The Embedded I/O Company

TEWS TECHNOLOGIES

TDRV010-SW-42

VxWorks Device Driver

Isolated 2x CAN Bus

Version 2.0.x

User Manual

Issue 2.0.2

September 2010

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

TDRV010-SW-42

VxWorks Device Driver

Isolated 2x CAN Bus

Supported Modules:
TPMC310
TPMC810

Issue	Description	Date
1.0.0	First Issue	October 12, 2005
1.0.1	Review	November 8, 2005
1.1.0	Updated CAN Controller Speed Range and General Review	May 30, 2006
1.1.1	Revision	November 1, 2006
2.0.0	Support for VxBus and API description added	November 16, 2009
2.0.1	Legacy vs. VxBus Driver modified	March 26, 2010
2.0.2	tdrv010Init() function added	September 22, 2010

Table of Contents

1 Introduction

The TDRV010-SW-42 VxWorks device driver software allows the operation of the TPMC310 and TPMC810 PMC conforming to the VxWorks I/O system specification.

The TRDV010-SW-42 release contains independent driver sources for the old legacy (pre-VxBus) and the new VxBus-enabled driver model. The VxBus-enabled driver is recommended for new developments with later VxWorks 6.x release and mandatory for VxWorks SMP systems.

Both drivers, legacy and VxBus, share the same application programming interface (API) and device-independent basic I/O interface with open(), close() and ioctl() functions. The basic I/O interface is only for backward compatibility with existing applications and should not be used for new developments.

Both drivers invoke a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

To prevent the application program for losing data, incoming messages will be stored in a message FIFO with a depth of 100 messages.

The TDRV010-SW-42 device driver supports the following features:

- Transmission and receive of Standard and Extended Identifiers
- Standard bit rates from 50 kbit up to 1.0 Mbit and user defined bit rates
- Message acceptance filtering
- Single-Shot transmission
- Listen only mode
- Message self reception
- Programmable error warning limit

The TDRV010-SW-42 supports the modules listed below:

TPMC310	Isolated 2x CAN	(PMC, Conduction Cooled, Silent Mode Options)
TPMC810	Isolated 2x CAN	(PMC)

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

TPMC310 and TPMC810 User manual
TPMC310 and TPMC810 Engineering Manual
SJA1000 CAN Controller Manual
VxWorks Device Driver Developer's Guide

2 Installation

Following files are located on the distribution media:

Directory path 'TDRV010-SW-42':

TDRV010-SW-42-2.0.2.pdf	PDF copy of this manual
TDRV010-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TDRV010-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TDRV010-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tdrv010':

tdrv010drv.c	TDRV010 device driver source
tdrv010def.h	TDRV010 driver include file
tdrv010.h	TDRV010 include file for driver and application
tdrv010api.c	TDRV010 API file
sja1000.h	CAN controller driver include file
Makefile	Driver Makefile
40tdrv010.cdf	Component description file for VxWorks development tools
tdrv010.dc	Configuration stub file for direct BSP builds
tdrv010.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tdrv010exa.c	Example application

The archive TDRV010-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tdrv010':

tdrv010drv.c	TDRV010 device driver source
tdrv010def.h	TDRV010 driver include file
tdrv010.h	TDRV010 include file for driver and application
tdrv010pci.c	TDRV010 device driver source for x86 based systems
tdrv010api.c	TDRV010 API file
tdrv010exa.c	Example application
include/tdhal.h	Hardware dependent interface functions and definitions
sja1000.h	CAN controller driver include file

2.1 Legacy vs. VxBus Driver

In later VxWorks 6.x releases, the old VxWorks 5.x legacy device driver model was replaced by VxBus-enabled device drivers. Legacy device drivers are tightly coupled with the BSP and the board hardware. The VxBus infrastructure hides all BSP and hardware differences under a well defined interface, which improves the portability and reduces the configuration effort. A further advantage is the improved performance of API calls by using the method interface and bypassing the VxWorks basic I/O interface.

VxBus-enabled device drivers are the preferred driver interface for new developments.

The checklist below will help you to make a decision which driver model is suitable and possible for your application:

Legacy Driver	VxBus Driver
▪ VxWorks 5.x releases ▪ VxWorks 6.5 and earlier releases ▪ VxWorks 6.x releases without VxBus PCI bus support	▪ VxWorks 6.6 and later releases with VxBus PCI bus ▪ SMP systems (only the VxBus driver is SMP safe!)

2.2 VxBus Driver Installation

Because Wind River doesn't provide a standard installation method for 3rd party VxBus device drivers the installation procedure needs to be done manually.

In order to perform a manual installation extract all files from the archive TDRV010-SW-42-VXBUS.zip to the typical 3rd party directory *installDir/vxworks-6.x/target/3rdparty* (whereas *installDir* must be substituted by the VxWorks installation directory).

After successful installation the TDRV010 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tdrv010.*

At this point the TDRV010 driver is not configurable and cannot be included with the kernel configuration tool in a Wind River Workbench project. To make the driver configurable the driver library for the desired processer (CPU) and build tool (TOOL) must be built in the following way:

(1) Open a VxWorks development shell (e.g. C:\WindRiver\wrenv.exe -p vxworks-6.7)

(2) Change into the driver installation directory
installDir/vxworks-6.x/target/3rdparty/tews/tdrv010

(3) Invoke the build command for the required processor and build tool
make CPU=cpuName TOOL=tool

For Windows hosts this may look like this:

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv010
C:> make CPU=PENTIUM4 TOOL=diab
```

To compile SMP-enabled libraries, the argument VXBUILD=SMP must be added to the command line

```
C:> make CPU=PENTIUM4 TOOL=diab VXBUILD=SMP
```

To integrate the TDRV010 driver with the VxWorks development tools (Workbench), the component configuration file *40tdrv010.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv010
C:> copy 40tdrv010.cdf \Windriver\vxworks-6.7\target\config\comps\vxWorks
```

In VxWorks 6.7 and newer releases the kernel configuration tool scans the CDF file automatically and updates the *CxrCat.txt* cache file to provide component parameter information for the kernel configuration tool as long as the timestamp of the copied CDF file is newer than the one of the *CxrCat.txt*. If your copy command preserves the timestamp, force to update the timestamp by a utility, such as *touch*.

In earlier VxWorks releases the CxrCat.txt file may not be updated automatically. In this case, remove or rename the original *CxrCat.txt* file and invoke the make command to force recreation of this file.

```
C:> cd \Windriver\vxworks-6.7\target\config\comps\vxWorks
C:> del CxrCat.txt
C:> make
```

After successful completion of all steps above and restart of the Wind River Workbench, the TDRV010 driver can be included in VxWorks projects by selecting the *"TEWS TDRV010 Driver"* component in the *"hardware (default) - Device Drivers"* folder with the kernel configuration tool.

2.2.1 Direct BSP Builds

In development scenarios with the direct BSP build method without using the Workbench or the vxprj command-line utility, the TDRV010 configuration stub files must be copied to the directory *installDir/vxworks-6.x/target/config/comps/src/hwif.* Afterwards the *vxbUsrCmdLine.c* file must be updated by invoking the appropriate make command.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tdrv010
C:> copy tdrv010.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tdrv010.dr \Windriver\vxworks-6.7\target\config\comps\src\hwif

C:> cd \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> make vxbUsrCmdLine.c
```

2.3 Legacy Driver Installation

2.3.1 Include device driver in Tornado IDE project

For Including the TDRV010-SW-42 device driver into a Tornado IDE project follow the steps below:

(1) Extract all files from the archive TDRV010-SW-42-LEGACY.zip to your project directory.

(2) Add the device drivers C-files to your project.
Make a right click to your project in the 'Workspace' window and use the 'Add Files ...' topic.
A file select box appears, and the driver files in the tdrv010 directory can be selected.

(3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your Tornado User's Guide.

2.3.2 Special installation for Intel x86 based targets

The TDRV010 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro **CPU_FAMILY**. If the content of this macro is equal to *I80X86* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, the required device memory spaces can't be accessed.

To solve this problem a MMU mapping entry has to be added for the required TDRV010 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tdrv010pci.c** contains the function *tdrv010PciInit().* This routine finds out all TDRV010 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*).

The right place to call the function *tdrv010PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window).

```
tdrv010PciInit();
```

Be sure that the function is called prior to MMU initialization otherwise the TDRV010 PCI spaces remains unmapped and an access fault occurs during driver initialization.

Modifying the sysLib.c file will change the sysLib.c in the BSP path. Remember this for future projects and recompilations.

2.3.3 System resource requirement

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

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 1 KB
Stack	< 1 KB	---
Semaphores	2	1

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

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

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

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

3 API Documentation

3.1 General Functions

3.1.1 tdrv010Open()

Name

tdrv010Open() – opens a device.

Synopsis

```
TDRV010_DEV tdrv010Open
(
     char *DeviceName
);
```

Description

Before I/O can be performed to a device, a file 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 CAN channel on the first TDRV010 device is named “/tdrv010/0/0”, the second channel is named “/tdrv010/0/1”. The first CAN channel on the second TDRV010 device is named “/tdrv010/1/0” and so on.

Example

```
#include “tdrv010.h”
TDRV010_DEV     pDev;

/*
** open file descriptor to device
*/
pDev = tdrv010Open( “/tdrv010/0/0” );
if (pDev == NULL)
{
     /* handle open error */
}
```

RETURNS

A device descriptor pointer, 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

```
STATUS tdrv010Close
(
     TDRV010_DEV pDev
);
```

Description

This function closes previously opened devices.

Parameters

pDev

This value specifies the file descriptor pointer to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tdrv010.h”
TDRV010_DEV   pDev;
STATUS        result;

/*
** close file descriptor to device
*/
result = tdrv010Close( pDev );
if (result == ERROR)
{
     /* handle close error */
}
```

RETURNS

OK, or ERROR 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.

3.2 Device Access Functions

3.2.1 tdrv010Read()

Name

tdrv010Read() – Read a CAN message

Synopsis

```
STATUS tdrv010Read
(
     TDRV010_DEV        pDev,
     int                Timeout,
     unsigned long      *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 block until the specified timeout has expired.

Parameters

pDev

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

Timeout

This parameter specifies the maximum time (in system ticks) the function will block and wait for data if no data is available.

pIdentifier

This parameter is a pointer to an *unsigned long* (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 as a set of bit flags are stored. 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 was 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 “tdrv010.h”

TDRV010_DEV        pDev;
STATUS             result;
int                Timeout;
unsigned long      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 for 300 ticks for incoming messages.
*/
Timeout = 300;
result = tdrv010Read(   pDev,
                        Timeout,
                        &Identifier,
                        &IOFlags,
                        &Status,
                        &Length,
                        &Data[0] );
if (result == ERROR)
{
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_TIMEOUT	Read was blocked and the allowed time has elapsed or NO_WAIT was setup.
S_tdrv010Dev_ENETDOWN	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 tdrv010ReadNoWait()

Name

tdrv010ReadNoWait() – Read a CAN message

Synopsis

```
STATUS tdrv010Read
(
     TDRV010_DEV        pDev,
     unsigned long      *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. This function returns immediately if no data is available.

Parameters

pDev

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

pIdentifier

This parameter is a pointer to an *unsigned long* (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 as a set of bit flags are stored. 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 was 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 “tdrv010.h”

TDRV010_DEV        pDev;
STATUS             result;
unsigned long      Identifier;
unsigned char      IOFlags;
unsigned char      Status;
int                Length;
unsigned char      Data[8];

/*
** Read a CAN message from the device
*/
result = tdrv010ReadNoWait(  pDev,
                             &Identifier,
                             &IOFlags,
                             &Status,
                             &Length,
                             &Data[0] );
```

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

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_TIMEOUT	No CAN message available.
S_tdrv010Dev_ENETDOWN	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.

3.2.3 tdrv010Write()

Name

tdrv010Write() – Write a CAN message

Synopsis

```
STATUS tdrv010Write
(
     TDRV010_DEV        pDev,
     int                Timeout,
     unsigned long      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

pDev

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

Timeout

Specifies the amount of time (in unit system ticks) the caller is willing to wait for execution of write request. A value of ***WAIT_FOREVER*** means wait indefinitely. If Timeout is set to ***NO_WAIT*** write 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 “tdrv010.h”

TDRV010_DEV        pDev;
STATUS             result;
int                Timeout;
unsigned long      Identifier;
unsigned char      IOFlags;
int                Length;
unsigned char      Data[8];

/*
** Write an extended CAN message to the device.
*/
Identifier    = 1234;
Timeout       = 600;
IOFlags       = TDRV010_EXTENDED | TDRV010_SINGLE_SHOT;
MsgLen        = 2;
Data[0]       = 0xaa;
Data[1]       = 0x55;

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

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_TIMEOUT	The allowed time to finish the write request is elapsed.
S_tdrv010Dev_ENETDOWN	The controller is in bus OFF state and unable to transmit messages.
S_tdrv010Drv_EINVAL	Illegal message length (valid range is 0...8).
S_tdrv010Drv_ECOMM	Send failed in single shot mode.

3.2.4 tdrv010WriteNoWait()

Name

tdrv010WriteNoWait() – Write a CAN message

Synopsis

```
STATUS tdrv010Write
(
     TDRV010_DEV        pDev,
     unsigned long      Identifier,
     unsigned char      IOFlags,
     int                Length,
     unsigned char      *pData
);
```

Description

This function writes a CAN message to the CAN bus. The function returns immediately after initiating the write action in the CAN controller.

Parameters

pDev

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

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 “tdrv010.h”

TDRV010_DEV        pDev;
STATUS             result;
unsigned long      Identifier;
unsigned char      IOFlags;
int                Length;
unsigned char      Data[8];

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

result = tdrv010WriteNoWait(     pDev,
                                 Identifier,
                                 IOFlags,
                                 Length,
                                 &Data[0] );
if (result == ERROR)
{
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_TIMEOUT	The allowed time to finish the write request is elapsed.
S_tdrv010Dev_ENETDOWN	The controller is in bus OFF state and unable to transmit messages.
S_tdrv010Drv_EINVAL	Illegal message length (valid range is 0...8).
S_tdrv010Drv_ECOMM	Send failed in single shot mode.

3.2.5 tdrv010SetFilter()

Name

tdrv010SetFilter() – Configure Acceptance Filter

Synopsis

```
STATUS tdrv010SetFilter
(
     TDRV010_DEV        pDev,
     int                SingleFilter,
     unsigned long      AcceptanceCode,
     unsigned long      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). Enter tdrv010Stop() first, otherwise you will get an error *S_tdrv010Drv_EACCES*.

Parameters

pDev

This parameter specifies the device descriptor 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 “tdrv010.h”

TDRV010_DEV        pDev;
STATUS             result;
int                SingleFilter;
unsigned long      AcceptanceCode;
unsigned long      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(   pDev,
                             SingleFilter,
                             AcceptanceCode,
                             AcceptanceMask );
if (result == ERROR)
{
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the bit timing.

All other returned error codes are system error conditions.

SEE ALSO

tdrv010exa.c for a programming example.

SJA1000 Product Specification Manual – *6.4.15 ACCEPTANCE FILTER*

3.2.6 tdrv010SetBitTiming()

Name

tdrv010SetBitTiming() – Modify CAN Bus transfer speed

Synopsis

```
STATUS tdrv010SetBitTiming
(
     TDRV010_DEV        pDev,
     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). Enter tdrv010Stop() first, otherwise you will get an error *S_tdrv010Drv_EACCES*.

Parameters

pDev

This parameter specifies the device descriptor 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 'tdrv010.h' contains predefined transfer rate symbols (TDRV010_5KBIT ... 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 “tdrv010.h”

TDRV010_DEV         pDev;
STATUS              result;
int                 UseThreeSamples;
unsigned short      TimingValue;

TimingValue         = TDRV010_100KBIT;
UseThreeSamples     = FALSE;

result = tdrv010SetBitTiming(    pDev,
                                 TimingValue,
                                 UseThreeSamples );
if (result == ERROR)
{
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the bit timing.

All other returned error codes are system error conditions.

SEE ALSO

tdrv010exa.c for a programming example.

tdrv010.h for predefined bus timing constants.

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

3.2.7 tdrv010Start()

Name

tdrv010Start() – Set CAN controller into BUSON state

Synopsis

```
STATUS tdrv010Start
(
     TDRV010_DEV    pDev
);
```

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

pDev

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

Example

```
#include “tdrv010.h”

TDRV010_DEV    pDev;
STATUS         result;

result = tdrv010Start( pDev );
if (result == ERROR)
{
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_ENETDOWN	Unable to enter the Bus ON mode.

All other returned error codes are system error conditions.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.2.8 tdrv010Stop()

Name

tdrv010Stop() – Set CAN controller into BUSOFF state

Synopsis

```
STATUS tdrv010Stop
(
     TDRV010_DEV    pDev
);
```

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

pDev

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

Example

```
#include “tdrv010.h”

TDRV010_DEV    pDev;
STATUS         result;

result = tdrv010Stop( pDev );
if (result == ERROR)
{
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_ENETDOWN	Unable to enter the Bus ON mode.

All other returned error codes are system error conditions.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.2.9 tdrv010FlushReceiveFifo()

Name

tdrv010FlushReceiveFifo() – Flush software receive FIFO

Synopsis

```
STATUS tdrv010FlushReceiveFifo
(
     TDRV010_DEV    pDev
);
```

Description

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

Parameters

pDev

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

Example

```
#include “tdrv010.h”

TDRV010_DEV    pDev;
STATUS         result;

result = tdrv010FlushReceiveFifo( pDev );
if (result == ERROR)
{
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

3.2.10 tdrv010GetControllerStatus()

Name

tdrv010GetControllerStatus() – Get CAN controller status information

Synopsis

```
STATUS tdrv010GetControllerStatus
(
    TDRV010_DEV         pDev,
    TDRV010_STATUS      *pCANStatus
);
```

Description

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

Parameters

pDev

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

pCANStatus

This parameter points to a TDRV010_STATUS 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, *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 TX 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 "tdrv010.h"

TDRV010_DEV        pDev;
STATUS             result;
TDRV010_STATUS     CanStatus;

result = tdrv010GetControllerStatus( pDev, &CanStatus );
if (result == ERROR)
{
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

SEE ALSO

SJA1000 Product Specification Manual

3.2.11 tdrv010SelftestEnable()

Name

tdrv010SelftestEnable() – Enable self test facility

Synopsis

```
STATUS tdrv010SelftestEnable
(
     TDRV010_DEV    pDev
);
```

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). Enter tdrv010Stop() first, otherwise you will get an error (S_tdrv010Drv_EACCES).

Parameters

pDev

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

Example

```
#include “tdrv010.h”

TDRV010_DEV   pDev;
STATUS        result;

result = tdrv010SelftestEnable( pDev );
if (result == ERROR)
{
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state first.

All other returned error codes are system error conditions.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.2.12 tdrv010SelftestDisable()

Name

tdrv010SelftestDisable() – Disable self test facility

Synopsis

```
STATUS tdrv010SelftestDisable
(
     TDRV010_DEV    pDev
);
```

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). Enter tdrv010Stop() first, otherwise you will get an error (S_tdrv010Drv_EACCES).

Parameters

pDev

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

Example

```
#include “tdrv010.h”

TDRV010_DEV    pDev;
STATUS         result;

result = tdrv010SelftestDisable( pDev );
if (result == ERROR)
{
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state first.

All other returned error codes are system error conditions.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.2.13 tdrv010ListenOnlyEnable()

Name

tdrv010ListenOnlyEnable() – Enable listen-only facility

Synopsis

```
STATUS tdrv010ListenOnlyEnable
(
     TDRV010_DEV    pDev
);
```

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). Enter tdrv010Stop() first, otherwise you will get an error (S_tdrv010Drv_EACCES).

Parameters

pDev

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

Example

```
#include “tdrv010.h”

TDRV010_DEV    pDev;
STATUS         result;

result = tdrv010ListenOnlyEnable( pDev );
if (result == ERROR)
{
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state first.

All other returned error codes are system error conditions.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.2.14 tdrv010ListenOnlyDisable()

Name

tdrv010ListenOnlyDisable() – Disable listen-only facility

Synopsis

```
STATUS tdrv010ListenOnlyDisable
(
     TDRV010_DEV    pDev
);
```

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). Enter tdrv010Stop() first, otherwise you will get an error (S_tdrv010Drv_EACCES).

Parameters

pDev

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

Example

```
#include “tdrv010.h”

TDRV010_DEV    pDev;
STATUS         result;

result = tdrv010ListenOnlyDisable( pDev );
if (result == ERROR)
{
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state first.

All other returned error codes are system error conditions.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.2.15 tdrv010Setlimit()

Name

tdrv010SetLimit() – Disable listen-only facility

Synopsis

```
STATUS tdrv010SetLimit
(
     TDRV010_DEV   pDev,
     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). Enter tdrv010Stop() first, otherwise you will get an error (S_tdrv010Drv_EACCES).

Parameters

pDev

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

ErrorLimit

This parameter specifies the new error warning limit.

Example

```
#include “tdrv010.h”

TDRV010_DEV   pDev;
STATUS        result;

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

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state first.

All other returned error codes are system error conditions.

SEE ALSO

tdrv010exa.c for a programming example.

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

3.2.16 tdrv010CanReset()

Name

tdrv010CanReset() – Set CAN controller into reset or operating mode

Synopsis

```
STATUS tdrv010CanReset
(
     TDRV010_DEV   pDev,
     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

pDev

This parameter specifies the device descriptor 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 “tdrv010.h”

TDRV010_DEV   pDev;
STATUS        result;

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

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_ICMD	Function not supported by the device.

All other returned error codes are system error conditions.

SEE ALSO

TPMC310 User Manual

3.2.17 tdrv010CanSel()

Name

tdrv010CanSel() – Set CAN transceiver into silent or operating mode

Synopsis

```
STATUS tdrv010CanSel
(
     TDRV010_DEV   pDev,
     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 operating mode.

This function is only available for TPMC310 devices.

Parameters

pDev

This parameter specifies the device descriptor 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 “tdrv010.h”

TDRV010_DEV   pDev;
STATUS        result;

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

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_ICMD	Function not supported by the device.

All other returned error codes are system error conditions.

SEE ALSO

TPMC310 User Manual

3.2.18 tdrv010CanInt()

Name

tdrv010CanInt() – Enable or disable CAN controller interrupts

Synopsis

```
STATUS tdrv010CanInt
(
     TDRV010_DEV   pDev,
     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

pDev

This parameter specifies the device descriptor 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 “tdrv010.h”

TDRV010_DEV   pDev;
STATUS        result;

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

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_ICMD	Function not supported by device.

All other returned error codes are system error conditions.

SEE ALSO

TPMC310 User Manual

4 Legacy I/O system functions

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

The legacy I/O system functions are only relevant for the legacy TDRV010 driver. For the VxBus-enabled TDRV010 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected CAN channels.

4.1 tdrv010Drv()

NAME

tdrv010Drv() - installs the TDRV010 driver in the I/O system

SYNOPSIS

#include “tdrv010.h”

STATUS tdrv010Drv(void)

DESCRIPTION

This function searches and initializes TDRV010 supported devices on the PCI bus and installs the TDRV010 driver in the I/O system.

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

EXAMPLE

```
#include "tdrv010.h”

/*--------------------
  Initialize Driver
  --------------------*/
status = tdrv010Drv();
if (status == ERROR)
{
     /* Error handling */
}
```

RETURNS

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

ERROR CODES

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

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

Error code	Description
S_tdrv010Drv_NOMEM	Unable to allocate memory for device control block
S_tdrv010Drv_EIO	Unable to enter the CAN controllers reset mode, controller seems to be faulty
S_tdrv010Drv_NXIO	Found no TDRV010 supported devices found on the PCI bus

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.2 tdrv010DevCreate()

NAME

tdrv010DevCreate() – Add a TDRV010 device to the VxWorks system

SYNOPSIS

#include “tdrv010.h”

```
STATUS tdrv010DevCreate
(
    char        *name,
    int         devIdx,
    int         funcType,
    void        *pParam
)
```

DESCRIPTION

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

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

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls. The first CAN channel on the first TDRV010 device should be named "/tdrv010/0/0", the second channel should be named "/tdrv010/0/1". The first CAN channel on the second TDRV010 device should be named "/tdrv010/1/0" and so on.

devIdx

This index number specifies the device to add to the system.
The index number depends on the search priority of the modules. The modules will be searched in the following order:

- TPMC310-10 (CAN1, CAN2)
- TPMC810-10 (CAN1, CAN2)

If modules of the same type are installed the channel numbers will be advised in the order the VxWorks *pciFindDevice()* function will find the devices.

Example: A system with one TPMC310-10 and one TPMC810-10 installed will assign the following device indexes:

Module	Device Index
TPMC310-10 (CAN 1)	0
TPMC310-10 (CAN 2)	1
TPMC810-10 (CAN 1)	2
TPMC810-10 (CAN 2)	3

funcType

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

pParam

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

EXAMPLE

```
#include "tdrv010.h”

STATUS    result;

/*-------------------------------------------------------
  Create the device "/tdrv010/0/0" for the first CAN device
  -------------------------------------------------------*/

result = tdrv010DevCreate(   "/tdrv010/0/0",
                             0,
                             0,
                             NULL);
if (result == OK)
{
     /* Device successfully created */
}
else
{
     /* Error occurred when creating the device */
}
```

RETURNS

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

ERROR CODES

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

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

Error code	Description
S_tdrv010Drv_NODRV	The device driver was not initialized by tdrv010Drv()
S_tdrv010Drv_NXIO	Found no device matching given *devIdx*
S_tdrv010Drv_DUPDEV	Device node already created

SEE ALSO

VxWorks Programmer’s Guide: I/O System

4.3 tdrv010PciInit()

NAME

tdrv010PciInit() – Generic PCI device initialization

SYNOPSIS

void tdrv010PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC310 and TPMC810 PCI spaces (base address registers) and to enable the TDRV010 supported device for access.

The global variable *tdrv010Status* obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

Value	Meaning
> 0	Initialization successful completed. The value of tdrv010Status is equal to the number of mapped PCI spaces
0	No TDRV010 device found
< 0	Initialization failed. The value of (tdrv010Status & 0xFF) is equal to the number of mapped spaces until the error occurs. Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[]. Remedy: Add dummy entries as necessary (syslib.c).

EXAMPLE

```
extern void tdrv010PciInit();

…

tdrv010PciInit();
```

4.4 tdrv010Init()

NAME

tdrv010Init() – initialize TDRV010 driver and devices

SYNOPSIS

#include “tdrv010.h”

STATUS tdrv010Init(void)

DESCRIPTION

This function is used by the TDRV010 example application to install the driver and to add all available devices to the VxWorks system.

See also 3.1.1 tdrv010Open() for the device naming convention for legacy devices.

After calling this function it is not necessary to call tdrv010Drv() and tdrv010DevCreate() explicitly.

EXAMPLE

```
#include "tdrv010.h”

STATUS    result;


result = tdrv010Init();

if (result == ERROR)
{
     /* Error handling */
}
```

RETURNS

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

ERROR CODES

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

See 4.1 and 4.2 for a description of possible error codes.

5 Basic I/O Functions

The VxWorks basic I/O interface functions are useable with the TDRV010 legacy and VxBus-enabled driver in a uniform manner.

5.1 open()

NAME

open() - open a device or file.

SYNOPSIS

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

DESCRIPTION

Before I/O can be performed to the TDRV010 supported devices, a file descriptor must be opened by invoking the basic I/O function *open().*

PARAMETER

name

Specifies the device which shall be opened.

flags

Not used

mode

Not used

EXAMPLE

```
int  fd;

/*----------------------------------------
  Open the device named "/tdrv010/0/0" for I/O
  ----------------------------------------*/
fd = open("/tdrv010/0/0", 0, 0);
if (fd == ERROR)
{
     /* Handle error */
}
```

RETURNS

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

ERROR CODES

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

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

SEE ALSO

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

5.2 close()

NAME

close() – close a device or file

SYNOPSIS

int close (int fd)

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

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

EXAMPLE

```
int  fd;
int  retval;

/*---- close the device ----*/
retval = close(fd);
if (retval == ERROR)
{
     /* Handle error */
}
```

RETURNS

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

ERROR CODES

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

SEE ALSO

ioLib, basic I/O routine - close()

5.3 ioctl()

NAME

ioctl() - performs an I/O control function.

SYNOPSIS

#include “tdrv010.h”

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

DESCRIPTION

Special I/O operation that do not fit to the standard basic VxWorks I/O calls (read, write) will be performed by calling the ioctl() function.

PARAMETER

fd

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

request

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

Function	Description
FIO_TDRV010_READ	Read a CAN message
FIO_TDRV010_WRITE	Write a CAN message
FIO_TDRV010_BITTIMING	Set CAN bus speed
FIO_TDRV010_SETFILTER	Setup message filtering
FIO_TDRV010_BUSON	Set the device bus active
FIO_TDRV010_BUSOFF	Set the device bus passive
FIO_TDRV010_FLUSH	Flush the message queues
FIO_TDRV010_CANSTATUS	Retrieve CAN controller status register content
FIO_TDRV010_ENABLE_SELFTEST	Enable controller self test
FIO_TDRV010_DISABLE_SELFTEST	Disable controller self test
FIO_TDRV010_ENABLE_LISTENONLY	Enable listen only mode
FIO_TDRV010_DISABLE_LISTENONLY	Disable listen only mode

FIO_TDRV010_SETLIMIT	Set error warning limit
FIO_TDRV010_CAN_RESET	Set reset/operating mode (TPMC310 only)
FIO_TDRV010_CAN_SEL	Set silent/operating mode (TPMC310 only)
FIO_TDRV010_CAN_INT	Enable/disable interrupts (TPMC310 only)

arg

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

RETURNS

Function dependent value (described with the function) or ERROR if the function fails an error code will be stored in *errno*.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_ICMD	Invalid IOCTL command. FIO_TDRV010_CAN_RESET, …CAN_SEL and …CAN_INT ioctl commands are only supported on TPMC310 modules or the certain ioctl command is unknown in general.

SEE ALSO

ioLib, basic I/O routine - ioctl()

5.3.1 FIO_TDRV010_READ

The read 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.

```
typedef struct {
        unsigned long     Identifier;
        unsigned char     IOFlags;
        unsigned char     MsgLen;
        unsigned char     Data[8];
        unsigned long     Timeout;
        unsigned char     Status;
} TDRV010_MSG_BUF, *PTDRV010_MSG_BUF;
```

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:

Flag	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.

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 unit ticks) the caller is willing to wait for execution of read request. A value of ***WAIT_FOREVER*** means wait indefinitely. If Timeout is set to ***NO_WAIT*** read will return immediately with error if the device is blocked by other read requests or no message is available.

Status

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

Value	Description
TDRV010_SUCCESS	No messages lost
TDRV010_FIFO_OVERRUN	One or more messages was overwritten in the receive queue FIFO. This problem occurs if the FIFO is too small for the application read interval.
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

```
#include “tdrv010.h”

int                fd;
int                result;
TDRV010_MSG_BUF    MsgBuf;

MsgBuf.Timeout = 300;

result = ioctl(fd, FIO_TDRV010_READ, (int)&MsgBuf);

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

RETURN VALUE

Number of databytes read if function succeeds, or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_TIMEOUT	Read was blocked and the allowed time has elapsed or NO_WAIT was setup.
S_tdrv010Dev_ENETDOWN	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.

5.3.2 FIO_TDRV010_WRITE

The write 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.

```
typedef struct {
        unsigned long    Identifier;
        unsigned char    IOFlags;
        unsigned char    MsgLen;
        unsigned char    Data[8];
        unsigned long    Timeout;
        unsigned char    Status;
} TDRV010_MSG_BUF, *PTDRV010_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 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.

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 unit ticks) the caller is willing to wait for execution of write request. A value of ***WAIT_FOREVER*** means wait indefinitely. If Timeout is set to ***NO_WAIT*** write will return immediately after initiating the write in the CAN controller.

Status

Unused set to 0.

EXAMPLE

```
#include “tdrv010.h”

int                fd;
int                result;
TDRV010_MSG_BUF    MsgBuf;

MsgBuf.Identifier  = 1234;
MsgBuf.Timeout     = 600;
MsgBuf.IOFlags     = TDRV010_EXTENDED | TDRV010_SINGLE_SHOT;
MsgBuf.MsgLen      = 2;
MsgBuf.Data[0]     = 0xaa;
MsgBuf.Data[1]     = 0x55;

result = ioctl(fd, FIO_TDRV010_WRITE, (int)&MsgBuf);

if (nBytes == ERROR) {
     /* process error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_TIMEOUT	The allowed time to finish the write request is elapsed.
S_tdrv010Dev_ENETDOWN	The controller is in bus OFF state and unable to transmit messages.
S_tdrv010Drv_EINVAL	Illegal message length (valid range is 0...8).
S_tdrv010Drv_ECOMM	Send failed in single shot mode.

5.3.3 FIO_TDRV010_BITTIMING

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.

```
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 50 Kbit per second and 1 Mbit per second. The include file 'tdrv010.h' contains predefined transfer rate symbols (TDRV010_5KBIT ... 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.

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 (BUSOFF). Enter FIO_TDRV010_BUSOFF first otherwise you will get an error *S_tdrv010Drv_EACCES*.

EXAMPLE

```
#include “tdrv010.h”

int                 fd;
STATUS              retval;
TDRV010_TIMING      BitTimingParam;

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

retval = ioctl(fd, FIO_TDRV010_BITTIMING, (int)&BitTimingParam);

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

RETURNS

OK if successful or ERROR otherwise.

Error codes are provided by the global variable *errno* or delivered by the function *errnoGet().*

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the bit timing.

All other returned error codes are system error conditions.

SEE ALSO

tdrv010.h for predefined bus timing constants.

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

5.3.4 FIO_TDRV010_SETFILTER

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.

```
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 (BUSOFF). Enter FIO_TDRV010_BUSOFF first otherwise you will get an error *S_tdrv010Drv_EACCES*.

EXAMPLE

```
#include “tdrv010.h”

int                fd;
STATUS             retval;
TDRV010_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;

retval = ioctl(fd, FIO_TDRV010_SETFILTER, (int)&AcceptFilter);

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

RETURNS

OK if successful or ERROR otherwise. Error codes are provided by the global variable *errno* or delivered by the function *errnoGet().*

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the bit timing.

All other returned error codes are system error conditions.

SEE ALSO

SJA1000 Product Specification Manual – *6.4.15 ACCEPTANCE FILTER*

5.3.5 FIO_TDRV010_BUSON

This 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 bus OFF 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.

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

EXAMPLE

```
#include “tdrv010.h”

int       fd;
STATUS    retval;

retval = ioctl(fd, FIO_TDRV010_BUSON, 0);

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

RETURNS

OK if successful or ERROR otherwise.

Error codes are provided by the global variable *errno* or delivered by the function *errnoGet().*

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_ENETDOWN	Unable to enter the Bus ON mode.

All other returned error codes are system error conditions.

SEE ALSO

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

5.3.6 FIO_TDRV010_BUSOFF

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 FIO_TDRV010_BUSON is executed.

EXAMPLE

```
#include “tdrv010.h”

int       fd;
STATUS    retval;

retval = ioctl(fd, FIO_TDRV010_BUSOFF, 0);

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

RETURNS

OK if successful or ERROR otherwise.

Error codes are provided by the global variable *errno* or delivered by the function *errnoGet().*

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EIO	Unable to enter the bus OFF mode.

All other returned error codes are system error conditions.

SEE ALSO

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

5.3.7 FIO_TDRV010_FLUSH

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

EXAMPLE

```
#include “tdrv010.h”

int       fd;
STATUS    retval;


retval = ioctl(fd, FIO_TDRV010_FLUSH, 0);

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

RETURNS

OK if successful or ERROR otherwise.

5.3.8 FIO_TDRV010_CANSTATUS

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 device.

```
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, *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 TX 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 "tdrv010.h"

int                fd;
STATUS             retval;
TDRV010_STATUS     CanStatus;


retval = ioctl(fd, FIO_TDRV010_CANSTATUS, (int)&CanStatus);

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

RETURNS

OK if successful or ERROR otherwise.

SEE ALSO

SJA1000 Product Specification Manual

5.3.9 FIO_TDRV010_ENABLE_SELFTEST

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). Enter FIO_TDRV010_BUSOFF first otherwise you will get an error (S_tdrv010Drv_EACCES).

EXAMPLE

```
#include "tdrv010.h"

int       fd;
STATUS    retval;

retval = ioctl(fd, FIO_TDRV010_ENABLE_SELFTEST, 0);
if (retval == ERROR) {
     /* process error */
}
```

RETURNS

OK if successful or ERROR otherwise. Error codes are provided by the global variable *errno* or delivered by the function *errnoGet().*

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the bit timing.

All other returned error codes are system error conditions.

SEE ALSO

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

5.3.10 FIO_TDRV010_DISABLE_SELFTEST

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). Enter FIO_TDRV010_BUSOFF first otherwise you will get an error (S_tdrv010Drv_EACCES).

EXAMPLE

```
#include "tdrv010.h"

int       fd;
STATUS    retval;

retval = ioctl(fd, FIO_TDRV010_DISABLE_SELFTEST, 0);

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

RETURNS

OK if successful or ERROR otherwise.

Error codes are provided by the global variable *errno* or delivered by the function *errnoGet().*

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the bit timing.

All other returned error codes are system error conditions.

SEE ALSO

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

5.3.11 FIO_TDRV010_ENABLE_LISTENONLY

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). Enter FIO_TDRV010_BUSOFF first otherwise you will get an error (S_tdrv010Drv_EACCES).

EXAMPLE

```
#include “tdrv010.h”

int       fd;
STATUS    retval;

retval = ioctl(fd, FIO_TDRV010_ENABLE_LISTENONLY, 0);
if (retval == ERROR) {
     /* process error */
}
```

RETURNS

OK if successful or ERROR otherwise.

Error codes are provided by the global variable *errno* or delivered by the function *errnoGet().*

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the bit timing.

All other returned error codes are system error conditions.

SEE ALSO

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

5.3.12 FIO_TDRV010_DISABLE_LISTENONLY

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). Enter FIO_TDRV010_BUSOFF first otherwise you will get an error (S_tdrv010Drv_EACCES).

EXAMPLE

```
#include "tdrv010.h"

int       fd;
STATUS    retval;

retval = ioctl(fd, FIO_TDRV010_DISABLE_LISTENONLY, 0);

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

RETURNS

OK if successful or ERROR otherwise.

Error codes are provided by the global variable *errno* or delivered by the function *errnoGet().*

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the bit timing.

All other returned error codes are system error conditions.

SEE ALSO

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

5.3.13 FIO_TDRV010_SETLIMIT

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 (BUSOFF). Enter FIO_TDRV010_BUSOFF first otherwise you will get an error (S_tdrv010Drv_EACCES).

EXAMPLE

```
#include "tdrv010.h"

int           fd;
STATUS        retval;
unsigned char ErrorLimit

ErrorLimit = 20;
retval = ioctl(fd, FIO_TDRV010_SETLIMIT, (int)&ErrorLimit);
if (retval == ERROR) {
     /* process error */
}
```

RETURNS

OK if successful or ERROR otherwise. Error codes are provided by the global variable *errno* or delivered by the function *errnoGet().*

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_EACCES	Permission denied. The controller is currently in BUS ON state. Please enter the BUS OFF state before changing the bit timing.

All other returned error codes are system error conditions.

SEE ALSO

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

5.3.14 FIO_TDRV010_CAN_RESET

This I/O control function sets the certain CAN controller in reset or operating mode. The function specific control parameter **arg** specifies the new configuration. This function is only available for TPMC310 devices.

arg

0 to set the certain CAN channel in reset mode
1 to set the certain CAN channel in operating mode

EXAMPLE

```
#include “tdrv010.h”

int                fd;
unsigned long      retval;

/*----------------------------------
  Execute ioctl() function
  Set the controller in reset mode
  ----------------------------------*/
retval = ioctl(fd, FIO_TDRV010_CAN_RESET, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_ICMD	Function not supported by device.

All other returned error codes are system error conditions.

5.3.15 FIO_TDRV010_CAN_SEL

This I/O control function sets the certain CAN controller in silent or operating mode. The function specific control parameter **arg** specifies the new configuration. This function is only available for TPMC310 devices.

arg

0 to set the certain CAN channel in silent mode
1 to set the certain CAN channel in operating mode

EXAMPLE

```
#include “tdrv010.h”

int                fd;
unsigned long      retval;

/*-----------------------------------------------
  Execute ioctl() function
  Set the certain CAN controller in silent mode
  -----------------------------------------------*/
retval = ioctl(fd, FIO_TDRV010_CAN_SEL, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_ICMD	Function not supported by device.

All other returned error codes are system error conditions.

5.3.16 FIO_TDRV010_CAN_INT

This I/O control function enables or disables the certain CAN controller interrupts. The function specific control parameter **arg** specifies the new configuration. This function is only available for TPMC310 devices.

arg

0 to enable the certain CAN channel interrupt
1 to disable the certain CAN channel interrupt

EXAMPLE

```
#include “tdrv010.h”

int                fd;
unsigned long      retval;

/*-----------------------------------------------------
  Execute ioctl() function
  Disable the interrupts of the certain CAN controller
  -----------------------------------------------------*/
retval = ioctl(fd, FIO_TDRV010_CAN_INT, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

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

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

Error code	Description
S_tdrv010Drv_ICMD	Function not supported by device.

All other returned error codes are system error conditions.