The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV010-SW-72

LynxOS Device Driver

Isolated 2 x CAN Bus

Version 1.0.x

User Manual

Issue 1.0.1

February 2007

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

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

TDRV010-SW-72

Isolated 2 x CAN Bus

LynxOS Device Driver

Issue	Description	Date
1.0.0	First Issue	March 3, 2006
1.0.1	New Address TEWS LLC	February 28, 2007

Table of Contents

1 Introduction

The TDRV010-SW-72 LynxOS device driver allows the operation of the TPMC810 and TPMC310 product family on LynxOS platforms with DRM based PCI interface.

The standard file (I/O) functions (open, close, ioctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and configuration operations.

The TDRV010 device driver includes the following functions:

- Reading received messages from input FIFO
- Sending messages
- Set channel Bus On/Off
- Configure Listen-Only mode On/Off
- Configure Selftest mode On/Off
- Extended and Standard Identifiers
- Configure Bitrate from 20 kbit up to 1 Mbit and user defined bit rates
- Configure Receive Mask
- Flush receive FIFO
- Read CAN status

The TDRV010-SW-72 supports the modules listed below:

TPMC310	Isolated 2x CAN Bus (Conduction Cooled) (64 pin connector for Back-IO)	PMC
TPMC810	Isolated 2x CAN Bus (2x SUBD 9 pin connectors for Front-Panel I/O) (64 pin connector for Back-I/O)	PMC

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

TPMC310/TPMC810 User manual

TPMC310/TPMC810 Engineering Manual

2 Installation

Following files are located in the directory TDRV010-SW-72 on the distribution media:

TDRV010-SW-72-1.0.1.pdf	This manual in PDF format
TDRV010-SW-72-SRC.tar	Device Driver and Example sources
Release.txt	Information about the Device Driver Release

The TAR archive TDRV010-SW-72-SRC.tar contains the following files and directories:

tdrv010.c	Driver source code
tdrv010.h	Definitions and data structures for driver and application
tdrv010def.h	Definitions and data structures for the driver
sja1000.h	Definitions for SJA1000 controller
tdrv010_info.c	Device information definition
tdrv010_info.h	Device information definition header
tdrv010.cfg	Driver configuration file include
tdrv010.import	Linker import file
Makefile	Device driver make file
example/tdrv010exa.c	Example application source
example/Makefile	Example application make file

In order to perform a driver installation first extract the TAR file to a temporary directory, then follow the steps below:

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/tdrv010 or /sys/drivers.cpci_x86/tdrv010

2. Copy the following files to this directory:
   - tdrv010.c
   - tdrv010def.h
   - sja1000.h
   - tdrv010.import
   - Makefile

3. Copy tdrv010.h to /usr/include/

4. Copy tdrv010_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tdrv010_info.h to /sys/dheaders/

6. Copy tdrv010.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform. For example: /sys/cfg.ppc or /sys/cfg.x86 ....

2.1 Device Driver Installation

The two methods of driver installation are as follows:

- Static Installation
- Dynamic Installation (only native LynxOS systems)

2.1.1 Static Installation

With this method, the driver object code is linked with the kernel routines and is installed during system start-up.

2.1.1.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tdrv010, where xxx represents the BSP that supports the target hardware.
2. To update the library /sys/lib/libdrivers.a   enter:

   ```
   make install
   ```

2.1.1.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).
2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = ... tdrv010_info.x
   ```

   And at the end of the Makefile

   ```
   tdrv010_info.o:$(DHEADERS)/tdrv010_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

2.1.1.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os/ respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry at the end of this file.

   ```
   I:tdrv010.cfg
   ```

2.1.1.4 Rebuild the Kernel

1. Change to the directory /sys/lynx.os/ (/sys/bsp.xxx)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

   The N flag instructs init to run mknod and create all the nodes mentioned in the new nodetab.

4. After reboot you should find the following new devices (depends on the device configuration): /dev/tdrv010a1, /dev/tdrv010a2, …

2.1.2 Dynamic Installation

This method allows you to install the driver after the operating system is booted. The driver object code is attached to the end of the kernel image and the operating system dynamically adds this driver to its internal structures. The driver can also be removed dynamically.

2.1.2.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tdrv010, where xxx represents the BSP that supports the target hardware.
2. To make the dynamic link-able driver enter :

   ```
   make dldd
   ```

2.1.2.2 Create Device Information Declaration

1. Change to the directory /sys/drivers.xxx/tdrv010, where xxx represents the BSP that supports the target hardware.
2. To create a device definition file for the major device (this works only on native systems)

   ```
   make t010info
   ```

3. To install the driver enter:

   ```
   drinstall –c tdrv010.obj
   ```

   If successful, drinstall returns a unique <driver-ID>
4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> t010info
   ```

   The <driver-ID> is returned by the drinstall command
5. To create nodes for both minor devices enter:

   ```
   mknod /dev/tdrv010a1 c <major_no> 0
   mknod /dev/tdrv010a2 c <major_no> 1
   ```

   The <major_no> is returned by the devinstall command.

If all steps are successfully completed, the TDRV010 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TDRV010 device enter the following commands:

```
devinstall –u –c <device-ID>
drinstall –u <driver-ID>
```

2.1.3 Device Information Definition File

The device information definition contains information necessary to install the TDRV010 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tdrv010_info.h*.

This structure contains the following parameter:

PCIBusNumber	Contains the PCI bus number at which the supported device is connected. Valid bus numbers are in range from 0 to 255.
PCIDeviceNumber	Contains the device number (slot) at which the supported device is connected. Valid device numbers are in range from 0 to 31.

If both PCIBusNumber and PCIDeviceNumber are –1 then the driver will auto scan for supported devices. The first device found in the scan order will be allocated by the driver for this major device.

Already allocated devices can't be allocated twice. This is important to know if there are more than one TDRV010 major devices.

A device information definition is unique for every TDRV010 major device. The file *tdrv010_info.c* on the distribution media contains two device information declarations, **tdrv010a_info** for the first major device and **tdrv010b_info** for the second major device.

If the driver should support more than two major devices it is necessary to copy and paste an existing declaration and rename it with a unique name, for example **tdrv010c_info**, **tdrv010d_info** and so on.

It is also necessary to modify the device and driver configuration file, respectively the configuration include file *tdrv010.cfg*.

The following device declaration information uses the auto find method to detect a supported device on the PCI bus.

```
TDRV010_INFO tdrv010a_info = {

     -1,           /*  Auto find the device on any PCI bus   */
     -1,

};
```

2.1.4 Configuration File: CONFIG.TBL

The device and driver configuration file CONFIG.TBL contains entries for device drivers and its major and minor device declarations. Each time the system is rebuild, the config utility read this file and produces a new set of driver and device configuration tables and a corresponding nodetab.

To install the TDRV010 driver and devices into the LynxOS system, the configuration include file tdrv010.cfg must be included in the CONFIG.TBL (see also chapter 2.1.1.3).

The file tdrv010.cfg on the distribution disk contains the driver entry (*C:tdrv010:\....*) and two major device entries (*D:TDRV010 1:tdrv010a_info:: and D:TDRV010 2:tdrv010b_info::*).

If the driver should support more than one major device, the following entries for major devices must be enabled by removing the comment character (#). By copy and paste an existing major and minor entries and renaming the new entries, it is possible to add any number of additional TDRV010 devices.

This example shows a driver entry with one major device and one minor device:

```
#    Format:
#    C:driver-name:open:close:read:write:select:control:install:uninstall
#    D:device-name:info-block-name:raw-partner-name
#    N:node-name:minor-dev

C:tdrv010:\
      :tdrv010open:tdrv010close:::\
      ::tdrv010ioctl:tdrv010install:tdrv010uninstall
D:TDRV010 1:tdrv010a_info::
N:tdrv010a1:0
N:tdrv010a2:1
D:TDRV010 2:tdrv010b_info::
N:tdrv010b1:0
N:tdrv010b2:1
```

The configuration above creates the following nodes in the /dev directory.

```
/dev/tdrv010a1  /dev/tdrv010a2  /dev/tdrv010b1  /dev/tdrv010b2
```

2.2 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 symbols in tdrv010def.h.

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

3 TDRV010 Device Driver Programming

LynxOS system calls are all available directly to any C program. They are implemented as ordinary function calls to "glue" routines in the system library, which trap to the OS code.

Note that many system calls use data structures, which should be obtained in a program from appropriate header files. Necessary header files are listed with the system call synopsis.

3.1 open()

NAME

open() - open a file

SYNOPSIS

#include <sys/file.h>
#include <sys/types.h>
#include <fcntl.h>

int open (char *path, int oflags[, mode_t mode])

DESCRIPTION

Opens a file (TDRV010 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TDRV010 device ***oflags*** must be set to **O_RDWR** to open the file for both reading and writing.

The ***mode*** argument is required only when a file is created. Because a TDRV010 device already exists this argument is ignored.

EXAMPLE

```
int fd

fd = open ("/dev/tdrv010a1", O_RDWR);
```

RETURNS

open returns a file descriptor number if successful, or –1 on error.

SEE ALSO

LynxOS System Call - open()

3.2 close()

NAME

close() – close a file

SYNOPSIS

int close(int fd)

DESCRIPTION

This function closes an opened device.

EXAMPLE

```
int  result;

result = close(fd);
```

RETURNS

close returns 0 (OK) if successful, or –1 on error

SEE ALSO

LynxOS System Call - close()

3.3 ioctl()

NAME

ioctl() – I/O device control

SYNOPSIS

```
#include <ioctl.h>
#include <tdrv010.h>

int ioctl (int fd, int request, char *arg)
```

DESCRIPTION

ioctl provides a way of sending special commands to a device driver. The call sends the value of request and the pointer arg to the device associated with the descriptor fd.

The following ioctl codes are supported by the driver and are defined in *tdrv010.h*:

Symbol	Meaning
TDRV010_READ	Read a CAN message
TDRV010_WRITE	Write a CAN message
TDRV010_BITTIMING	Setup bit-timing
TDRV010_SETFILTER	Setup acceptance filter
TDRV010_BUSON	Enter bus ON mode
TDRV010_BUSOFF	Enter bus OFF mode
TDRV010_FLUSH	Flush receive FIFO
TDRV010_CANSTATUS	Read CAN status from controller
TDRV010_ENABLE_SELFTEST	Enable self test mode
TDRV010_DISABLE_SELFTEST	Disable self test mode
TDRV010_ENABLE_LISTENONLY	Enable listen only mode
TDRV010_DISABLE_LISTENONLY	Disable listen only mode
TDRV010_SETLIMIT	Set warning limit

See behind for more detailed information on each control code.

RETURNS

ioctl returns 0 if successful, or –1 on error.

On error, *errno* will contain a standard error code (see also LynxOS System Call – ioctl).

SEE ALSO

LynxOS System Call - ioctl().

tdrv010exa.c programming example

3.3.1 TDRV010_READ

NAME

TDRV010_READ – Read a CAN message

DESCRIPTION

This function reads a CAN message from the device driver receive queue. A pointer to the callers message buffer (*TDRV010_MSG_BUF*) must be passed by the parameter *arg* to the device.

The *TDRV010_MSG_BUF* structure has the following layout:

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

Members

Identifier

Obtains the message identifier of the read CAN message.

IOFlags

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

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

MsgLen

Obtains 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.

Timeout

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

Status

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

TDRV010_SUCCESS	No messages lost
TDRV010_FIFO_OVERRUN	One or more messages were overwritten in the receive queue FIFO. This problem occurs if the FIFO is too small for the application read interval.
TDRV010_MSGOBJ_OVERRUN	One or more messages were overwritten in the CAN controller message FIFO because the interrupt latency is too large. Reduce the CAN bit rate or upgrade the system speed.

EXAMPLE

```
int  fd;
int  result;
TDRV010_MSG_BUF    MsgBuf;


MsgBuf.Timeout = 1000; /* ticks */

result = ioctl(fd, TDRV010_READ, (char*)&MsgBuf);

if (result < 0) {
     /* process read error */
}
```

ERRORS

EINTR	The function was cancelled.
ENXIO	Illegal minor device number
ETIMEDOUT	The maximum allowed time to finish the read request is exhausted.
ENETDOWN	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.

Other returned error codes are system error conditions.

3.3.2 TDRV010_WRITE

NAME

TDRV010_WRITE – Write a CAN message

DESCRIPTION

This function writes a CAN message to the CAN bus. A pointer to the callers message buffer (*TDRV010_MSG_BUF*) must be passed by the argument *arg* to the device.

The *TDRV010_MSG_BUF* structure has the following layout:

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

Members

Identifier

Contains the message identifier of the CAN message to write.

IOFlags

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

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

MsgLen

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

Data[8]

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

Unused, set to 0.

EXAMPLE

```
int  fd;
int  result;
TDRV010_MSG_BUF    MsgBuf;


MsgBuf.Identifier  = 1234;
MsgBuf.Timeout     = 200;
MsgBuf.IOFlags     = TDRV010_EXTENDED | TDRV010_SELF_RECEPTION;
MsgBuf.MsgLen      = 2;
MsgBuf.Data[0]     = 0xaa;
MsgBuf.Data[1]     = 0x55;


result = ioctl(fd, TDRV010_WRITE, (char*)&MsgBuf);

if (result < 0) {
     /* process write error */
}
```

ERRORS

EINTR	The function was cancelled.
ENXIO	Illegal minor device number
ETIMEDOUT	The allowed time to finish the write request is elapsed. This occurs if the CAN bus is overloaded and the priority of the message identifier is too low, no other node is online or the controller enters the bus OFF state.
ENETDOWN	The controller is in bus OFF state and unable to transmit messages.

Other returned error codes are system error conditions.

3.3.3 TDRV010_BITTIMING

NAME

TDRV010_BITTIMING – Setup bit timing

DESCRIPTION

This function modifies the bit timing registers of the CAN controller to setup a new CAN bus transfer speed. A pointer to the callers parameter buffer (*TDRV010_TIMING*) must be passed by the argument *arg* to the device.

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

The *TDRV010_TIMING* structure has the following layout:

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

TimingValue

This parameter holds the new value for the bit timing register 0 (bit 0...7) and for the bit timing register 1 (bit 8...15). Possible transfer rates are between 20 kbit per second and 1 Mbit per second. The include file 'tdrv010.h' contains predefined transfer rate symbols (TDRV010_20KBIT ... TDRV010_1MBIT).
For other transfer rates please follow the instructions of the *SJA1000 Product Specification*, which is also part of the engineering kit TPMC310-EK and TPMC810-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.

This function will be accepted only in reset mode (bus OFF). Enter *TDRV010_BUSOFF* first, otherwise you will get an error *EACCES*.

EXAMPLE

```
int  fd;
int  result;
TDRV010_TIMING  BitTimingParam;


BitTimingParam.TimingValue   = TDRV010_100KBIT;
BitTimingParam.ThreeSamples = FALSE;

result = ioctl(fd, TDRV010_TIMING, (char*)&BitTimingParam);

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

ERRORS

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

Other returned error codes are system error conditions.

3.3.4 TDRV010_SETFILTER

NAME

TDRV010_SETFILTER – Setup acceptance filter

DESCRIPTION

This 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 receive FIFO.

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 callers parameter buffer (*TDRV010_FILTER*) must be passed by the argument *arg* to the device.

The *TDRV010_FILTER* structure has the following layout:

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

SingleFilter

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

AcceptanceCode

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

AcceptanceMask

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

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

This function will be accepted only in reset mode (bus OFF). Enter *TDRV010_BUSOFF* first, otherwise you will get an error *EACCES*.

EXAMPLE

```
int  fd;
int  result;
TDRV010_FILTER AcceptFilter;


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

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

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


result = ioctl(fd, TDRV010_SETFILTER, (char*)&AcceptFilter);

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

ERRORS

EACCES	Permission denied. The controller is currently in bus ON state. Please enter the bus OFF state first.

Other returned error codes are system error conditions.

3.3.5 TDRV010_BUSON

NAME

TDRV010_BUSON – Enter the bus ON state

DESCRIPTION

This function sets the 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 bus OFF recovery sequence and resets both 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.

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 = ioctl(fd, TDRV010_BUSON, NULL);

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

ERRORS

ENETDOWN	Unable to enter the bus ON mode.

Other returned error codes are system error conditions.

3.3.6 TDRV010_BUSOFF

NAME

TDRV010_BUSOFF – Enter the bus OFF state

DESCRIPTION

This function sets the specified CAN controller into the bus OFF state.

After execution of this control function the CAN controller is completely removed from the CAN bus and cannot communicate until the control function *TDRV010_BUSON* is executed.

EXAMPLE

```
int  fd;
int  result;


result = ioctl(fd, TDRV010_BUSOFF, NULL);

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

ERRORS

EIO	Unable to enter the bus OFF mode.

Other returned error codes are system error conditions.

3.3.7 TDRV010_FLUSH

NAME

TDRV010_FLUSH – Flush the message receive FIFO

DESCRIPTION

This function flushes the FIFO buffer of received messages.

EXAMPLE

```
int  fd;
int  result;


result = ioctl(fd, TDRV010_FLUSH, NULL);

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

ERRORS

No driver specific errors will be returned.

3.3.8 TDRV010_CANSTATUS

NAME

TDRV010_CANSTATUS – Read CAN controller status information

DESCRIPTION

This function returns the actual contents of several CAN controller registers for diagnostic purposes.

A pointer to the callers status buffer (*TDRV010_STATUS*) must be passed by the argument *arg* to the driver.

The *TDRV010_STATUS* structure has the following layout:

```
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;
} TDRV010_STATUS, *PTDRV010_STATUS;
```

ArbitrationLostCapture

Contents of the arbitration lost capture register. This register contains information about the bit position of losing arbitration.

ErrorCodeCapture

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

TxErrorCounter

Contents of the TX error counter register. This register contains the current value of the transmit error counter.

RxErrorCounter

Contents of the RX error counter register. This register contains the current value of the receive error counter.

ErrorWarningLimit

Contents of the error warning limit register.

StatusRegister

Contents of the status register.

ModeRegister

Contents of the mode register.

RxMessageCounterMax

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

EXAMPLE

```
int  fd;
int  result;
TDRV010_STATUS  CanStatus;


result = ioctl(fd, TDRV010_STATUS, (char*)&CanStatus);

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

ERRORS

No driver specific errors will be returned.

3.3.9 TDRV010_ENABLE_SELFTEST

NAME

TDRV010_ ENABLE_SELFTEST – Enable self test mode

DESCRIPTION

This 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.

This function will be accepted only in reset mode (bus OFF). Enter *TDRV010_BUSOFF* first, otherwise you will get an error *EACCES*.

EXAMPLE

```
int  fd;
int  result;


result = ioctl(fd, TDRV010_ENABLE_SELFTEST, NULL);

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

ERRORS

EACCES	Permission denied. The controller is currently in bus ON state. Please enter the bus OFF state first.

Other returned error codes are system error conditions.

3.3.10 TDRV010_DISABLE_SELFTEST

NAME

TDRV010_ DISBALE_SELFTEST – Disable self test mode

DESCRIPTION

This function disables the self test facility of the SJA1000 CAN controller, which was enabled before with the function *TDRV010_ENABLE_SELFTEST*.

The optional argument pointer can be NULL.

This function will be accepted only in reset mode (bus OFF). Enter *TDRV010_BUSOFF* first, otherwise you will get an error *EACCES*.

EXAMPLE

```
int  fd;
int  result;


result = ioctl(fd, TDRV010_DISABLE_SELFTEST, NULL);

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

ERRORS

EACCES	Permission denied. The controller is currently in bus ON state. Please enter the bus OFF state first.

Other returned error codes are system error conditions.

3.3.11 TDRV010_ENABLE_LISTENONLY

NAME

TDRV010_ENABLE_LISTENONLY – Enable listen only mode

DESCRIPTION

This 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'.

This function will be accepted only in reset mode (bus OFF). Enter *TDRV010_BUSOFF* first, otherwise you will get an error *EACCES*.

EXAMPLE

```
int  fd;
int  result;


result = ioctl(fd, TDRV010_ENABLE_LISTENONLY, NULL);

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

ERRORS

EACCES	Permission denied. The controller is currently in bus ON state. Please enter the bus OFF state first.

Other returned error codes are system error conditions.

3.3.12 TDRV010_DISABLE_LISTENONLY

NAME

TDRV010_DISABLE_LISTENONLY – Disable listen only mode

DESCRIPTION

This function disables the self test facility of the SJA1000 CAN controller, which was enabled before with the function *TDRV010_ENABLE_LISTENONLY*.

This function will be accepted only in reset mode (bus OFF). Enter *TDRV010_BUSOFF* first, otherwise you will get an error *EACCES*.

EXAMPLE

```
int  fd;
int  result;


result = ioctl(fd, TDRV010_DISABLE_LISTENONLY, NULL);

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

ERRORS

EACCES	Permission denied. The controller is currently in bus ON state. Please enter the bus OFF state first.

Other returned error codes are system error conditions.

3.3.13 TDRV010_SET_LIMIT

NAME

TDRV010_SET_LIMIT – Set new error warning limit

DESCRIPTION

This 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 function will be accepted only in reset mode (bus OFF). Enter *TDRV010_BUSOFF* first, otherwise you will get an error *EACCES*.

EXAMPLE

```
int  fd;
int  result;
unsigned char ErrorLimit ;


ErrorLimit = 20;
result = ioctl(fd, TDRV010_SET_LIMIT, (char*)&ErrorLimit);

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

ERRORS

EACCES	Permission denied. The controller is currently in bus ON state. Please enter the bus OFF state first.

Other returned error codes are system error conditions.

4 Debugging and Diagnostic

If the driver will not work properly, please enable debug outputs by defining the symbols *DEBUG, DEBUG_TPMC,* and *DEBUG_PCI* in file tdrv010.c.

The debug output should appear on the console. If not, please check the symbol *KKPF_PORT* in *uparam.h*. This symbol should be configured to a valid COM port (e.g. *SKDB_COM1*).

The debug output displays the device information data for the current major device like this.

```
TDRV010 Device Driver Install
Bus = 1  Dev = 2  Func = 0
[00] = 01361498
[04] = 02800000
[08] = 02800000
[0C] = 00000000
[10] = 84000000
[14] = 00804001
[18] = 84001000
[1C] = 84002000
[20] = 00000000
[24] = 00000000
[28] = 00000000
[2C] = 000A1498
[30] = 00000000
[34] = 00000040
[38] = 00000000
[3C] = 0000010B
```

The debug output above is only an example. Debug output on other systems may be different for addresses and data in some locations.