LIN driver user guide

TRAVEO™ T2G family

About this document

Scope and purpose

This guide describes the architecture, configuration, and use of the local interconnect network (LIN) driver. This document explains the functionality of the driver and provides a reference to the driver's API.

The installation, the build process, and general information on the use of the EB tresos Studio are not within the scope of this document. See the *EB tresos Studio for ACG8 user's guide* **[9]** for a detailed discussion of these topics.

Intended audience

This document is intended for anyone who uses the LIN driver of the TRAVEO™ T2G family.

Document structure

Chapter **1 General overview** gives a brief introduction to the LIN driver, explains the embedding in the AUTOSAR environment, and describes the supported hardware and development environment.

Chapter **2 Using the LIN driver** details the steps on how to use the LIN driver in your application.

Chapter **3 Structure and dependencies** describes the file structure and the dependencies for the LIN driver.

Chapter **4 EB tresos Studio configuration interface** describes the driver's configuration.

Chapter **5 Functional description** gives a functional description of all services offered by the LIN driver.

Chapter **6 Hardware resources** gives a description of all hardware resources used.

The Appendix A and Appendix B provides a complete API reference and access register table.

Abbreviations and definitions

Table 1 Abbreviations

Abbreviation	Description
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASIL	ASIL Automotive Safety Integrity Level
AUTOSAR	Automotive Open System Architecture
Basic Software	Standardized part of software which does not fulfill a vehicle functional job.
Baud rate	Data transfer rate. Since only one bit is represented by a state on the LIN bus this is used as an equivalent for bps (bits per seconds).
DEM	Diagnostic Event Manager
DET	Default Error Tracer
EcuM	ECU State Manager
GCE	Generic Configuration Editor

About this document

Abbreviation	Description
ISR	Interrupt Service Routine
LIN	Local Interconnect Network
LIN bus	The medium on which the LIN messages are transported. Sometimes also used to refer to the whole LIN cluster.
LIN channel	The interface of the LIN master node that is used to communicate with the LIN cluster. (This is what is called “Interface” in the LIN 2.1 Specification.).
LIN cluster	The LIN bus and all its nodes
µC	Microcontroller
MCAL	Microcontroller Abstraction Layer
MCU	Micro Controller Unit
OS	Operating System
EB tresos Studio	Elektrobit Automotive configuration framework
UART	Universal Asynchronous Receiver-Transmitter
UTF-8	8-Bit Universal Character Set Transformation Format

Related documents

AUTOSAR requirements and specifications

Bibliography

[1] General specification of basic software modules, AUTOSAR release 4.2.2.

[2] Specification of LIN driver, AUTOSAR release 4.2.2.

[3] Specification of LIN interface, AUTOSAR release 4.2.2.

[4] Specification of ECU state manager, AUTOSAR release 4.2.2.

[5] Specification of standard types, AUTOSAR release 4.2.2.

[6] Specification of ECU configuration parameters, AUTOSAR release 4.2.2.

[7] Specification of default error tracer, AUTOSAR release 4.2.2.

[8] Layered software architecture, AUTOSAR release 4.2.2.

Elektrobit automotive documentation

Bibliography

[9] EB tresos Studio for ACG8 user’s guide.

Hardware documentation

The hardware documents are listed in the delivery notes.

Related standards and norms

Bibliography

[10] LIN specification package, revision 2.1.

Table of contents

Table of contents

Table of contents

1 General overview

1.1 Introduction to the LIN driver

The LIN driver abstracts the LIN hardware of TRAVEO™ T2G family microcontrollers and provides API functions. The LIN driver is responsible for the transfer of LIN frames to a specific microcontroller. The LIN interface is the main part of the AUTOSAR LIN implementation as it is responsible for the processing of the LIN schedule tables. The LIN driver processes the transmission and reception of the LIN frames.

1.2 User profile

This guide is intended for users with a basic knowledge of the following domains:

- LIN protocol
- Automotive embedded systems
- C programming language
- AUTOSAR standard
- Target hardware architecture

1.3 Embedding in the AUTOSAR environment

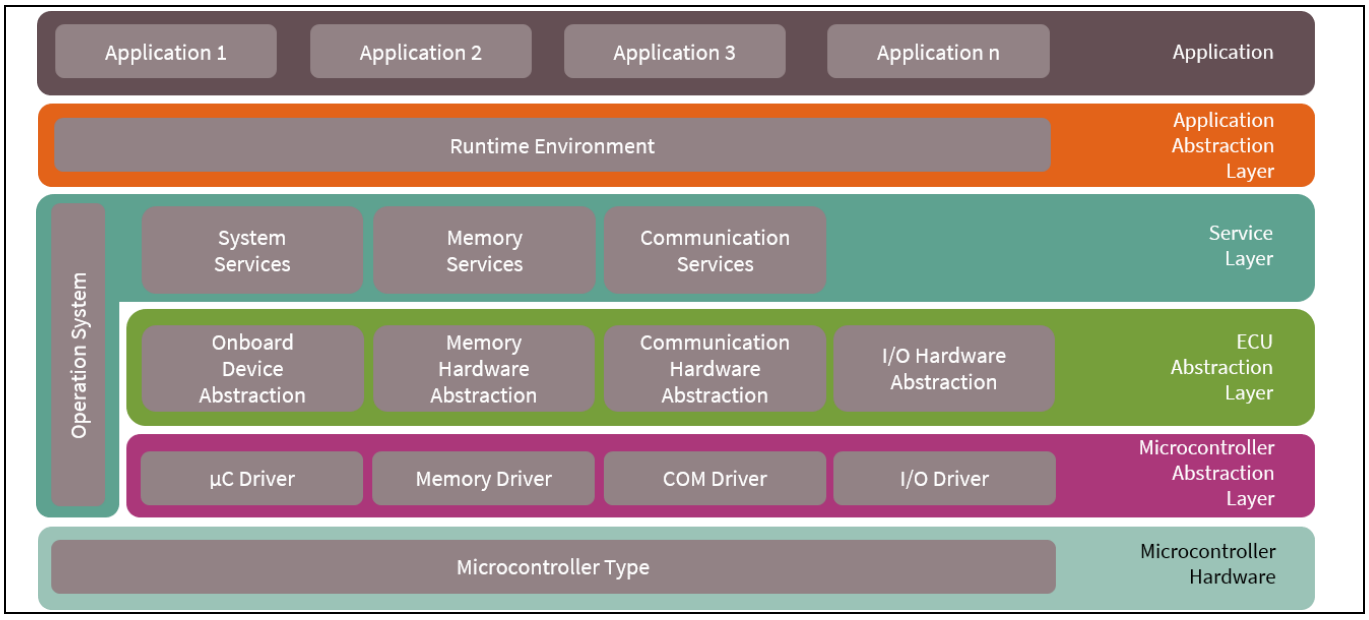

Figure 1 Overview of AUTOSAR software layers

Figure 1 depicts the layered AUTOSAR software architecture. The LIN driver (**Figure 2**) is part of the microcontroller abstraction layer (MCAL), the lowest layer of basic software in the AUTOSAR environment.

As a communication driver, LIN driver accesses the hardware directly and provides a standardized and hardware independent API for the LIN interface.

For an exact overview of the AUTOSAR layered software architecture, see *Layered software architecture* **[8]**.

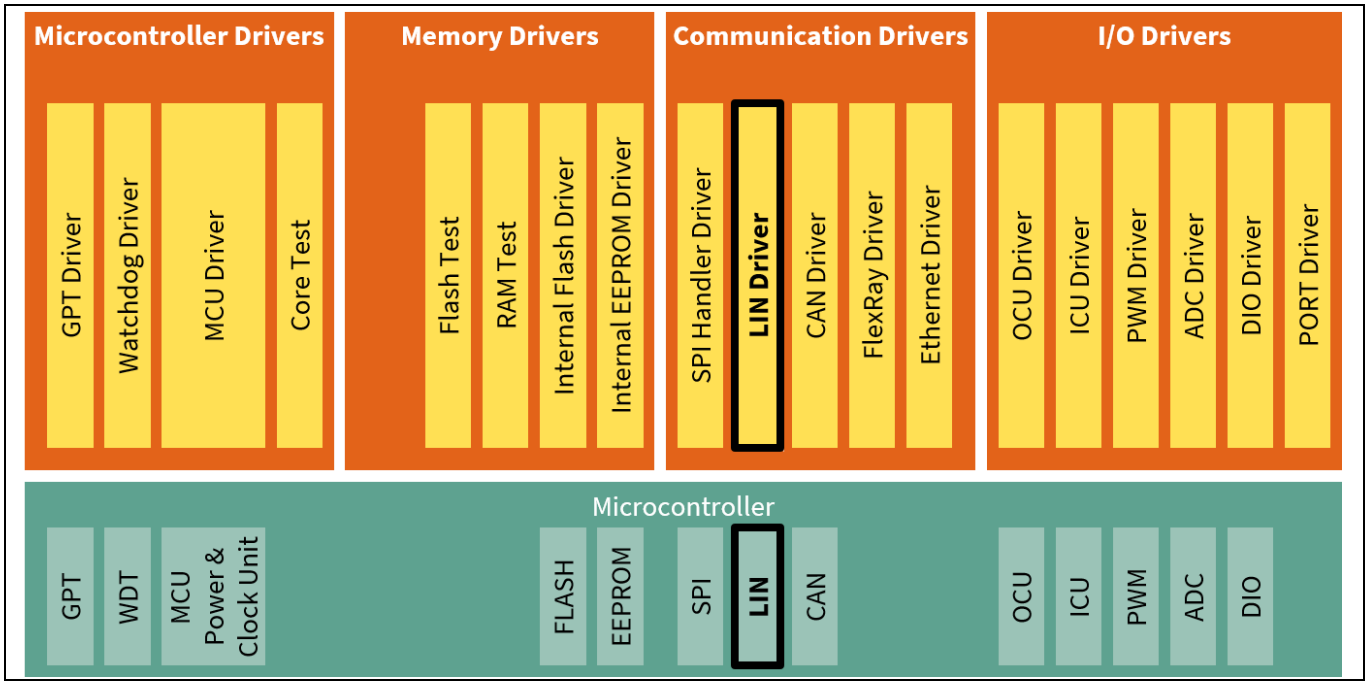

Figure 2 LIN driver in MCAL layer

1.4 Supported hardware

This version of the LIN driver supports TRAVEO™ T2G family microcontrollers. However, the SCB instances is not supported. No special external hardware devices are required.

The supported derivatives are listed in the release notes.

1.5 Development environment

The development environment corresponds to AUTOSAR release 4.2.2. The modules Base, Make, Mcu, Port and Resource are needed for proper functionality of the LIN driver.

According to AUTOSAR release 4.2.2., the AUTOSAR environment must provide file *Lin_GeneralTypes.h* for proper LIN functionality. Set the include path correctly for *Lin_GeneralTypes.h.*

1.6 Character set and encoding

All source code files of the MCU driver are restricted to the ASCII character set. The files are encoded in UTF-8 format, with only the 7-bit subset (values 0x00 … 0x7F) being used.

2 Using the LIN driver

This chapter describes all necessary steps to incorporate the LIN driver into your application.

2.1 Installation and prerequisites

Note: Before you start, see the EB tresos Studio for ACG8 user's guide [9] for the following information.

1. *The installation procedure of EB tresos ECU AUTOSAR components*
2. *The usage of the EB tresos Studio*
3. *The usage of the EB tresos ECU AUTOSAR build environment (It includes the steps to setup and integrate the own application within the EB tresos ECU AUTOSAR build environment)*

The installation of the LIN driver compiles with the general installation procedure for EB tresos ECU AUTOSAR components given in the documents mentioned above. If the driver has been successfully installed, the driver will appear in the module list of the EB tresos Studio (see *EB tresos Studio for ACG8 user's guide* **[9]**).

This document assumes that the project is properly set up and is using the application template as described in the *EB tresos Studio for ACG8 user's guide* **[9]**. This template provides the necessary folder structure, project and makefiles needed to configure and compile your application within the build environment. You must be familiar with the use of the command shell.

2.2 Configuring the LIN driver

The LIN driver can be configured with any AUTOSAR compliant GCE tool. Save the configuration in a separate file named e.g., *Lin.xdm*. More information about the LIN driver configuration can be found in chapter **4 EB tresos Studio configuration interface**.

2.2.1 Architecture specifics

See section **4.3 Vendor and driver specific parameters**, for all vendor- and driver-specific configuration parameters.

2.3 Adapting your application

The LIN driver is normally used via the LIN interface and may not be accessed directly. See the AUTOSAR *Specification of the LIN interface* **[3]** for more information.

To access the LIN driver's functions directly from the application, do the following:

Include headers for the PORT driver and LIN driver:

```
#include "Mcu.h"
#include "Port.h"
#include "Lin.h"
```

Initialize the MCU and PORT driver before the LIN driver with:

```
Mcu_Init(&Mcu_Config[0]);
Port_Init(&Port_Config[0]);
```

The function `Mcu_Init()` is called with a pointer to a structure of type `Mcu_ConfigType`, which is published by the MCU driver itself.

The function `Port_Init()` is called with a pointer to a structure of type `Port_ConfigType`, which is published by the PORT driver itself.

Then, initialize the LIN driver with:

```
Lin_Init(NULL_PTR);
```

The PORT driver must be properly configured. The procedure to configure is explained in section **6.1 Ports and pins**.

Do not forget to configure the interrupts as explained in section **6.3 Interrupts**.

Finally, depending on the configuration, the LIN driver requires at least some of the callback functions mentioned in section **7.5 Required callback functions**. If you want to experiment with the only the LIN driver, you must provide stubs to these callback functions.

2.4 Starting the build process

Do the following to build your application:

Note: For a clean build, use the build command with target `clean_all`. before (`make clean_all`).:

1. On the command shell, type the following command to generate the necessary configuration-dependent files. See **3.3 Generated files**.

   ```
   > make generate
   ```

2. Type the following command to resolve required file dependencies:

   ```
   > make depend
   ```

3. Type the following command to compile and link the application:

   ```
   > make (optional target: all)
   ```

2.5 Measuring stack consumption

Do the following to measure stack consumption. It requires the Base module for proper measurement.

Note: All files (including library files) should be rebuilt with the dedicated compiler option. The executable file built in this step must be used only to measure stack consumption.

1. Add the following compiler option to the Makefile to enable stack consumption measurement:

   ```
   -DSTACK_ANALYSIS_ENABLE
   ```

2. Type the following command to clean library files:

   ```
   > make clean_lib
   ```

3. Follow the build process described in **Starting the build process**.
4. Follow the instructions in the release notes and measure the stack consumption.

2.6 Memory mapping

The *Lin_MemMap.h* file in the *$(TRESOS_BASE)/plugins/MemMap_TS_T40D13M0I0R0/include* directory is a sample. This file is replaced by the file generated by MEMMAP module. Input to MEMMAP module is generated as *Lin_Bswmd.arxml* in the *$(PROJECT_ROOT)/output/generated/swcd* directory of your project folder.

2.6.1 Memory allocation keyword

- `LIN_START_SEC_CODE_ASIL_B` / `LIN_STOP_SEC_CODE_ASIL_B`

  The memory section type is CODE. All executable code is allocated in this section.

- `LIN_START_SEC_CONST_ASIL_B_UNSPECIFIED` / `LIN_STOP_SEC_CONST_ASIL_B_UNSPECIFIED`

  The memory section type is CONST. The following constants are allocated in this section:

  - LIN whole configuration setting
  - LIN channel configuration setting

- `LIN_START_SEC_VAR_INIT_ASIL_B_UNSPECIFIED` / `LIN_STOP_SEC_VAR_INIT_ASIL_B_UNSPECIFIED`

  The memory section type is VAR. The following variable is allocated in this section:

  - Information for Lin status

- `LIN_START_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED` / `LIN_STOP_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED`

  The memory section type is VAR. The following variable is allocated in this section:

  - Information for LIN channel status

3 Structure and dependencies

The LIN driver consists of static, configuration and generated files.

3.1 Static files

- $(PLUGIN_PATH)=$(TRESOS_BASE)/plugins/Lin_TS_* is the path to the LIN driver plugin.
- *$(PLUGIN_PATH)/lib_src* contains all static source files of the LIN driver. These files contain the functionality of the driver which does not depend on the current configuration. The files are grouped into a static library.
- *$(PLUGIN_PATH)/src* comprises configuration dependent source files or special derivate files. Each file will be rebuilt when the configuration is changed.

All necessary source files will automatically be compiled and linked during the build process and all include paths will be set if the LIN driver is enabled.

- *$(PLUGIN_PATH)/include* is the basic public include directory you need to include in *Lin.h*.
- *$(PLUGIN_PATH)/autosar* directory contains the AUTOSAR ECU parameter definition with vendor, architecture and derivative specific adaptations to create a correct matching parameter configuration for the LIN driver.

3.2 Configuration files

The configuration of the LIN driver is done via EB tresos Studio. The file containing the LIN driver's configuration is named *Lin.xdm* and is in the *$(PROJECT_ROOT)/config* directory. This file serves as an input for the generation of the configuration dependent source and header files during the build process.

3.3 Generated files

During the build process the following files are generated based on the current configuration description. They are in the *output/generated* sub folder of your project folder:

- include/Lin_Cfg.h
- include/Lin_Defines.h
- include/Lin_Enable_Api.h
- src/Lin_Irq.c

Note: *Generated source files need not to be added to your application make file. These files will be compiled and linked automatically during the build process.*

- swcd/Lin_Bswmd.arxml

Note: *Additional steps are required for the generation of BSW module description. In EB tresos Studio, follow the menu path **Project** > **Build Project** and click **generate_swcd**.*

3.4 Dependencies

As part of the AUTOSAR framework, the LIN driver needs to interact with several other AUTOSAR modules. This section lists modules that have a relation with the LIN driver and gives information about the type of connection between those modules and the LIN driver (**Figure 3**).

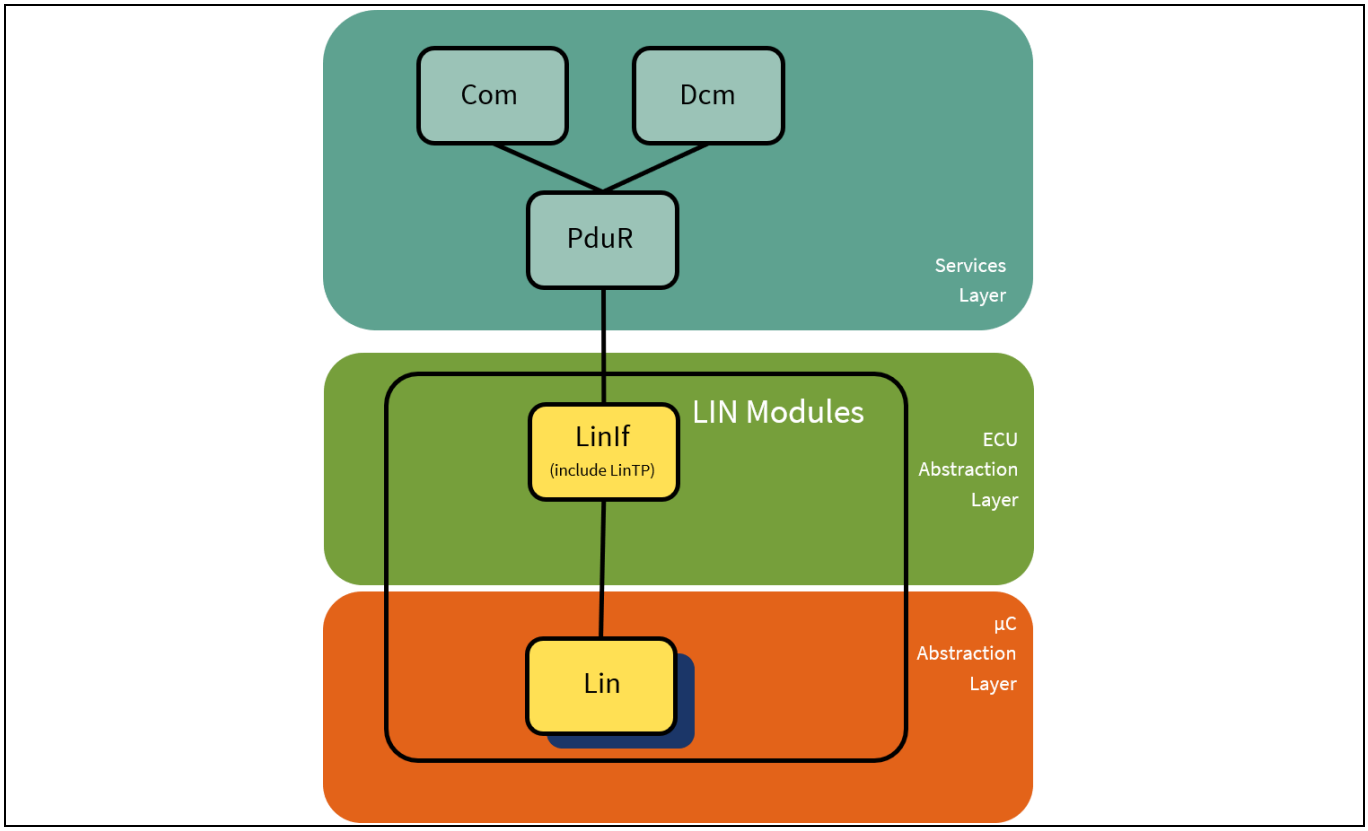

Figure 3 Position of the LIN driver in the AUTOSAR layers

3.4.1 PORT driver

The PORT driver (not displayed) needs to be correctly configured to allow the use of the appropriate microcontroller pins. The PORT driver needs to be initialized before the LIN driver is initialized.

3.4.2 MCU driver

The MCU driver needs to be initialized and all MCU clock reference points referenced by the LIN driver channels via configuration parameter `LinClockRef` must have been activated (via calls of MCU API functions) before initializing the LIN driver. See the MCU driver's user guide for details.

Note: This LIN clock divided by 16 is used as the baud rate of LIN channel. The LIN driver does not calculate the baud rate from configuration parameter `LinClockRef`.

3.4.3 LIN interface

The LIN interface is part of the ECU abstraction layer which is located above the LIN driver. It is the only module that calls the LIN driver functions. The LIN driver provides the ability to notify the LIN interface in case a wakeup event is detected on an individual channel by calling the callback function `LinIf_WakeupConfirmation()`. When this feature is to be used, the LIN interface needs to be configured properly.

3.4.4 AUTOSAR OS

The AUTOSAR operating system handles the interrupts used by the LIN driver. See section **6.3 Interrupts** for more information.

3.4.5 ECU state manager

The LIN driver provides the ability to notify the EcuM in case a wakeup event is detected on an individual channel by calling the callback functions `EcuM_SetWakeupEvent()` and `EcuM_CheckWakeup()`. When this feature is to be used, the EcuM needs to be configured properly.

3.4.6 DET

If default error detection is enabled in the LIN driver configuration, the DET module needs to be installed, configured, and integrated with the application as well.

3.4.7 DEM

The production error `LIN_E_TIMEOUT` can be defined for the LIN driver in the LIN module DEM configuration:

Note: *The production error `LIN_E_TIMEOUT` is currently not reported by any function of the LIN driver. (There is no blocking function, e.g., polling of a hardware register in a while loop, where it could be used.)*

3.4.8 Error callout handler

The error callout handler is called on every error that is detected, regardless of whether default error detection is enabled. The error callout handler is an ASIL safety extension that is not specified by AUTOSAR. It is configured via configuration parameter `LinErrorCalloutFunction`.

4 EB tresos Studio configuration interface

The GUI is not part of this delivery. For further information, see *EB tresos Studio for ACG8 user's guide* [9].

4.1 General configuration

The module comes preconfigured with default settings. These settings must be adapted when necessary.

4.2 Standard parameters

This section lists all configuration parameters required by AUTOSAR. Deviations from the specification or hardware specific restrictions are mentioned where applicable.

4.2.1 Container LinGeneral

4.2.1.1 LinDevErrorDetect

Description

Enables or disables default error notification for the LIN driver.

Remarks

Setting this parameter to FALSE will disable the notification of development errors via DET. However, in contrast to AUTOSAR specification, detection of development errors is still enabled and errors will be reported via `LinErrorCalloutFunction`.

4.2.1.2 LinIndex

Description

Represents the LIN driver's ID that is used as a reference by the upper software layer.

Remarks

None

4.2.1.3 LinTimeoutDuration

Description

Specifies the maximum number of polling loops within a function until a timeout is raised.

Remarks

This parameter is not used by the LIN driver and therefore not being evaluated. There is no blocking function (e.g., polling of a hardware register in a while loop) where it could be used.

4.2.1.4 LinVersionInfoApi

Description

This parameter controls the availability of the API function `Lin_GetVersionInfo`.

Remarks

None

4.2.2 Container LinChannel

4.2.2.1 LinChannelBaudRate

Description

Specifies the baud rate of the LIN channel in kbits/s.

Remarks

This parameter is not used by the LIN driver. The LIN clock set by MCU divided by 16 is used as the baud rate of LIN channel.

4.2.2.2 LinChannelId

Description

Identifies the LIN channel in each LIN driver.

Remarks

The valid range of `LinChannelId` has been restricted to 0...255.

The value of `LinChannelId` must be unique among all configured channels.

4.2.2.3 LinChannelWakeupSupport

Description

Enables or disables the “wakeup by slave” support of a LIN channel. Note that due to the hardware architecture this wakeup functionality is not available while the MCU is in Deep Sleep mode and Hibernate mode.

Remarks

None

4.2.2.4 LinChannelEcuMWakeupSource

Description

If `LinChannelWakeupSupport` is true, `LinChannelEcuMWakeupSource` specifies a reference to the wakeup source defined in EcuM.

Remarks

None

4.2.2.5 LinClockRef

Description

Reference to the LIN clock source configuration, which is set in the MCU driver configuration.

Remarks

The LIN driver does not calculate the baud rate from this configuration parameter. The LIN clock which set by MCU divided by 16 is used as the baud rate of LIN channel.

4.2.3 Container LinDemEventParameterRefs

4.2.3.1 LIN_E_TIMEOUT

Description

Reference to the DemEventParameter which will be issued when a timeout caused by hardware error has occurred.

Remarks

The production error `LIN_E_TIMEOUT` is currently not reported by any function of the LIN driver.

4.3 Vendor and driver specific parameters

4.3.1 Container LinGeneral

4.3.1.1 LinErrorCalloutFunction

Description

```
Error callout function. Syntax:
void ErrorCalloutHandler
(
  uint16 ModuleId,
  uint8 InstanceId,
  uint8 ApiId,
  uint8 ErrorId
);
The error callout function is called on every error. The ASIL level of this function
limits the ASIL level of the LIN Driver.
```

Type

`EcucFunctionNameDef`

4.3.1.2 LinIncludeFile

Description

Lists the file names that will be included within the driver. Any application specific symbol that is used by the Lin configuration (e.g., error callout function) should be included by configuring this parameter.

Type

`EcucStringParamDef`

4.3.1.3 LinDeInitApi

Description

Controls the availability of API function `Lin_DeInit`.

Type

`EcucBooleanParamDef`

4.3.2 Container LinChannel

4.3.2.1 LinChannelHwUsed

Description

Selects the physical interface of the channel.

Type

`EcucStringParamDef`

Range

LIN00: LIN Instance 0, LIN Channel 0

LIN01: LIN Instance 0, LIN Channel 1

...

LINmn: LIN Instance m, LIN Channel n

Note: The number of available LIN instances and channels depends on the configured target derivative.

4.3.2.2 LinChannelBreakLength

Description

Length of LIN break on the bus in bits.

Type

`EcucIntegerParamDef`

Range

13...26

4.3.2.3 LinChannelBreakDelimiter

Description

Length of the LIN break delimiter field on the bus, represented in bits.

Type

`EcucIntegerParamDef`

Range

1...4

4.3.2.4 LinChannelWakeupLength

Description

Length of the reception wakeup on the bus, represented in bits.

Type

`EcucIntegerParamDef`

Range

1...31

4.3.2.5 LinChannelRxNoiseFilter

Description

Controls the availability of receiver noise filter.

Type

`EcucBooleanParamDef`

4.3.2.6 LinInstance

Description

Instance to be used for this LIN channel.

Type

`EcucIntegerParamDef`

Range

0...9

4.4 Other modules

4.4.1 PORT driver

The pins given in section **6.1 Ports and pins** must be configured in the PORT driver.

4.4.2 LIN interface

The LIN interface must be configured to match the LIN driver's configuration.

4.4.3 DET

DET must be configured if default error detection is activated.

4.4.4 AUTOSAR OS

The LIN driver's interrupts (listed in section **6.3 Interrupts**) must be configured in the AUTOSAR operating system.

Note: The AUTOSAR OS must configure only the interrupts that are used by the LIN driver.

5 Functional description

The LIN driver is intended to be used as a LIN master node on a LIN bus with one or more connected LIN slaves. The LIN driver's purpose is to provide a known API for the LIN interface regardless of the underlying hardware.

The LIN driver's main functions are to send and receive LIN frames, generate and verify checksums, put the devices on a LIN bus in Sleep mode, and generate/detect wakeup signals.

A single LIN driver can manage several independent LIN buses simultaneously.

5.1 Function of the module

5.1.1 Frame transmission and reception

Communication on the LIN bus is mediated by the LIN master. For reception, the function `Lin_SendFrame()` must be called with a response type set to `LIN_SLAVE_RESPONSE` to induce the LIN slaves to answer. For transmission, the function `Lin_SendFrame()` must be called with a response type set to `LIN_MASTER_RESPONSE` to initiate the transmission.

For a more detailed description of the functionality of the LIN interface and driver, see the functional description of the LIN interface in *Specification of LIN interface* **[3]**.

5.1.2 Sleep mode

The LIN cluster connected to a channel can be put into Sleep mode by calling the function `Lin_GoToSleep()` or `Lin_GoToSleepInternal()`. `Lin_GoToSleep()` sends a sleep frame before entering the Sleep mode while, `Lin_GoToSleepInternal()` enters the Sleep mode immediately.

Once in Sleep mode, the cluster can be woken up either by a call to the function `Lin_Wakeup()` or `Lin_WakeupInternal ()` or by a wakeup signal received from LIN bus. If wakeup support is enabled in the configuration, the EcuM callback functions `EcuM_SetWakeupEvent()` or `EcuM_CheckWakeup()`, and the LIN interface callback functions `LinIf_WakeupConfirmation ()` will be called by the LIN.

The LIN driver detects wakeup when a dominant pulse longer than the time specified by configuration `LinChannelWakeupLength` is received.

`Lin_Wakeup()` causes a dominant pulse of 5 bit times. Depending on the configured baud rate, the pulse length is
0.25 ms to 5 ms.

Note: *Note that due to the hardware architecture, the wakeup functionality is not available while the MCU is in Deep Sleep mode and Hibernate mode.*

Note: *The minimum of wakeup signal is 150 µs. If the 1 bit time of the baud rate is longer than 150 µs, the following procedure is necessary.*

1. Call `Lin_DeInit()`.
2. Change source clock of LIN from MCU module.
3. Call `Lin_Init()`.
4. Call `Lin_GotoSleepInternal()`.
5. Change the of baud rate according to steps 1 to 3 after wakeup.

5.1.3 Status acquisition

The LIN driver can be polled for the state of each of its channels. This is done by using the `Lin_GetStatus()` function, which returns a `Lin_StatusType`. The possible return values have the meaning as listed in **Table 2**.

Table 2 Possible return values of Lin_GetStatus()

Value	Description
`LIN_NOT_OK`	Development or production error occurred
`LIN_TX_OK`	Transmission successful
`LIN_TX_BUSY`	Transmission in progress
`LIN_TX_HEADER_ERROR`	Erroneous header transmission
`LIN_TX_ERROR`	Erroneous response transmission
`LIN_RX_OK`	Reception successful
`LIN_RX_BUSY`	Reception in progress
`LIN_RX_ERROR`	Erroneous response reception
`LIN_RX_NO_RESPONSE`	No data received
`LIN_OPERATIONAL`	Channel ready for transmission
`LIN_CH_SLEEP`	Channel is in Sleep mode

5.2 Initialization

As with most modules, the LIN driver needs to be initialized once before use. Initialization of the LIN driver is made by a call to the `Lin_Init()` function.

Note: This LIN driver does not support post-build-time configuration, thus the parameter passed to the function `Lin_Init()` must be a NULL pointer.

Not all initializations are performed by the LIN driver. The PORT module needs to be called before the LIN driver is used. See section **2.3 Adapting your application**.

5.3 Runtime reconfiguration

The LIN driver does not support reconfiguration during runtime.

The LIN baud rate can be changed by the following procedure:

1. Call `Lin_DeInit()`.
2. Change source clock of LIN from MCU module.
3. Call `Lin_Init()`.

5.4 API parameter checking

The driver's services perform regular error checks.

When an error occurs, the error hook routine (configured via `LinErrorCalloutFunction`) is called and the error code, service ID, module ID, and instance ID are passed as parameters.

If default error detection is enabled, all errors are also reported to DET, a central error hook function within the AUTOSAR environment. The checking itself cannot be deactivated for safety reasons.

Table 3 shows the development error checks that are performed by the services of the LIN driver.

Table 3 Development error codes

Related error code	Value	Type of error
LIN_E_UNINIT	0	API service used without module initialization.
LIN_E_INVALID_CHANNEL	2	API service used with an invalid or inactive channel parameter.
LIN_E_INVALID_POINTER	3	API service called with invalid configuration pointer.
LIN_E_STATE_TRANSITION	4	Invalid transition for the current state.
LIN_E_PARAM_POINTER	5	API service called with a NULL pointer.
LIN_E_PARAM_POINTERCONTENT	8	API service called with invalid pointer content.

5.5 Reentrancy

AUTOSAR does not require reentrancy for the LIN driver except `Lin_GetVersionInfo()`. It is not guaranteed for any non-reentrant API functions of this driver.

5.6 Debugging support

The LIN driver does not support debugging.

5.7 Execution time dependencies

The execution of the API function is dependent on certain factors. **Table 4** lists these dependencies.

Table 4 Execution time dependencies

Affected function	Dependency
`Lin_Init()` `Lin_DeInit()`	Runtime depends on the number of configured channels, because all configured channels are initialized in this function.

`Lin_GoToSleep()`
`Lin_GoToSleepInternal()`
`Lin_SendFrame()`
`Lin_Wakeup()`
`Lin_WakeupInternal()`
`Lin_CheckWakeup()`
`Lin_GetStatus()`
	Runtime of internal translation from channel ID to index depends on the number of configured channels.

`Lin_GetStatus()`	Runtime depends on the channel status and the received data length.
`Lin_SendFrame()`	Runtime depends on the length of passed PDU.

5.8 Environment restrictions

The LIN driver's environment must take care of the following restrictions:

- LIN driver requires the MCU and PORT to be initialized, as the LIN driver relies on the MCU and PORT settings.
- The LIN module's environment will not call any function of the LIN module before calling `Lin_Init()` except `Lin_GetVersionInfo()`.
- The LIN module's environment will only call `Lin_SendFrame()` on a channel which is in the `LIN_CH_OPERATIONAL` state or in one of the sub-states of `LIN_CH_OPERATIONAL`.
- After using the `Lin_SendFrame()` function, and if data is received, the LIN interface has to wait for the corresponding response part of the LIN frame (by polling with the function `Lin_GetStatus()`).

- The upper layer of the LIN driver must keep the transmit buffer data consistent until the return of the transmit function call.
- The LIN driver's environment will only call `Lin_Wakeup()` or `Lin_WakeupInternal()` when the LIN channel is in the `LIN_CH_SLEEP` state.
- The configuration variant is pre-compile selectable (PC).

6 Hardware resources

6.1 Ports and pins

The LIN driver uses the LIN hardware of TRAVEO™ T2G family microcontrollers. To allow the use of those, the PORT driver needs to be configured properly. Therefore, the pins corresponding to the LIN hardware need to be configured according to **Table 5**.

The pin numbers depend on the target derivative and its package, see the corresponding hardware manual for the correct pins.

Table 5 Port and pins

Pin	PortPinDirection	PortPinInitialMode	PortPinOutputDrive
LIN_TX	OUT	LIN_TX	PULLUP
LIN_RX	IN	LIN_RX	HIGHZ
LIN_EN	OUT	GPIO	STRONG

Note: The LIN driver does not control `LIN_EN`. The `PortPinInitialMode` of `LIN_EN` must be in the GPIO mode.

6.2 Timer

The LIN driver does not use hardware timers.

6.3 Interrupts

Each LIN channel is associated with one LIN hardware. For each LIN hardware, one interrupt line is available.

The LIN driver uses interrupts only for the detection of the following events:

- Completion of go-to-sleep command
- Wakeup event detection

For the corresponding interrupt vector numbers, see the list of interrupt assignments in the target device's hardware manual. The association with the driver's interrupt services routines (ISRs) of category1 or category2 needs to be done according to **Table 6**. Note that ISRs are only available for the used interrupt lines of the configured LIN channels.

Table 6 ISR names

ISR name cat1	ISR name cat2
`Lin_IsrWrapper_LIN<n>_TxRx_Cat1()`	`Lin_IsrWrapper_LIN<n>_TxRx_Cat2()`

Note: The OS must be associated the named ISRs (declared in Lin_Cfg.h) with the corresponding LIN interrupt.
For example, consider a LIN channel is configured to use LIN02. Then, `Lin_IsrWrapper_LIN02_TxRx_Cat2()` must be called from the (OS) interrupt service routine of `LIN hardware ch.2` interrupt. If category1 is used, the address of `Lin_IsrWrapper_LIN02_TxRx_Cat1()` must be the entry for `LIN hardware ch.2` interrupt in the (OS) interrupt vector table.

Note: *On the Arm® Cortex®-M4 CPU, priority inversion of interrupts may occur under specific timing conditions in the integrated system with TRAVEO™ T2G MCAL. For more details, see the following errata notice.*

Arm® Cortex®-M4 Software Developers Errata Notice - 838869:
“Store immediate overlapping exception return operation might vector to incorrect interrupt”

If the user application cannot tolerate the priority inversion, a DSB instruction should be added at the end of the interrupt function to avoid the priority inversion.

TRAVEO™ T2G MCAL interrupts are handled by an ISR wrapper (handler) in the integrated system. Thus, if necessary, the DSB instruction should be added just before the end of the handler by the integrator.

7 Appendix A – API reference

7.1 Include files

If the LIN driver is used without the LIN interface, you must include *Lin.h* within your application.

7.2 Data types

7.2.1 Lin_FramePidType

Type

```
typedef uint8 Lin_FramePidType
```

Description

This type represents all valid protected identifiers used by the `Lin_SendFrame()` Function.

7.2.2 Lin_FrameCsModelType

Type

```
typedef enum
{
  LIN_ENHANCED_CS,
  LIN_CLASSIC_CS
}  Lin_FrameCsModelType;
```

Description

This type is used to specify the checksum model to be used for the LIN frame.

7.2.3 Lin_FrameResponseType

Type

```
typedef enum
{
  LIN_MASTER_RESPONSE,
  LIN_SLAVE_RESPONSE,
  LIN_SLAVE_TO_SLAVE
}  Lin_FrameResponseType;
```

Description

This type is used to specify whether the frame processor is required to transmit the response part of the LIN frame.

7.2.4 Lin_FrameDlType

Type

```
typedef uint8 Lin_FrameDlType;
```

Description

This type is used to specify the number of SDU data bytes to copy.

7.2.5 Lin_PduType

Type

```
typedef struct
{
  Lin_FramePidType      Pid;
  Lin_FrameCsModelType  Cs;
  Lin_FrameResponseType Drc;
  Lin_FrameDlType       Dl;
  uint8                 *SduPtr;
}  Lin_PduType;
```

Description

This type is used to provide PID, checksum model, response type, data length, and SDU pointer from the LIN interface to the LIN driver.

7.2.6 Lin_StatusType

Type

```
typedef enum
{
  LIN_NOT_OK,
  LIN_TX_OK,
  LIN_TX_BUSY,
  LIN_TX_HEADER_ERROR,
  LIN_TX_ERROR,
  LIN_RX_OK,
  LIN_RX_BUSY,
  LIN_RX_ERROR,
  LIN_RX_NO_RESPONSE,
  LIN_OPERATIONAL,
  LIN_CH_SLEEP
}  Lin_StatusType;
```

Description

LIN operation states for a LIN channel or frame, as returned by the `Lin_GetStatus()` function.

7.3 Constants

7.3.1 Error codes

A service may return one of the error codes listed in **Table 7** if default error detection is enabled.

Table 7 Error codes

Name	Value	Description
LIN_E_UNINIT	0	LIN driver is not initialized
LIN_E_INVALID_CHANNEL	2	Invalid channel given
LIN_E_INVALID_POINTER	3	Invalid pointer given
LIN_E_STATE_TRANSITION	4	Driver or channel is in the wrong state
LIN_E_PARAM_POINTER	5	NULL pointer given

7.3.2 Vendor specific error codes

In addition to the error codes listed in **Table 7**, this LIN driver defines the errors listed in **Table 8**.

Table 8 Vendor specific error codes

Name	Value	Description
LIN_E_PARAM_POINTERCONTENT	8	Invalid pointer content given

7.3.3 Version information

Table 9 lists the version information published in the driver's header file.

Table 9 Version information

Name	Value	Description
LIN_AR_RELEASE_MAJOR_VERSION	4	Major version number (AUTOSAR)
LIN_AR_RELEASE_MINOR_VERSION	2	Minor version number (AUTOSAR)
LIN_AR_RELEASE_REVISION_VERSION	2	Patch version number (AUTOSAR)
LIN_SW_MAJOR_VERSION	see release notes	Vendor specific major version number
LIN_SW_MINOR_VERSION	see release notes	Vendor specific minor version number
LIN_SW_PATCH_VERSION	see release notes	Vendor specific patch version number

7.3.4 Module information

Table 10 Module information

Name	Value	Description
LIN_MODULE_ID	82	Module ID (Lin)
LIN_VENDOR_ID	66	Vendor ID

7.3.5 API service IDs

Table 11 API service IDs

Name	Value	API name
LIN_INIT_API_ID	0x0	Lin_Init
LIN_GETVERSIONINFO_API_ID	0x1	Lin_GetVersionInfo
LIN_SENDFRAME_API_ID	0x4	Lin_SendFrame
LIN_GOTOSLEEP_API_ID	0x6	Lin_GoToSleep
LIN_WAKEUP_API_ID	0x7	Lin_Wakeup
LIN_GETSTATUS_API_ID	0x8	Lin_GetStatus
LIN_GOTOSLEEPINTERNAL_API_ID	0x9	Lin_GoToSleepInternal
LIN_CHECKWAKEUP_API_ID	0xA	Lin_CheckWakeup
LIN_WAKEUPINTERNAL_API_ID	0xB	Lin_WakeupInternal
LIN_DEINIT_API_ID	0xC	Lin_DeInit

7.4 Functions

7.4.1 Lin_Init

Syntax

```
void Lin_Init
(
  const Lin_ConfigType* Config
)
```

Service ID

0x0

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

- `ConfigPtr` – Pointer to the LIN driver's configuration (must be a NULL pointer).

Parameters (out)

None

Return value

None

DET errors

- `LIN_E_STATE_TRANSITION` – LIN driver is already initialized.

- `LIN_E_INVALID_POINTER` – Invalid configuration pointer (non-NULL pointer).

DEM errors

None

Description

Initializes the LIN driver and the configured LIN channels.

7.4.2 Lin_GetVersionInfo

Syntax

```
void Lin_GetVersionInfo
(
  Std_VersionInfoType* versioninfo
)
```

Service ID

0x1

Sync/Async

Synchronous

Reentrancy

Reentrant

Parameters (in)

None

Parameters (out)

`versioninfo` – Pointer to where the version information of this module is stored.

Return value

None

DET errors

- `LIN_E_PARAM_POINTER` – NULL pointer given.

DEM errors

None

Description

Returns the version of the LIN driver in the `Std_VersionInfoType` structure.

7.4.3 Lin_SendFrame

Syntax

```
Std_ReturnType Lin_SendFrame
(
  uint8 Channel,
  Lin_PduType* PduInfoPtr
)
```

Service ID

0x4

Sync/Async

Asynchronous

Reentrancy

Non-reentrant

Parameters (in)

- `Channel` – Selected channel.
- `PduInfoPtr` – Pointer to the `Lin_PduType` structure.

Parameters (out)

None

Return value

- `E_NOT_OK` – DET error occurred.
- `E_OK` – Otherwise.

DET errors

- `LIN_E_UNINIT` – LIN driver is not initialized.
- `LIN_E_INVALID_CHANNEL` – Invalid channel is given.
- `LIN_E_PARAM_POINTER` – PDU pointer is NULL pointer.
- `LIN_E_STATE_TRANSITION` – Driver is in Sleep mode.
- `LIN_E_PARAM_POINTERCONTENT` – PDU is invalid.

DEM errors

None

Description

Starts the transmission of a LIN header (and optionally additionally a LIN response) on the selected channel.

7.4.4 Lin_GoToSleep

Syntax

```
Std_ReturnType Lin_GoToSleep
(
  uint8 Channel
)
```

Service ID

0x6

Sync/Async

Asynchronous

Reentrancy

Non-reentrant

Parameters (in)

- `Channel` – Selected channel.

Parameters (out)

None

Return value

- `E_NOT_OK` – DET error occurred.
- `E_OK` – Otherwise.

DET errors

- `LIN_E_UNINIT` – LIN driver not initialized.
- `LIN_E_INVALID_CHANNEL` – Invalid channel given.

DEM errors

None

Description

Sends a LIN “go-to-sleep” command on the selected channel.

7.4.5 Lin_Wakeup

Syntax

```
Std_ReturnType Lin_Wakeup
(
  uint8 Channel
)
```

Service ID

0x7

Sync/Async

Asynchronous

Reentrancy

Non-reentrant

Parameters (in)

- `Channel` – Selected channel.

Parameters (out)

None

Return value

- `E_NOT_OK` – DET error occurred.
- `E_OK` – Otherwise.

DET errors

- `LIN_E_UNINIT` – LIN driver is not initialized.
- `LIN_E_INVALID_CHANNEL` – Invalid channel is given.
- `LIN_E_STATE_TRANSITION` – Driver is not in Sleep mode.

DEM errors

None

Description

Sends a wakeup signal on the selected channel.

7.4.6 Lin_GetStatus

Syntax

```
Lin_StatusType Lin_GetStatus
(
  uint8 Channel,
  uint8** Lin_SduPtr
)
```

Service ID

0x8

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

- `Channel` – Selected channel.

Parameters (out)

- `Lin_SduPtr` – Pointer to pointer to the data buffer.

Return value

- `LIN_NOT_OK` – DET error occurred.

`Channel's status` – Otherwise. (See **Table 2**)

DET errors

- `LIN_E_UNINIT` – LIN driver is not initialized.
- `LIN_E_INVALID_CHANNEL` – Invalid channel is given.
- `LIN_E_PARAM_POINTER` – NULL pointer is given.

DEM errors

None

Description

Returns the current status of the selected channel. If a reception has been completed, the data received is copied into the buffer denoted by `Lin_SduPtr`.

7.4.7 Lin_GoToSleepInternal

Syntax

```
Std_ReturnType Lin_GoToSleepInternal
(
  uint8 Channel
)
```

Service ID

0x9

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

- `Channel` – Selected channel.

Parameters (out)

None

Return value

- `E_NOT_OK` – DET error occurred.
- `E_OK` – Otherwise.

DET errors

- `LIN_E_UNINIT` – LIN driver is not initialized.
- `LIN_E_INVALID_CHANNEL` – Invalid channel is given.

DEM errors

None

Description

Sets the selected channel into Sleep mode without issuing a “go-to-sleep” command on the bus.

7.4.8 Lin_CheckWakeup

Syntax

```
Std_ReturnType Lin_CheckWakeup
(
  uint8 Channel
)
```

Service ID

0xA

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

- `Channel` – Selected channel.

Parameters (out)

None

Return value

- `E_NOT_OK` – DET error occurred.
- `E_OK` – Otherwise.

DET errors

- `LIN_E_UNINIT` – LIN driver is not initialized.
- `LIN_E_INVALID_CHANNEL` – Invalid channel is given.

DEM errors

None

Description

Checks if a wakeup has occurred on the addressed LIN channel. When a wakeup event on the addressed LIN channel is detected, this function notifies the EcuM and LIN interface immediately via the callback functions `EcuM_SetWakeupEvent()` and `LinIf_WakeupConfirmation()`.

7.4.9 Lin_WakeupInternal

Syntax

```
Std_ReturnType Lin_WakeupInternal
(
  uint8 Channel
)
```

Service ID

0xb

Sync/Async

Asynchronous

Reentrancy

Non-reentrant

Parameters (in)

- `Channel` – Selected channel.

Parameters (out)

None

Return value

- `E_NOT_OK` – DET error occurred.
- `E_OK` – Otherwise.

DET errors

- `LIN_E_UNINIT` – LIN driver is not initialized.
- `LIN_E_INVALID_CHANNEL` – Invalid channel given.
- `LIN_E_STATE_TRANSITION` – Driver is not in Sleep mode.

DEM errors

None

Description

Wakes up without sending a wakeup signal on the bus.

7.4.10 Lin_DeInit

Syntax

```
void Lin_DeInit
(
  void
)
```

Service ID

0xc

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (out)

None

Return value

None

DET errors

- `LIN_E_STATE_TRANSITION` – LIN driver is already de-initialized.

DEM errors

None

Description

De-initializes the LIN driver and the configured LIN channels.

7.5 Required callback functions

7.5.1 DET

If default error detection is enabled, the LIN driver uses the following callback function provided by DET. If you do not use DET, you must implement this function within your application.

7.5.1.1 Det_ReportError

Syntax

```
Std_ReturnType Det_ReportError
(
    uint16 ModuleId,
    uint8 InstanceId,
    uint8 ApiId,
    uint8 ErrorId
)
```

Reentrancy

Reentrant

Parameters (in)

- `ModuleId` – Module ID of the calling module.
- `InstanceId` – Instance ID of the calling module.
- `ApiId` – ID of the API service that calls this function.
- `ErrorId` – ID of the detected development error.

Return value

Returns `E_OK` (is required for services) always.

Description

Service for reporting development errors.

7.5.2 EcuM

If wakeup support by slave is enabled for a channel (i.e., if `LinChannelWakeupSupport` is TRUE), the LIN driver uses the following EcuM functions to notify detected wakeup by slave events (see *Specification of ECU state manager* **[4]** for more information).

7.5.2.1 EcuM_SetWakeupEvent

Syntax

```
void EcuM_SetWakeupEvent
(
   EcuM_WakeupSourceType sources
)
```

Reentrancy

Non-Reentrant

Parameters (in)

- `sources` – Bitfield of wakeup events.

Return value

None

Description

Service for notifying wakeup events (called from LIN API functions).

7.5.2.2 EcuM_CheckWakeup

Syntax

```
void EcuM_CheckWakeup
(
   EcuM_WakeupSourceType wakeupsource
)
```

Reentrancy

Non-Reentrant

Parameters (in)

- `sources` – Bitfield of wakeup events.

Return value

None

Description

Service for notifying wakeup events (called from Lin ISR).

7.5.3 LIN interface

If wakeup support by slave is enabled for a channel (i.e., if `LinChannelWakeupSupport` is TRUE), the LIN driver uses the following LIN interface function to notify detected wakeup by slave events (see *Specification of LIN interface* **[3]** for more information).

7.5.3.1 LinIf_WakeupConfirmation

Syntax

```
void LinIf_WakeupConfirmation
(
   EcuM_WakeupSourceType wakeupsource
)
```

Reentrancy

Non-Reentrant

Parameters (in)

- `wakeupsource` – Bitfield of wakeup events.

Return value

None

Description

Service for notifying wakeup events (called from LIN API functions).

7.5.4 Callout functions

7.5.4.1 Error callout API

The AUTOSAR LIN module requires an error callout handler. Each error is reported to this handler; error checking cannot be switched OFF. The name of the function to be called can be configured by `LinErrorCalloutFunction` parameter.

Syntax

```
void Error_Handler_Name
(
    uint16 ModuleId,
    uint8 InstanceId,
    uint8 ApiId,
    uint8 ErrorId
)
```

Reentrancy

Reentrant

Parameters (in)

- `ModuleId` – Module ID of the calling module.
- `InstanceId` – Instance ID of the calling module.
- `ApiId` – ID of the API service that calls this function.
- `ErrorId` – ID of the detected error.

Return value

None

Description

Service for reporting errors.

8 Appendix B – Access register table

8.1 LIN

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CTL0	31:0	Word (32 bits)	0x00000001	Channel disable	Lin_Init, Lin_Wakeup, Lin_GoToSleep, Lin_SendFrame, Lin_WakeupInternal, Lin_GoToSleepInternal, Lin_GetStatus, Lin_IsrWrapper_LINxx_TxRx_Cat1, Lin_IsrWrapper_LINxx_TxRx_Cat2	0x09000010 (AUTO_EN[4], MODE[24], BIT_ERROR_IGNORE[27])	0x00000000
			0x80000001 | (break delimiter length)<<8 | (break length)<<16 | (rx noise filter enable)<<30	Channel enable	Lin_Init, Lin_GoToSleep, Lin_SendFrame, Lin_Wakeup, Lin_WakeupInternal, Lin_IsrWrapper_LINxx_TxRx_Cat1(detect Wakeup), Lin_IsrWrapper_LINxx_TxRx_Cat2(detect Wakeup)		
			0x80000001 | (wakeup length)<<16	Channel enable for wakeup detection	Lin_GoToSleepInternal, Lin_GetStatus(after Lin_GoToSleep), Lin_IsrWrapper_LINxx_TxRx_Cat1(detect GoToSleep complete), Lin_IsrWrapper_LINxx_TxRx_Cat2(detect GoToSleep complete)		
			0x80040101	Channel enable for Tx wakeup	Lin_Wakeup		
			0x400C0101	Channel disable for de-initialization	Lin_DeInit		

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CTL1	31:0	Word (32 bits)	0x00000100	Initialization	Lin_Init, Lin_DeInit	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
			0x00000000 | (data length)<<0	Set classic checksum, Set Tx data	Lin_GoToSleep		
			0x00000100 | (data length)<<0	Set enhanced checksum, Set Tx data	Lin_SendFrame		
STATUS	31:0	Word (32 bits)	-	Status	Read only	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
CMD	31:0	Word (32 bits)	0x00000000	Clear command	Lin_Init, Lin_DeInit	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
			0x00000001	Set Tx header	Lin_SendFrame		
			0x00000001|0x00000002	Set Tx header, Set Tx response	Lin_GoToSleep, Lin_SendFrame		
			0x00000001|0x00000200	Set Tx header, Set Rx response	Lin_SendFrame		
			0x00000004	Set Tx wakeup	Lin_Wakeup		
TX_RX_STATUS	31:0	Word (32 bits)	0x04000000	Clear Tx/Rx status	Lin_Init, Lin_DeInit, Lin_GoToSleep, Lin_SendFrame, Lin_Wakeup, Lin_WakeupInternal , Lin_GoToSleepInternal, Lin_GetStatus, Lin_IsrWrapper_LINxx_TxRx_Cat1, Lin_IsrWrapper_LINxx_TxRx_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
PID_CHECKSUM	31:0	Word (32 bits)	0x00000000	Clear Tx PID	Lin_Init, Lin_DeInit	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
			0x00000000 | (PID)<<0	Set Tx PID	Lin_GoToSleep, Lin_SendFrame		
DATA0	31:0	Word (32 bits)	0x00000000	Clear data 0	Lin_Init, Lin_DeInit	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
			0x00000000 | send data[0-3]<<0	Set data 0	Lin_GoToSleep, Lin_SendFrame		
DATA1	31:0	Word (32 bits)	0x00000000	Clear data 1	Lin_Init, Lin_DeInit	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
			0x00000000 | send data[4-7]<<0	Set data 1	Lin_GoToSleep, Lin_SendFrame		
INTR	31:0	Word (32 bits)	0xFFFFFFFF	Clear all interrupt flag to '0'	Lin_Init, Lin_DeInit, Lin_GoToSleep, Lin_SendFrame, Lin_Wakeup, Lin_WakeupInternal, Lin_GoToSleepInternal, Lin_GetStatus, Lin_IsrWrapper_LINxx_TxRx_Cat1, Lin_IsrWrapper_LINxx_TxRx_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
INTR_SET	31:0	Word (32 bits)	-	Interrupt set	Do not use	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
INTR_MASK	31:0	Word (32 bits)	0x00000000	Clear interrupt mask to '0'	Lin_Init, Lin_DeInit, Lin_GoToSleep, Lin_SendFrame, Lin_Wakeup, Lin_WakeupInternal, Lin_IsrWrapper_LINxx_TxRx_Cat1, Lin_IsrWrapper_LINxx_TxRx_Cat2	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
			0x00000400	Set wakeup detect interrupt mask	Lin_IsrWrapper_LINxx_TxRx_Cat1(detect GoToSleep complete), Lin_IsrWrapper_LINxx_TxRx_Cat2(detect GoToSleep complete), Lin_GoToSleepInternal, Lin_GetStatus (after Lin_GoToSleep)		
			0x00030002	Set Tx complete interrupt mask for GoToSleep	Lin_GoToSleep		
INTR_MASKED	31:0	Word (32 bits)	-	Interrupt masked	Read only	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

Revision history

Revision	Issue date	Description of change
**	2018-06-28	New Spec.
*A	2018-09-20	Added some document in Hardware Documentation. Deleted the datasheet in Hardware Documentation. Added LinInstance configuration in following sections. 4.3.2 Container LinChannel
*B	2018-12-17	Changed LinChannelBreakLength configuration in following sections. 4.3.2 Container LinChannel
*C	2019-06-11	Updated hardware documentation information.
*D	2020-09-05	Changed a memmap file include folder in section "Memory Mapping".
*E	2020-11-19	MOVED TO INFINEON TEMPLATE.
*F	2021-08-19	Added a note in **6.3 Interrupts**
*G	2021-12-07	Updated to the latest branding guidelines

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2021-12-07
Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Go to www.cypress.com/support

Document reference
002-23396 Rev. *G

IMPORTANT NOTICE
The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”) .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer’s technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies’ products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.