caller's parameter buffer (*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          TimingValue;
        unsigned short          ThreeSamples;
} T810_TIMING, *PT810_TIMING;
```

TimingValue

This parameter holds the new value for the bit timing register 0 (bit 0...7) and for the bit timing register 1 (bit 8...15). Possible transfer rates are between 5 Kbit per second and 1 Mbit per second. The include file 'tip810.h' contains predefined transfer rate symbols (T810_5KBIT ... 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.

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 rates to make the CAN bus more immune against noise spikes.

EXAMPLE

```
#include <tip810.h>

int          fd;
int          result;
T810_TIMING  BitTimingParam;

...

BitTimingParam.TimingValue = T810_100KBIT;
BitTimingParam.ThreeSamples = FALSE;

result = ioctl(fd, T810_IOCSBITTIMING, (char*)&BitTimingParam);

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

ERRORS

EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.
EFAULT	An error occurred during memory data transfer between kernel and user space.

SEE ALSO

tip810.h for predefined bus timing constants

SJA1000 Product Specification Manual – 6.5.1/2 BUS TIMING REGISTER

3.5.2 T810_IOCSSETFILTER

NAME

T810_IOCSSETFILTER - Setup acceptance filter

DESCRIPTION

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

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

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

The corresponding acceptance mask registers allow defining certain bit positions to be "don't care" (an 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                 SingleFilter;
        unsigned long       AcceptanceCode;
        unsigned long       AcceptanceMask;
}T810_FILTER, *PT810_FILTER;
```

SingleFilter

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

AcceptanceCode

The contents of this parameter will be written to acceptance code register of the controller. This value must be shifted to match the desired identifier (refer to SJA1000 manual).

AcceptanceMask

The contents of this parameter will be written to the acceptance mask register of the controller. This value must be shifted to match the desired identifier (refer to SJA1000 manual).

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 AcceptFilter;

...

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

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

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

result = ioctl(fd, T810_IOCSSETFILTER, (char*)&AcceptFilter);

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

ERRORS

EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.
EFAULT	An error occurred during memory data transfer between kernel and user space.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.15 ACCEPTANCE FILTER*

3.5.3 T810_IOCBUSON

NAME

T810_IOCBUSON - Enter the bus on state

DESCRIPTION

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

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

The optional argument can be omitted for this ioctl function.

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

EXAMPLE

```
#include <tip810.h>

int fd;
int result;

...

result = ioctl(fd, T810_IOCBUSON);

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

ERRORS

This ioctl function returns no specific error codes.

SEE ALSO

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

3.5.4 T810_IOCBUSOFF

NAME

T810_IOCBUSOFF - Enter the bus off state

DESCRIPTION

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

After execution of this control function the CAN controller is completely removed from the CAN bus and cannot communicate until the control function T810_IOCBUSON is executed. Note: During a pending write of another concurrent thread it's not possible to set the device bus off.

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

EXAMPLE

```
#include <tip810.h>

int fd;
int result;

...

result = ioctl(fd, T810_IOCBUSOFF);

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

ERRORS

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

SEE ALSO

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

3.5.5 T810_IOCFLUSH

NAME

T810_IOCFLUSH - Flush the received message FIFO

DESCRIPTION

This ioctl function flushes the FIFO buffer of received messages.

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

EXAMPLE

```
#include <tip810.h>

int fd;
int result;

...

result = ioctl(fd, T810_IOCFLUSH);

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

ERRORS

This ioctl function returns no function specific error codes.

3.5.6 T810_IOCGCANSTATUS

NAME

T810_IOCGCANSTATUS - Returns CAN controller status information

DESCRIPTION

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

```
typedef struct
{
        unsigned char       ArbitrationLostCapture;
        unsigned char       ErrorCodeCapture;
        unsigned char       TxErrorCounter;
        unsigned char       RxErrorCounter;
        unsigned char       ErrorWarningLimit;
        unsigned char       StatusRegister;
        unsigned char       ModeRegister;
        unsigned char       RxMessageCounterMax;
} T810_STATUS, *PT810_STATUS;
```

ArbitrationLostCapture

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

ErrorCodeCapture

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

TxErrorCounter

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

RxErrorCounter

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

ErrorWarningLimit

This parameter receives content of the error warning limit register.

StatusRegister

This parameter receives content of the status register.

ModeRegister

This parameter receives the content of the mode register.

RxMessageCounterMax

This parameter 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 CanStatus;

...

result = ioctl(fd, T810_IOCGCANSTATUS, (char*)&CanStatus);

if (result < 0) {
  /* handle ioctl error */
} else {
  printf("ModeRegister = 0x%02X\n", CanStatus.ModeRegister);
}
...
```

ERRORS

EFAULT	An error occurred during memory data transfer between kernel and user space.

SEE ALSO

SJA1000 Product Specification Manual

3.5.7 T810_IOCENABLE_SELFTEST

NAME

T810_IOCENABLE_SELFTEST - Enable self test mode

DESCRIPTION

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

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

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

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

This ioctl command will be accepted only in reset mode (BUSOFF). Enter T810_IOCBUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include <tip810.h>

int fd;
int result;

result = ioctl(fd, T810_IOCENABLE_SELFTEST);

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

ERRORS

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

SEE ALSO

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

3.5.8 T810_IOCDISABLE_SELFTEST

NAME

T810_IOCDISABLE_SELFTEST - Disable self test mode

DESCRIPTION

This ioctl function disables the self test facility of the SJA1000 CAN controller, which was before enabled with the ioctl command T810_IOCENABLE_SELFTEST.

The optional argument pointer can be omitted for this function.

This ioctl command will be accepted only in reset mode (BUSOFF). Enter T810_IOCBUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include <tip810.h>

int fd;
int result;

...

result = ioctl(fd, T810_IOCDISABLE_SELFTEST);

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

ERRORS

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

SEE ALSO

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

3.5.9 T810_IOCENABLE_LISTENONLY

NAME

T810_IOCENABLE_LISTENONLY - Enable listen only mode

DESCRIPTION

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

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

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

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

This ioctl command will be accepted only in reset mode (BUSOFF). Enter T810_IOCBUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include <tip810.h>

int fd;
int result;

...

result = ioctl(fd, T810_IOCENABLE_LISTENONLY);

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

ERRORS

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

SEE ALSO

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

3.5.10 T810_IOCDISABLE_LISTENONLY

NAME

T810_IOCDISABLE_LISTENONLY - Disable listen only mode

DESCRIPTION

This ioctl function disables the listen only facility of the SJA1000 CAN controller, which was before enabled with the ioctl command T810_IOCENABLE_LISTENONLY.

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

This ioctl command will be accepted only in reset mode (BUSOFF). Enter T810_IOCBUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include <tip810.h>

int fd;
int result;

...

result = ioctl(fd, T810_DISABLE_LISTENONLY);

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

ERRORS

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

SEE ALSO

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

3.5.11 T810_IOCSSETLIMIT

NAME

T810_IOCSSETLIMIT - Set new error warning limit

DESCRIPTION

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

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

This ioctl command will be accepted only in reset mode (BUSOFF). Enter T810_IOCBUSOFF first otherwise you will get an error (EACCES).

EXAMPLE

```
#include <tip810.h>

int fd;
int result;
unsigned char limit;

limit = 200;

result = ioctl(fd, T810_IOCSSETLIMIT, (char*)&limit);

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

ERRORS

EACCES	The CAN controller is in operating mode. This mode can be changed only in reset mode.
EFAULT	An error occurred during memory data transfer between kernel and user space.

SEE ALSO

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

3.6 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 = T810_100KBIT;
BitTimingParam.ThreeSamples = FALSE;

result = ioctl(fd, T810_IOCSBITTIMING, (char*)&BitTimingParam);



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

result = ioctl(fd, T810_IOCSSETFILTER, (char*)&AcceptFilter);



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

result = ioctl(fd, T810_IOCBUSON);
```

Now you should be able to send and receive CAN messages with appropriate calls to write() and read() functions.

4 Debugging

For debugging output see driver file tip810.c. You will find the two following symbols:

```
#undef TIP810_DEBUG_INTR
#undef TIP810_DEBUG_VIEW
```

To enable a debug output replace “undef” with “define”.

The TIP810_DEBUG_INTR symbol controls debugging output from the ISR.

```
TIP810 : Get bus error interrupt. tip810_0
TIP810 : Get transmit interrupt. tip810_0
TIP810 : Get receive interrupt. tip810_1
```

The TIP810_DEBUG_VIEW symbol controls debugging output from the remaining part of the driver.

```
TIP810 - CAN Bus IP - version 1.2.3 (2007-11-26)<6>
TIP810 :  Probe new TIP810 mounted on <TEWS TECHNOLOGIES - (Compact)PCI
IPAC Carrier> at slot C

TIP810 :  IP I/O Memory Space
00000000 : FF 01 FF 00 FF 3C FF 00 FF 00 FF 00 FF 2F FF 43
00000010 : FF DA FF 00 FF 00 FF 00 FF 00 FF 60 FF 00 FF 00
00000020 : FF 00 FF 00 FF 00 FF 00 FF FF FF FF FF FF FF FF
00000030 : FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 80
00000040 : FF 05 FF 05 FF 05 FF 05 FF 05 FF 05 FF 05 FF 05
00000050 : FF 05 FF 05 FF 05 FF 05 FF 05 FF 05 FF 05 FF 05
00000060 : FF 05 FF 05 FF 05 FF 05 FF 05 FF 05 FF 05 FF 05
00000070 : FF 05 FF 05 FF 05 FF 05 FF 05 FF 05 FF 05 FF 05
00000080 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000090 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000A0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000B0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000C0 : FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000D0 : FF FF FF FF FF FF FF FF FF FF FF FF

TIP810 :  IP ID Memory Space
00000000 : FF 49 FF 50 FF 41 FF 43 FF B3 FF 01 FF 20 FF 00
00000010 : FF 00 FF 00 FF 0C FF 9D FF 0A FF 00 FF 00 FF 00
00000020 : FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
00000030 : FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00
TIP810 : Enable Ints for 0xca06f000
```

If you have trouble with the driver please enable the debug output and send us a copy of the results. The kernel context output is generated with “printk” and is appended to /var/log/messages or wherever it is piped in your system.

For debugging please run

tail –f /var/log/messages

at first and then install the driver.