MICROSAR LIN Driver

Technical Reference

Generic

Version 5.00.00

Authors	Jan Gaukel
Status	Released

Document Information

History

Author	Date	Version	Remarks
Jan Gaukel	2018-07-06	4.00.00	Initial version of the Technical Reference
Jan Gaukel	2019-06-28	5.00.00	Add Master / Slave support

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_LINDriver.pdf	2.2.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	4.2.0
[4]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0

Scope of the Document

This technical reference describes the general use of the LIN driver.

Caution
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

1. Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
4.00.00	Initial version for AR4.0
5.00.00	Master / Slave support

Table 1-1 Component history

2. Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module LIN as specified in [1].

Supported AUTOSAR Release*:	4	
Vendor ID:	LIN_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	LIN_MODULE_ID	82 decimal (according to ref. [4])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

2.1 Architecture Overview

The following figure shows where the LIN is located in the AUTOSAR architecture.

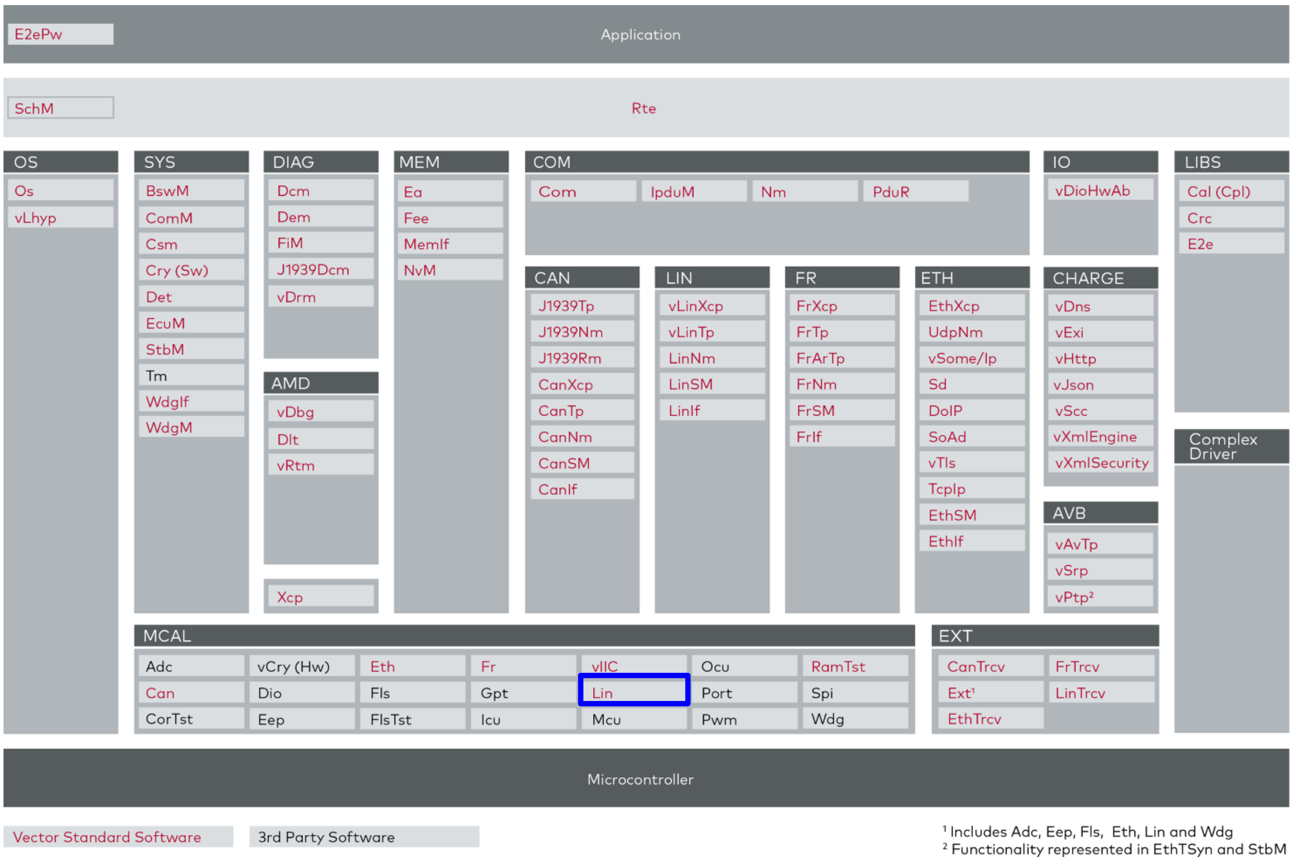

Figure 2-1 AUTOSAR 4.1 Architecture Overview

The next figure shows the interfaces to adjacent modules of the LIN. These interfaces are described in chapter 5.

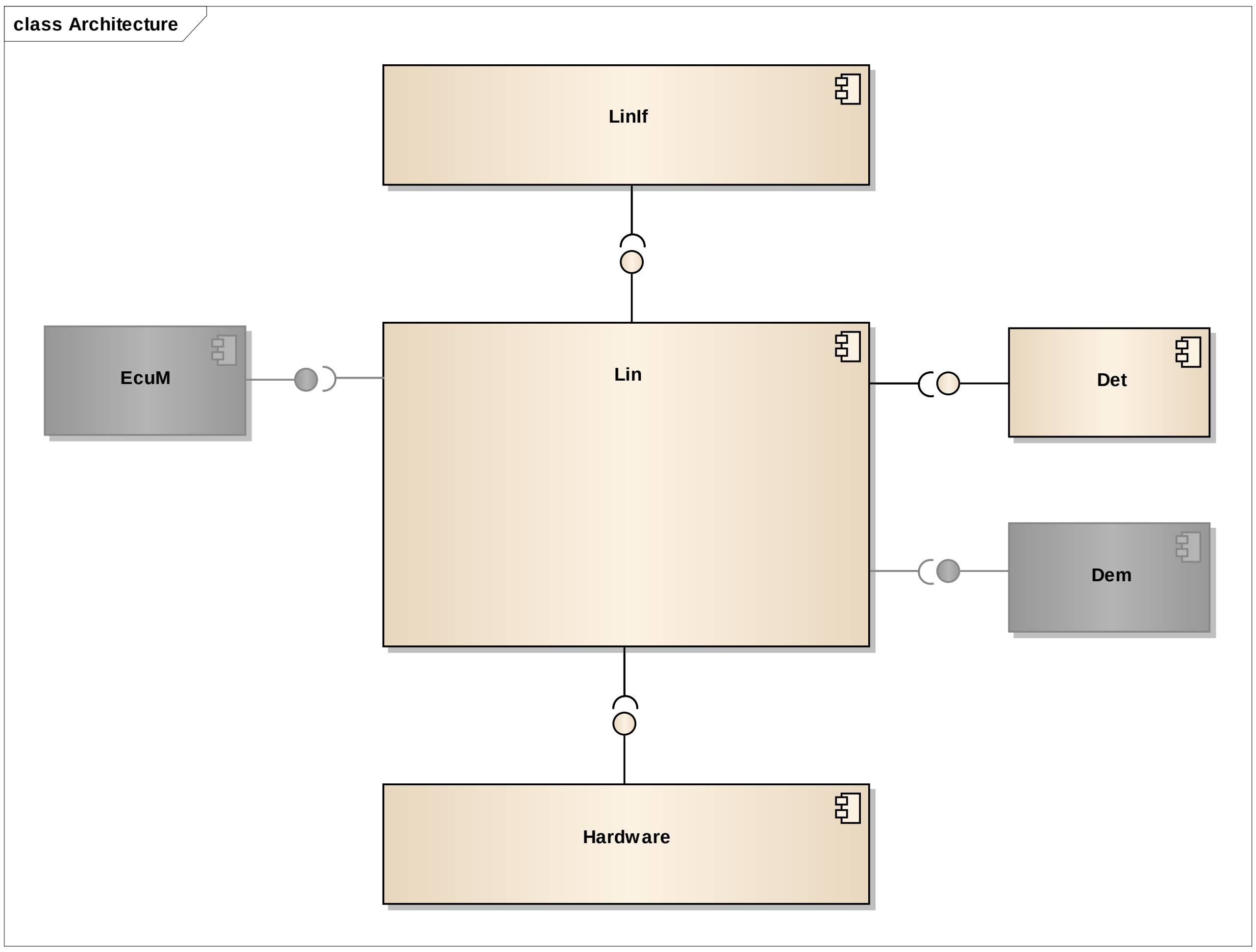

Figure 2-2 Interfaces to adjacent modules

3. Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the LIN.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 3-1 Supported AUTOSAR standard conform features

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Service to initialize itself
Service to handle a frame on channel
Service to provide the frame status
Service to set the channel to sleep and optionally transmit a go-to-sleep-command
Service to set the channel to wake and optionally generate a wake up pulse
Service to evaluate a wake up event of channel
Service to access the version information
Multi channel usage

Table 3-1 Supported AUTOSAR standard conform features

3.2 Initialization

After power on the LIN hardware has to be initialized. Therefore the LIN Driver provides two service functions.

> `Lin_Init()` has to be called to initialize the LIN driver at power on.
> `Lin_InitMemory()` is an additional service function to reinitialize the memory to bring the driver back to a pre-power-on state (not initialized). Afterwards Lin_Init() have to be called again. It is recommended to use this function before calling Lin_Init() to secure that no startup-code specific pre-initialized variables affect the driver startup behavior.

3.3 Wake up handling

Wakeup frame handling is only applicable in state sleep. A wakeup frame can be transmitted by calling the function Lin_Wakeup(). When calling Lin_WakeupInternal() no wakeup frame is transmitted. If a wakeup frame is received the EcuM is informed by calling the function EcuM_CheckWakeupEvent().

3.4 Sleep handling

Sleep mode frame handling is only applicable in state wake. A sleep mode frame can be transmitted by calling the function Lin_GoToSleep(), only applicaple for channels with node type Master. When calling Lin_GoToSleepInternal() the LIN Driver enters sleep mode without transmitting a sleep mode frame.

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `LIN_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported LIN ID is 82.

The reported service IDs identify the services which are described in chapter 5. The following table presents the service IDs and the related services:

Service ID	Service
0x00	Lin_Init
0x01	Lin_GetVersionInfo
0x04	Lin_SendFrame
0x06	Lin_GoToSleep
0x07	Lin_Wakeup
0x08	Lin_GetStatus
0x09	Lin_GoToSleepInternal
0x0A	Lin_CheckWakeup
0x0B	Lin_WakeupInternal
0x90	Lin_Interrupt

Table 3-2 Service IDs

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

Error Code		Description
0x00	LIN_E_UNINIT	API service used without module initialization.
0x02	LIN_E_INVALID_CHANNEL	API service used with an invalid or inactive channel parameter.
0x03	LIN_E_INVALID_POINTER	API service called with invalid configuration pointer
0x04	LIN_E_STATE_TRANSITION	Invalid state transition for the current state
0x05	LIN_E_PARAM_POINTER	API service called with a NULL pointer
0x06	LIN_E_PARAM_VALUE	API service called with invalid parameter value

Error Code		Description
0x10	LIN_E_TIMEOUT	Timeout caused by hardware error (currently not required and implemented)

Table 3-3 Errors reported to DET

3.5.2 Production Code Error Reporting

By default, production code related errors are not reported. If production error reporting is enabled the error is reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3].

If another module is used for production code error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Dem_ReportErrorStatus()`.

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

Error Code	Description
LIN_E_TIMEOUT	Timeout caused by hardware error Availability is hardware dependent

Table 3-4 Errors reported to DEM

Caution
In this Generic driver only the possibility for this Error Reporting is implemented into the generator. The static implementation must be done manually.

To enable the error reporting of LIN_E_TIMEOUT create a sub container ‘LinDemEventParameterRefs’ in the ‘LinGlobalConfig’ container (right click ‘LinGlobalConfig’ on DaVinci Configurator) and select a valid target reference for ‘E TIMEOUT’. For disabling delete the ‘LinDemEventParameterRefs’ container.

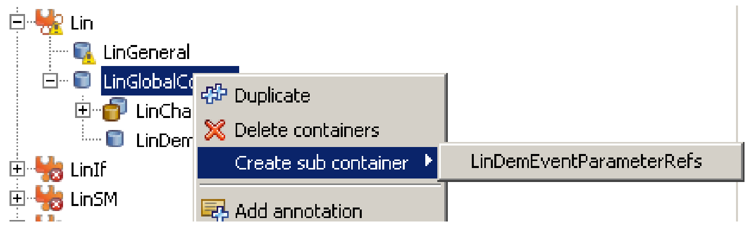

Figure 3-1 Creating LindemEventParameterRefs on DaVinci Configurator

The necessary output in the Lin_30_Generic_Cfg.h file looks like the following:

```
#define LIN_30_GENERIC_E_TIMEOUT_TYPE_DEM                       STD_ON
#define LIN_30_GENERIC_E_TIMEOUT                                65535U
```

Note
Bothe defines will be generated in every case. If the sub container exists the `_TIMEOUT_TYPE_DEM` define will be `STD_ON`. If not, it will be `STD_OFF`. The value will be the Dem value if the container exists, otherwise it will be 16.

4. Integration

This chapter gives necessary information for the integration of the MICROSAR LIN into an application environment of an ECU.

4.1 Embedded Code Files

The delivery of the LIN contains the files which are described in the chapters 4.1.1 and 4.1.2.

4.1.1 Static Files

File Name	Description	Integration Tasks
Lin_GeneralTypes.h	Header containing commonly used type definitions of the LIN cluster.	-
Lin_30_Generic.h	Header containing the interface of the LIN Driver.	-
Lin_30_Generic.c	C code file with template areas that must be extended/filled by the user.	Adapt the dedicated code areas within that file. See hints within that file.
Lin_30_Generic_Types.h	Header containing internal type definitions with template areas that must be extended/filled by the user.	Adapt the dedicated code areas within that file. See hints within that file.

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool.

File Name	Description
Lin_30_Generic_Cfg.h	Generated header adapting the LIN Driver to project requirements.
Lin_30_Generic_LCfg.c	Generated C code containing tables with link time variables and the implementation of interrupt functions.

Table 4-2 Generated files

4.2 Critical Sections

The MICROSAR LIN Driver is either running in interrupt context or is called from LinIf. The LinIf already prevents from interruption by means of exclusive areas. Thus no exclusive area handling is done within the LIN Driver.

4.3 Interruptmapping

The `Lin_Interrupt()` function requires the Hardware channel number as Parameter. Therefore, a mapping is necessary. There are two primarily used options for the interrupts distribution. The first would be, that each channel has its own interrupt or more and the second would be that all channels have the same interrupt. For each of these possibilities an example is shown below.

Example
Here is an example for the first option.

```
ISR( LinIsr_3 ){ Lin_Interrupt( 0 ); }
ISR( LinIsr_0 ){ Lin_Interrupt( 1 ); }
```

Example
Here is an example for the second option.

```
ISR( LinIsr){
  /* Check if interrupts belongs to Uart Channel 0 */
  if( (Lin_GetGlobalInterruptStatus() & 0x00000002UL) != 0 )
  { Lin_Interrupt( 0UL ); }
  /* Check if interrupts belongs to Uart Channel 1 */
  if( (Lin_GetGlobalInterruptStatus() & 0x00000008UL) != 0 )
  { Lin_Interrupt( 1UL ); }
}
```

5. API Description

For an interfaces overview please see Figure 2-2.

5.1 Type Definitions

The types defined by the LIN are described in this chapter.

Type Name	C-Type	Description	Value Range
Lin_u8PtrType	uint8	Pointer to a uint8 variable. Use for 'uint8**' definition in Lin_GetStatus	not applicable
Lin_StatusType	enum	LIN operation states for a LIN channel or frame, as returned by the API service Lin_GetStatus().	`LIN_NOT_OK` LIN frame operation return value. Development or production error occurred.
			`LIN_TX_OK` LIN frame operation return value. Successful transmission.
			`LIN_TX_BUSY` LIN frame operation return value. Ongoing transmission (Header or Response).
			`LIN_TX_HEADER_ERROR` LIN frame operation return value. Erroneous header transmission such as: > Mismatch between sent and read back data > Identifier parity error or > Physical bus error
			`LIN_TX_ERROR` LIN frame operation return value. Erroneous response transmission such as: > Mismatch between sent and read back data > Physical bus error
			`LIN_RX_BUSY` LIN frame operation return value. Ongoing reception: at least one response byte has been received, but the checksum byte has not been received.

Type Name	C-Type	Description	Value Range
			`LIN_RX_OK` LIN frame operation return value. Reception of correct response.
			`LIN_RX_ERROR` LIN frame operation return value. Erroneous response reception such as: > Framing error > Overrun error > Checksum error or > Short response
			`LIN_RX_NO_RESPONSE` LIN frame operation return value. No response byte has been received so far.
			`LIN_OPERATIONAL` LIN channel state return value. Normal operation; the related LIN channel is ready to transmit next header. No data from previous frame available (e.g. after initialization).
			`LIN_CH_SLEEP` LIN channel state return value. Sleep state operation; in this state wake-up detection from slave nodes is enabled.
Lin_SlaveErrorType	enum	Lin Slave channel error indication values for the reporting to the Lin interface.	`LIN_ERR_HEADER` LIN Slave channel error return value. An error in the header occurred.
			`LIN_ERR_RESP_STOPBIT` LIN Slave channel error return value. A framing error in the response occurred.
			`LIN_ERR_RESP_CHKSUM` LIN Slave channel error return value. A checksum error in the response occurred.
			`LIN_ERR_RESP_DATABIT` LIN Slave channel error return value. A monitoring error in the response occurred.

Type Name	C-Type	Description	Value Range
			`LIN_ERR_NO_RESP` LIN Slave channel error return value. No response was received.
			`LIN_ERR_INC_RESP` LIN Slave channel error return value. An incomplete response was received.
Lin_ConfigType	struct	Global Configuration	A structure type is present for data in each configuration class.

Table 5-1 Type definitions

Lin_PduType

This Type is used to provide PID, checksum model, data length and SDU pointer of a LIN frame from the LIN Interface to the LIN driver.

Struct Element Name	C-Type	Description	Value Range
Pid	uint8 (Lin_FramePidType)	Valid protected identifier.	Represents all valid protected identifier used by Lin_SendFrame().
Cs	enum (Lin_FrameCsModelType)	Specified Checksum model	`LIN_ENHANCED_CS` Enhanced checksum model.
			`LIN_CLASSIC_CS` Classic checksum model.
Drc	enum (Lin_FrameResponseType)	Type of response part	`LIN_FRAMERESPONSE_TX` Response is generated from this node.
			`LIN_FRAMERESPONSE_RX` Response is generated by an other node.
			`LIN_FRAMERESPONSE_IGNORE` Response is ignored by this node.
Dl	uint8 (Lin_FrameDlType)	Number of SDU data bytes to copy	This type is used to specify the number of SDU data bytes to copy. Range: 1 - 8, data length of a LIN frame.
SduPtr	uint8*	Pointer to SDU data bytes	Valid pointer to SDU data.

Table 5-2 Lin_PduType

5.2 Services provided by LIN

5.2.1 Lin_InitMemory

Prototype	
`void Lin_InitMemory (void)`	
Parameter	
-	
Return code	
void	-
Functional Description	
Sets the module state to uninitialized.	
Particularities and Limitations	
Function must be called in case LIN_VAR_ZERO_INIT variables are not initialized with 0 after reset (i.e. by startup code). This service function has to be called before Lin_Init() function.	
Call Context	
Called by upper layer.	

Table 5-3 Lin_InitMemory

5.2.2 Lin_Init

Prototype	
`void Lin_Init (const Lin_ConfigType *Config)`	
Parameter	
Config	Pointer to a selected configuration structure
Return code	
void	-
Functional Description	
Initializes the LIN module channel hardware and sets the state to initialize.	
Particularities and Limitations	
This service function has to be called before any other LIN driver function. The service ID of this function is LIN_SID_INIT_ID	
Call Context	
Called by upper layer.	

Table 5-4 Lin_Init

5.2.3 Lin_GetVersionInfo

Prototype	
`void Lin_GetVersionInfo (Std_VersionInfoType *versioninfo)`	
Parameter	
versioninfo	Pointer to where to store the version information of this module.
Return code	
void	-
Functional Description	
This service returns version information as decimal, vendor ID and AUTOSAR module ID of the component.	
Particularities and Limitations	
This function shall be pre compile time configurable On/Off by the configuration parameter: LIN_VERSION_INFO_API The service ID of this function is LIN_SID_GETVERSIONINFO_ID	
Call Context	
Called by upper layer.	

Table 5-5 Lin_GetVersionInfo

5.2.4 Lin_SendFrame

Prototype	
`Std_ReturnType Lin_SendFrame (uint8 Channel, Lin_PduType *PduInfoPtr)`	
Parameter	
Channel	LIN channel to be addressed
PduInfoPtr	Pointer to PDU containing the PID, Checksum model, Response type, Dl and SDU data pointer
Return code	
Std_ReturnType	E_OK: send command has been accepted E_NOT_OK: send command has not been accepted, development or production error occurred
Functional Description	
The function Lin_SendFrame generates a LIN frame on the addressed LIN channel.	
Particularities and Limitations	
The service ID of this function is LIN_SID_SENDFRAME_ID Only for Master channels.	
Call Context	
Called by upper layer.	

Table 5-6 Lin_SendFrame

5.2.5 Lin_GoToSleep

Prototype	
`Std_ReturnType **Lin_GoToSleep** (uint8 Channel)`	
Parameter	
Channel	LIN channel to be addressed
Return code	
Std_ReturnType	E_OK: Sleep command has been accepted E_NOT_OK: Sleep command has not been accepted, development or production error occurred
Functional Description	
Transmits a go-to-sleep command on the addressed LIN channel and sets the channel into sleep mode.	
Particularities and Limitations	
If supported by HW the LIN hardware unit maybe set to reduced power operation mode. The service ID of this function is LIN_SID_GOTOSLEEP_ID Only for Master channels.	
Call Context	
Called by upper layer.	

Table 5-7 Lin_GoToSleep

5.2.6 Lin_GoToSleepInternal

Prototype	
`Std_ReturnType **Lin_GoToSleepInternal** (uint8 Channel)`	
Parameter	
Channel	LIN channel to be addressed
Return code	
Std_ReturnType	E_OK: Sleep command has been accepted E_NOT_OK: Sleep command has not been accepted, development or production error occurred
Functional Description	
Sets the channel to sleep mode without sending a go-to-sleep command.	
Particularities and Limitations	
The service ID of this function is LIN_SID_GOTOSLEEPINTERNAL_ID	
Call Context	
Called by upper layer.	

Table 5-8 Lin_GoToSleepInternal

5.2.7 Lin_Wakeup

Prototype	
`Std_ReturnType` **`Lin_Wakeup`** `(uint8 Channel)`	
Parameter	
Channel	LIN channel to be addressed
Return code	
Std_ReturnType	E_OK: Wake-up request has been accepted E_NOT_OK: Wake-up request has not been accepted, development or production error occurred
Functional Description	
Sends a wakeup frame on the on the addressed LIN channel.	
Particularities and Limitations	
The service ID of this function is LIN_SID_WAKEUP_ID	
Call Context	
Called by upper layer.	

Table 5-9 Lin_Wakeup

5.2.8 Lin_WakeupInternal

Prototype	
`Std_ReturnType` **`Lin_WakeupInternal`** `(uint8 Channel)`	
Parameter	
Channel	LIN channel to be addressed
Return code	
Std_ReturnType	E_OK: Wake-up request has been accepted E_NOT_OK: Wake-up request has not been accepted, development or production error occurred
Functional Description	
Sets the channel state to LIN_CH_OPERATIONAL without generating a wake up pulse.	
Particularities and Limitations	
The service ID of this function is LIN_SID_WAKEUPINTERNAL_ID	
Call Context	
Called by upper layer.	

Table 5-10 Lin_WakeupInternal

5.2.9 Lin_CheckWakeup

Prototype	
`Std_ReturnType` **`Lin_CheckWakeup`** `(uint8 Channel)`	
Parameter	
Channel	LIN channel to be addressed
Return code	
Std_ReturnType	E_OK: No error has occurred during execution of the API E_NOT_OK: An error has occurred during execution of the API
Functional Description	
After a wake up caused by LIN bus transceiver or LIN driver the function Lin_CheckWakeup will be called by the LIN Interface module to identify the corresponding LIN channel.	
Particularities and Limitations	
The service ID of this function is LIN_SID_CHECKWAKEUP_ID	
Call Context	
Called by upper layer.	

Table 5-11 Lin_CheckWakeup

5.2.10 Lin_GetStatus

Prototype	
`Lin_StatusType` **`Lin_GetStatus`** `(uint8 Channel, Lin_u8PtrType *Lin_SduPtr)`	
Parameter	
Channel	LIN channel to be addressed
Lin_SduPtr	Pointer to pointer to shadow buffer or memory mapped LIN Hardware receive buffer
Return code	
Lin_StatusType	Lin_StatusType: Information about the current message state.
Functional Description	
The function Lin_GetStatus shall return the current transmission, reception or operation status of the LIN driver. Only for Master channels.	
Particularities and Limitations	
The service ID of this function is LIN_SID_GETSTATUS_ID.	
Call Context	
Called by upper layer.	

Table 5-12 Lin_GetStatus

5.2.11 Lin_Interrupt

Prototype	
`void Lin_Interrupt (uint8 ChannelHw)`	
Parameter	
ChanneHw	ChannelHw index (not Channel ID) of configuration from where the interrupt occurred.
Return code	
void	-
Functional Description	
Interrupt processing function. Handles the internal state machine.	
Particularities and Limitations	
The service ID of this function is LIN_SID_INTERRUPT_ID.	
Call Context	
Called by LinIsr_<x>. The user should not call this function directly.	

Table 5-13 Lin_Interrupt

5.3 Services used by LIN

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

Component	API
DET	Det_ReportError()
DEM	Dem_ReportErrorStatus()

EcuM	EcuM_CheckWakeup()
	EcuM_SetWakeupEvent()

LinIf	LinIf_WakeupConfirmation()
	LinIf_HeaderIndication()
	LinIf_LinErrorIndication()
	LinIf_RxIndication()
	LinIf_TxConfirmation()

Table 5-14 Services used by the LIN

6. Configuration

In the LIN driver the attributes can be configured according to/ with the following methods/ tools:

> Configuration in DaVinci Configurator 5

6.1 Configuration Variants

The LIN driver supports the configuration variants

> `VARIANT-PRE-COMPILE`

7. Glossary and Abbreviations

7.1 Glossary

Term	Description
Channel	A channel defines the assignment (1:1) between a physical communication interface and a physical layer on which different modules are connected to.
Interrupt	Processor-specific event which can interrupt the execution of a current program section.
Transceiver	A transceiver adapts the physical layer to the communication interface.

Table 7-1 Glossary

7.2 Abbreviations

Term	Description
API	Application Program Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basic Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
HW	Hardware
ID	Identifier (e.g. Identifier of a CAN message)
ISR	Interrupt Service Routine
LIN	Local Interconnect Network
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
RTM	Runtime Measurement
HIS	Hersteller Initiative Software

Table 7-2 Abbreviations

8. Contact

Visit our website for more information on

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

www.vector.com