VECTOR

MICROSAR CAN Transceiver driver

Technical Reference

SBC

Version 2.01.01

Authors	Robert Schelkle, Jan Hammer
Status	Released

1 Document Information

1.1 History

Author	Date	Version	Remarks
Robert Schelkle	2015-06-30	1.00.00	Creation
Robert Schelkle	2015-09-22	1.01.00	Add support for PN
Robert Schelkle	2016-01-27	2.00.00	Add section 8.3.5, Adapt section 3.2, add section 3.8
Robert Schelkle	2017-01-17	2.01.00	Adapt section 4.4.1
Jan Hammer	2020-01-07	2.01.01	Update document format

Table 1 History of the document

1.2 Reference Documents

No.	Title	Version
[1]	AUTOSAR_SWS_CAN_TransceiverDriver.pdf	3.0.0
[2]	AUTOSAR_SWS_DET.pdf	2.2.1
[3]	AUTOSAR_BasicSoftwareModules.pdf	v1.0.0
[4]	TechnicalReference_Sbc_<HardwareName>.pdf	N. A.

Table 2 Reference documents

1.3 Scope of the Document

This technical reference describes the specific use of the CAN transceiver driver on System Basis Chips (SBCs).

Please note
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module CANTRCV as specified in [1]. The CAN transceiver driver provides an abstraction layer for the used CAN transceiver hardware. It offers a hardware independent interface to the upper layer components.

Supported AUTOSAR Release:	4	
Supported Configuration Variants:	Pre-compile, Post-build selectable	
Vendor ID:	CANTRCV_30_SBC_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	CANTRCV_30_SBC_MODULE_ID	70 decimal (according to ref. **[3]**)
Instance ID:	CANTRCV_30_SBC_INSTANCE_ID	Specified as pre-compile parameter in the Generation Tool.

2.1 Supported Hardware

The CanTrcv driver acts as a peripheral handler that implements the behavior and API defined by Autosar [1]. Since the CanTrcv hardware is located on the SBC module, the CanTrcv driver uses the API of the SBC driver for hardware accesses. The SBC driver is responsible for access and synchronization aspects with the SBC hardware and peripheral components that are integrated on the chip.

So the CanTrcv driver acts as a universal handler and can be re-used for different SBCs.

3 Functional Description

3.1 Features

The features listed in this chapter cover the complete functionality specified in [1].

The supported and not supported features are presented in the following two tables.

Supported features
CAN Transceiver initialization.
CAN Transceiver control via SBC component (see 5.1).
Detection of wakeup.
Getting and setting operation mode of CAN transceiver.

Table 3 Supported SWS features

Additional supported MICROSAR features
MICROSAR Identity Manager using Post-build selectable.

Table 4 Additional supported MICROSAR features

Not supported features
CAN Transceiver control via SPI.
CAN Transceiver control via DIO.
CAN Transceiver self diagnostics.

Table 5 Not supported SWS features

The CAN transceiver driver provides service functions for initialization, operation mode change and operation mode detection of the used CAN transceiver hardware. Optional service functions and callback functions are provided to detect wakeup by bus events and report them to the upper layer components.

3.2 Initialization

After power on the CAN transceiver hardware has to be initialized. Therefore the CAN transceiver driver provides two service functions.
The function `CanTrcv_30_Sbc_InitMemory` initializes all necessary values for the transceiver driver. This function has to be called first after a power-on or reset.
The function `CanTrcv_30_Sbc_Init` initializes all CAN transceiver channels which are selected by the generation tool. Each CAN transceiver channel is switched into operating

mode: NORMAL. Note that CanTrcv_30_Sbc_InitMemory has to be called before the function CanTrcv_30_Sbc_Init is called.
If a wakeup event was pending before the initialization, the CAN transceiver driver does store this event and notifies it by calling of EcuM_SetWakeupEvent().

3.3 Set operation mode

The operation mode of the CAN transceiver hardware is changed by service function CanTrcv_30_Sbc_SetOpMode. It can be switched from normal into standby mode and vice versa. Note that this transceiver does not support the sleep mode.

If the function is called with the same operation mode as already set, no mode change occurs and E_OK is returned.

3.4 Get operation mode

To retrieve the current operation mode of a specified CAN transceiver hardware, the service function CanTrcv_30_Sbc_GetOpMode has to be called.

3.5 Get version info

The service function CanTrcv_30_Sbc_GetVersionInfo can be called to get the version info of the software module. This function must be enabled in the configuration tool by setting the checkbox **Version Info Api**.
The version of the CAN transceiver driver module can be acquired in two different ways. Calling the function CanTrcv_30_Sbc_GetVersionInfo will return the version of the module in the structure Std_VersionInfoType which additionally includes the VendorID and the ModuleID. Accessing the version defines which are specified in the header file CanTrcv_30_Sbc.h:

Autosar Revision:

```
CANTRCV_30_SBC_AR_RELEASE_MAJOR_VERSION
CANTRCV_30_SBC_AR_RELEASE_MINOR_VERSION
CANTRCV_30_SBC_AR_RELEASE_PATCH_VERSION
```

Module Version:

```
CANTRCV_30_SBC_SW_MAJOR_VERSION
CANTRCV_30_SBC_SW_MINOR_VERSION
CANTRCV_30_SBC_SW_PATCH_VERSION
```

3.6 Wakeup by bus event detection

If wakeup by bus detection is enabled in the configuration tool the callback function `CanTrcv_30_Sbc_CheckWakeup` has to be called by the lower layer in case of a wakeup. This function checks the specified CAN transceiver channel. If the transceiver hardware is in standby mode and a wakeup by bus event is detected the function returns `E_OK`, otherwise `E_NOT_OK`.

3.6.1 Get bus wakeup reason

The service function `CanTrcv_30_Sbc_GetBusWakeupReason` returns the reason which caused the wakeup.

3.6.2 Set wakeup mode

The service function `CanTrcv_30_Sbc_SetWakeupMode` sets the wakeup mode which is required by the CAN interface.

3.7 Development Error Reporting

Development errors are reported to DET using the service `Det_ReportError`, if the pre-compile parameter `CANTRCV_30_SBC_DEV_ERROR_DETECT == STD_ON`.

The reported CANTRCV instance ID has to be specified in configuration tool. For details please refer to chapter Configuration with DaVinci Configurator 5.

The reported service IDs identify the services which are described in 6.1. The following table presents the service IDs and the related services:

Service ID	Service
0x00	CanTrcv_30_Sbc_Init
0x01	CanTrcv_30_Sbc_SetOpMode
0x02	CanTrcv_30_Sbc_GetOpMode
0x03	CanTrcv_30_Sbc_GetBusWuReason
0x05	CanTrcv_30_Sbc_SetWakeupMode
0x07	CanTrcv_30_Sbc_CheckWakeup
0x04	CanTrcv_30_Sbc_GetVersionInfo
0x06	CanTrcv_30_Sbc_MainFunction
0x09	CanTrcv_30_Sbc_GetTrcvSystemData

Table 6 Mapping of service IDs to services

The errors reported to DET are described in the following table:

Error Code		Description
0x01	CANTRCV_30_SBC_E_INVALID_TRANSCEIVER	Invalid channel index is used in function argument list.
0x02	CANTRCV_30_SBC_E_PARAM_POINTER	Invalid pointer NULL_PTR is used in function argument list.
0x11	CANTRCV_30_SBC_E_UNINIT	CAN transceiver hardware is not initialized.
0x21	CANTRCV_30_SBC_E_TRCV_NOT_STANDBY	CAN transceiver hardware is not in standby mode.
0x22	CANTRCV_30_SBC_E_TRCV_NOT_NORMAL	CAN transceiver hardware is not in normal operation mode.
0x23	CANTRCV_30_SBC_E_PARAM_TRCV_WAKEUP_MODE	The requested wakeup mode is not valid.
0x24	CANTRCV_30_SBC_E_PARAM_TRCV_OPMODE	The requested opmode is not supported by the underlying transceiver hardware.
0x40	CANTRCV_30_SBC_E_NO_TRCV_CONTROL	If the CAN transceiver is not under control, which means the transceiver does remain in an invalid state, this production error is raised.

Table 7 Errors reported to DET

3.8 Function calls across partitions

The CAN transceiver driver performs hardware access by calling service functions of the lower layer component Sbc driver, see 5.1.

If the CAN transceiver is located in the same partition than the SBC driver there is no special protection requirement due to the fact that there are no partition crossing function calls. This is the default behavior of the CAN Transceiver driver.

In protected systems it may occur that the CAN transceiver driver is not located in the same partition than the SBC driver. For such scenarios the CAN transceiver driver must use special calls to get access to the corresponding SBC driver functions. Depending on the safety classification of the Sbc, the CAN transceiver driver must use either trusted or non-trusted function calls for partition crossing. Those function calls must be implemented by the application. By setting the define `CANTRCV_30_SBC_CALLBACK_FUNCTIONS` to `STD_ON` in the user configuration file the SBC driver function calls are mapped to callback functions, see Table 8, that must be implemented by the application according to the description of the Autosar OS Technical Reference.

SBC driver function call	Callback function
`Sbc_CanTrcv_SetMode(params);`	`CanTrcv_30_Sbc_Hw_SetMode(params);`
`Sbc_CanTrcv_ReadStatus(params);`	`CanTrcv_30_Sbc_Hw_ReadStatus(params);`
`Sbc_CanTrcv_ClearEvents(params);`	`CanTrcv_30_Sbc_Hw_ClearEvents(params);`
`Sbc_CanTrcv_WritePnConfig(params);`	`CanTrcv_30_Sbc_Hw_WritePnConfig(params);`

Table 8 Callback mapping of SBC function calls

In the following table the different combinations are listed:

SBC class	Trcv location	Function call
trusted	Same partition	Sbc_CanTrcv_SetMode(params); Sbc_CanTrcv_ReadStatus(params); Sbc_CanTrcv_ClearEvents(params); Sbc_CanTrcv_WritePnConfig(params);
	Different partition	CallTrustedFunction(IDX_SBC_CANTRCV_SETMODE, (TrustedFunctionParameterRefType)&params); CallTrustedFunction(IDX_SBC_CANTRCV_READSTATUS, (TrustedFunctionParameterRefType)&params); CallTrustedFunction(IDX_SBC_CANTRCV_CLEAREVENTS, (TrustedFunctionParameterRefType)&params); CallTrustedFunction(IDX_SBC_CANTRCV_WRITEPNCONFIG, (TrustedFunctionParameterRefType)&params);
non-trusted	Same partition	Sbc_CanTrcv_SetMode(params); Sbc_CanTrcv_ReadStatus(params); Sbc_CanTrcv_ClearEvents(params); Sbc_CanTrcv_WritePnConfig(params);

	Different partition	osCallNonTrustedFunction(IDX_SBC_CANTRCV_SETMODE, (NonTrustedFunctionParameterRefType)&params);
		osCallNonTrustedFunction(IDX_SBC_CANTRCV_READSTATUS, (NonTrustedFunctionParameterRefType)&params);
		osCallNonTrustedFunction(IDX_SBC_CANTRCV_CLEAREVENTS, (NonTrustedFunctionParameterRefType)&params);
		osCallNonTrustedFunction(IDX_SBC_CANTRCV_WRITEPNCONFIG, (NonTrustedFunctionParameterRefType)&params);

Table 9 Function calls across partitions

4 Integration

This chapter gives necessary information for the integration of the MICROSAR CANTRCV into an application environment of an ECU. This component is only applicable on CAN transceiver hardware mentioned in chapter 2.1.

4.1 Scope of Delivery

The delivery of the CANTRCV contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
`CanTrcv_30_Sbc.h`	Header file which has to be included by higher layers.
`CanTrcv_30_Sbc.c`	Implementation

Table 10 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool.

File Name	Description
`CanTrcv_30_Sbc_Cfg.h`	Header file contains type definitions and external data declarations. It is included by `CanTrcv_30_Sbc.h`.
`CanTrcv_30_Sbc_Cfg.c`	Contains the configuration data.
`CanTrcv_GeneralTypes.h`	Header file with all CanTrcv types specified by SWS. This file can be included by `Can_GeneralTypes.h`.

Table 11 Generated files

4.2 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions defined for the CANTRCV and illustrates their assignment among each other.

Memory Mapping Sections \ Compiler Abstraction Definitions	CANTRCV_30_SBC_VAR	CANTRCV_30_SBC_APPL_VAR	CANTRCV_30_SBC_CONST	CANTRCV_30_SBC_CODE	CANTRCV_30_SBC_APPL_CODE
CANTRCV_30_SBC_START_SEC_CODE CANTRCV_30_SBC_STOP_SEC_CODE				■	■
CANTRCV_30_SBC_START_SEC_CONST_UNSPECIFIED CANTRCV_30_SBC_STOP_SEC_CONST_UNSPECIFIED			■		
CANTRCV_30_SBC_START_SEC_VAR_NOINIT_UNSPECIFIED CANTRCV_30_SBC_STOP_SEC_VAR_NOINIT_UNSPECIFIED	■	■			

Table 12 Compiler abstraction and memory mapping

4.3 Data consistency

The CAN transceiver driver calls service functions of upper layers in order to prevent interruption when accessing the CAN transceiver pins.

These service functions have to be provided by the components VStdLib, Schedule Manager or OSEK OS depending on which of these components is used for interrupt disable / restore handling. The component for interrupt control handling has to be selected in the configuration tool.

4.4 Exclusive Areas

The transceiver driver uses the following exclusive areas:

4.4.1 CANTRCV_EXCLUSIVE_AREA_0

This section ensures consistent handling of CanTrcv hardware via the SBC driver interface.

- Protects: atomic / consistent usage of SBC driver interface.
- Used in: `CanTrcv_30_Sbc_Init()`, `CanTrcv_30_Sbc_SetOpMode()`, `CanTrcv_30_Sbc_GetOpMode()`, `CanTrcv_30_Sbc_CheckWakeup()` and `CanTrcv_30_Sbc_MainFunction()`.
- Exclude call of CanTrcv driver APIs: `CanTrcv_30_Sbc_Init()`, `CanTrcv_30_Sbc_SetOpMode()`, `CanTrcv_30_Sbc_GetOpMode()`, `CanTrcv_30_Sbc_CheckWakeup()`and `CanTrcv_30_Sbc_MainFunction()`.
- Duration: Call to Sbc module – LONG duration

4.5 Cyclic Tasks

The function CanTrcv_30_Sbc_Mainfunction must only be called periodically is if Wakeup Support is set to Polling in generation tool.

5 Dependencies to other components

5.1 Sbc driver

The CAN Transceiver driver performs hardware access by calling service functions of the lower layer component Sbc driver:

- Function `Sbc_CanTrcv_ReadStatus` is used to read the current status of the CanTrcv peripheral.
- Function `Sbc_CanTrcv_SetMode` is used to change the mode of the CanTrcv peripheral to the given mode.
- Function `Sbc_CanTrcv_ClearEvents` is used to clear pending events of the CanTrcv peripheral.
- Function `Sbc_CanTrcv_WritePnConfig` is used to write the selective wake-up configuration to the CanTrcv peripheral.

5.2 Icu driver

The CAN transceiver driver performs hardware access by calling service functions of the lower layer component Icu driver:

- Function `Icu_DisableNotification()` is used by the transceiver driver to disable the ICU notification after wakeup event notification.
- Function `Icu_EnableNotification()` is used by the transceiver driver to enable the ICU notification during the transition into the STANDBY mode.

Please note
The following chapter applies only to this use case:

- the used CAN controller does not support the feature "wakeup by CAN bus", i.e. the CAN controller can not detect incoming CAN messages and generate a so-called wakeup-interrupt
- the ECU shall wake up and start its own communication due to detected communication on the CAN bus

The proposal described in this chapter enables the user to support the use-case by using an additional µC I/O port to generate the wakeup information via the ICU driver. This I/ O port is connected either parallel to the CAN Rx port or is directly attached to the CAN transceivers ERR port (depending on the used CAN transceiver). The following steps have to be done for every CAN channel that shall be able to wake up via the CAN bus:

5.2.1 ICU configuration

Add an ICU channel

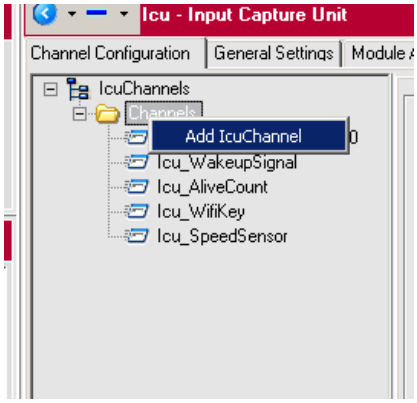

Figure 5-1 Add an ICU channel

The user has to configure the ICU channel and to add a signal notification function.

Additionally the wakeup source for the CAN channel which shall be handled via this ICU channel have to be chosen.

In dependency of the provided ICU configuration options it is possible to configure the “IcuDefaultStartEdge”. This option should be configured to ICU_FALLING_EDGE, because the wakeup event is provided by the Rx pin and/ or the ERR pin via a level change from recessive to dominant.

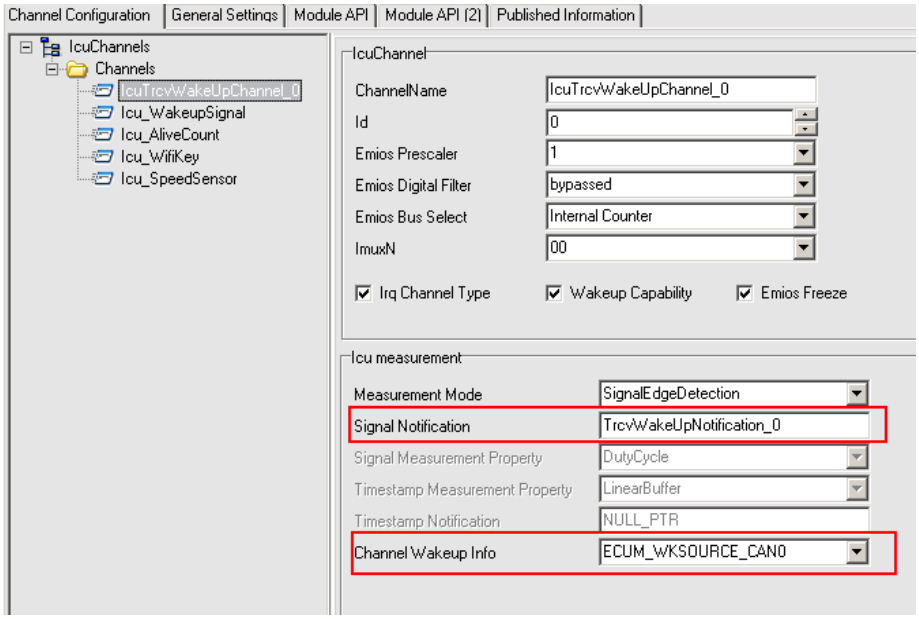

Figure 5-2 ICU channel configuration for wakeup via transceiver

If the transceiver shall also wake up the ECU and not only the CAN channel then it is necessary to activate the “Wakeup Capability” for this ICU channel.

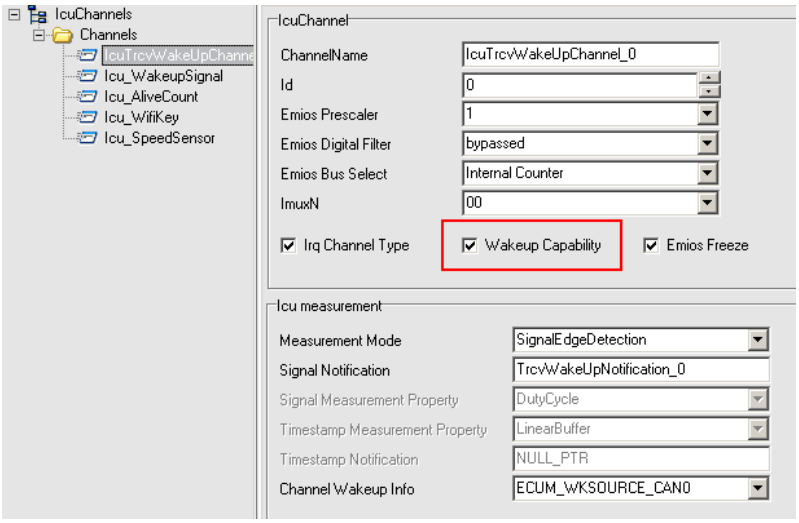

Figure 5-3 ICU wakeup capability

5.2.2 Implementation of the signal notification function

The user has to implement the signal notification function as shown in the following example:

Example

```
void Icu_TrcvWakeUpNotification_0(void)
{

  /* inform the EcuM about the wakeup event, the parameter
   * is the configured transceiver wakeup source */
  EcuM_CheckWakeup(ECUM_WKSOURCE_CAN0);
}
```

Please note

If no wakeup validation is used for the configured wakeup source, then it is possible to call `EcuM_SetWakeupEvent()` instead of `EcuM_CheckWakeup()`.

6 API Description

6.1 Services provided by CANTRCV

The CANTRCV API consists of services, which are realized by function calls.

6.1.1 CanTrcv_30_Sbc_InitMemory

Prototype	
`void` **`CanTrcv_30_Sbc_InitMemory`** `( void )`	
Parameter	
-	-
Return code	
-	-
Functional Description	
This function initializes the memory and needed values of the CAN transceiver driver.	
Particularities and Limitations	
> This function must be called before any other functionality of the CAN transceiver driver.	
Expected Caller Context	
This function must be called from task level and is not reentrant.	

Table 13 CanTrcv_30_Sbc_InitMemory

6.1.2 CanTrcv_30_Sbc_Init

Prototype	
`void` **`CanTrcv_30_Sbc_Init`** `( CanTrcv_30_Sbc_ConfigType* ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to the `CanTrcv_30_Sbc_Config` struct. If multiple configurations are available, the active configuration can be selected by using the related `CanTrcv_30_Sbc_Config_<IdentityName>_Ptr` struct
Return code	
-	-
Functional Description	
This function initializes all channels of the CAN Transceiver driver which are configured in the configuration tool.	
Particularities and Limitations	
> The function `CanTrcv_30_Sbc_InitMemory` must be called before the function `CanTrcv_30_Sbc_Init` can be called. > This function must be called before any other service functionality of the Transceiver driver.	
Expected Caller Context	
This function must be called from task level and is not reentrant.	

Table 14 CanTrcv_30_Sbc_Init

6.1.3 CanTrcv_30_Sbc_SetOpMode

Prototype	
`Std_ReturnType` **`CanTrcv_30_Sbc_SetOpMode`** `(uint8 CanTrcvIndex, CanTrcv_TrcvModeType OpMode )`	
Parameter	
`CanTrcvIndex`	Index of the selected transceiver.
`OpMode`	Pointer which contains the desired operation mode.
Return code	
`E_OK / E_NOT_OK`	`E_OK`: is returned if the transceiver state has accepted the request to change to the requested mode. `E_NOT_OK`: is returned if the transceiver state cannot be accepted or the parameter is out of the allowed range. The previous state will not change.
Functional Description	
Sets the CAN transceiver to the requested operation mode. These operation modes are: > `CANTRCV_TRCVMODE_NORMAL` > `CANTRCV_TRCVMODE_STANDBY` If return code is E_OK the notification function `CanIf_30_Sbc_TrcvModeIndication` is called if mode change has completed. Note that the notification occurs in context of `CanTrcv_30_Sbc_SetOpMode`.	
Particularities and Limitations	
> The CAN transceiver driver must be initialized. > Not each transition from one mode in any other is allowed. For these limitations please refer to chapter Set operation mode.	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 15 CanTrcv_30_Sbc_SetOpMode

6.1.4 CanTrcv_30_Sbc_GetOpMode

Prototype	
`Std_ReturnType` **`CanTrcv_30_Sbc_GetOpMode`** `( uint8 CanTrcvIndex, CanTrcv_TrcvModeType *OpMode )`	
Parameter	
`CanTrcvIndex`	Index of the selected transceiver.
`OpMode`	Pointer to a variable the current operation mode is saved in.
Return code	
`E_OK / E_NOT_OK`	`E_OK`: is returned if the operation mode was detected. `E_NOT_OK`: is returned if the operation mode was not detected.
Functional Description	
Stores the current operation mode of the selected CAN transceiver to `OpMode`. These operation modes are: > `CANTRCV_TRCVMODE_NORMAL` > `CANTRCV_TRCVMODE_STANDBY`	
Particularities and Limitations	
> The CAN transceiver driver must be initialized. > If a mode change was requested the reported operation mode may not be valid until the mode change is completed and `CanIf_30_Sbc_TrcvModeIndication` was called.	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 16 CanTrcv_30_Sbc_GetOpMode

6.1.5 CanTrcv_30_Sbc_GetBusWuReason

Prototype	
`Std_ReturnType` **`CanTrcv_30_Sbc_GetBusWuReason`** `(uint8 CanTrcvIndex, CanTrcv_TrcvWakeupReasonType *Reason )`	
Parameter	
`CanTrcvIndex`	Index of the selected transceiver.
`Reason`	Pointer to a variable to save the wakeup reason to.
Return code	
`E_OK / E_NOT_OK`	`E_OK`: is returned if the wake up reason was detected. `E_NOT_OK`: is returned if the wake up reason was not detected.
Functional Description	
Stores the last wakeup reason for the channel `CanTrcvIndex` to `Reason`. These wakeup reasons are: > `CANTRCV_WU_INTERNALLY`: The wakeup was caused by setting the CAN transceiver in operation mode via `CanTrcv_30_Sbc_SetOpMode`. > `CANTRCV_WU_ERROR`: No wakeup was detected by the transceiver and no reason is stored. The function returns `E_NOT_OK`. > `CANTRCV_WU_NOT_SUPPORTED`: No wakeup detection supported by this CAN transceiver. The function returns `E_NOT_OK`. > `CANTRCV_WU_BY_BUS`: The wakeup was caused by an external bus wakeup. > `CANTRCV_WU_RESET`: The wakeup was detected after a reset.	
Particularities and Limitations	
> The CAN transceiver driver must be initialized. > The wakeup reason represents always the last detected wakeup reason. If there was more than one wakeup detected only the last one is reported.	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 17 CanTrcv_30_Sbc_GetBusWuReason

6.1.6 CanTrcv_30_Sbc_SetWakeupMode

Prototype	
`Std_ReturnType CanTrcv_30_Sbc_SetWakeupMode (uint8 CanTrcvIndex, CanTrcv_TrcvWakeupModeType TrcvWakeupMode )`	
Parameter	
`CanTrcvIndex`	Index of the selected transceiver.
`TrcvWakeupMode`	Requested transceiver wakeup reason.
Return code	
`E_OK / E_NOT_OK`	`E_OK`: is returned, if the wakeup state has been changed to the requested mode. `E_NOT_OK`: is returned, if the wakeup state change has failed or the parameter is out of the allowed range. The previous state has not been changed.
Functional Description	
Enables and disables the reporting from wakeup events on the channel `CanTrcvIndex`. > `CANTRCV_WUMODE_ENABLE`: Report wakeup events to upper layer. > `CANTRCV_WUMODE_DISABLE`: Do not report wakeup events to upper layer. > `CANTRCV_WUMODE_CLEAR`: Clear a pending wakeup event.	
Particularities and Limitations	
> The CAN transceiver driver must be initialized. > If the wakeup handling is not enabled in configuration tool the function always returns `E_NOT_OK`. > If a wakeup was detected between DISABLE and ENABLE, it will not be reported actively by this function. Always call CanTrcv_30_Sbc_GetBusWuReason after ENABLING wakeup reporting again.	
Expected Caller Context	
This function is called from task or interrupt level and not reentrant.	

Table 18 CanTrcv_30_Sbc_SetWakeupMode

6.1.7 CanTrcv_30_Sbc_GetVersionInfo

Prototype	
`void` **`CanTrcv_30_Sbc_GetVersionInfo`** `(Std_VersionInfoType VersionInfo)`	
Parameter	
`VersionInfo`	Pointer to version information of this module.
Return code	
-	-
Functional Description	
Gets the version of the module and returns it to `VersionInfo`.	
Particularities and Limitations	
> The CAN transceiver driver must be initialized.	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 19 CanTrcv_30_Sbc_GetVersionInfo

6.1.8 CanTrcv_30_Sbc_CheckWakeup

Prototype	
`Std_ReturnType` **`CanTrcv_30_Sbc_CheckWakeup`** `(uint8 CanTrcvIndex)`	
Parameter	
`CanTrcvIndex`	Index of the selected transceiver.
Return code	
`E_OK / E_NOT_OK`	`E_OK` a wakeup-by-bus event was detected `E_NOT_OK` no wakeup was detected or an error occured
Functional Description	
This function requests the CanTrcv driver to check for wakeups and report them. If a wakeup was detected, the CanTrcv reports it by calling `EcuM_SetWakeupEvent`.	
Particularities and Limitations	
> The CAN transceiver driver must be initialized. > Do not use return value `E_OK` for wakeup detection. A wakeup can be considered as valid only if `EcuM_SetWakeupEvent` was called for the corresponding wakeup source.	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 20 CanTrcv_30_Sbc_CheckWakeup

6.1.9 CanTrcv_30_Sbc_GetTrcvSystemData

Prototype	
`Std_ReturnType CanTrcv_30_Sbc_GetTrcvSystemData (uint8 CanTrcvIndex, uint32* TrcvSysData)`	
Parameter	
`CanTrcvIndex`	Index of the selected transceiver
`TrcvSysData`	Pointer to the diagnosis data buffer to store the information in.
Return code	
`E_OK/E_NOT_OK`	`E_OK` if the diagnostic register was successfully read, `E_NOT_OK` otherwise.
Functional Description	
Reads the diagnosis registers of the underlying transceiver hardware and stores them in `TrcvSysData`. `TrcvSysData` contains the following register contents: Bit 07-00: actual mode of the CanTrcv Bit 15-08: actual states of the CanTrcv Bit 31-16: actual events of the CanTrcv See [4] for detailed information about mode, states and events of the CanTrcv on the SBC.	
Particularities and Limitations	
■ The CAN transceiver driver must be initialized. ■ SysDiagData contains only valid information if `E_OK` was returned.	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 21 CanTrcv_30_Sbc_GetTrcvSystemData

6.1.10 CanTrcv_30_Sbc_ClearTrcvWufFlag

Prototype	
`Std_ReturnType` **`CanTrcv_30_Sbc_ClearTrcvWufFlag`** `(uint8 CanTrcvIndex)`	
Parameter	
`CanTrcvIndex`	Index of the selected transceiver
Return code	
`E_OK` / `E_NOT_OK`	`E_OK` if the WUF flag was cleared, `E_NOT_OK` otherwise.
Functional Description	
This service requests the transceiver driver to clear all wakeup flags from the underlying transceiver hardware. If E_OK is returned, the driver will call the notification function `CanIf_30_Sbc_ClearTrcvWufFlagIndication` as soon as the request has been completed. The notification occurs in context of `CanTrcv_30_Sbc_ClearTrcvWufFlag`.	
Particularities and Limitations	
■ The CAN transceiver driver must be initialized. > The underlying transceiver hardware must support Selective Wakeup / Partial Networking > Due to the architecture of the hardware, this API will clear ALL wakeup flags.	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 22 CanTrcv_30_Sbc_ClearTrcvWufFlag

6.1.11 CanTrcv_30_Sbc_ReadTrcvTimeoutFlag

Prototype	
`Std_ReturnType` **`CanTrcv_30_Sbc_ReadTrcvTimeoutFlag`** `(uint8 CanTrcvIndex, CanTrcv_30_Sbc_TrcvFlagStateType* FlagState)`	
Parameter	
`CanTrcvIndex`	Index of the selected transceiver
`FlagState`	State of the timeout flag.
Return code	
`E_OK` / `E_NOT_OK`	`E_OK` if status of the timeout flag is successfully read. `E_NOT_OK` otherwise.
Functional Description	
Reads the status of the timeout flag from the underlying transceiver hardware and stores it to `FlagState`.	
Particularities and Limitations	
■ The CAN transceiver driver must be initialized.	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 23 CanTrcv_30_Sbc_ReadTrcvTimeoutFlag

6.1.12 CanTrcv_30_Sbc_ClearTrcvTimeoutFlag

Prototype	
`Std_ReturnType` **`CanTrcv_30_Sbc_ClearTrcvTimeoutFlag`** `(uint8 CanTrcvIndex)`	
Parameter	
`CanTrcvIndex`	Index of the selected transceiver
Return code	
`E_OK` / `E_NOT_OK`	`E_OK` if the timeout flag was cleared, `E_NOT_OK` otherwise.
Functional Description	
Clears the timeout flag from the underlying transceiver hardware.	
Particularities and Limitations	
■ The CAN transceiver driver must be initialized.	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 24 CanTrcv_30_Sbc_ClearTrcvTimeoutFlag

6.1.13 CanTrcv_30_Sbc_ReadTrcvSilenceFlag

Prototype	
`Std_ReturnType` **`CanTrcv_30_Sbc_ReadTrcvSilenceFlag`** `(uint8 CanTrcvIndex, CanTrcv_30_Sbc_TrcvFlagStateType* FlagState)`	
Parameter	
`CanTrcvIndex`	Index of the selected transceiver
`FlagState`	State of the silence flag.
Return code	
`E_OK` / `E_NOT_OK`	`E_OK` if status of the silence flag is successfully read. `E_NOT_OK` otherwise.
Functional Description	
Reads the status of the silence flag from the underlying transceiver hardware and stores it to `FlagState`.	
Particularities and Limitations	
■ The CAN transceiver driver must be initialized.	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 25 CanTrcv_30_Sbc_ReadTrcvSilenceFlag

6.1.14 CanTrcv_30_Sbc_CheckWakeFlag

Prototype	
`Std_ReturnType` **`CanTrcv_30_Sbc_CheckWakeFlag`** `(uint8 CanTrcvIndex)`	
Parameter	
`CanTrcvIndex`	Index of the selected transceiver
Return code	
`E_OK` / `E_NOT_OK`	`E_OK` if request for checking wakeup flag was accepted, `E_NOT_OK` otherwise.
Functional Description	
Requests the driver to check the status of the wake flag of the underlying transceiver hardware. Any detected wakeup is reported through service `EcuM_SetWakeupEvent`. The correct wakeup reason can be determined by using `CanTrcv_30_Sbc_GetBusWuReason`. If E_OK is returned, the driver calls the notification function `CanIf_30_Sbc_CheckTrcvWakeFlagIndication` as soon as the request is completed. The notification may occur in context of `CanTrcv_30_Sbc_CheckWakeFlag`.	
Particularities and Limitations	
■ The CAN transceiver driver must be initialized.	
Expected Caller Context	
This function can be called from task or interrupt level and is not reentrant.	

Table 26 CanTrcv_30_Sbc_CheckWakeFlag

6.1.15 CanTrcv_30_Sbc_MainFunction

Prototype	
`void CanTrcv_30_Sbc_MainFunction (void)`	
Parameter	
–	-
Return code	
–	–
Functional Description	
This service can be called from task level and periodically checks if a wakeup was detected by the underlying transceiver hardware.	
Particularities and Limitations	
■ The CAN transceiver driver must be initialized. ■ This service has only effect if Wakeup support is set to polling.	
Expected Caller Context	
This function can be called from task context and is not reentrant.	

Table 27 CanTrcv_30_Sbc_MainFunction

6.1.16 CanTrcv_30_Sbc_SetPNActivationState

Prototype	
`Std_ReturnType` **`CanTrcv_30_Sbc_SetPNActivationState`** `(CanTrcv_30_Sbc_PNActivationType ActivationState)`	
Parameter	
`ActivationState`	Specifies whether to enable or disable selective wakeup.
Return code	
`E_OK / E_NOT_OK`	`E_OK` will be returned if PN activation sate was successfully changed. `E_NOT_OK` will be returned if PN activation state could not be changed or an error occurred.
Functional Description	
This service changes the state of the selective wakeup feature. If called with `PN_ENABLED` CanTrcv hardware is configured to wake up on configured frames only (WUF). If called with `PN_DISABLED` CanTrcv hardware is configured to wake up on any bus activity (WUP). In order to enable selective wakeup it must be also configured as enabled in generation tool.	
Particularities and Limitations	
■ The CAN transceiver driver must be initialized. > State change to PN_DISABLED will be effective after next mode change. > Selective wakeup can be enabled only for channels that have PN activated in generation tool. > It is strongly recommended to use this API only directly after `CanTrcv_30_Sbc_Init()`. > State change to PN_ENABLE does not have any effect on the SBC hardware. Reinitialization is required to enable selective wakeup again.	
Expected Caller Context	
This function must be called from task and is not reentrant. The API must not be interrupted by any other CanTrcv API.	

Table 28 CanTrcv_30_Sbc_SetPNActivationState

6.2 Services used by CANTRCV

In the following table services provided by other components, which are used by the CANTRCV are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET	Det_ReportError
SBC	Sbc_CanTrcv_ReadStatus
SBC	Sbc_CanTrcv_SetMode
SBC	Sbc_CanTrcv_ClearEvents
SBC	Sbc_CanTrcv_WritePnConfig
CANIF	CanIf_TrcvModeIndication
ICU	Icu_EnableNotification
ICU	Icu_DisableNotification

Table 29 Services used by the CANTRCV

7 Configuration

7.1 Configuration with DaVinci Configurator 5

Refer to the integrated online help and parameter descriptions of Configurator 5.

8 AUTOSAR Standard Compliance

8.1 Additions/ Extensions

8.1.1 Memory initialization

To have an independent memory initialization for this BSW module the additional function `CanTrcv_30_Sbc_InitMemory` was added.

8.2 Limitations

8.2.1 Support of SPI and DIO

This CAN Transceiver driver does not support the connection to a CAN transceiver via SPI or DIO. The CAN Transceiver uses API of SBC driver for hardware access (see 5.1). The SBC driver is responsible to perform and synchronize the accesses to the CanTrcv component that is located on the SBC.

8.3 Deviations

While the driver is implemented according [1], some requirements could not be fulfilled in order to ensure proper functionality. The following chapter lists these deviations.

8.3.1 Notification functions

According to SWS [1] the transceiver shall call notification functions in CanIf. As the given `CanTrcvChannelId` is valid only for one transceiver driver instance, it was decided to call the following notification functions instead:

SWS API	Used API
`CanIf_TrcvModeIndication`	`CanIf_30_Sbc_TrcvModeIndication`
`CanIf_ConfirmPnAvailability`	`CanIf_30_Sbc_ConfirmPnAvailability`
`CanIf_ClearTrcvWufFlagIndication`	`CanIf_30_Sbc_ClearTrcvWufFlagIndication`
`CanIf_CheckTrcvWakeFlagIndication`	`CanIf_30_Sbc_CheckTrcvWakeFlagIndication`

Table 30 Deviation of APIs used by CanTrcv

Within these functions recalculation of the given `CanTrcvIndex` has to be performed so that the local index of the driver matches the global index of the CanIf.

Affected requirements: CanTrcv086, CanTrcv222, CanTrcv239, CanTrcv238

8.3.2 CanIf_CheckTrcvWakeFlagIndication is always called

According to SWS [1], `CanIf_CheckTrcvWakeFlagIndication` shall be called only if a wakeup was detected. As this would lock CanSm, it was decided to call this notification function in every case, even if no wakeup was detected. If a wakeup was detected, `EcuM_SetWakeupEvent` is called by the transceiver driver.

Affected requirements: CanTrcv238

8.3.3 Const removed from API CanTrcv_GetTrcvSystemData

According to SWS [1], the API CanTrcv_GetTrcvSystemData shall have a parameter TrcvSysData of type const uint32*. As this parameter has to be filled by the transceiver driver it was decided to remove this const qualifier.

Affected requirements: CanTrcv152

8.3.4 Unused BSWMD parameters

According to SWS [1], the parameters `CanTrcvSPICommRetries` and `CanTrcvSPICommTimeout` shall have the multiplicity 1. As the SWS does not describe where to use these values, it was decided to set multiplicity to 0..1 so they do not have to be used.

Affected requirements: CanTrcv172, CanTrcv171

8.3.5 Initialization of operating mode

According to SWS [1] each CAN transceiver channel shall be initialized to operating mode which is configured by parameter CanTrcvInitState. Independent of configuration each CAN transceiver channel is initialized into operating mode NORMAL.

Affected requirements: CanTrcv100, CanTrcv148.

9 Glossary and Abbreviations

9.1 Glossary

Term	Description
Cfg5	Davinci Configurator 5

Table 31 Glossary

9.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
SBC	System Basis Chip
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 32 Abbreviations

10 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com