The Embedded I/O Company

TDRV010-SW-82

Linux Device Driver

Isolated 2x CAN Bus

Version 2.1.x

User Manual

Issue 2.1.0

January 2024

TEWS Technologies GmbH
Eggerstedter Weg 14, 25421 Pinneberg
Phone: +49 (0) 4101 4058 0
info@tews.com www.tews.com

TDRV010-SW-82

Linux Device Driver

Isolated 2x CAN Bus

Supported Modules:
TPMC310
TPMC810

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

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

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

©2007-2024 by TEWS Technologies GmbH

Issue	Description	Date
1.0.0	First Issue	May 9, 2007
1.0.1	Bitrate values corrected	March 9, 2009
2.0.0	API functions documented, legacy functions removed	March 27, 2013
2.0.1	Engineering Documentation removed	February 24, 2014
2.0.2	Startup configuration of Transceiver Mode corrected (TPMC310 only) Step by Step Driver Initialization modified	July 8, 2015
2.0.3	File-List modified	November 15, 2017
2.1.0	New Address of TEWS Technologies GmbH, general Revision, Support for TPMC810-20R added	January 25, 2024

Table of Contents

1 Introduction

The TDRV010-SW-82 Linux device driver allows the operation of the TDRV010 2x CAN PMC devices conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with open(), close() and ioctl() functions.

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

Supported features:

- Transmission and reception of Standard and Extended Identifiers
- Standard bit rates from 50 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
- Creates devices with dynamically allocated or fixed major device numbers
- DEVFS and SYSFS (UDEV) support for automatic device node creation

The TDRV010-SW-82 device driver supports the modules listed below:

TPMC310	Isolated 2 x CAN Bus	(PMC, Conduction Cooled)
TPMC810	Isolated 2 x CAN Bus	(PMC)

In this document all supported modules and devices will be called TDRV010. Specials for certain devices will be advised.

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
SJA1000 CAN Controller Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV010-SW-82':

TDRV010-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
TDRV010-SW-82-2.1.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

For installation the files have to be copied to the desired target directory.

The GZIP compressed archive TDRV010-SW-82-SRC.tar.gz contains the following files and directories:

Directory path 'tdrv010':

tdrv010.c	Driver source code
tdrv010def.h	Driver include file
tdrv010.h	Driver include file for application program
sja1000.h	Driver include file (CAN Controller Spec.)
Makefile	Device driver make file
makenode	Script to create device nodes in the file system
COPYING	Copy of the GNU Public License (GPL)
api/tdrv010api.h	API include file
api/tdrv010api.c	API source file
example/tdrv010exa.c	Example application
example/Makefile	Example application make file
include/tpxxxhwdep.c	Hardware dependent library
include/tpxxxhwdep.h	Hardware dependent library header file
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
include/config.h	Driver independent library header file

In order to perform an installation, extract all files of the archive TDRV010-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TDRV010-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tdrv010.h and api/tdrv010api.h to */usr/include*

2.1 Build and install the Device Driver

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

  # make install

- Only after the first build we have to execute *depmod* to create a new dependency description for loadable kernel modules. This dependency file is later used by *modprobe* to automatically load dependent kernel modules.

  # depmod –aq

2.2 Uninstall the Device Driver

- Login as *root*
- Change to the target directory
- To remove the driver from the module directory */lib/modules/<version>/misc* enter:

  # make uninstall

- Update kernel module dependency description file

  # depmod –aq

2.3 Install the Device Driver in the running Kernel

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

  # **modprobe tdrv010drv**

- 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 dynamic 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 TDRV010 CAN Channel found. The first TDRV010 CAN Channel can be accessed with device node /dev/tdrv010_0, the second with /dev/tdrv010_1, the third with /dev/tdrv010_2 and so on.

The assignment of device nodes to physical TDRV010 modules depends on the search order of the PCI bus driver. For more details on channel assignment see # *cat /proc/tews-tdrv010*.

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 tdrv010drv –r

If your kernel has enabled a dynamic device file system like devfs or sysfs (udev), all /dev/tdrv010_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 "*tdrv010drv: 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 TDRV010 driver uses dynamic allocation of major device numbers by default. If this isn't suitable for the application it is possible to define a major number for the driver. If the kernel has enabled devfs the driver will not use the symbol TDRV010_MAJOR.

To change the major number edit the file *tdrv010def.h*, change the following symbol to an appropriate value and enter **make install** to create a new driver.

TDRV010_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TDRV010_MAJOR       122
```

Be sure that the desired major number isn't used by other drivers. Please check */proc/devices* to see which numbers are free.

2.6 Receive Queue Configuration

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

3.1 General Functions

3.1.1 tdrv010Open

NAME

tdrv010Open – opens a device.

SYNOPSIS

```
TDRV010_HANDLE tdrv010Open
(
    char        *DeviceName
)
```

DESCRIPTION

Before I/O can be performed to a device, a device descriptor must be opened by a call to this function.

PARAMETERS

DeviceName

This parameter points to a null-terminated string that specifies the name of the device. The first TDRV010 CAN channel device is named "/dev/tdrv010_0" the second channel is named "/dev/tdrv010_1" and so on.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;

/*
** open the specified device
*/
hdl = tdrv010Open(“/dev/tdrv010_0”);
if (hdl == NULL)
{
     /* handle open error */
}
```

RETURNS

A device handle, or NULL if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno.*

The error code is a standard error code set by the I/O system.

3.1.2 tdrv010Close

NAME

tdrv010Close – closes a device.

SYNOPSIS

```
TDRV010_STATUS tdrv010Close
(
    TDRV010_HANDLE        hdl
)
```

DESCRIPTION

This function closes previously opened devices.

PARAMETERS

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;

/*
** close the device
*/
result = tdrv010Close(hdl);
if (result != TDRV010_OK)
{
     /* handle close error */
}
```

RETURNS

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.3 tdrv010GetModuleInfo

NAME

tdrv010GetModuleInfo – get information of the module

SYNOPSIS

```
TDRV010_STATUS tdrv010GetModuleInfo
(
        TDRV010_HANDLE              hdl,
        unsigned int                *pModuleType,
        unsigned int                *pChannelNo,
        TDRV010_PCIINFO_BUF         *pPciInfoBuf
)
```

DESCRIPTION

This function returns information about the module, including module type, local channel number and PCI header as well as the PCI localization.

PARAMETERS

hdl

This argument specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pModuleType

This argument is a pointer to an *unsigned int* (32bit) data buffer, where the module type is returned. Possible values are:

Value	**Description**
TDRV010_MODTYPE_TPMC310	Current module is a TPMC310
TDRV010_MODTYPE_TPMC810	Current module is a TPMC810

pChannelNo

This argument is a pointer to an *unsigned int* (32bit) data buffer, where the local channel number of the device is returned. Possible values are 0 or 1.

pPciInfoBuf

This argument is a pointer to the structure TDRV010_PCIINFO_BUF that receives information of the module PCI header.

```
typedef struct
{
        unsigned short      vendorId;
        unsigned short      deviceId;
        unsigned short      subSystemId;
        unsigned short      subSystemVendorId;
        int                 pciBusNo;
        int                 pciDevNo;
        int                 pciFuncNo;
} TDRV010_PCIINFO_BUF;
```

vendorId

PCI module vendor ID.

deviceId

PCI module device ID

subSystemId

PCI module sub system ID

subSystemVendorId

PCI module sub system vendor ID

pciBusNo

Number of the PCI bus, where the module resides.

pciDevNo

PCI device number

pciFuncNo

PCI function number

EXAMPLE

```
#include “tdrv010api.h”

TDRV010_HANDLE          hdl;
TDRV010_STATUS          result;
unsigned int            moduleType;
unsigned int            channelNo;
TDRV010_PCIINFO_BUF     pciInfoBuf

/*
** get module information
*/
result = tdrv010GetModuleInfo( hdl, &moduleType, &channelNo, &pciInfoBuf );

if (result != TDRV010_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid
TDRV010_ERR_INVAL	Specified pointer is invalid.

3.1.4 tdrv010GetControllerStatus

Name

tdrv010GetControllerStatus – Get CAN controller status information

Synopsis

```
TDRV010_STATUS tdrv010GetControllerStatus
(
        TDRV010_HANDLE              hdl,
        TDRV010_STATUS_BUF          *pCANStatus
)
```

Description

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

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

pCANStatus

This parameter points to a TDRV010_STATUS_BUF buffer, which receives the CAN controller status:

```
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;
        unsigned char       PLDControl;
} TDRV010_STATUS_BUF;
```

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.

PLDControl

If it’s available this parameter retrieves the content of the PLD Control Register. For non TPMC310 modules this parameter retrieves a value greater or equal 0x80 (means invalid). On TPMC310 devices the retrieved value will describe exactly the content of PLDControlReg[5:0].

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE          hdl;
TDRV010_STATUS          result;
TDRV010_STATUS_BUF      CanStatus;

result = tdrv010GetControllerStatus( hdl, &CanStatus );
if (result != TDRV010_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.

SEE ALSO

SJA1000 Product Specification Manual

3.2 Communication Functions

3.2.1 tdrv010Read

Name

tdrv010Read – Read a CAN message

Synopsis

```
TDRV010_STATUS tdrv010Read
(
        TDRV010_HANDLE              hdl,
        int                         Timeout,
        unsigned int                *pIdentifier,
        unsigned char               *pIOFlags,
        unsigned char               *pStatus,
        int                         *pLength,
        unsigned char               *pData
)
```

Description

This function reads a CAN message from the device driver receive queue. If no data is available, the function blocks until data is received or the specified timeout has expired.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

Timeout

This parameter specifies the maximum time (in milliseconds) the function will block and wait for data if no data is available. Specify -1 to wait indefinitely, or 0 to return immediately.

pIdentifier

This parameter is a pointer to an *unsigned int* (32bit) value where the CAN message identifier is stored.

pIOFlags

This parameter is a pointer to an *unsigned char* (8bit) value where CAN message attributes are stored as a set of bit flags. The following attribute flags are possible:

Value	**Description**
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.

pStatus

This parameter is a pointer to an *unsigned char* (8bit) value where status information about overrun conditions either in the CAN controller or intermediate software FIFO is stored. The following values are possible:

Value	**Description**
TDRV010_SUCCESS	No messages lost
TDRV010_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.
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.

pLength

This parameter is a pointer to an *int* value where the length of the received CAN message (number of bytes) is stored. Possible values are 0..8.

pData

This parameter is a pointer to an *unsigned char* array where the received CAN message is stored. This buffer receives up to 8 data bytes. pData[0] receives message Data 0, pData[1] receives message Data 1 and so on.

Example

```
#include "tdrv010api.h"

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;
int                Timeout;
unsigned int       Identifier;
unsigned char      IOFlags;
unsigned char      Status;
int                Length;
unsigned char      Data[8];

/*
** Read a CAN message from the device.
** If no data is available, wait 5000ms for incoming messages.
*/
Timeout = 5000;
result = tdrv010Read(   hdl,
                        Timeout,
                        &Identifier,
                        &IOFlags,
                        &Status,
                        &Length,
                        &Data[0] );
if (result != TDRV010_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_TIMEOUT	Read was blocked and the allowed time has elapsed.
TDRV010_ERR_BUSOFF	The controller is in bus OFF state and no message is available in the receive queue. Note, as long as CAN messages are available in the receive queue FIFO, bus OFF conditions were not reported by the read function. This means you can read all CAN messages out of the receive queue FIFO during bus OFF state without an error result.

3.2.2 tdrv010Write

Name

tdrv010Write – Write a CAN message

Synopsis

```
TDRV010_STATUS tdrv010Write
(
        TDRV010_HANDLE              hdl,
        int                         Timeout,
        unsigned int                Identifier,
        unsigned char               IOFlags,
        int                         Length,
        unsigned char               *pData
)
```

Description

This function writes a CAN message to the CAN bus. The function waits for the message to be sent until the specified timeout has expired.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

Timeout

Specifies the amount of time (in milliseconds) the caller is willing to wait for execution of write request. A value of -1 means wait indefinitely. If Timeout is set to 0 the function will return immediately after initiating the write in the CAN controller.

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.

Value	Description
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.

Length

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

pData

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

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;
int                Timeout;
unsigned int       Identifier;
unsigned char      IOFlags;
int                Length;
unsigned char      Data[8];

/*
** Write an extended CAN message to the device.
*/
Identifier    = 1234;
Timeout       = 5000;
IOFlags       = TDRV010_EXTENDED | TDRV010_SINGLE_SHOT;
MsgLen        = 2;
Data[0]       = 0xaa;
Data[1]       = 0x55;

result = tdrv010Write(  hdl,
                        Timeout,
                        Identifier,
                        IOFlags,
                        Length,
                        &Data[0] );
if (result != TDRV010_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_TIMEOUT	The allowed time to finish the write request is elapsed.
TDRV010_ERR_BUSOFF	The controller is in bus OFF state and unable to transmit messages.
TDRV010_ERR_INVAL	Illegal message length (valid range is 0...8).

3.3 Configuration Functions

3.3.1 tdrv010SetFilter

Name

tdrv010SetFilter – Configure Acceptance Filter

Synopsis

```
TDRV010_STATUS tdrv010SetFilter
(
        TDRV010_HANDLE          hdl,
        int                     SingleFilter,
        unsigned int            AcceptanceCode,
        unsigned int            AcceptanceMask
)
```

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 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 (BUSOFF). Use function tdrv010Stop() first.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

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.

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;
int                SingleFilter;
unsigned int       AcceptanceCode;
unsigned int       AcceptanceMask;

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

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

/* Single Filter Mode */
SingleFilter = 1;

result = tdrv010SetFilter(   hdl,
                             SingleFilter,
                             AcceptanceCode,
                             AcceptanceMask );
if (result != TDRV010_OK)
{
    /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_ACCESS	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the acceptance filter.

SEE ALSO

tdrv010exa.c for a programming example.

SJA1000 Product Specification Manual – *6.4.15 ACCEPTANCE FILTER*

3.3.2 tdrv010SetBitTiming

Name

tdrv010SetBitTiming – Modify CAN Bus transfer speed

Synopsis

```
TDRV010_STATUS tdrv010SetBitTiming
(
        TDRV010_HANDLE              hdl,
        unsigned short              TimingValue,
        int                         UseThreeSamples
)
```

Description

This function modifies the bit timing registers of the CAN controller to setup a new CAN bus transfer speed.

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 (BUSOFF). Use function tdrv010Stop() first.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

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 50 Kbit per second and 1 Mbit per second. The include file 'tdrv010api.h' contains predefined transfer rate symbols (TDRV010_50KBIT ... TDRV010_1MBIT).
For other transfer rates please follow the instructions of the *SJA1000 Product Specification*, which is also part of the TPMC310 or TPMC810 engineering documentation.

UseThreeSamples

If this parameter is TRUE (1) the CAN bus is sampled three times per bit time instead of one.

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE      hdl;
TDRV010_STATUS      result;
int                 UseThreeSamples;
unsigned short      TimingValue;

TimingValue         = TDRV010_100KBIT;
UseThreeSamples     = FALSE;

result = tdrv010SetBitTiming(    hdl,
                                 TimingValue,
                                 UseThreeSamples );
if (result != TDRV010_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_ACCESS	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the bit timing.

SEE ALSO

tdrv010exa.c for a programming example.

tdrv010api.h for predefined bus timing constants.

SJA1000 Product Specification Manual – 6.5.1/2 BUS TIMING REGISTER.

3.3.3 tdrv010Start

Name

tdrv010Start – Set CAN controller into BUSON state

Synopsis

```
TDRV010_STATUS tdrv010Start
(
    TDRV010_HANDLE          hdl
)
```

Description

This function sets the specified CAN controller into the BUSON state.

After an abnormal rate of occurrences of errors on the CAN bus or after driver startup, the CAN controller enters the BUSOFF state. This control function resets the "reset mode" bit in the mode register. The CAN controller begins the bus OFF recovery sequence and resets 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.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;

result = tdrv010Start( hdl );
if (result != TDRV010_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_BUSOFF	Unable to enter the Bus ON mode.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.3.4 tdrv010Stop

Name

tdrv010Stop – Set CAN controller into BUSOFF state

Synopsis

```
TDRV010_STATUS tdrv010Stop
(
        TDRV010_HANDLE hdl
)
```

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 tdrv010Start() is executed.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;

result = tdrv010Stop( hdl );
if (result != TDRV010_OK)
{
    /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_IO	Unable to enter the Bus OFF mode.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.3.5 tdrv010FlushReceiveFifo

Name

tdrv010FlushReceiveFifo – Flush software receive FIFO

Synopsis

```
TDRV010_STATUS tdrv010FlushReceiveFifo
(
    TDRV010_HANDLE          hdl
)
```

Description

This function flushes the software FIFO buffer of received CAN messages.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;

result = tdrv010FlushReceiveFifo( hdl );
if (result != TDRV010_OK)
{
    /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.

3.3.6 tdrv010SelftestEnable

Name

tdrv010SelftestEnable – Enable self test facility

Synopsis

```
TDRV010_STATUS tdrv010SelftestEnable
(
        TDRV010_HANDLE                  hdl
)
```

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 (BUSOFF). Use function tdrv010Stop() first.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;

result = tdrv010SelftestEnable( hdl );
if (result != TDRV010_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_ACCESS	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state first.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.3.7 tdrv010SelftestDisable

Name

tdrv010SelftestDisable – Disable self test facility

Synopsis

```
TDRV010_STATUS tdrv010SelftestDisable
(
    TDRV010_HANDLE          hdl
)
```

Description

This function disables the self test facility of the SJA1000 CAN controller, which was enabled before with the function tdrv010SelftestEnable().

This function will be accepted only in reset mode (BUSOFF). Use function tdrv010Stop() first.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;

result = tdrv010SelftestDisable( hdl );
if (result != TDRV010_OK)
{
    /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_ACCESS	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state first.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.3.8 tdrv010ListenOnlyEnable

Name

tdrv010ListenOnlyEnable – Enable listen-only facility

Synopsis

```
TDRV010_STATUS tdrv010ListenOnlyEnable
(
    TDRV010_HANDLE            hdl
)
```

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 (BUSOFF). Use function tdrv010Stop() first.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;

result = tdrv010ListenOnlyEnable( hdl );
if (result != TDRV010_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_ACCESS	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state first.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.3.9 tdrv010ListenOnlyDisable

Name

tdrv010ListenOnlyDisable – Disable listen-only facility

Synopsis

```
TDRV010_STATUS tdrv010ListenOnlyDisable
(
    TDRV010_HANDLE          hdl
)
```

Description

This function disables the self test facility of the SJA1000 CAN controller, which was enabled before with the function FIO_TDRV010_ENABLE_SELFTEST.

This function will be accepted only in reset mode (BUSOFF). Use function tdrv010Stop() first.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;

result = tdrv010ListenOnlyDisable( hdl );
if (result != TDRV010_OK)
{
    /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_ACCESS	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state first.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.3.10 tdrv010SetLimit

Name

tdrv010SetLimit – Disable listen-only facility

Synopsis

```
TDRV010_STATUS tdrv010SetLimit
(
	TDRV010_HANDLE		hdl,
	unsigned char		ErrorLimit
)
```

Description

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

This function will be accepted only in reset mode (BUSOFF). Use function tdrv010Stop() first.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

ErrorLimit

This parameter specifies the new error warning limit.

Example

```
#include "tdrv010api.h"

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;

/*
** Set Error Warning Limit to 20
*/
result = tdrv010SetLimit( hdl, 20 );
if (result != TDRV010_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_ACCESS	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state first.

SEE ALSO

tdrv010exa.c for a programming example.

SJA1000 Product Specification Manual – *6.4.10 ERROR WARNING LIMIT REGISTER (EWLR)*

3.3.11 tdrv010CanReset

Name

tdrv010CanReset – Set CAN controller into reset or operating mode

Synopsis

```
TDRV010_STATUS tdrv010CanReset
(
        TDRV010_HANDLE              hdl,
        unsigned char               CanReset
)
```

Description

This function sets the certain CAN controller in reset or operating mode. After driver startup, the CAN controllers are configured to operating mode.

This function is only available for TPMC310 devices.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

CanReset

This parameter specifies the controller operating mode.

Value	**Description**
TDRV010_CANRESET_RESET	Set the certain CAN channel into reset mode
TDRV010_CANRESET_OPERATING	Set the certain CAN channel into operating mode

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;

/*
** Set Controller into operating mode
*/
result = tdrv010CanReset( hdl, TDRV010_CANRESET_OPERATING );
if (result != TDRV010_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_NOTSUP	Function not supported by the device.

SEE ALSO

TPMC310 User Manual

3.3.12 tdrv010CanSel

Name

tdrv010CanSel – Set CAN transceiver into silent or operating mode

Synopsis

```
TDRV010_STATUS tdrv010CanSel
(
      TDRV010_HANDLE            hdl,
      unsigned char             CanSel
)
```

Description

This function sets the certain CAN transceivers into silent or operating mode. After driver startup, the CAN transceivers are configured to silent mode.

Before communication is possible, the transceivers must be set to operating mode.

This function is only available for TPMC310 devices.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

CanSel

This parameter specifies the controller operating mode.

Value	**Description**
TDRV010_CANSEL_SILENT	Set the certain CAN channel into silent mode
TDRV010_CANSEL_OPERATING	Set the certain CAN channel into operating mode

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;

/*
** Set Transceiver into operating mode
*/
result = tdrv010CanSel( hdl, TDRV010_CANSEL_OPERATING );
if (result != TDRV010_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_NOTSUP	Function not supported by the device.

SEE ALSO

TPMC310 User Manual

3.3.13 tdrv010CanInt

Name

tdrv010CanInt – Enable or disable CAN controller interrupts

Synopsis

```
TDRV010_STATUS tdrv010CanInt
(
        TDRV010_HANDLE              hdl,
        unsigned char               CanInt
)
```

Description

This function enables or disables certain CAN controller interrupts. After driver startup, the CAN controller interrupts are enabled.

This function is only available for TPMC310 devices.

Parameters

hdl

This value specifies the device handle to the hardware module retrieved by a call to the corresponding open-function.

CanInt

This parameter specifies the controller operating mode.

Value	**Description**
TDRV010_CANINT_ENABLE	Enable interrupt of a certain CAN channel
TDRV010_CANINT_DISABLE	Disable interrupt of a certain CAN channel

Example

```
#include “tdrv010api.h”

TDRV010_HANDLE     hdl;
TDRV010_STATUS     result;

/*
** Enable CAN controller interrupts
*/
result = tdrv010CanInt( hdl, TDRV010_CANINT_ENABLE );
if (result != TDRV010_OK)
{
     /* handle error */
}
```

RETURN VALUE

On success, TDRV010_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TDRV010_ERR_INVALID_HANDLE	The specified device handle is invalid.
TDRV010_ERR_NOTSUP	Function not supported by device.

SEE ALSO

TPMC310 User Manual

3.4 Step by Step Driver Initialization

The following code example illustrates all necessary steps to initialize a CAN device for communication.

```
/*
**   ( 0.) Set Transceivers to “Operating Mode”
**         (required for TPMC310 only)
*/
result = tdrv010CanSel(hdl, TDRV010_CANSEL_OPERATING);

/*
**   ( 1.) Setup CAN bus bit timing
*/
TimingValue        = TDRV010_100KBIT;
UseThreeSamples    = 0;      /* FALSE */

result = tdrv010SetBitTiming(    hdl,
                                 TimingValue,
                                 UseThreeSamples );

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

result = tdrv010SetFilter(   hdl,
                             SingleFilter,
                             AcceptanceCode,
                             AcceptanceMask );



/*
**   ( 3.) Enter Bus On State
*/
result = tdrv010Start( hdl );
```

Now you should be able to send and receive CAN messages with appropriate calls to tdrv010Write() and tdrv010Read() functions.

4 Diagnostic

If the TDRV010 does not work properly it is helpful to get some status information from the driver respective kernel. To get debug output from the driver enable the following symbols in '*tdrv010.c*' by replacing "#undef" with "#define":

```
#define DEBUG_TDRV010
#define DEBUG_TDRV010_INTR
```

The Linux */proc* file system provides information about kernel, resources, driver, devices and so on. The following screen dumps display information of a correct running TDRV010 driver (see also the proc man pages).

```
# tail –f /var/log/messages /* before modprobing the TDRV010 driver */


May  9 09:03:30 linuxsmp2 kernel: TEWS TECHNOLOGIES - TDRV010 Isolated 2x
CAN Bus -  version 2.0.x (<Release Date>)
May  9 09:03:30 linuxsmp2 kernel: TDRV010:  Probe new device
(vendor=0x1498, device=0x0136, type=310)
May  9 09:03:30 linuxsmp2 kernel: TDRV010:  Probe new device
(vendor=0x1498, device=0x032A, type=810)
/* if SYSFS + UDEV is present */
May  9 09:03:30 linuxsmp2 udev[3674]: creating device node '/dev/tdrv010_0'
May  9 09:03:30 linuxsmp2 udev[3676]: creating device node '/dev/tdrv010_1'
May  9 09:03:30 linuxsmp2 udev[3688]: creating device node '/dev/tdrv010_2'
May  9 09:03:30 linuxsmp2 udev[3689]: creating device node '/dev/tdrv010_3'
...

/* after modprobing the TDRV010 driver */

# cat /proc/tews-tdrv010 /* advanced CAN channel status information */
TEWS TECHNOLOGIES - TDRV010 Isolated 2x CAN Bus -  version 1.0.0 (2007-05-
09)
Supported modules: TPMC310, TPMC810

Registered SJA1000 CAN controller channels:
/dev/tdrv010_0 (phy: TPMC310 #0, mod:01 stat:3C rec:00 tec:00 alc:00 ecc:00
ewl:60, RxFIFO[rd:0,wr:0,pk#0:])
/dev/tdrv010_1 (phy: TPMC310 #1, mod:01 stat:3C rec:00 tec:00 alc:00 ecc:00
ewl:60, RxFIFO[rd:0,wr:0,pk#0:])
/dev/tdrv010_2 (phy: TPMC810 #0, mod:01 stat:3C rec:00 tec:00 alc:00 ecc:00
ewl:60, RxFIFO[rd:0,wr:0,pk#0:])
/dev/tdrv010_3 (phy: TPMC810 #1, mod:01 stat:3C rec:00 tec:00 alc:00 ecc:00
ewl:60, RxFIFO[rd:0,wr:0,pk#0:])

/*
     phy = carrier + #channel
     mod = mode register
```

```
        stat = status register
        rec = receive error counter
        tec = transmit error counter
        alc = arbitration lost capture
        ecc = error code capture
        ewl = actual error warning limit
        RxFIFO
             rd = FIFO read pointer
             wr = FIFO write pointer
             pk = FIFO message counter peak value
   */

   # cat /proc/pci
   .../* TPMC310 */
     Bus  2, device   8, function  0:
       Class 0280: PCI device 1498:0136 (rev 0).
         IRQ 177.
         Non-prefetchable 32 bit memory at 0xff5fe400 [0xff5fe47f].
         I/O at 0xa800 [0xa87f].
         Non-prefetchable 32 bit memory at 0xff5fe000 [0xff5fe00f].
         Non-prefetchable 32 bit memory at 0xff5fdc00 [0xff5fddff].
   .../* TPMC810 */
     Bus  2, device   9, function  0:
       Class 0280: PCI device 1498:032a (rev 0).
         IRQ 169.
         Non-prefetchable 32 bit memory at 0xff5fec00 [0xff5fec7f].
         I/O at 0xa880 [0xa8ff].
         Non-prefetchable 32 bit memory at 0xff5fe800 [0xff5fe9ff].
```

```
# cat /proc/interrupts
           CPU0       CPU1
  0:    5860733    5901379  IO-APIC-edge  timer
  1:       2099       1872  IO-APIC-edge  i8042
  2:          0          0  XT-PIC  cascade
  8:          0          1  IO-APIC-edge  rtc
  9:          2          0  IO-APIC-level  acpi
 12:      50793      50084  IO-APIC-edge  i8042
 14:     155677     148926  IO-APIC-edge  ide0
169:     712307     709746  IO-APIC-level  radeon@PCI:1:0:0, TDRV010
177:          0          2  IO-APIC-level  uhci_hcd, AMD AMD8111, TDRV010
185:      25775         31  IO-APIC-level  uhci_hcd, eth0
193:          0          1  IO-APIC-level  libata, ehci_hcd, ..., TDRV010
NMI:          0          0
LOC:   11763048   11763049
ERR:          0
MIS:          0

# cat /proc/iomem
...
  /* TPMC310 */
  ff5fdc00-ff5fddff : 0000:02:08.0
    ff5fdc00-ff5fddff : TDRV010CAN
  ff5fe000-ff5fe00f : 0000:02:08.0
    ff5fe000-ff5fe00f : TDRV010PLD
  ff5fe400-ff5fe47f : 0000:02:08.0
  /* TPMC810 */
  ff5fe800-ff5fe9ff : 0000:02:09.0
    ff5fe800-ff5fe9ff : TDRV010CAN
...
```

5 Known Issues

For some Linux installations, there might be a kernel driver claiming the TDRV010 devices. This prevents the TDRV010-SW-82 device driver from supporting the installed modules, as the supported physical devices may already be claimed by the kernel driver. In this case, the TDRV010-SW-82 device driver starts up well, but no device nodes are created for the CAN channels.

To suppress the automatic start of the kernel driver, add the following lines to the blacklist file /etc/modprobe.d/blacklist.conf:

```
# do not start the PLX_PCI kernel driver automatically
blacklist plx_pci
```

If the plx_pci device driver is required by other components of the system, it may be started after the TDRV010-SW-82 device driver using startup scripts.