MICROSAR Ethernet Transceiver Driver

Technical Reference

Vector Ethernet Transceiver Driver

Version 2.1.0

Authors	David Röder
Status	Released

Document Information

History

Author	Date	Version	Remarks
visdrr	2017-04-04	1.00.00	Initial creation
visdrr	2017-07-20	1.01.00	Reworked auto negotiation section
visdrr	2018-01-30	2.00.00	SafeBSW TASK-63385: Update technical reference: New file structure, review integration
visdrr	2018-10-15	2.01.00	STORYC-6117: Provide IEEE 802.3 clause 45 access over clause 22 registers

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_EthernetTransceiver.pdf	4.2.1
[2]	AUTOSAR	AUTOSAR_SWS_DET.pdf	3.3.0
[3]	AUTOSAR	AUTOSAR_SWS_DEM.pdf	5.0.0
[4]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[5]	Vector	TechnicalReference_EthTrcv_<Driver>.pdf	see delivery
[6]	Vector	TechnicalReference_Asr4Rtm.pdf	1.01.01
[7]	IEEE Computer Society	Standard for Ethernet – Amendment 1: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100BASE-T1) 802.3bw-2015	802.3bw-2015

1.1 Scope of the Document

This technical reference describes the aspects of the Ethernet Transceiver Driver basis software that are common for all supported derivatives, thus it is called ‘Core Technical Reference’. It is complemented by the hardware dependent lower layer technical reference [5] which is also contained in your delivery. Please refer to your Release Notes to get a detailed description of the platform (host, compiler, controller and transceiver hardware) your Vector Ethernet Bundle has been configured for.

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

2 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
1.00.xx	Initial creation
1.01.xx	FEAT-2234: Ethernet Link Quality Monitoring
1.02.xx	Support of Marvell 88Q1010 and 88Q211x transceivers
1.03.xx	ESCAN00095946: Split negotiation settings of Master/Slave and Speed mode
2.00.xx	Proc. 3.0 Release of DrvTrans__coreEthAsr
3.00.xx	FEAT-2877,S-464 [SAFE] SafeBSW for EthTrcv (ASIL B)
3.01.xx	Proc. 3.0 Release for 88Q2112
3.02.xx	STORYC-6117: IEEE 802.3 clause 45 access over clause 22 registers Support & Proc 3.0 Release for Kd10xx

Table 2-1 Component history

3 Introduction

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

Supported AUTOSAR Release*	4.1.x	
Supported Configuration Variants:	pre-compile	
Vendor ID:	ETHTRCV_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	ETHTRCV_MODULE_ID	73 decimal (according to ref. [4])

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

The EthTrcv (Ethernet Transceiver Driver) provides hardware independent access to control connected transceivers in a generic way. It offers the functionality to control the mode of operation of connected transceivers as well as to determine their current state, e.g. if events like link status change or bus errors happened.

The transceiver itself is a hardware device, which mainly transforms the logical I/0 signals of the Ethernet Controller to the bus compliant electrical levels, currents and timings.

The aspects discussed in this technical reference address all hardware independent parts of the EthTrcv while the derivative specific aspects are discussed in [5].

3.1 Architecture Overview

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

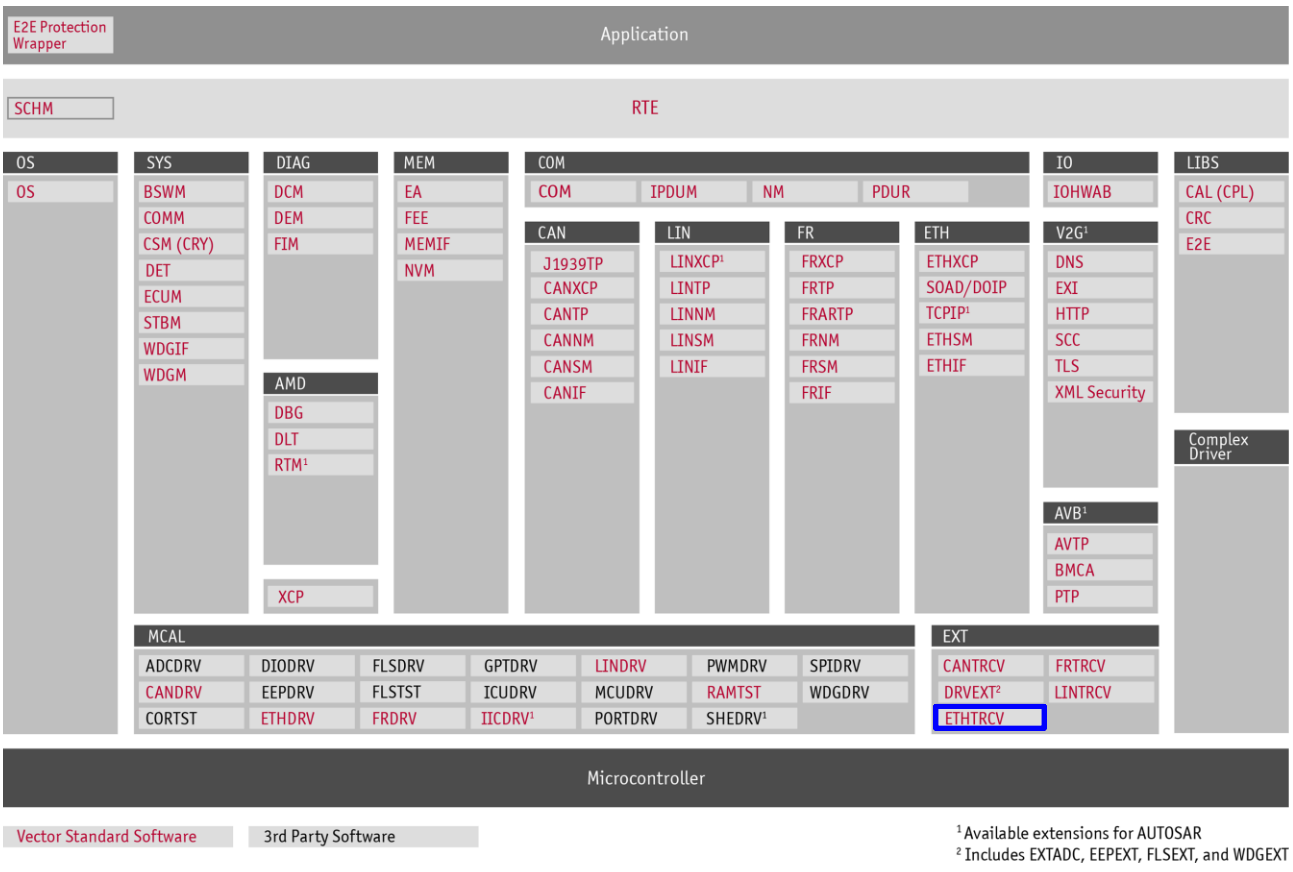


Figure 3-1 AUTOSAR 4.x Architecture Overview

Figure Figure 3-1 shows the interfaces to adjacent modules of the EthTrcv. These interfaces are described in chapter 6.

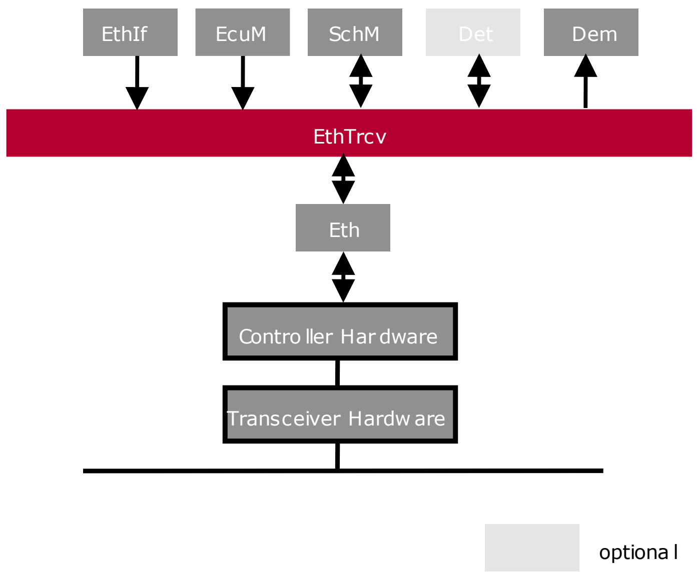


Figure 3-2 Interfaces to adjacent modules of the EthTrcv

Note
The Transceiver Hardware is not directly accessible but via the Controller Hardware. Thus the EthTrcv does use the Ethernet Driver or the Ethernet Switch Driver to access the transceiver hardware.

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE.

4 Functional Description

4.1 Features

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

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

> Table 4-1 Supported AUTOSAR standard conform features

> Table 4-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further EthTrcv functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 4-3 Features provided beyond the AUTOSAR standard

Some of the features described in the following chapters are not or not completely supported on all supported transceiver hardwares. Please refer to the hardware dependent technical reference [5], which is also contained in your delivery, if and to which extent the following features are supported.

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Initialization of the Ethernet Transceiver
Setting and getting the Transceiver Mode
Restart auto negotiation
Get link state
Get baud rate
Get duplex mode
Set test mode
Set loopback mode
Get signal quality
Set TX mode
Get cable diagnostics result
Get PHY identifier
Retrieve version info
Report development errors to DET
Report production errors to DEM
(See [5] for additional derivative specific features)

Table 4-1 Supported AUTOSAR standard conform features

4.1.1 Deviations

The following features specified in [1] are not supported:

Not Supported AUTOSAR Standard Conform Features
EthTrcv_WriteMiiIndication and EthTrcv_ReadMiiIndication for asynchronous reading and writing of the Mii interfaces are not implemented
EthTrcv_TransceiverInit ist not merged into EthTrcv_Init
Only pre-compile configuration is supported
The APIs EthTrcv_GetLinkState, EthTrcv_GetTransceiverMode and EthTrcv_SetTransceiverMode cannot be deactivated by preprocessor-switches because they are required for the Ethernet interface
(See [5] for additional derivative specific deviations)

Table 4-2 Not supported AUTOSAR standard conform features

4.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Pre-/Post-TransceiverInit User-Callouts
Wakeup support using activation line according to AUTOSAR 4.2.1
Setting and getting the Transceiver wakeup mode according to AUTOSAR 4.2.1
(See [5] for additional derivative specific Additions/Extensions)

Table 4-3 Features provided beyond the AUTOSAR standard

4.1.3 Limitations

Limitations
4.1.4 None

4.2 Initialization

4.2.1 High-Level Initialization

The Ethernet Transceiver Driver is initialized by calling the `EthTrcv_30_<Driver>_Init` service. The transceiver hardware itself is initialized by calling the `EthTrcv_30_<Driver>_TransceiverInit` service with the corresponding index for each transceiver.

Note
When using a MICROSAR SIP the initialization is done by the stack itself without any user interaction.

Caution
If startup code does not initialize the RAM and therefore some data is not set to a predefined value the Ethernet Transceiver Driver relies on (e.g. zero initialized data), the function `EthTrcv_30_<Driver>_InitMemory()` must be called during startup. This function sets the predefined values usually set by the startup code.

4.2.2 Low-Level Initialization

The Port Driver has to be configured to initialize the I/O-ports of the controller to which the transceiver is connected.

4.3 States

The transceiver should be set to a defined state during initialization by the upper layer software component (Ethernet Interface) to avoid that the initial state is undefined. These states are listed in table Table 4-4:

State	Description
`ETHTRCV_STATE_UNINIT`	The driver is uninitialized and all its state variables are undefined.
`ETHTRCV_STATE_INIT`	The driver is initialized and all its state variables have a defined value allowing the initialization of the transceiver hardware.

Table 4-4 Ethernet Transceiver Driver states

After the driver initialization the transceiver hardware can be initialized and enabled. The corresponding modes are listed in table Table 4-5:

Mode	Description
`ETHTRCV_MODE_DOWN`	The Ethernet transceiver is turned off and cannot be used for communication.
`ETHTRCV_MODE_ACTIVE`	The Ethernet transceiver is turned on and can be used for communication.

Table 4-5 Ethernet transceiver modes

Most APIs of the driver are only allowed to be called in specific states or modes of the Ethernet transceiver. For details see the service descriptions in 6.

4.4 Main functions

In case wakeup support is configured to `ETHTRCV_WAKEUP_BY_POLLING` the Ethernet Transceiver driver has a main function that must be called periodically.

EthTrcv_30_<Driver>_MainFunction()		
Existence	Timing	Description
Allways	Scheduled according to the period provided by the configuration parameter `EthTrcvMainFunctionPeriod`.	Processes inspection of wake-up reasons if wake-up is enabled.

Table 4-6 EthTrcv_Main function description

For derivatives that need a specific link-up procedure with delays or timeouts the additional main function for link handling is provided and must be called periodically. Please refer to [5].

EthTrcv_30_<Driver>_MainFunctionLinkHandling()		
Existence	Timing	Description
Allways	Scheduled according to the period provided by the configuration parameter `EthTrcvMainFunctionLinkHandlingPeriod`.	Processes timing and state machine of link startup if necessary.

Table 4-7 EthTrcv_MainFunctionLinkHandling description

4.5 Error Handling

4.5.1 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 `ETHTRCV_<Driver>_DEV_ERROR_REPORT == 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 EthTrcv ID is 73.

The reported service IDs identify the services which are described in section 6.2. The following table presents the service IDs and the related services. In the code, every ID define has the additional prefix “ETHTRCV_30_<Driver>_” and every service the corresponding prefix “EthTrcv_30_<Driver>_” which are omitted due to better readability in table Table 4-8.

Service ID define	Service
SID_INIT	Init()
SID_TRANSCEIVER_INIT	TransceiverInit()
SID_SET_TRANSCEIVER_MODE	SetTransceiverMode()
SID_GET_TRANSCEIVER_MODE	GetTransceiverMode()
SID_START_AUTO_NEG	StartAutoNegotiation()
SID_GET_LINK_STATE	GetLinkState()
SID_GET_BAUD_RATE	GetBaudRate()
SID_GET_DUPLEX_MODE	GetDuplexMode()
SID_GET_VERSION_INFO	GetVersionInfo()
SID_MAIN_FUNCTION	MainFunction()
SID_MAIN_FUNCTION_LINK_HANDLING	MainFunctionLinkHandling()
SID_SET_TRANSCEIVER_WAKEUP_MODE	SetTransceiverWakeupMode()
SID_GET_TRANSCEIVER_WAKEUP_MODE	GetTransceiverWakeupMode()
SID_CHECK_WAKEUP	CheckWakeup()
SID_SET_PHY_TEST_MODE	SetPhyTestMode()
SID_SET_PHY_LOOPBACK_MODE	SetPhyLoopbackMode()
SID_GET_PHY_SIGNAL_QUALITY	GetPhySignalQuality()
SID_SET_PHY_TX_MODE	SetPhyTxMode()
SID_GET_CABLE_DIAGNOSTICS_RESULT	GetCableDiagnosticsResult()
SID_GET_PHY_IDENTIFIER	GetPhyIdentifier()

Table 4-8 Service IDs

The errors reported to DET are described in the following table. They are prefixed with “ETHTRCV_30_<Driver>_” in the code, which is again omitted in table Table 4-9.

Error Code	Description
E_NO_ERROR	No error occurred
E_INV_TRCV_IDX	API service has been called with invalid transceiver index
E_NOT_INITIALIZED	API service used without required initialization
E_INV_POINTER	API service used with invalid pointer parameter
E_INV_PARAM	API service used with invalid value for parameter
E_NOT_SUPPORTED	API service not supported by the hardware

Table 4-9 Errors reported to DET

4.5.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3], if production error reporting is enabled (i.e. pre-compile parameter `ETHTRCV_30_<Driver>_DEM_ERROR_DETECT == STD_ON`).

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
ETHTRCV_E_ACCESS	Accessing the transceiver hardware failed

Table 4-10 Errors reported to DEM

4.6 PHY Diagnostic Modes

The Ethernet transceiver driver provides several PHY diagnostic modes which can be enabled during runtime. The possible modes are discussed in the following sections and are categorized in PHY test modes, loopback modes, transmitter modes, and cable diagnostics.

Caution
The PHY diagnostic modes can only be accessed when the Ethernet transceiver driver is in mode ACTIVE. This mode is reached when the initialization of driver and hardware is completed and the transceiver has been activated by calling `EthTrcv_30_<Driver>_SetTransceiverMode()` with the mode parameter set to ACTIVE. If a diagnostic mode is enabled, the transceiver is reported to be in mode DOWN for other modules and the link state is reported DOWN, even if the hardware reports an active link. Thus normal communication can only be continued by disabling the currently active diagnostic mode. Afterwards the Ethernet transceiver driver returns to mode ACTIVE automatically.

Note
The Ethernet transceiver driver only supports a certain diagnostic mode if it is supported by the transceiver hardware. The availability of a certain diagnostic mode can be seen in the derivative specific part of the technical reference [5].

4.6.1 PHY Test Modes

By using the API `EthTrcv_30_<Driver>_SetPhyTestMode()`, the transceiver driver enables the chosen PHY test mode based on the parameter `TestMode (EthTrcv_PhyTestModeType)`. The test modes which are available on the specific derivative are listed in the derivative specific technical reference [5]. The following Table 4-11 shows the generally available test modes and a short description.

TestMode value	Description
`ETHTRCV_PHYTESTMODE_NONE`	Return to normal operation
`ETHTRCV_PHYTESTMODE_1`	Test the transmitter droop
`ETHTRCV_PHYTESTMODE_2`	Test the master timing jitter
`ETHTRCV_PHYTESTMODE_3`	Test the slave timing jitter
`ETHTRCV_PHYTESTMODE_4`	Test the transmitter distortion
`ETHTRCV_PHYTESTMODE_5`	Test the power spectral density

Table 4-11 PHY Test Modes

4.6.2 PHY Loopback Modes

By using the API `EthTrcv_30_<Driver>_SetPhyLoopbackMode()`, the transceiver driver enables the chosen PHY loopback mode based on the parameter `LoopbackMode (EthTrcv_PhyLoopbackModeType)`. The loopback modes which are available on the specific derivative are listed in the derivative specific technical reference [5]. The following Table 4-12 shows the generally available loopback modes and a short description according to [7]. The difference between the loopback modes is based on the point where in the PHY the sent frames are looped back to the sender.

LoopbackMode value	Description
`ETHTRCV_PHYLOOPBACK_NONE`	Return to normal operation
`ETHTRCV_PHYLOOPBACK_INTERNAL`	The PCS (Physical Coding Sublayer) accepts data on the transmit path from the MII and returns it on the receive path to the MII. Additionally, the PHY receive circuitry is isolated from the network medium and no data is transmitted to the network medium

`ETHTRCV_PHYLOOPBACK_EXTERNAL`	The PMA (Physical Medium Attachment) receive function utilizes the echo signals from the unterminated MDI (Medium Dependent Interface) and decodes these signals to pass the data back to the MII Receive interface
`ETHTRCV_PHYLOOPBACK_REMOTE`	The packets that arrive at the MDI are looped through the PMA and PCS Receive and Transmit functions back to the MDI. It depends on the used hardware derivative, if the data is also made available on the connected MII

Table 4-12 PHY Loopback Modes

4.6.3 PHY TX Modes

By using the API `EthTrcv_30_<Driver>_SetPhyTxMode()`, the transceiver driver enables the chosen PHY loopback mode based on the parameter `TxMode (EthTrcv_PhyTxModeType)`. The TX modes which are available on the specific derivative are listed in the derivative specific technical reference [5]. The following Table 4-13 shows the generally available TX modes and a short description.

TxMode value	Description
`ETHTRCV_PHYTXMODE_NORMAL`	Return to normal operation
`ETHTRCV_PHYTXMODE_TX_OFF`	Disable the transmitter
`ETHTRCV_PHYTXMODE_SCRAMBLER_OFF`	Disable the scrambler

Table 4-13 PHY Tx Modes

4.6.4 Cable Diagnostics

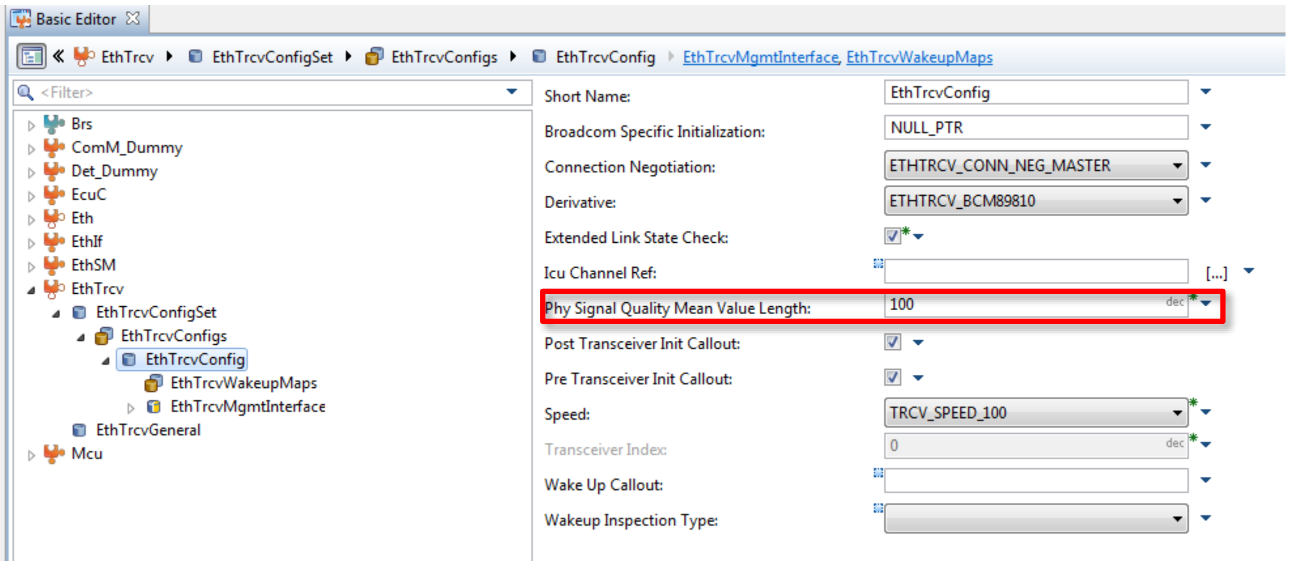
If the underlying transceiver hardware is able to perform cable diagnostics (see [5]), the API `EthTrcv_30_<Driver>_GetCableDiagnosticsResult()` can be used to retrieve the result of the diagnostic measurement. During the runtime of the test the Ethernet transceiver driver reports to be in mode DOWN. After completion of the diagnostics the mode is automatically reset to ACTIVE. The results that are possible can be seen in Table 4-14.

CableDiagResultPtr Value	Description
`ETHTRCV_CABLEDIAG_OK`	Cable diagnostics result is ok, no short or open detected
`ETHTRCV_CABLEDIAG_ERROR`	The cable diagnostics result could not be retrieved
`ETHTRCV_CABLEDIAG_SHORT`	A short circuit was detected during cable diagnostics
`ETHTRCV_CABLEDIAG_OPEN`	A open circuit was detected during cable diagnostics

Table 4-14 Possible results of EthTrcv_30_<Driver>_GetCableDiagnosticsResult()

4.7 Get PHY Signal Quality

If the underlying transceiver hardware is able to measure the signal quality (see [5]), the API `EthTrcv_30_<Driver>_GetPhySignalQuality()` can be used to retrieve an averaged value of the signal quality in percent, where 100% denotes the best signal quality. The number of measurements the driver performs the averaging over can be configured. Figure 4-1 shows an example, if 100 measured values should be used to calculate the average.



Short Name:	EthTrcvConfig
Broadcom Specific Initialization:	NULL_PTR
Connection Negotiation:	ETHTRCV_CONN_NEG_MASTER
Derivative:	ETHTRCV_BCM89810
Extended Link State Check:	✓
Icu Channel Ref:	
Phy Signal Quality Mean Value Length:	100
Post Transceiver Init Callout:	✓
Pre Transceiver Init Callout:	✓
Speed:	TRCV_SPEED_100
Transceiver Index:	0
Wake Up Callout:	
Wakeup Inspection Type:	

Figure 4-1 PHY Signal Quality Mean Value Length

4.8 PHY Identifier

The API `EthTrcv_30_<Driver>_GetPhyIdentifier()` can be used to retrieve the content of the two identifier registers of the PHY. The following Table 4-15 shows how the register contents are returned.

Returned Variable	Description
`OrgUniqueIdPtr`	Bits [31:22]: 0 Bits [21:6]: The content of the PHY identifier register 1 Bits [5:0]: The 6 MSB of the PHY identifier register 2
`ModelNrPtr`	Bits [8:6]: 0 Bits [5:0]: Bits [9:4] of the PHY identifier register 2
`RevisionNrPtr`	Bits [8:4]: 0 Bits [3:0]: Bits [3:0] of the PHY identifier register 2

Table 4-15 Results of EthTrcv_30_<Driver>_GetPhyIdentifier()

4.9 Wakeup Support

The Ethernet Transceiver Driver supports wakeup detection during the following states of operation:

- Initialization
- Runtime

For wakeup during runtime the following two methods can be utilized.

- Wakeup by Polling
- Wakeup by Interrupt

The detection for wakeup events during initialization is always performed once the wakeup support is enabled.

This section describes how the wakeup detection is processed during initialization or runtime and, in case of detection during runtime, how the methods “Wakeup by Polling“ and “Wakeup by Interrupt“ behave.

Note
In all cases the internal API `EthTrcv_30_<Driver>_InspectWakeupReasons()`, which is located in the `EthTrcv_30_<Driver>_Wakeup.c` source file, is utilized for the wakeup detection. Dependent on the configuration the integrator may be required to provide additional callout code within this function to perform proper wakeup detection. For more information please refer to 5.4.3 and 7.5.

4.9.1 Wakeup Detection during Initialization

The detection of wakeup events during initialization is performed by the Transceiver driver within the `EthTrcv_30_<Driver>_Init()` function. This function is called either during power on of the ECU, after a reset of the ECU or after the ECU transitions out of sleep mode.

During the initialization phase the driver checks if a wakeup event has occurred. If a wakeup event has occurred it is matched to a driver internal representation called wakeup reason. This reason can be mapped to an EcuM Wakeup Source by configuration. If such a mapping for detection of a wakeup event exists and a corresponding wakeup reason was set the transceiver driver informs the EcuM about the detection of a wakeup event by utilizing the corresponding EcuM Wakeup Source. Otherwise, if there is no wakeup event detected within the function call, the default wakeup reason `ETHTRCV_WUR_RESET` is used. What applies in this case too is that a mapping between the `ETHTRCV_WUR_RESET` to an EcuM Wakeup Source must exist to report the wakeup event to the EcuM.

4.9.2 Wakeup Detection during Runtime

During runtime the wakeup detection can be performed by two methods which are described in this section.

Wakeup by Polling

The transceiver driver will cyclically check for the occurrence of a wakeup event with each call of its main function. The wakeup detection is performed similar to the one performed during initialization. If an event is detected and a mapping for the corresponding wakeup reason to an EcuM Wakeup Source exists the EcuM is informed about a wakeup event for the corresponding Wakeup Source.

Wakeup by Interrupt

Instead of cyclically checking for wakeup event like in “Wakeup by Polling” the transceiver driver only performs a wakeup detection if the function `EthTrcv_30_<Driver>_CheckWakeup()` is called by an asynchronous event (e.g. Interrupt). This call is initiated by the Icu module, which listens for events on GPIO pins, that are related to signal lines being able to drive a wakeup signal. This event is propagated by user code (Icu Notification) to the EcuM by calling the `EcuM_CheckWakeup()` API. This API again contains integration code which calls the function `EthIf_CheckWakeup()` to eventually redirect the call and therefore the wakeup detection to the `EthTrcv_30_<Driver>_CheckWakeup()` API of the transceiver driver. The check for wake-up events is performed similar to the cases initialization and polling..

4.10 Runtime Measurement

Runtime measurement allows to measure the processing time of specific functions of the driver. Therefore so called measurement points are used. These points are automatically created in the Rtm module, if the module is contained in the configuration.

This description only shows the Ethernet Transceiver Driver specific aspect of the runtime measurement. For a more detailed description of the runtime measurement feature and the Rtm module providing the service see [6].

The Ethernet Transceiver Driver provides the following measurement points:

Measurement Point	Description
EthTrcv_30_<Driver>_TransceiverInit	Measures the runtime of the `EthTrcv_30_<Driver>_TransceiverInit()` function. The API is used for transceiver initialization during runtime.

Table 4-16 Runtime Measurement points

5 Integration

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

5.1 Embedded Implementation

The delivery of the EthTrcv consists out of the following files. If a driver needs additional source files the list of the additions can be found in [5].

File Name	Description	Integration Tasks
`EthTrcv_30_<Driver>.c`	Static source file that contains public API implementation.	-
`EthTrcv_30_<Driver>.h`	Static header file that contains public API declaration.	-
`EthTrcv_30_<Driver>_Wakeup.c`	Static source file that contains public wake-up related API implementation.	-
`EthTrcv_30_<Driver>_Wakeup.h`	Static header file that contains public wake-up related API declaration.	-
`EthTrcv_30_<Driver>_Hw.c`	Static source file that contains public hardware access related API implementation.	-
`EthTrcv_30_<Driver>_Hw.h`	Static header file that contains public hardware access related API declaration.	-
`EthTrcv_30_<Driver>_Int.h`	Static header file that contains internal general declaration and definitions.	-
`EthTrcv_30_<Driver>_IntDef.h`	Static header file that contains internal general macro declarations.	-
`EthTrcv_30_<Driver>_Wakeup_Int.h`	Static header file that contains internal wake-up related declarations.	-
`EthTrcv_30_<Driver>_Wakeup_UsrIfc.h`	Static header file that contains declarations providing a user-interface to detect wake-up reasons during wake-up inspection.	-
`EthTrcv_30_<Driver>_Hw_Int.h`	Static header file that contains hardware access related internal declaration, definitions and macro declarations.	-
`EthTrcv_30_<Driver>_LL.h`	Static header file that contains hardware dependent declaration, defines and implementation.	-
`EthTrcv_30_<Driver>_LL_Regs.h`	Static header file that contains driver specific register declaration.	-
`EthTrcv_30_<Driver>_Types.h`	Static header file that contains public types.	-
`EthTrcv_30_<Driver>_Cfg.h`	Generated header file that contains the pre-compile time configuration	-

File Name	Description	Integration Tasks
`EthTrcv_30_<Driver>_Lcfg.c`	Generated source file that contains the link-time parameter configuration	-
`EthTrcv_30_<Driver>_Lcfg.h`	Generated header file that contains the link-time parameter declaration and config access declarations and definitions.	-
`EthTrcv_30_<Driver>_GenTypes.h`	Generated header file that contains the types of the generated data.	-

Table 5-1 List of delivered files

5.2 Exclusive Areas

Note
When using DaVinci Configurator PRO and the respective MICROSAR stack there is no need to configure these areas manually. It is done by the tool automatically. The prefix `ETHTRCV_30_<Driver>_` is omitted to increase readability.

Exclusive Area	Description
`EXCLUSIVE_AREA_DATA`	This exclusive area ensures the state and data consistency between the transceiver hardware and software.

5.3 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 which are defined for the EthTrcv and illustrates their assignment among each other.

Memory Mapping Sections / Compiler Abstraction Definitions	ETHTRCV_CONST	ETHTRCV_VAR	ETHTRCV_CODE
`ETHTRCV_30_<Driver>_START_SEC_CONST_UNSPECIFIED`	■		
`ETHTRCV_30_<Driver>_START_SEC_CONST_32BIT`	■		
`ETHTRCV_30_<Driver>_START_SEC_CONST_16BIT`	■		
`ETHTRCV_30_<Driver>_START_SEC_CONST_8BIT`	■		

ETHTRCV_30_<Driver>_START_SEC_VAR_NOINIT_UNSPECIFIED		■	
ETHTRCV_30_<Driver>_START_SEC_VAR_NOINIT_32BIT		■	
ETHTRCV_30_<Driver>_START_SEC_VAR_NOINIT_16BIT		■	
ETHTRCV_30_<Driver>_START_SEC_VAR_NOINIT_8BIT		■	
ETHTRCV_30_<Driver>_START_SEC_CODE			■

Table 5-2 Compiler abstraction and memory mapping

5.4 Wakeup Support

The feature Wakeup needs adaptions in configuration of multiple modules and also User-/Integration-Code must be provided by the integrator to realize the proper execution of the wakeup procedure.

This section describes which User-/Integration-Code must be provided for the related modules. The configuration adaptions, which must be performed within the Ethernet Transceiver driver configuration, for the Wakeup feature are described in 7.5 and the different modes of operation are introduced in 4.9.

5.4.1 Icu Notification

The Icu Channel utilized to detect a wakeup event issued on a signal line must be extended by a call to a Icu Notification function. The notification is a function implemented by the integrator. For proper operation this function shall implement a call to `EcuM_CheckWakeup()` passing the EcuM Wakeup Source related to the Activation Line.

The following example shows a sample source code used for detecting a wakeup event on an activation line.

Note
This integration code must only be provided if the wakeup support of the Ethernet transceiver driver is configured to `ETHTRCV_WAKEUP_BY_INTERRUPT`.

Example
The following code snippet provides an example for an Icu Notification implementation.

```
void EthernetWakeup_IcuActivationLineNotification( void )
{
  /* Trigger wakeup detection by EcuM with the corresponding
   * EcuM Wakeup Source related to the Activation Line. */

EcuM_CheckWakeup(EcuMConf_EcuMWakeupSource_WakeupSource_Activat
ionLine);
}
```

5.4.2 EcuM Check Wakeup

The EcuM generates the source file `EcuM_Callout_Stubs.c`. This file contains multiple EcuM functions which can be extended by an integrator. In case of wakeup detection for Ethernet the function `EcuM_CheckWakeup()` must be extended by an integration code redirecting the call to the `EthIf_CheckWakeup()` API if it is called with the EcuM Wakeup Source related to the signal line observed by the Icu Channel.

The implementation must be done within the comment block with the starting comment containing the tag `<USERBLOCK …>` and the closing comment containing the tag `</USERBLOCK>`.

The following example shows sample source code used for detecting a wakeup event on an activation line.

Note
This integration code must only be provided if the wakeup support of the Ethernet transceiver driver is configured to `ETHTRCV_WAKEUP_BY_INTERRUPT`.

Example

The following code snippet provides an example for the extension of the `EcuM_CheckWakeup()` API by a call to `EthIf_CheckWakeup()` for the corresponding EcuM Wakeup Source.

```
FUNC(void, ECUM_CODE) EcuM_CheckWakeup(EcuM_WakeupSourceType
wakeupSource)
/* PRQA S 1503 */ /* MD_MSR_14.1 */
{
#if (ECUM_CHECKWAKEUPTIMEOFWAKEUPSOURCELIST == STD_ON)
  /* Do not remove the following function call. It is necessary 28yclic28
   * CheckWakeup timeout mechanism */
  EcuM_StartCheckWakeup(wakeupSource);
#endif

#if (ECUM_ALARM_CLOCK_PRESENT == STD_ON)
  if((ECUM_ALARM_WKSOURCE & wakeupSource) != 0)
  {
    EcuM_AlarmCheckWakeup();
  }
#endif

/*****************************************************************************
 * DO NOT CHANGE THIS COMMENT! <USERBLOCK …> DO NOT CHANGE
THIS COMMENT!
*****************************************************************************/
#if (ECUM_USE_DUMMY_STATEMENT == STD_ON)
  /* dummy assignment to prevent compiler warnings on most of the compilers.
*/
  (void)wakeupSource; /* PRQA S 3112 */ /* MD_EcuM_3112 */
#endif
  /* Add implementation of EcuM_CheckWakeup() */
  switch(wakeupSource)
  {
  case EcuMConf_EcuMWakeupSource_WakeupSource_ActivationLine:
    /* If corresponding EcuM Wakeup Source is passed redirect the call to
     * EthIf for checkup detection. */
    EthIf_CheckWakeup(wakeupSource);
    break;
  default:
    break;
  }

  return;
/*****************************************************************************
 * DO NOT CHANGE THIS COMMENT! </USERBLOCK> DO NOT CHANGE
THIS COMMENT!
*****************************************************************************/
} /* End of EcuM_CheckWakeup() */
```

5.4.3 Transceiver Wakeup Reason Inspection by user code

The transceiver driver provides a method to detect a wakeup reason to be mapped to an EcuM Wakeup Source, if a mapping is provided by configuration. One option is to use user-defined integration code provided in a configurable callout.

Note
This integration code must only be provided if the wakeup inspection type parameter `EthTrcvConfig/EthTrcvWakeupInspectionType` for a transceiver is configured to `ETHTRCV_USER_CODE`.

The wake-up inspection callout can be used to set a transceiver wake-up reason if a corresponding wakeup event ocurred. The found wakeup reason is then correlated with an EcuM Wakeup Source if configuration provides a corresponding mapping. These reasons, defined by AUTOSAR, can be set by functions provided by the driver implementation that are made available by including their declaration with the header file "`EthTrcv_30_<Driver>_Wakeup_UserIfc.h`" into the file where the configured Wakeup Inspection Callout is implemented. Please see section 7.3 on how to configure such a user callout in general. The corresponding parameter in the configuration is called EthTrcvWakeupInspectionCallout. Table 5-3 shows an overview of the usable functions and their relation to the wakeup reason.

Wakeup Reason	Function to set reason
ETHTRCV_WUR_GENERAL	EthTrcv_30_<Driver>_SetWakeupReasonToGeneral(TrcvIdx)
ETHTRCV_WUR_BUS	EthTrcv_30_<Driver>_SetWakeupReasonToBus(TrcvIdx)
ETHTRCV_WUR_INTERNAL	EthTrcv_30_<Driver>_SetWakeupReasonToInternal(TrcvIdx)
ETHTRCV_WUR_RESET	EthTrcv_30_<Driver>_SetWakeupReasonToReset(TrcvIdx)
ETHTRCV_WUR_POWER_ON	EthTrcv_30_<Driver>_SetWakeupReasonToPowerOn(TrcvIdx)
ETHTRCV_WUR_PIN	EthTrcv_30_<Driver>_SetWakeupReasonToPin(TrcvIdx)
ETHTRCV_WUR_SYSERR	EthTrcv_30_<Driver>_SetWakeupReasonToSysErr(TrcvIdx)

Table 5-3 Wakeup Reason Functions

Caution
Further processing and propagating the wakeup event to EcuM is done by the driver itself. The user must not use wakeup related APIs of EcuM or the transceiver driver within the user code. He is only allowed to use the given functions to influence the wakeup detection.

The following example describes an implementation for general wakeup event detection. The user code evaluates if a general wakeup was detected and utilizes the provided functions to initiate a propagation of the wakeup event to the EcuM. The wake-up inspection callout can be enabled by providing an individual value for the parameter EthTrcvConfig/EthTrcvWakeUpInspectionCallout in the configuration tool. The function can then be implemented by the user, an example implementation for a callout called "WakeupInspectionCallout" is provided in the following. The inspection callout configuration itself is similar to any other callout available in the Ethernet transceiver driver and can be seen in 7.3.

Example
The following code snippet provides an example for a user code which is called by the internal wakeup reason inspection API located in `EthTrcv_30_<Driver>_Wakeup.c` source file. The file implementing the WakeupInspectionCallout has to include the declaration for the wake-up user interface.

```
#include "EthTrcv_30_<Driver>_Wakeup_UsrIfc.h"

void WakeupInspectionCallout(uint8 TrcvIdx)
{
  boolean IsWakeupEventDetected = FALSE;

  /* User defined code to check if a general wakeup event has occurred */
  DetectGeneralWakeupEvent(&IsWakeupEventDetected);

  /* Set wakeup reason only if wakeup event has occurred */
  if (IsWakeupEventDetected == TRUE )
  {
    /* Set wakeup reason with predefined functions */
    EthTrcv_30_<Driver>_SetWakeupReasonToGeneral(TrcvIdx);
  }
}
```

Note
The function “DetectGeneralWakeupEvent(boolean *result)” used in the previous example also has to be realized by the user in some way dependent on the used setup. It is not mandatory to stick to the example which is only used to clarify the idea.

6 API Description

6.1 Type Definitions

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

Type Name	C-Type	Description	Value Range
`EthTrcv_ConfigType`	struct	Transceiver configuration	`CFG_PTR` Pointer to configuration
`EthTrcv_ModeType`	uint8	Defines all possible transceiver modes	`ETHTRCV_MODE_DOWN` Transceiver inactive
			`ETHTRCV_MODE_ACTIVE` Normal operation mode
`EthTrcv_LinkStateType`	uint8	Defines all possible transceiver link states	`ETHTRCV_LINK_STATE_DOWN` Transceiver link disconnected
			`ETHTRCV_LINK_STATE_ACTIVE` Transceiver link connected
`EthTrcv_BaudRateType`	uint8	Defines all possible transceiver baud rates	`ETHTRCV_BAUD_RATE_10MBIT` 10Mbit baud rate
			`ETHTRCV_BAUD_RATE_100MBIT` 100Mbit baud rate
			`ETHTRCV_BAUD_RATE_1000MBIT` 1000Mbit baud rate
`EthTrcv_DuplexModeType`	uint8	Defines all possible transceiver duplex modes	`ETHTRCV_DUPLEX_MODE_HALF` Half duplex connection
			`ETHTRCV_DUPLEX_MODE_FULL` Full duplex connection
`EthTrcv_StateType`	uint8	Defines all possible transceiver states	`ETHTRCV_STATE_UNINIT` Ethernet Transceiver Driver not initialized
			`ETHTRCV_STATE_INIT` Ethernet Transceiver Driver initialized
			`ETHTRCV_STATE_ACTIVE` Ethernet Transceiver Driver active
			`ETHTRCV_STATE_DOWN` Ethernet Transceiver Driver down
`EthTrcv_WakeupModeType`	uint8	Defines all	`ETHTRCV_WUM_DISABLE`

Type Name	C-Type	Description	Value Range
		possible wakeup modes	Disable wakeup
			ETHTRCV_WUM_ENABLE Enable wakeup
			ETHTRCV_WUM_CLEAR Clear wakeup reasons
EthTrcv_WakeupReasonType	uint8	Defines all possible wakeup reasons	ETHTRCV_WUR_NONE No wakeup reason
			ETHTRCV_WUR_GENERAL General wakeup reason
			ETHTRCV_WUR_BUS Bus wakeup reason
			ETHTRCV_WUR_INTERNAL Internal wakeup reason
			ETHTRCV_WUR_RESET Reset wakeup reason
			ETHTRCV_WUR_POWER_ON Power on wakeup reason
			ETHTRCV_WUR_PIN Pin wakeup reason
			ETHTRCV_WUR_SYSERR System error wakeup reason
EthTrcv_ WakeupReasonFlagsType	uint8	Defines masks for all possible bit positions of the AUTOSAR wakeup reasons	ETHTRCV_WUR_GENERAL_FLAGS_MASK Mask for general wakeup reason
			ETHTRCV_WUR_BUS_FLAGS_MASK Mask for bus wakeup reason
			ETHTRCV_WUR_INTERNAL_FLAGS_MASK Mask for internal wakeup reason
			ETHTRCV_WUR_RESET_FLAGS_MASK Mask for reset wakeup reason
			ETHTRCV_WUR_POWER_ON_FLAGS_MASK Mask for power on wakeup reason
			ETHTRCV_WUR_PIN_FLAGS_MASK Mask for pin wakeup reason
			ETHTRCV_WUR_SYSERR_FLAGS_MASK Msk for system error wakeup reason
			ETHTRCV_WUR_CLEAR_MASK Mask to clear all wakeup reason flags
EthTrcv_PhysAddrType	uint8[]	Defines Ethernet physical addresses	00:00:00:00:00:00 - FF:FF:FF:FF:FF:FF Byte array containing 6 physical address bytes for Ethernet communication
EthTrcv_PhyTestModeType	uint8	Defines all possible PHY	ETHTRCV_PHYTESTMODE_NONE Mask to clear all wakeup reason

Type Name	C-Type	Description	Value Range
		test modes	flags
			ETHTRCV_PHYTESTMODE_1 PHY Test transmitter droop
			ETHTRCV_PHYTESTMODE_2 PHY Test master timing jitter
			ETHTRCV_PHYTESTMODE_3 PHY Test slave timing jitter
			ETHTRCV_PHYTESTMODE_4 PHY Test transmitter distortion
			ETHTRCV_PHYTESTMODE_5 PHY Test power spectral density
EthTrcv_ PhyLoopbackModeType	uint8	Defines all possible PHY loopback modes	ETHTRCV_PHYLOOPBACK_NONE Normal operation
			ETHTRCV_PHYLOOPBACK_INTERNAL Internal loopback
			ETHTRCV_PHYLOOPBACK_EXTERNAL External loopback
			ETHTRCV_PHYLOOPBACK_REMOTE Remote loopback
EthTrcv_PhyTxModeType	uint8	Defines all possible PHY TX modes	ETHTRCV_PHYTXMODE_NORMAL Normal operation
			ETHTRCV_PHYTXMODE_TX_OFF Transmitter disabled
			ETHTRCV_PHYTXMODE_SCRAMBLER_OFF Scrambler disabled
EthTrcv_ CableDiagResultType	uint8	Defines all possible cable diagnostics results	ETHTRCV_CABLEDIAG_OK Cable diagnostics result: OK
			ETHTRCV_CABLEDIAG_ERROR Cable diagnostics result invalid
			ETHTRCV_CABLEDIAG_SHORT Cable diagnostics result: Short circuit detected
			ETHTRCV_CABLEDIAG_OPEN Cable diagnostics result: Open circuit detected

Table 6-1 Type definitions

6.2 Services provided by EthTrcv

6.2.1 EthTrcv_30_Core_InitMemory

Prototype	
`void` **`EthTrcv_30_Core_InitMemory`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Function for *_INIT_*-variable initialization. Service to initialize module global variables at power up. This function initializes the variables in *_INIT_* sections. Used in case they are not initialized by the startup code.	
Particularities and Limitations	
Module is uninitialized.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 6-2 EthTrcv_30_Core_InitMemory

6.2.2 EthTrcv_30_Core_Init

Prototype	
`void` **`EthTrcv_30_Core_Init`** `(const EthTrcv_30_Core_ConfigType *CfgPtr)`	
Parameter	
CfgPtr [in]	Configuration structure for initializing the module
Return code	
void	none
Functional Description	
Initialization function. This function initializes the module EthTrcv_30_Core. It initializes all variables and sets the module state to initialized.	
Particularities and Limitations	
Specification of module initialization CREQ-137828 > Module is uninitialized.EthTrcv_30_Core_InitMemory has been called unless EthTrcv_30_Core_ModuleInitialized is initialized by start-up code.	
Call context	
> TASK > This function is Synchronous	

> This function is Non-Reentrant

Table 6-3 EthTrcv_30_Core_Init

6.2.3 EthTrcv_30_Core_TransceiverInit

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_TransceiverInit`** `(uint8 TrcvIdx, uint8 CfgIdx)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver
CfgIdx [in]	Configuration index
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameters or at least one of the hardware operations (read/write) has failed.
Functional Description	
Initializes an Ethernet transceiver (register configuration). This function initializes the transceiver's registers to realize its functionality according to the configuration.	
Particularities and Limitations	
Module is initialized.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 6-4 EthTrcv_30_Core_TransceiverInit

6.2.4 EthTrcv_30_Core_SetTransceiverMode

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_SetTransceiverMode`** `(uint8 TrcvIdx, EthTrcv_ModeType TrcvMode)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver
TrcvMode [in]	Transceiver mode to set: ETHTRCV_MODE_DOWN - shut down the Ethernet transceiver ETHTRCV_MODE_ACTIVE - activate the Ethernet transceiver
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameters or at least one of the hardware operations (read/write) has failed.

Functional Description
Sets the transceiver's mode. This function sets the transceiver to active or down.
Particularities and Limitations
> Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdx
Call context
> ANY > This function is Synchronous > This function is Reentrant

Table 6-5 EthTrcv_30_Core_SetTransceiverMode

6.2.5 EthTrcv_30_Core_GetTransceiverMode

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_GetTransceiverMode`** `(uint8 TrcvIdx, EthTrcv_ModeType *TrcvModePtr)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver.
TrcvModePtr [out]	Pointer for retrieved transceiver mode: ETHTRCV_MODE_DOWN - transceiver is turned off ETHTRCV_MODE_ACTIVE - transceiver is active
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameters or at least one of the hardware operations (read/write) has failed.
Functional Description	
Gets the transceiver's mode. This function returns the transceiver's current mode.	
Particularities and Limitations	
> Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdx	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 6-6 EthTrcv_30_Core_GetTransceiverMode

6.2.6 EthTrcv_30_Core_StartAutoNegotiation

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_StartAutoNegotiation`** `(uint8 TrcvIdx)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver.
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Starts automatic negotiation. This function starts the process to automatically negotiate the transceivers master-slave role, duplex mode and link speed, if available.	
Particularities and Limitations	
> Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdx	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 6-7 EthTrcv_30_Core_StartAutoNegotiation

6.2.7 EthTrcv_30_Core_GetLinkState

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_GetLinkState`** `(uint8 TrcvIdx, EthTrcv_LinkStateType *LinkStatePtr)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver.
LinkStatePtr [out]	Pointer for the retrieved link state value: ETHTRCV_LINK_STATE_DOWN - link is down ETHTRCV_LINK_STATE_ACTIVE - link is up
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Gets the transceiver's link state. This function returns the transceiver's current link state. It is called by the EthIf_MainFunctionState with a deterministic polling cycle time and and shall not be called directly to avoid side-effects	

Particularities and Limitations
> Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdx
Call context
> ANY > This function is Synchronous > This function is Reentrant

Table 6-8 EthTrcv_30_Core_GetLinkState

6.2.8 EthTrcv_30_Core_GetBaudRate

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_GetBaudRate`** `(uint8 TrcvIdx, EthTrcv_BaudRateType *BaudRatePtr)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver.
BaudRatePtr [out]	Pointer for the retrieved baud rate value: ETHTRCV_BAUD_RATE_10MBIT - Linkspeed 10 Mbit/s ETHTRCV_BAUD_RATE_100MBIT - Linkspeed: 100 Mbit/s ETHTRCV_BAUD_RATE_1000MBIT - Linkspeed: 1Gb/s
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Gets the transceiver's baud rate. This function returns the transceiver's current baud rate.	
Particularities and Limitations	
> Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdx	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 6-9 EthTrcv_30_Core_GetBaudRate

6.2.9 EthTrcv_30_Core_GetDuplexMode

Prototype
`Std_ReturnType` **`EthTrcv_30_Core_GetDuplexMode`** `(uint8 TrcvIdx, EthTrcv_DuplexModeType *DuplexModePtr)`

Parameter	
TrcvIdx [in]	Zero based index of the transceiver
DuplexModePtr [out]	Pointer for the retrieved duplex mode value: ETHTRCV_DUPLEX_MODE_HALF - transceiver operates in half duplex mode ETHTRCV_DUPLEX_MODE_FULL - transceiver operates in full duplex mode
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Gets the transceiver's duplex mode. This function returns the transceiver's current duplex mode.	
Particularities and Limitations	
> Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdx	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 6-10 EthTrcv_30_Core_GetDuplexMode

6.2.10 EthTrcv_30_Core_SetPhyTestMode

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_SetPhyTestMode`** `(uint8 TrcvIdx, EthTrcv_PhyTestModeType TestMode)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver
TestMode [in]	Test mode to be activated ETHTRCV_PHYTESTMODE_NONE - Normal operation ETHTRCV_PHYTESTMODE_1 - Test transmitter drop ETHTRCV_PHYTESTMODE_2 - Test master timing jitter ETHTRCV_PHYTESTMODE_3 - Test slave timing jitter ETHTRCV_PHYTESTMODE_4 - Test transmitter distortion ETHTRCV_PHYTESTMODE_5 - Test power spectral density
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Sets the phy test mode. This function activates the given phy test mode	

Particularities and Limitations
> Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdx
Call context
> ANY > This function is Synchronous > This function is Reentrant

Table 6-11 EthTrcv_30_Core_SetPhyTestMode

6.2.11 EthTrcv_30_Core_SetPhyLoopbackMode

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_SetPhyLoopbackMode`** `(uint8 TrcvIdx, EthTrcv_PhyLoopbackModeType LoopbackMode)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver
LoopbackMode [in]	Loopback mode to be activated ETHTRCV_PHYLOOPBACK_NONE - Normal operation ETHTRCV_PHYLOOPBACK_INTERNAL - Internal Loopback ETHTRCV_PHYLOOPBACK_EXTERNAL - External Loopback ETHTRCV_PHYLOOPBACK_REMOTE - Remote loopback
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Sets the phy loopback mode. This function activates the given phy loopback mode	
Particularities and Limitations	
> Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdx	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 6-12 EthTrcv_30_Core_SetPhyLoopbackMode

6.2.12 EthTrcv_30_Core_GetPhySignalQuality

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_GetPhySignalQuality`** `(uint8 TrcvIdx, uint8 *SignalQualityPtr)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver
SignalQualityPtr [out]	Pointer to the memory where the signal quality in percent shall be stored
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Gets the signal quality. This function obtains the current signal quality of the link of the indexed transceiver	
Particularities and Limitations	
> Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdx	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 6-13 EthTrcv_30_Core_GetPhySignalQuality

6.2.13 EthTrcv_30_Core_SetPhyTxMode

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_SetPhyTxMode`** `(uint8 TrcvIdx, EthTrcv_PhyTxModeType TxMode)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver
TxMode [in]	Transmission mode to be activated ETHTRCV_PHYTXMODE_NORMAL - Normal Operation ETHTRCV_PHYTXMODE_TX_OFF - Transmitter disabled ETHTRCV_PHYTXMODE_SCRAMBLER_OFF - Scrambler disabled
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Activates a given Tx Mode.	

This function activates the given transmission mode
Particularities and Limitations
> Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdxThe transmission mode to be activated is supported by the hardware
Call context
> ANY > This function is Synchronous > This function is Reentrant

Table 6-14 EthTrcv_30_Core_SetPhyTxMode

6.2.14 EthTrcv_30_Core_GetCableDiagnosticsResult

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_GetCableDiagnosticsResult`** `(uint8 TrcvIdx, EthTrcv_CableDiagResultType *CableDiagResultPtr)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver
CableDiagResultPtr [out]	Pointer to the memory where the signal quality in percent shall be stored ETHTRCV_CABLEDIAG_OK - Cable diagnostics result was OK ETHTRCV_CABLEDIAG_ERROR - Cable diagnostics failed ETHTRCV_CABLEDIAG_SHORT - Cable diagnostics detected a short on the MDI ETHTRCV_CABLEDIAG_OPEN - Cable diagnostics detected an open on the MDI
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Retrieves cable diagnostics result. This function retrieves the cable diagnostics result of a given transceiver	
Particularities and Limitations	
> Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdx	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 6-15 EthTrcv_30_Core_GetCableDiagnosticsResult

6.2.15 EthTrcv_30_Core_GetPhyIdentifier

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_GetPhyIdentifier`** `(uint8 TrcvIdx, uint32 *OrgUniqueIdPtr, uint8 *ModelNrPtr, uint8 *RevisionNrPtr)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver
OrgUniqueIdPtr [out]	Pointer to the memory where the Organizational Unique Identifier shall be stored
ModelNrPtr [out]	Pointer to the memory where the Manufacturer's Model Number shall be stored
RevisionNrPtr [out]	Pointer to the memory where the Revision Number shall be stored
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - Function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Obtains PHY identifier. This function obtains the PHY identifier of the Ethernet Transceiver according to IEEE 802.3-2015 chapter 22.2.4.3.1 PHY Identifier.	
Particularities and Limitations	
> Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdx	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 6-16 EthTrcv_30_Core_GetPhyIdentifier

6.2.16 EthTrcv_30_Core_GetVersionInfo

Prototype	
`void` **`EthTrcv_30_Core_GetVersionInfo`** `(Std_VersionInfoType *VersionInfoPtr)`	
Parameter	
VersionInfoPtr [out]	Pointer to where to store the version information. Parameter must not be NULL.
Return code	
void	none
Functional Description	
Returns the version information. EthTrcv_30_Core() returns version information, vendor ID and AUTOSAR module ID of the component.	

Particularities and Limitations
-
Call context
> ANY > This function is Synchronous > This function is Reentrant

Table 6-17 EthTrcv_30_Core_GetVersionInfo

6.2.17 EthTrcv_30_Core_SetTransceiverWakeupMode

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_SetTransceiverWakeupMode`** `(uint8 TrcvIdx, EthTrcv_WakeupModeType TrcvWakeupMode)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver
TrcvWakeupMode [in]	Operation that shall be performed: ETHTRCV_WUM_DISABLE - disable the transceiver wake-up ETHTRCV_WUM_ENABLE - enable the transceiver wake-up ETHTRCV_WUM_CLEAR - clear wake-up reason
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Main Function. Main function to handle cyclical tasks of the Ethernet transceiver driver	
Particularities and Limitations	
> Module is initialized.Module is initialized.EthTrcv_30_Core_TransceiverInit() has been called for the transceiver with index TrcvIdx.	
Call context	
> TASK > ANY > This function is Synchronous > This function is Non-Reentrant	

Table 6-18 EthTrcv_30_Core_SetTransceiverWakeupMode

6.2.18 EthTrcv_30_Core_GetTransceiverWakeupMode

Prototype
`Std_ReturnType` **`EthTrcv_30_Core_GetTransceiverWakeupMode`** `(uint8 TrcvIdx,`

`EthTrcv_WakeupModeType *TrcvWakeupModePtr)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver
TrcvWakeupMode [out]	Pointer to the memory where the transceivers current wake-up mode is stored: ETHTRCV_WUM_DISABLE - Transceiver wake-up disabled ETHTRCV_WUM_ENABLE - Transceiver wake-up enabled
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Gets the current wake-up mode. This function allows to retrieve if the transceiver wake-up is enabled or disabled.	
Particularities and Limitations	
Module is initialized.	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 6-19 EthTrcv_30_Core_GetTransceiverWakeupMode

6.2.19 EthTrcv_30_Core_CheckWakeup

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_CheckWakeup`** `(uint8 TrcvIdx)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameter or at least one of the hardware operations (read/write) has failed.
Functional Description	
Performs a wake-up check. This function performs a check of the transceiver's wake-up function.	
Particularities and Limitations	
Module is initialized.	
Call context	
> TASK > This function is Synchronous	

> This function is Non-Reentrant

Table 6-20 EthTrcv_30_Core_CheckWakeup

6.2.20 EthTrcv_30_Core_ReadTrcvReg

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_ReadTrcvReg`** `(uint8 TrcvIdx, uint8 RegAddr, uint16 *RegValPtr)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver. Range: 0 to number of configured transceiver instances minus one.
RegAddr [in]	Address of the register to be read
RegValPtr [out]	Pointer to the address of the read register value
Return code	
Std_ReturnType	RetVal E_OK: Operation successful E_NOT_OK: Operation not successful
Functional Description	
Main function implementation which is used for link handling and link startup. This function is used to perform timing relevant initialization and link startup and/or link handling tasks.	
Particularities and Limitations	
> Module is initialized.Module is initialized. CREQ-170739	
Call context	
> TASK > TASK > This function is Synchronous > This function is Non-Reentrant	

Table 6-21 EthTrcv_30_Core_ReadTrcvReg

6.2.21 EthTrcv_30_Core_WriteTrcvReg

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_WriteTrcvReg`** `(uint8 TrcvIdx, uint8 RegAddr, uint16 RegVal)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver. Range: 0 to number of configured transceiver instances minus one.
RegAddr [in]	Address of the register to be read
RegVal [in]	Value to be written to the register
Return code	
Std_ReturnType	RetVal E_OK: Operation successful E_NOT_OK: Operation not successful

Functional Description
Writes a given value to a register of the transceiver with index TrcvIdx. -
Particularities and Limitations
Module is initialized. CREQ-170740
Call context
> TASK > This function is Synchronous > This function is Reentrant

Table 6-22 EthTrcv_30_Core_WriteTrcvReg

6.2.22 EthTrcv_30_Core_ReadCl45TrcvReg

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_ReadCl45TrcvReg`** `(uint8 TrcvIdx, uint8 DeviceAddr, uint16 RegAddr, uint16 *RegValPtr)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver. Range: 0 to number of configured transceiver instances minus one.
DeviceAddr [in]	Address of the MMD
RegAddr [in]	Address of the register to be read
RegValPtr [out]	Pointer to the address of the read register value
Return code	
Std_ReturnType	RetVal E_OK: Operation successful E_NOT_OK: Operation not successful
Functional Description	
Retrieves the value of a IEEE 802.3 Clause 45 register of the transceiver with index TrcvIdx over the Clause 22 registers MMD_ACCESS_CONTROL and MMD_ACCESS_ADDRESS_DATA as described in IEEE 802.3 Annex 22D. -	
Particularities and Limitations	
Module is initialized. CREQ-170741	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 6-23 EthTrcv_30_Core_ReadCl45TrcvReg

6.2.23 EthTrcv_30_Core_WriteCl45TrcvReg

Prototype	
`Std_ReturnType` **`EthTrcv_30_Core_WriteCl45TrcvReg`** `(uint8 TrcvIdx, uint8 DeviceAddr, uint16 RegAddr, uint16 RegVal)`	
Parameter	
TrcvIdx [in]	Zero based index of the transceiver. Range: 0 to number of configured transceiver instances minus one.
DeviceAddr [in]	Address of the MMD
RegAddr [in]	Address of the register to be read
RegVal [in]	Value to be written to the register
Return code	
Std_ReturnType	RetVal E_OK: Operation successful E_NOT_OK: Operation not successful
Functional Description	
Writes a given value to a IEEE 802.3 Clause 45 register of the transceiver with index TrcvIdx over the Clause 22 registers MMD_ACCESS_CONTROL and MMD_ACCESS_ADDRESS_DATA as described in IEEE 802.3 Annex 22D. -	
Particularities and Limitations	
Module is initialized. CREQ-170742	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 6-24 EthTrcv_30_Core_WriteCl45TrcvReg

6.3 Services used by EthTrcv

In the following table services provided by other components, which are used by the EthTrcv are listed. The usage of the APIs by the transceiver drivers depends on configuration options also given in the table. For details about prototype and functionality refer to the documentation of the providing component.

Component	Option	API
Det	Development Error Detection	Det_ReportError()
Dem	Production Error Detection	Dem_ReportErrorStatus()
Eth	Hardware Access by MII interface of Ethernet Controller	Eth_ReadMii()
		Eth_WriteMii()
EthSwt	Hardware Access by configuration interface of Ethernet Switch driver	EthSwt_ReadTrcvRegister()
		EthSwt_WriteTrcvRegister()

Table 6-25 Services used by the EthTrcv

7 Configuration

The EthTrcv is configured with the help of the tool DaVinci Configurator Pro. This chapter describes the configuration process for some selected features. For information for features not introduced here please refer to the `Properties` view of the Configurator Pro.

The configuration classes of the EthTrcv parameters depend on the supported configuration variants. For their definitions please see the EthTrcv_bswmd.arxml file.

7.1 Configuration Variants

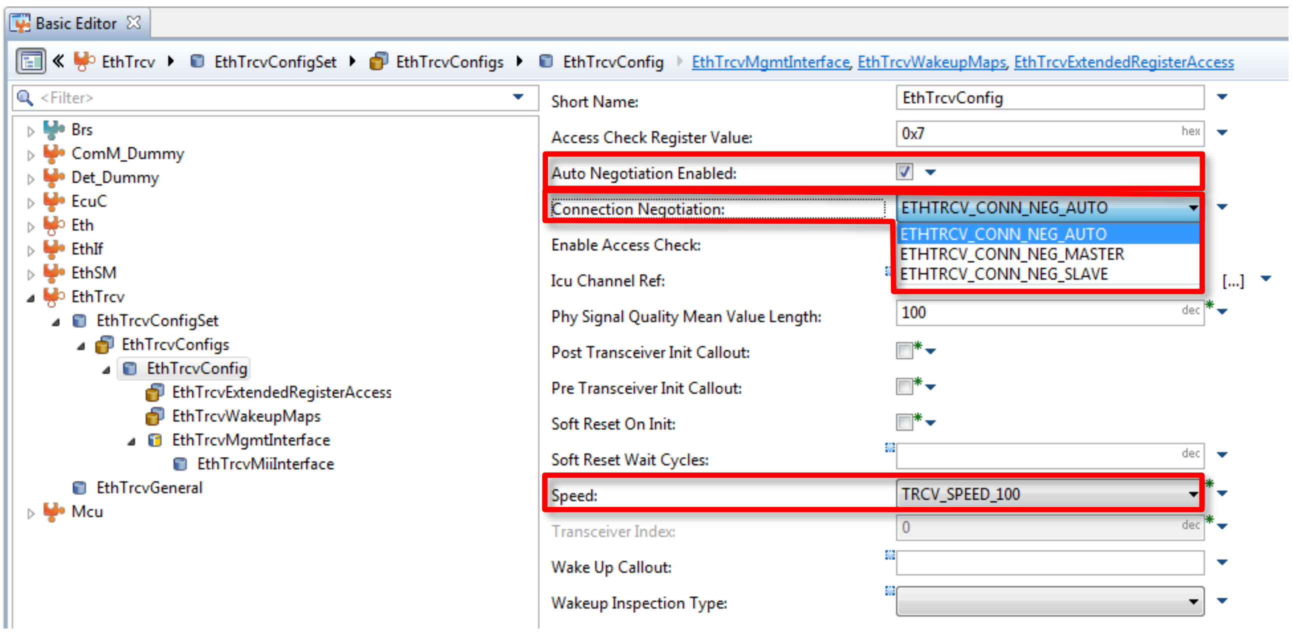
The EthTrcv supports the configuration variants

> `VARIANT-PRE-COMPILE`

7.2 Connection Negotiation

Some Ethernet Transceivers offer the possibility to enable auto negotiation for the automatic configuration of settings like link speed, duplex mode and master-slave settings with the link partner. Auto negotiation can be enabled by setting the parameter “Auto Negotiation Enabled” in the container “EthTrcvConfig”. If it is disabled, the value configured for “Speed” is forced as link speed, else the setting for “Speed” is corresponding to the maximum advertised value during auto-negotiation.

If available, for example for 100/1000BASE-T1-transceivers, the Master/Slave setting can be configured with the enumeration “Connection Negotiation”, as it is shown in figure Figure 7-1.



Short Name:	EthTrcvConfig
Access Check Register Value:	0x7 hex
Auto Negotiation Enabled:	✓
Connection Negotiation:	ETHTRCV_CONN_NEG_AUTO (ETHTRCV_CONN_NEG_AUTO, ETHTRCV_CONN_NEG_MASTER, ETHTRCV_CONN_NEG_SLAVE)
Enable Access Check:	
Icu Channel Ref:	[...]
Phy Signal Quality Mean Value Length:	100 dec
Post Transceiver Init Callout:	☐
Pre Transceiver Init Callout:	☐
Soft Reset On Init:	☐
Soft Reset Wait Cycles:	dec
Speed:	TRCV_SPEED_100
Transceiver Index:	0 dec
Wake Up Callout:	
Wakeup Inspection Type:	

Figure 7-1 Configuration of auto- and connection negotiation

The enumeration values of the parameter EthTrcvConnNeg can be found in Table 7-1.

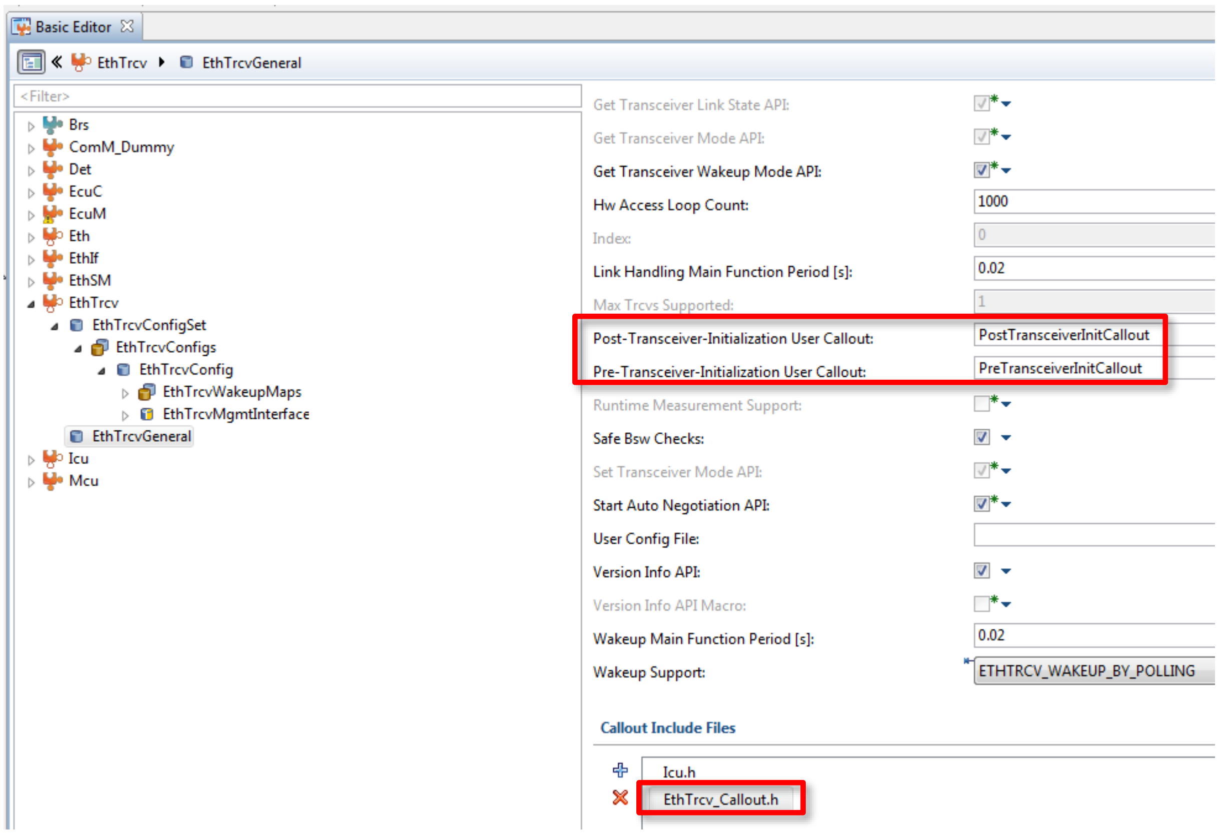
Connection Negotiation	Description
`ETHTRCV_CONN_NEG_AUTO`	Auto negotiation for the Master/Slave setting is enabled. Depending on the transceiver's technology ability the roles are negotiated during link startup.
`ETHTRCV_CONN_NEG_MASTER`	The transceiver is configured as master.
`ETHTRCV_CONN_NEG_SLAVE`	The transceiver is configured as slave.

Table 7-1 Connection Negotiation

7.3 User callouts

The driver allows to inject integration code, for example in the processing of the `EthTrcv_30_<Driver>_TransceiverInit()` API. In the following a configuration example is provided:

This integration code is executed either before the actual code of the `EthTrcv_30_<Driver>_TransceiverInit()`, after the actual code or both (Pre- and Post – Transceiver Init Callout). Figure 7-2 shows a configuration example, if both callouts are enabled.



Get Transceiver Link State API: ✓
Get Transceiver Mode API: ✓
Get Transceiver Wakeup Mode API: ✓
Hw Access Loop Count: 1000
Index: 0
Link Handling Main Function Period [s]: 0.02
Max Trcvs Supported: 1
Post-Transceiver-Initialization User Callout: PostTransceiverInitCallout
Pre-Transceiver-Initialization User Callout: PreTransceiverInitCallout
Runtime Measurement Support: ☐
Safe Bsw Checks: ✓
Set Transceiver Mode API: ✓
Start Auto Negotiation API: ✓
User Config File:
Version Info API: ✓
Version Info API Macro: ☐
Wakeup Main Function Period [s]: 0.02
Wakeup Support: ETHTRCV_WAKEUP_BY_POLLING

Callout Include Files
Icu.h
EthTrcv_Callout.h

Figure 7-2 Enabling Pre- and Post-Transceiver Init Callouts

These callouts can be used to integrate derivative specific code that must be executed for a proper operation of the Ethernet Transceiver. The corresponding user-functions are accessed over function pointers declared and defined in the `EthTrcv_30_<Driver>_Lcfg.h/.c`. The names of the functions can be given individually and they can be declared and defined in separate files. The declaration has to be included as shown in Figure 7-2. In the “Callout include files”. In the following an example is provided:

Example EthTrcv_Callout.h:

Example

```
#ifndef ETHTRCV_CALLOUT_H
# define ETHTRCV_CALLOUT_H
# include "Std_Types.h"


void PreTransceiverInitCallout(uint8 TrcvIdx);
void PostTransceiverInitCallout(uint8 TrcvIdx);
#endif /* ETHTRCV_CALLOUT_H */
```

Example EthTrcv_Callout.c:

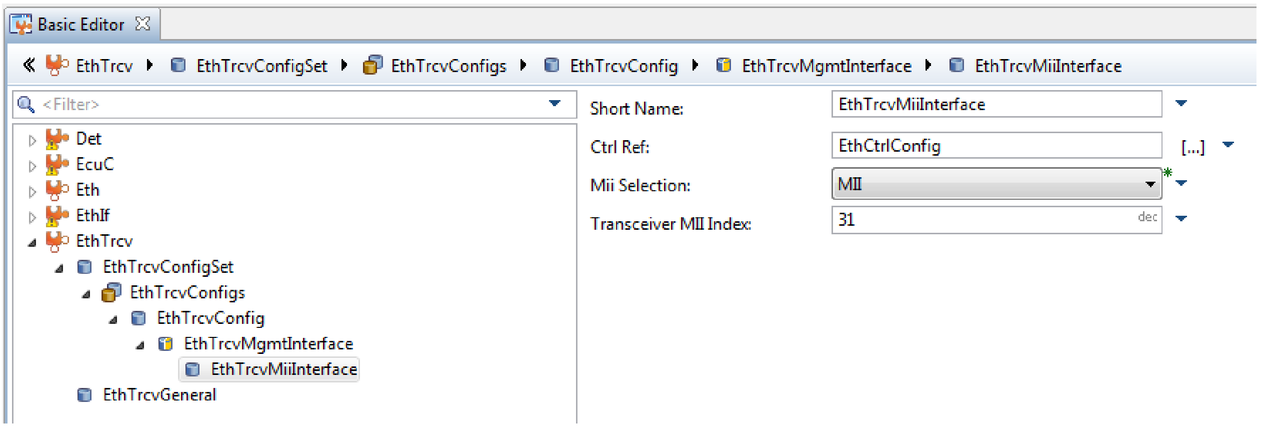
Example

```
#include "EthTrcv_Callout.h"


void PreTransceiverInitCallout(uint8 TrcvIdx)
{
  /* Implementation of callout */
}

void PostTransceiverInitCallout(uint8 TrcvIdx)
{
  /* Implementation of callout */
}
```

The same mechanism can be used for the “WakeupCallout” and “WakeupInspectionCallout” parameters that can be found in the EthTrcvConfig-Container.

7.4 Transceiver Hardware Access Interface

The MII- or port-index for the transceiver can be set in the EthTrcvMgmtInterface container, after the subcontainer is chosen. Possible types are EthTrcvMIIInterface or EthTrcvSwitchInterface, depending on whether the transceiver shall access the hardware by the MII-API of the Ethernet Controller driver or the general hardware access API of the Ethernet Switch driver.



Short Name: EthTrcvMiiInterface
Ctrl Ref: EthCtrlConfig
Mii Selection: MII
Transceiver MII Index: 31 dec

Figure 7-3 Management Interface Configuration

The value chosen for the Transceiver MII Index is in the range between 0 and 31 and represents the transceiver's address used for IEEE 802.3 Clause 22 MDIO register read and write operations. If this management method is chosen, the user has to make sure that the configured address is correct for the transceiver in focus by checking the corresponding manual or data sheet or by performing a transceiver register read operation and checking the result for validity (an invalid MII Index could cause that all transceiver register read operations only return 1's or 0's.). If the MDIO management access fails, transceiver initialization and other important operations will fail as well and communication is not possible.

7.5 Wakeup Support

This section describes how to configure the fundamental configuration elements needed for the wakeup support. However, the scope is only on the transceiver driver related configuration elements. For elements of the other modules involved in the wakeup procedure please refer to the corresponding technical reference.

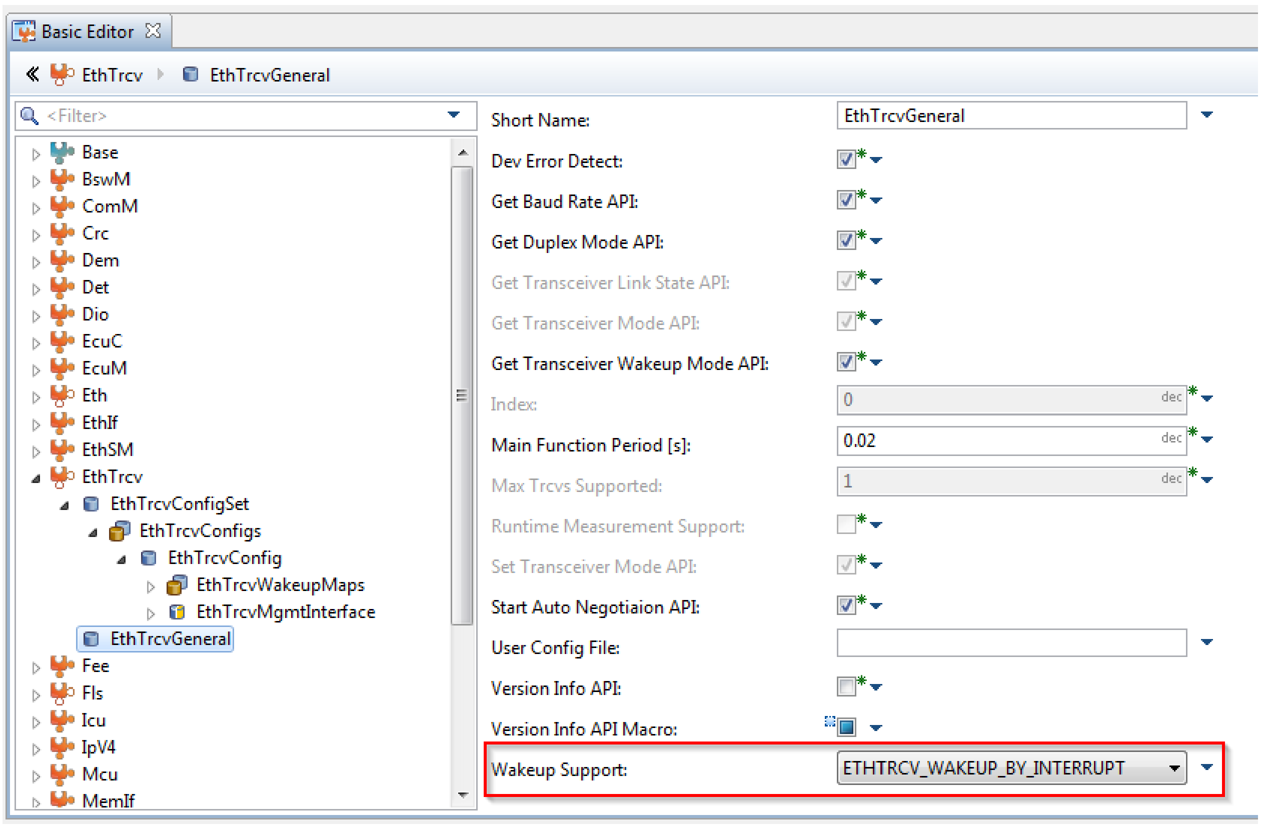
Wakeup Method

The wakeup support is enabled by setting the corresponding method that shall be used for wakeup detection. These options are:

- `ETHTRCV_WAKEUP_BY_POLLING`
- `ETHTRCV_WAKEUP_BY_INTERRUPT`

For a detailed description of the behavior of the two methods please refer to section 4.9 or the detailed description of the parameter within the Configurator Pros `Properties` view.

Following screenshot shows an example where wakeup support by interrupt is enabled.



Short Name:	EthTrcvGeneral
Dev Error Detect:	✓
Get Baud Rate API:	✓
Get Duplex Mode API:	✓
Get Transceiver Link State API:	✓
Get Transceiver Mode API:	✓
Get Transceiver Wakeup Mode API:	✓
Index:	0 dec
Main Function Period [s]:	0.02 dec
Max Trcvs Supported:	1 dec
Runtime Measurement Support:	
Set Transceiver Mode API:	✓
Start Auto Negotiaion API:	✓
User Config File:	
Version Info API:	
Version Info API Macro:	
Wakeup Support:	ETHTRCV_WAKEUP_BY_INTERRUPT

Figure 7-4 General Wakeup support configuration

Wakeup Detection Mechanism

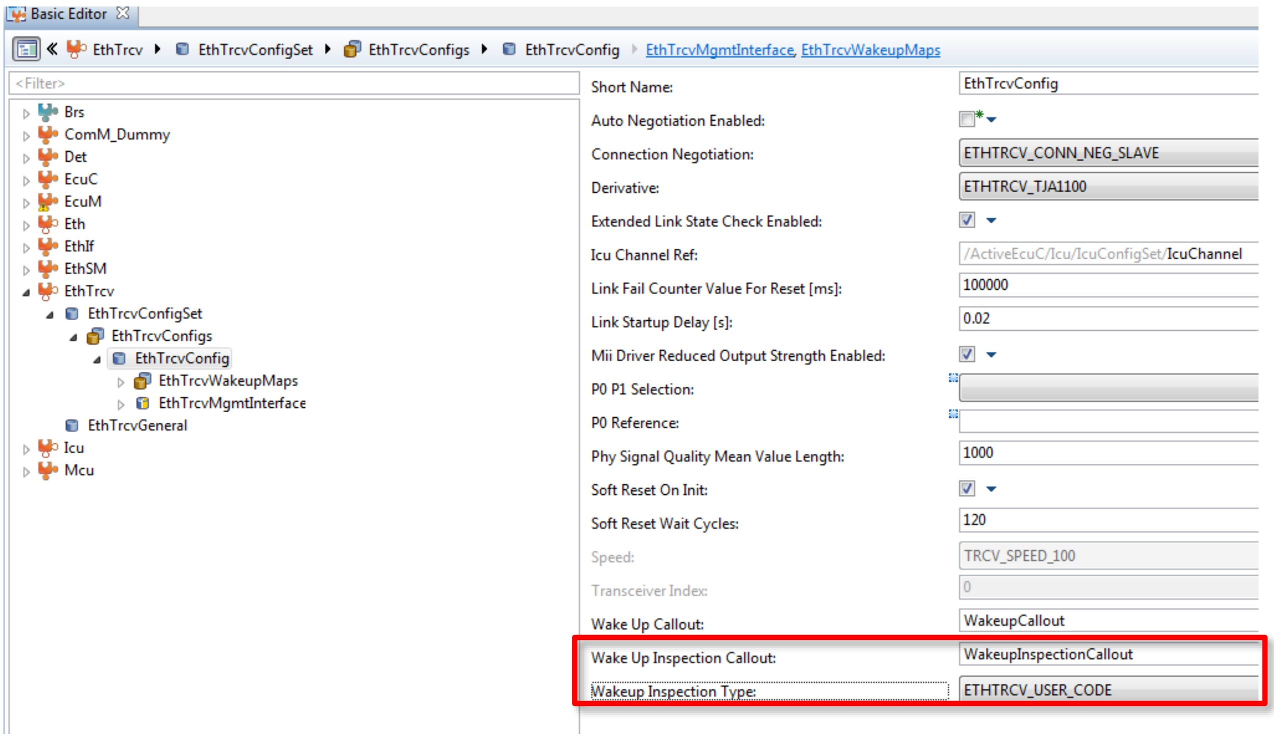
The driver provides the possibility to configure how the wakeup detection shall be performed. Therefore two supported detection mechanisms can be selected.

- `ETHTRCV_ICU_CHANNEL`
- `ETHTRCV_USER_CODE`

The third option `ETHTRCV_TRCV_REGISTER` is not supported by the driver.

For a detailed description of how the mechanisms work please refer to the detailed description of the parameter in the `Properties` view of Configurator Pro. In case of `ETHTRCV_USER_CODE` additional information can be found in section 5.4 and 7.5.

As shown in figure Figure 7-5 the parameter can be configured for every transceiver instance therefore the driver is able to perform adapted wakeup detection for the configured transceivers. If `ETHTRCV_USER_CODE` is configured, the WakeupInspectionCallout has to be provided as well. The name for the callout can be defined by the user, the header containing the corresponding declaration can be included with the mechanism described in 7.3.

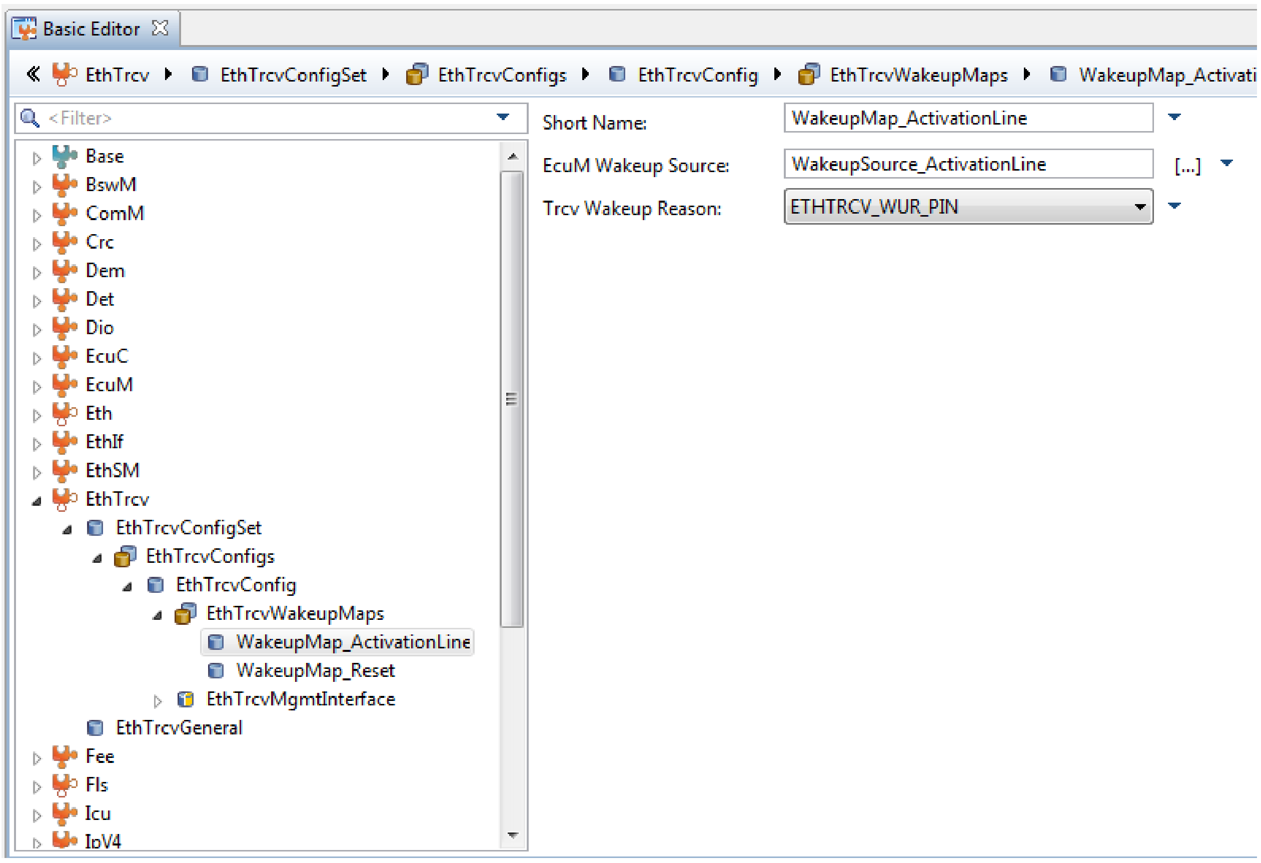


Short Name:	EthTrcvConfig
Auto Negotiation Enabled:	☐
Connection Negotiation:	ETHTRCV_CONN_NEG_SLAVE
Derivative:	ETHTRCV_TJA1100
Extended Link State Check Enabled:	✓
Icu Channel Ref:	/ActiveEcuC/Icu/IcuConfigSet/IcuChannel
Link Fail Counter Value For Reset [ms]:	100000
Link Startup Delay [s]:	0.02
Mii Driver Reduced Output Strength Enabled:	✓
P0 P1 Selection:	
P0 Reference:	
Phy Signal Quality Mean Value Length:	1000
Soft Reset On Init:	✓
Soft Reset Wait Cycles:	120
Speed:	TRCV_SPEED_100
Transceiver Index:	0
Wake Up Callout:	WakeupCallout
Wake Up Inspection Callout:	WakeupInspectionCallout
Wakeup Inspection Type:	ETHTRCV_USER_CODE

Figure 7-5 Selection of the Wakeup detection mechanism

Wakeup Map

To achieve proper wakeup detection a wakeup map must be configured. The map relates an EcuM Wakeup Source with a wakeup reason used by the driver internally. The following screenshot shows an example configuration, which maps the wakeup reason `ETHTRCV_WUR_PIN` to a wakeup source related to the wakeup event on an activation line.



Short Name: WakeupMap_ActivationLine
EcuM Wakeup Source: WakeupSource_ActivationLine
Trcv Wakeup Reason: ETHTRCV_WUR_PIN

Figure 7-6 Wakeup map configuration

8 Glossary and Abbreviations

8.1 Glossary

Term	Description
DaVinci Configurator PRO	Generation tool for MICROSAR components
Wakeup Event	The wakeup event is a common event that triggers the wakeup detection procedure to evaluate, which event has occurred. In common it is initiated by a level change on a signal line, which is either be driven by another ECU (then it's called an activation line) or by a transceiver (then it's called a transceiver interrupt line).
Wakeup Reason	The wakeup reason is specified by AUTOSAR and defines the possible wakeup events the transceiver driver can detect. It is so to say a more specific representation of a wakeup event used transceiver driver internal.
Wakeup Source	The wakeup source is the representation of a wakeup event on EcuM level. It is used for initiating the wakeup detection and also to trigger further processes if a corresponding wakeup event was detected.

Table 8-1 Glossary

8.2 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
ICU	Input Capture Unit
MDI	Media Dependent Interface
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)

MII	Media Independent Interface
PCS	Physical Coding Sublayer
PMA	Physical Media Attachment
RTE	Runtime Environment
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 8-2 Abbreviations

9 Contact

Visit our website for more information on

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

www.vector.com