MICROSAR Ethernet Interface

Technical Reference

Version 6.0.0

Author	Harald Walter, Mark Harsch
Status	Released

Document Information

History

Author	Date	Version	Remarks
Alex Lunkenheimer	2008-10-08	1.0	Creation of document
Alex Lunkenheimer	2009-10-05	2.0	Tool based configuration
Alex Lunkenheimer	2011-07-25	2.1	ESCAN00051966, Documentation of EthIf_Transmit not correct
Alex Lunkenheimer	2011-07-25	2.1.1	No changes
Alex Lunkenheimer	2012-02-22	2.1.2	Released
Harald Walter	2012-08-31	2.2.0	ESCAN00061126: Discard VLAN Tags within received Ethernet frames originated by PLC chips
Harald Walter	2013-04-16	3.0.0	ESCAN00066678: ASR4.1.1 extensions
Harald Walter	2014-01-31	3.1.0	ESCAN00073156: Implemented zero-copy extensions
Harald Walter	2014-03-04	3.2.0	ESCAN00074063: Introduced PTP support
Harald Walter	2014-07-30	3.4.0	ESCAN00077150: AR4-500: AUTOSAR CONC_600_SwitchConfiguration
Harald Walter	2014-09-19	3.5.0	▶ ESCAN00077496: AR4-897: Support overwriting of the Ethernet header during transmission and support reception of Ethernet header information for upper layers of the Ethernet Interface ▶ ESCAN00077345: AR3-2679: Description BCD-coded return-value of XXX_GetVersionInfo() in TechRef
Mark Harsch	2015-03-11	3.6.0	▶ ESCAN00080875: Rename Technical Reference according to SPEC-63403 ▶ ESCAN00081283: FEAT-705: Ethernet wakeup based on Activation Line [AR4-1006] ▶ Some minor editorial corrections
Mark Harsch	2015-06-17	3.7.0	▶ ESCAN00083462: FEAT-1457: SRP module development ▶ Corrected/extended information about generated files
Mark Harsch	2015-07-31	3.8.0	ESCAN00084013: Support of Mirroring/Gateway functionality
Mark Harsch	2015-12-22	4.0.0	▶ Introduction of detailed feature section ▶ Minor adaptions to fit latest version of Technical Reference template ▶ ESCAN00086805: Allow establishing a link on a EthIf Controller if not all related EthSwt Ports have a link established ▶ ESCAN00086741: FEAT-1529: Support

			Ethernet Switches for Ethernet Time Sync
Mark Harsch	2017-01-13	5.1.0	▶ General rework ▶ ESCAN00093542: Introduced description of Ethernet switch time synchronization ▶ ESCAN00093543: Introduced description of Ethernet switch multicast to port assignment ▶ ESCAN00092838: Exclusive Areas in configuration tool does not match with description in Technical Reference
Mark Harsch	2017-02-08	5.2.0	FEAT-2354: Firewall concept for Ethernet
Mark Harsch	2017-02-17	5.2.1	Review integration
Mark Harsch	2017-02-20	6.0.0	▶ Updated Error IDs of Default Error Reporting ▶ FEAT-2151: Extended Ethernet Bus Diagnostic

Reference Documents

No.	Title	Version
[1]	AUTOSAR_SWS_EthernetInterface.pdf	4.1.1
[2]	AUTOSAR_SWS_EthernetInterface.pdf	4.2.1
[3]	AUTOSAR_SWS_DET.pdf	2.2.1
[4]	AUTOSAR_SWS_DEM.pdf	2.2.0
[5]	AUTOSAR_BasicSoftwareModules.pdf	1.0.0
[6]	TechnicalReference_EthFw.pdf	1.0.0

Scope of the Document

This technical reference describes the general use of the Ethernet Interface basis software. Please refer to your Release Notes to get a detailed description of the platform (host, compiler) 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

1 Component History

Component Version	New Features
01.00.xx	Created
02.00.xx	Vector Coding Rules Applied
03.00.xx	Incompatible IPv6 Adaption
04.00.xx	VLAN + AUTOSAR 4.1.1 compliance
04.01.xx	Zero-copy extensions
04.02.xx	PTP support
04.04.xx	EthSwt support
04.05.xx	Support for overwriting the Ethernet header during transmission
04.06.xx	Wakeup support
04.07.xx	Timestamp type according to AUTOSAR 4.2.1
04.08.xx	Bandwidth manipulation support (e.g. for usage by SRP)
04.09.xx	TcpIp according to AUTOSAR 4.2.1 support
04.10.xx	▶ Traffic Mirroring/Gateway support ▶ Ethernet driver API infixing (support of different Ethernet drivers within one configuration)
05.00.xx	▶ Ethernet Switch Frame Management support ▶ Lax link aggregation support for Ethernet switch use case
06.00.xx	Support of Vector Ethernet drivers able to handle receive buffer segments
07.00.xx	▶ Ethernet Switch Time Synchronization support ▶ Support for switching Ethernet switch port groups either during communication request or by BswM (PnC related)
07.01.xx	Ethernet switch multicast to port assignment support (e.g. for usage by SRP)
07.02.xx	Ethernet firewall support
08.00.xx	▶ Extended Ethernet Bus Diagnostics ▶ Migrated Vector specific EthIf_GetGlobalTime() to ASR4.3 API EthIf_GetCurrentTime()

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the Ethernet Interface.

Supported AUTOSAR Release*:	4.1.1	
Supported Configuration Variants:	pre-compile	
Vendor ID:	ETHIF_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	ETHIF_MODULE_ID	65 decimal

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

The Ethernet Interface provides hardware independent access to control connected Ethernet Controller Drivers and Ethernet Transceiver Drivers in a generic way. It offers the functionality to

- Control the operation modes of the lower layer drivers
- Obtain status information of the lower layer drivers
- Send and receive Ethernet frames.

2.1 Architecture Overview

The following figure shows the location of the Ethernet Interface in the AUTOSAR architecture.

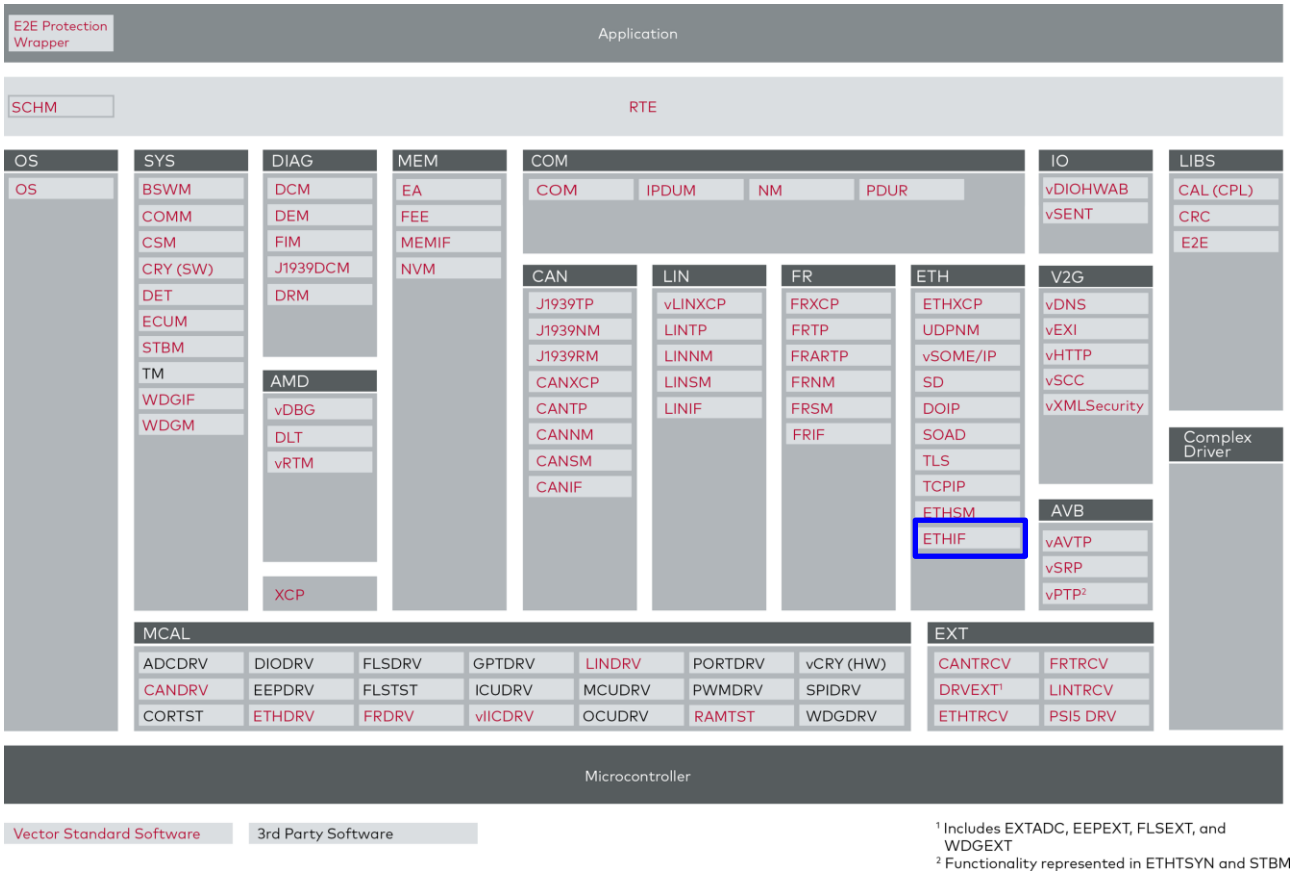

Figure 2-1 MICROSAR 4.1 Architecture Overview

The next figure shows the interfaces to adjacent modules of the Ethernet Interface. These interfaces are described in chapter 4.3.

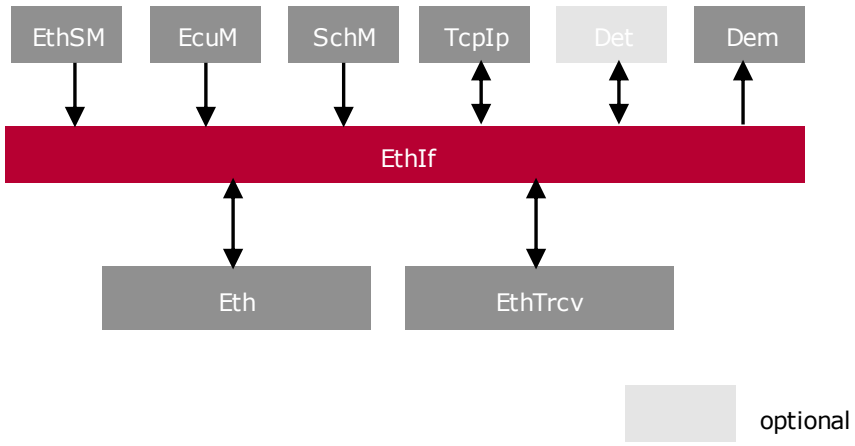

Figure 2-2 Interfaces to adjacent modules of the Ethernet Interface

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

3 Functional Description

3.1 Features

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

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

- Table 3-1 Supported AUTOSAR standard conform features
- Table 3-2 Not supported AUTOSAR standard conform features

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

- Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Ethernet Interface initialization
Mode Management for Ethernet driver and Ethernet Transceiver driver
Transmission/Reception of Ethernet frames
Handling of untagged, priority tagged and VLAN tagged Ethernet frames
Recognition and filtering of Ethernet frames based on Frame Type
Recognition and filtering of Ethernet frames based on VLAN
Insertion/Removal of Frame Type for upper layer modules
Insertion/Removal of VLAN tag for upper layer modules
Insertion of VLAN priority during transmission
Manipulation/Retrieval of the MAC address of Ethernet controllers
Adaption of the MAC filter of Ethernet controllers
Retrieval of the link of Ethernet transceivers
Configurable upper layer interface
Interrupt operation mode
Polling operation mode
Multiple Ethernet Transceiver support
Multiple Ethernet Controller support
Pre-compile configuration variant
Development error reporting by Det

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations AUTOSAR 4.1.1

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

Not Supported AUTOSAR Standard Conform Features
Link-time configuration variant
Post-build configuration variant
Transceiver mode management interface for upper layers not provided (please see 3.1.1.1 for detailed explanation)
Configuration deviations related to the Ethernet switch support (please see 3.1.1.2 for detailed explanation)

Table 3-2 Not supported AUTOSAR standard conform features

3.1.1.1 Transceiver mode management deviation

In contradiction to AUTOSAR standard 4.1.1 the APIs `EthIf_TransceiverInit()`, `EthIf_SetTransceiverMode()` and `EthIf_GetTransceiverMode()` are not provided to the upper layer.

The APIs are removed because the interface to the upper layer is defined by a so called Ethernet Interface Controller, which is an abstraction for the tuple composited of the underlying hardware elements (e.g. Ethernet controller, Ethernet transceiver or Ethernet switch). So there is no need for the upper layer to manage the underlying hardware explicitly but use the API for the Ethernet Interface Controller for implicit management. Therefore the functionality of the removed APIs is implemented by the APIs `EthIf_ControllerInit()`, `EthIf_SetControllerMode()` and `EthIf_GetControllerMode()`.

3.1.1.2 Configuration deviation related to the Ethernet switch support

In contradiction to AUTOSAR standard 4.1.1 the configuration structure was adapted to meet the requirements to support Ethernet switches according to a preliminary AUTOSAR 4.3 specification (based on an intermediate result of AUTOSAR RfC 67878 available at implementation time).

An additional configuration container called `EthIfSwitchPortGroup` composing Ethernet switch ports forming an entity able to be controlled and retrieved information for as a whole was introduced.

The `EthIfController` container was changed with respect to its parameters. `EthIfEthCtrlRef` and `EthIfEthTrcvRef` were renamed to `EthIfPhysControllerRef` and `EthIfTransceiverRef`. Additionally two new parameters were introduced. `EthIfSwitchRef` allow referencing an Ethernet switch and `EthIfSwitchPortGroupRef` allow referencing the newly introduced container.

This structural change allows the Ethernet Interface to handle Ethernet switches without changing the interface for the upper layer modules with respect to the mode management functionality. It implicitly initializes and changes the mode of a Ethernet switch during the calls to the Ethernet Interface Controller APIs `EthIf_ControllerInit()`, `EthIf_SetControllerMode()` and `EthIf_GetControllerMode()`.

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Ethernet switch support according to AUTOSAR 4.2.1
Ethernet switch frame management support (feature must also be supported by Ethernet switch driver)
Ethernet switch time synchronization support (feature must also be supported by Ethernet switch driver)
Multicast to switch port assignment manipulation (feature must also be supported by Ethernet switch driver)
Ethernet wakeup support according to AUTOSAR 4.2.1 (feature must also be supported by Ethernet transceiver driver)
Precision Time Protocol (PTP) support (feature must also be supported by Ethernet driver)
Forwarding and Queuing Enhancements for Time-Sensitive Streams (FQTSS) support (feature must also be supported by Ethernet driver)
Reception buffer locking mechanism based on VLAN and Frame Type (feature must also be supported by Ethernet driver)
External transmission buffer provision (feature must also be supported by Ethernet driver)
Ethernet frame mirroring functionality on Ethernet Interface Controller level (feature must also be supported by Ethernet driver)
Ethernet frame gateway functionality on Ethernet Interface Controller level (feature must also be supported by Ethernet driver)
Retrieval of the Ethernet header information for reception and transmission buffers (feature must also be supported by Ethernet driver)
Ethernet firewall support (Vector specific BSW)
Extended Ethernet Rx/Tx Statistics

Table 3-3 Features provided beyond the AUTOSAR standard

For a detailed description of the Additions/Extensions please refer to the related subsection in 3 - Functional Description.

3.1.3 Limitations

No limitations beside the ones stated in Table 3-2.

3.2 Initialization

The Ethernet Interface is initialized by calling the `EthIf_Init()` service with `NULL_PTR` as configuration pointer due to Pre-Compile-Configuration-Variant only support.

Caution
If start-up code doesn't initialize the RAM and therefore some data isn't set to a predefined value EthIf relies on (e.g. zero initialized data), the function `EthIf_InitMemory()` must be called during the startup. This function sets the predefined values usually set by the start-up code.

3.3 States

Each EthIf Controller has a state machine with respect to its link state. The following figure shows the state machine and its transitions.

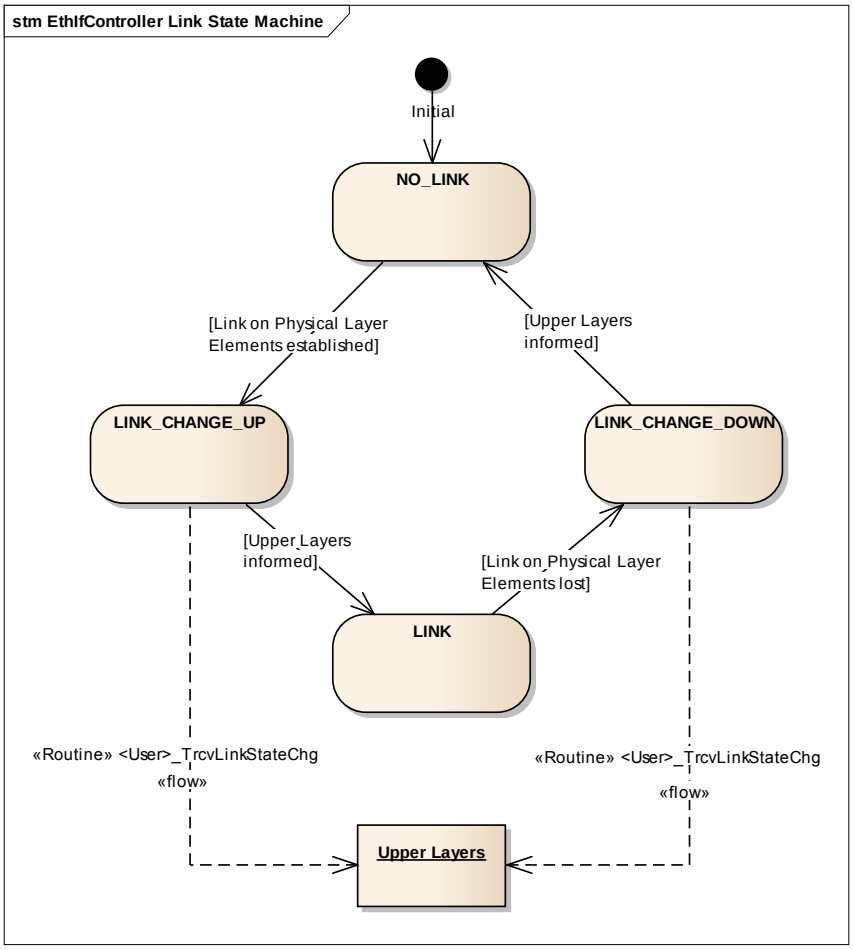

Figure 3-1 EthIf Controller Link State Machine

Initially the EthIf Controller is in state `NO_LINK`. If all of the underlying physical layer elements (i.e. Ethernet Transceivers and Ethernet Switch Ports) have established a link the transition to state `LINK_CHANGE_UP` is done. In this state, the upper layers are informed about the link change to an active link for this EthIf Controller. After the indication to the upper layers this state is left to state `LINK`.

If now at least one of the underlying physical layer elements mapped to the EthIf Controller reports a link down (e.g. forced by a mode change or some possible issue on hardware

layer) the transition to state LINK_CHANGE_DOWN is done. In this state the upper layers are informed about the link change to an inactive link for this EthIf Controller. After the indication to the upper layers this state is left to state NO_LINK.

Note
The condition for the transitions from state NO_LINK to state LINK_CHANE_UP and from state LINK to LINK_CHANGE_DOWN can be influenced by the feature ‘Lax Link Aggregation’ described in 3.10 Lax Link Aggregation.

3.4 Main Functions

The Ethernet Interface has multiple main functions. Dependent on receive and transmit mode (polling or interrupt) they exist or not.

Following table lists all main functions, when they exist and describes which processing is done by them.

Note
The main functions aren’t declared by EthIf itself but declaration is provided by the SchM during generation with DaVinci Configurator PRO.

EthIf_MainFunctionState()		
Existence	Timing	Description
EthIfEnableMainFunctionState is activated.	Scheduled according the period provided by configuration parameter EthIfMainFunctionPeriod and EthIfTrcvLinkStateChgMainReload. Period = EthIfMainFunctionPeriod * EthIfTrcvLinkStateChgMainReload	Processes the link state observation of the connected transceivers/switch-ports and propagates changes to the upper layers.

Table 3-4 EthIf_MaionFunctionState() description

EthIf_MainFunctionRx()		
Existence	Timing	Description
EthIfEnableRxInterrupt is deactivated.	Scheduled according the period provided by configuration parameter EthIfMainFunctionPeriod.	Processing of reception handling (polling mode). The controller drivers are polled for received frames and the frames are

		passed to the upper layers by an indication call.

Table 3-5 EthIf_MaionFunctionRx() description

EthIf_MainFunctionTx()		
Existence	Timing	Description
`EthIfEnableTxInterrupt` is deactivated.	Scheduled according the period provided by configuration parameter `EthIfMainFunctionPeriod`.	Processing of transmission confirmation handling (polling mode). The controller drivers are polled for the indication that the frames triggered for transmission were transmitted. If there is an unconfirmed transmission that has finished the upper layer is informed that the transmission was successful. Additionally (if `EthIf_MainFunctionState()` doesn't exist) the link state observation normally performed by the state main function is done here. The observation is performed each n-th run of the main function were n is the value of `EthIfTrcvLinkStateChgMainReload`.

Table 3-6 EthIf_MaionFunctionTx() description

3.5 VLAN

By enabling the VLAN feature the currently known controller indexes do not reference physical controllers anymore. Instead the controller index references a virtual controller.

A virtual controller is in a manner of speaking an abstract superclass. There are two different specializations of the virtual controller:

- A VLAN specialization and
- A physical controller specialization.

Figure 3-2 shows the abstract virtual controller and its specializations in form of a class diagram.

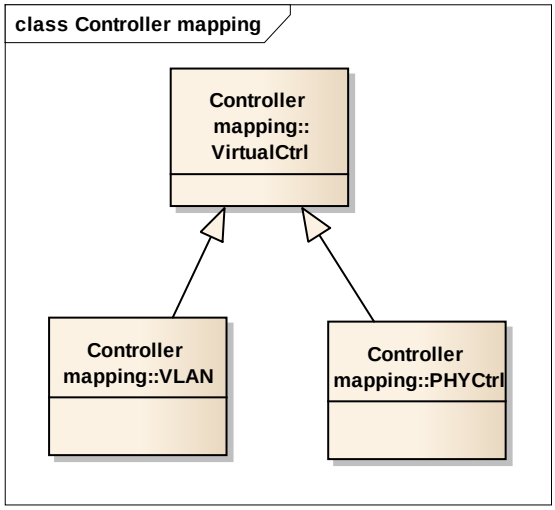

Figure 3-2 Virtual Controllers

During configuration the user specifies the type of the virtual controller. If the virtual controller is directly mapped onto a physical controller, all frames are received that do not have an IEEE802.1Q VLAN tag. The user is allowed to set up only one virtual controller per physical controller that uses a direct mapping.

If the virtual controller is from type VLAN, the user specifies the VLAN Identifier (VID) during configuration time. The VID specifies which VLAN tagged frames are forwarded to the respective virtual controller.

During runtime the priority (PCP) can be inserted into the VLAN tag. The result of providing an explicit priority is a different handling of the frame on Ethernet driver level. Depending on the configuration the frame will be mapped to a so called traffic class and treated by the Ethernet driver with higher or lesser priority.

Note
Special treatment of priority is supported by Ethernet drivers with QoS (Quality of Service) support only.

Note: VID=0
When VLAN is enabled and VID is specified to zero the sent Ethernet frame does not belong to any VLAN. The appended IEEE802.1Q tag only specifies the priority of the frame.

During reception a virtual controller with VID=0 behaves similar to a virtual controller that is directly mapping onto a physical controller. All frames without a VLAN tag are received. Additionally frames with VLAN.VID=0 are received.
The user is only allowed to set up one virtual controller per physical controller that is directly mapped onto a physical controller or that has a VID=0.

3.6 Zero-Copy extensions

The zero-copy extensions allow handling Ethernet-frame reception and transmission without the need for copying relevant data of an Ethernet frame that shall be used after leaving the reception or transmission context.

Note
The feature must be supported by the Vector Ethernet driver used.
See technical reference of the respective driver for more about the support.

3.6.1 Functional description

The zero-copy extensions comprise two different core features:

- On reception path an explicit unlock mechanism is provided. Usually all reception buffers are released right after the receive indication callback is called in interrupt context of the receive interrupt. The application (or any higher layers) would be forced to consume the data immediately in interrupt context.
  With the additional unlock mechanism the buffers must be released explicitly by the application. The application may consume the data whenever it wants to.
- On transmission path the application (or any higher layers) can inject further transmission buffers that are hosted by its own. Usually all transmission buffers are hold by the driver. The application would ask about a buffer, fill it with data and transmit it. To fill the data into the buffer a copy routine would be necessary.

3.6.2 Reception path

During reception two kinds of buffers are differentiated:

- Implicitly released buffers and
- Explicitly released buffers.

The differentiation bases on either the EtherType of the containing Ethernet frame or the VLAN ID. In terms of configuration the integrator must setup so-called buffer filters. Each VLAN or EtherType buffer filter enables explicit unlocking of a specific class of Ethernet

frames. The absence of a buffer filter for frames with a certain EtherType or VLAN ID enables implicit buffer locking.

Example

Explicit unlocking for frames with EtherType 0xC0CA and VLAN ID 0x321 or VLAN ID 0x123 is required. Two buffer filters need to be configured in the configuration tool. The following screenshots show the configuration viewed by Vector Configurator 5.

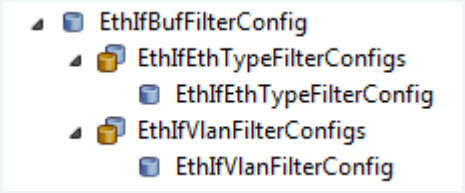

Figure 3-3 Screenshot of an EtherType and a VLAN buffer filter container

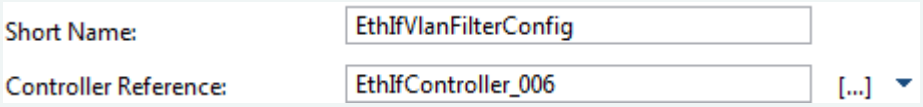

Figure 3-4 Screenshot of a VLAN ID buffer filter configuration.

The configuration shows a reference to an Ethernet Interface Controller configuration. The Controller Configuration specifies the VLAN ID 0x123.

Consequently all Ethernet frames with VLAN ID 0x123 must be explicitly unlocked.

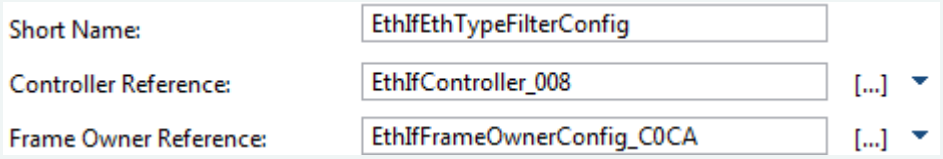

Figure 3-5 Screenshot of an EtherType buffer filter configuration.

The configuration shows a reference to a frame owner container with EtherType 0xC0CA. The reference to the controller restricts the buffer filter furthermore to an Ethernet Interface Controller Configuration, here with VLAN ID 0x321.

Consequently all frames received on the referenced Ethernet Controller with EtherType 0xC0CA and VLAN ID 0x321 must be explicitly unlocked.

Summarized a VLAN buffer filter identifies Ethernet frames on its `VLAN ID` only. An EtherType buffer filter identifies Ethernet frames that match the tuple `<EtherType, VLAN ID>`. An EtherType buffer filter is more restrictive than a VLAN buffer filter.

Buffers that require explicit unlocking, defined by configuration, require a call to `EthIf_ReleaseRxBuffer()`. The API call is allowed to be done at any time and context (interrupt or task). As far as the API call isn't performed the buffer does not participate on the reception process anymore. Be aware that a disadvantageous unlocking strategy may lead to a buffer underflow. In these situations the unlocking strategy must be reworked or the amount of receive buffers must be increased.

3.6.3 External transmission buffers

Further external transmission buffers need to be announced to the Ethernet Driver by configuration. The integrator must specify the maximum amount of external buffers that may be transmitted at the same time in the Ethernet driver configuration. Each buffer requires an administration structure that needs to be pre-allocated by the driver. If an underflow of pre-allocated administration structures occurs, no external buffers can be injected anymore until the transmission of another external buffer is completed.

Injecting external buffers is done via `EthIf_ProvideExtTxBuffer()` API. The API is comparable with `EthIf_ProvideTxBuffer()`, which provides an internal buffer. As far as one of the mentioned APIs returned a buffer pointer and index, these parameters are used for transmission and transmission callback.

Using an external buffer is described in 3 steps:

Practical Procedure

1. External buffer pointer and length must be provided via `EthIf_ProvideExtTxBuffer()`. The pointer to the buffer and its length are adapted by the function to point to and returning the actual length of the Ethernet frames payload (amount of bytes needed for Destination-/Source-MAC, Ether-Type and if needed the VLAN tag are taken into account).
   The API additionally returns a buffer index which is now used to identify the buffer. After the return of the function the buffer should not be modified anymore expect for the memory space addressable by the adapted pointer and length.
2. Call `EthIf_Transmit()` with the buffer index returned by `Eth_ProvideExtTxBuffer()` if transmission shall be triggered.
3. Wait until transmission callback (`<Ul>_TxConfirmation()`) was called for the buffer associated with the buffer index returned by `Eth_ProvideExtTxBuffer()`. The buffer is supposed to be successfully transmitted and may be reused from now on.

Note

Steps 1 to 3 must be processed in temporal order. No step must be omitted. In case `EthIf_ProvideExtTxBuffer()` was called and the buffer shall not be transmitted, call `EthIf_Transmit()` with a length parameter of 0. Afterwards step 3 is omitted.

For a more detailed description of the API parameters, refer to chapter 5.

3.7 Hardware Access APIs

The Hardware Access APIs allow access to the Ethernet frame header. Usually the Ethernet header is cut off during reception or is assembled during transmission by the Ethernet driver.

Note
The feature must be supported by the Vector Ethernet driver used.
See technical reference of the respective driver for more about the support.

To read the header information during reception or to overwrite the header that is assembled by the driver during transmission, further APIs are provided:

API	Description
`EthIf_GetTxHeaderPtr()`	Returns a pointer to the Ethernet transmission header. The API must be called after `EthIf_ProvideTxBuffer()` or `EthIf_ProvideExtTxBuffer()`. The pointer returned by ProvideTxBuffer is used to write the payload, the pointer returned by `EthIf_GetTxHeaderPtr()` is used to write the Ethernet header. Please refer to chapter 5 API Description to see the detailed signature of the function.
`EthIf_GetRxHeaderPtr()`	Returns a pointer to the first octet of a received Ethernet frame. After frame reception, the `<User>_RxIndication()` callback function is called with a pointer to the payload portion of a RX frame. That pointer is passed into the `EthIf_GetRxHeaderPtr()` function call and is decremented by the length of the Ethernet header. The returned pointer finally points to the beginning of the entire Ethernet frame. Please refer to chapter 5 API Description to see the detailed signature of the function.

Table 3-7 Hardware Access APIs

3.8 Wakeup Support

The Ethernet Interface supports the wakeup functionality needed for transceivers with wakeup capability either by the transceiver itself or by an activation line triggered by another ECU.

In case the transceiver performs the wakeup detection with the method “Wakeup by Interrupt” the Ethernet Interface must be involved for redirecting the wakeup detection call of the EcuM to the correct transceiver driver.

The wakeup detection is performed by a procedure, which involves multiple modules (Icu, EcuM, EthIf, EthTrcv). For a detailed description please refer to the Technical Reference of the transceiver driver used. Information about if the transceiver driver supports wakeup at all and how it is integrated is located there.

Note
The following procedure is only involved if the transceiver performs “Wakeup by Interrupt”.

The EthIf API `EthIf_CheckWakeup()` is called during the wakeup procedure by EcuM within an integration code. This integration code performs a mapping from the EcuM Wakeup Source related to the wakeup to the call of the function.

An example for an integration code that must be placed into the generated files of EcuM can be found in the Technical Reference of the transceiver driver.

Within the function the EthIf will redirect the call to the corresponding transceiver driver according to a map, which is provided by configuration. For further details to this so called wakeup map please refer 6.1.1 Wakeup Support.

3.9 Extended Traffic Handling

The Ethernet Interface supports extended traffic handling mechanisms that allow either to mirror frames received or transmitted on a specific EthIf controllers. Additionally traffic can be routed from one EthIf controller to another.

This section describes the functionality of the supported traffic handling mechanisms. For a detailed description on how to configure the features please refer to 6.1.2.

3.9.1 Traffic Mirroring

The traffic mirroring feature allows to capture receive and transmission traffic of EthIf controllers and outputs the traffic on an Ethernet controller.

The mirrored traffic can be selected on an EthIf controller level and is parted into receive and transmit traffic. This allows for example to only capture received frames or transmitted frames for one or more EthIf controllers.

The frames itself aren’t modified by neither the EthIf nor the underlying hardware to achieve a real mirroring behavior. So if there is for example a VLAN tag contained in the frame to be mirrored the same VLAN tag is contained in the mirrored frame. Same for all other header and payload information of the Ethernet frame. However it is possible to change the behavior with respect to the source MAC address. The source MAC address can be exchanged by the source MAC address of the Ethernet controller used as mirroring destination (configuration option).

The following illustration shows how a mirroring use case could look like.

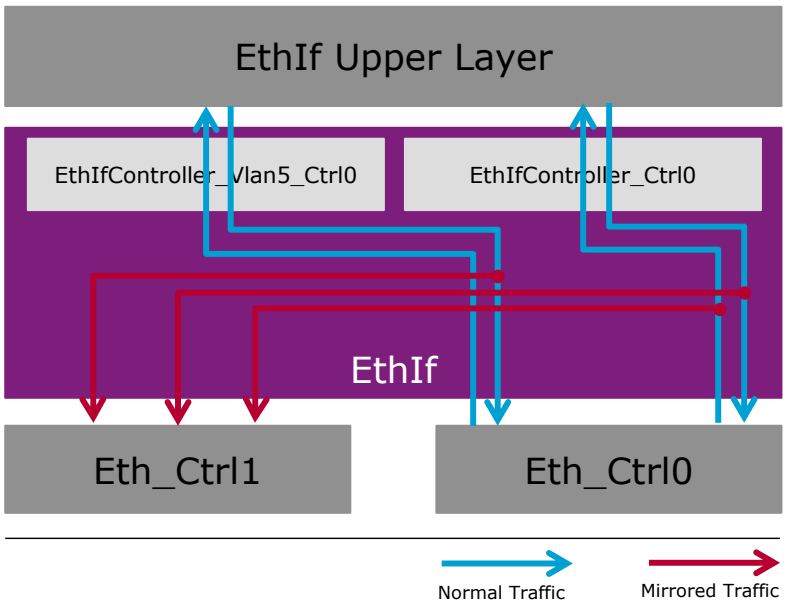

Figure 3-6 Traffic Mirroring

The frames transmitted and received over EthIfController_Ctrl0 are mirrored to Eth_Ctrl1. For EthIfController_Vlan5_Ctrl0 only the transmitted frames are taken into account for mirroring.

3.9.2 Traffic Gateway

The traffic gateway feature allows to route traffic from one EthIf controller to another. The feature can be used to route traffic from one Ethernet controller managed by the ECU to another Ethernet controller (e.g. used for media conversion, 100BaseTx <-> PLC, 100BaseTx <-> BroadR-Reach, and so on). Because the routing is based on EthIf controllers additionally VLAN tags can be inserted, removed or modified dependent on the EthIf controllers involved in the traffic gateway.

The mechanism is based on traffic gateway routes which define how packets will be redirected. Every route involves exactly two EthIf controllers. Additionally the EthIf controllers can only be used by one route exclusively.

The following illustration shows an example of a traffic gateway route.

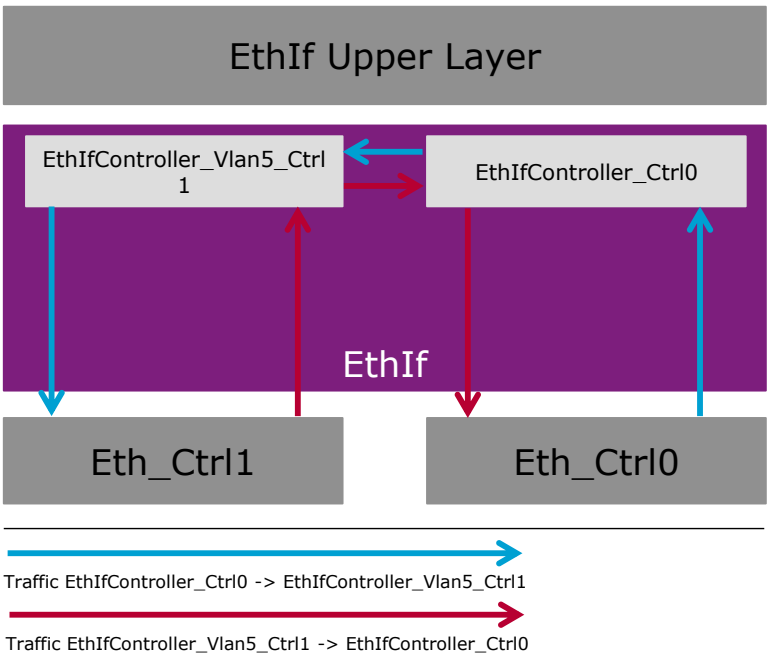

Figure 3-7 Traffic Gateway Route

The traffic gateway route involves the EthIf controllers EthIfController_Vlan5_Ctrl1 and EthIfController_Ctrl0 and the underlying Ethernet controllers Eth_Ctrl0 and Eth_Ctrl1.

EthIfController_Vlan5_Ctrl1 represents tagged traffic with VLAN ID 5 transferred over Eth_Ctrl1 and EthIfController_Ctrl0 represents untagged traffic transferred over Eth_Ctrl0.

Traffic directed to EthIfController_Ctrl0 (which is untagged traffic received on Eth_Ctrl0) will be redirected by the traffic gateway route to EthIfController_Vlan5_Ctrl1. EthIfCtonroller_Vlan5_Ctrl1. EthIfController_Vlan5_Ctrl1 will tag it with VLAN ID 5 and transmit it over Eth_Ctrl1. The same mechanism applies for traffic received on Eth_Ctrl1 which is designated for EthIfController_Vlan5_Ctrl1. However the mechanism will work vice versa. The VLAN tag will be removed and the frame is routed to Eth_Ctrl0.

As soon as an EthIf controller is involved into a traffic gateway route the traffic received will not be passed to an upper layer module of EthIf anymore. To avoid this behavior for special upper layers (e.g. QCA7000 driver) that must communicate on the same EthIf controller a MAC source address black list can be defined.

The following illustration shows an example of how the mechanism works.

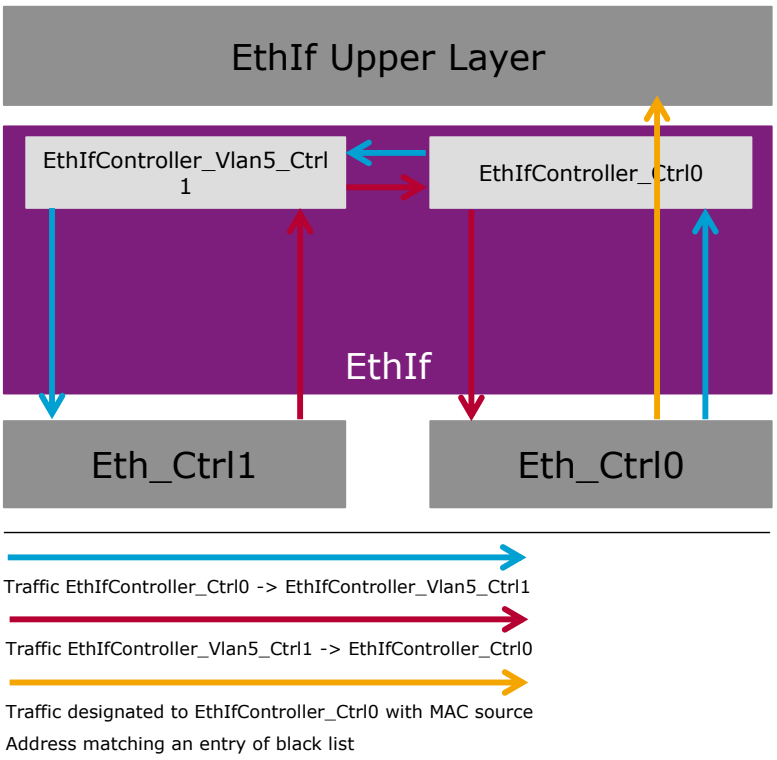

Figure 3-8 Traffic Gateway Route with Source MAC address Black List match

The traffic gateway route operates like in the example presented before. However if a frame designated to EthIfController_Ctrl0 with a source MAC address matching an entry of the black list is received it will not be routed to EthIfController_Vlan5_Ctrl1 but processed and if possible passed to an EthIf upper layer.

3.10 Lax Link Aggregation

The Lax Link Aggregation feature can be used to lower the requirements for a link on EthIf Controller level. This section describes how a link on EthIf Controller level is defined if the feature is activated and which conditions must be met for a link up/down.

Note
The feature is feasible in an Ethernet Switch use-case only. Therefore it is not possible to activate it if the underlying hardware configuration of the EthIf Controller doesn't contain any Ethernet Switch related elements.

The common link aggregation mechanism defines a link as a conjunction of the link state of all physical layer elements (i.e. Ethernet Transceivers and Ethernet Switch Ports), mapped to an EthIf Controller. That is to say that each physical layer element has to report an active link to have a link on EthIf Controller level and the link gets lost if only one physical layer element reports a link down.

With the feature active for an EthIf Controller, the link for this controller is defined in a different way. It is defined as a possible communication path between at least two nodes connected to an Ethernet Switch. Is this minimal communication path ensured (by an established link on at least two Ethernet Switch Ports) the network is considered as ‘communication ready’. This possibility of at least two nodes communicating with each other is provided to the upper layers as link.

Note
The feature is disabled by default and must be explicitly enabled for each EthIf Controller by setting the parameter `EthIfEnableLaxLinkAggregation`.

3.11 Ethernet Switch Frame Management

This section describes the ability to retrieve and apply frame management information for received and transmitted Ethernet frames that are or must be treated special by an Ethernet Switch.

This management information contains data like the switch port the frame was received on or shall be directed to.

Note
The feature is highly related to the Ethernet Switch drivers’ abilities. If the feature is supported and enabled in the Ethernet Switch driver, EthIf implicitly enables it too. Therefore there is no need to explicitly enable it within the EthIf configuration.

3.11.1 Retrieval of Frame Management Information

The retrieval of frame management information is done by an indication mechanism.

An upper layer is able to configure a callout function in the Ethernet Interface, which is called every time management information for an Ethernet frame previously received is provided.

This callout is configured by creating an `EthIfSwitchMgmtInfoIndicationConfig` container in the configuration tool and defining the callout function with the parameter `EthIfSwitchMgmtInfoIndicationFunction`.

The following sequence diagram shows the reception process and the provision of the frame management information.

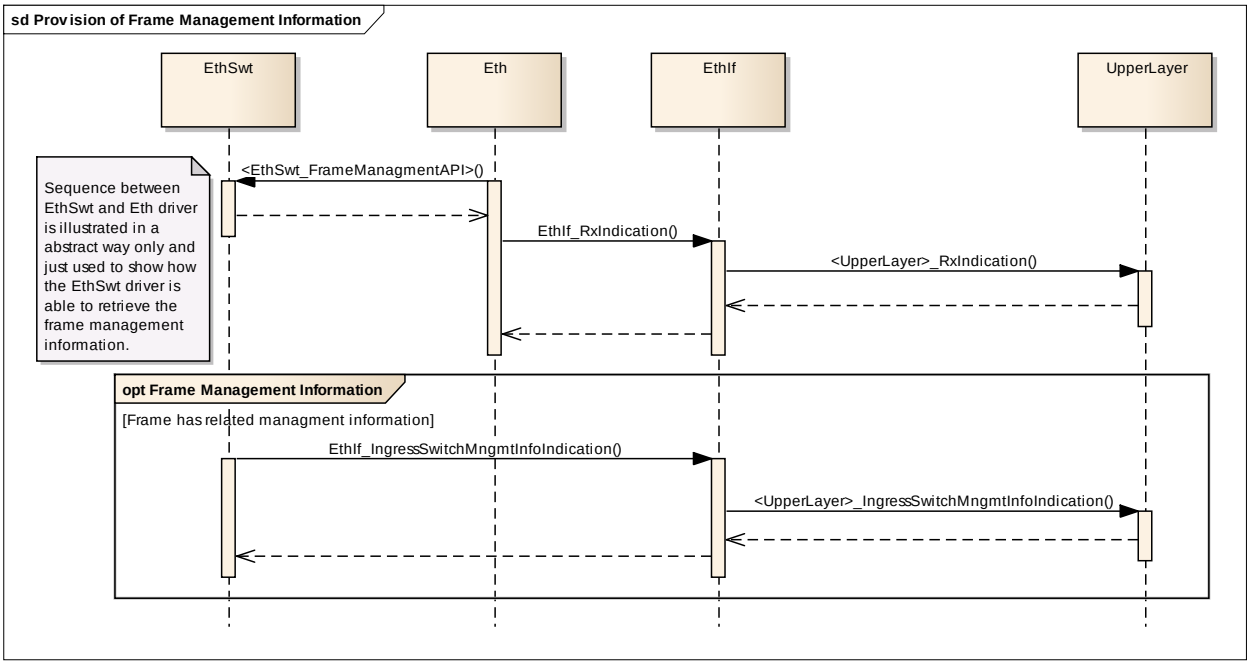

Figure 3-9 Sequence Diagram: Retrieving Frame Management Information

The reception of a frame is issued to the Ethernet driver (by polling or interrupt). The Ethernet driver calls the Ethernet Switch drivers API to preprocess the frame for further handling during the reception process. If the frame has related management information the Ethernet Switch driver will store it for later use and inform the Ethernet driver if the frame must be processed further (if it also contains common payload data which must be provided to upper layers) or if the frame shall not be processed further (if it was intended for providing management information only).

The management information is then provided (if it was contained in the frame) in the context of the ‘Ingress Switch Management Information Indication’ callout context.

3.11.2 Setting Frame Management Information

In addition to the retrieval of frame management information an upper layer can also define which management information shall be applied to an Ethernet frame.

This ability allows, for example, sending the Ethernet frame on only one port of the Ethernet Switch and bypassing the common switching engine.

The following sequence diagram shows the transmission process allowing an upper layer to provide management information for a specific Ethernet frame.

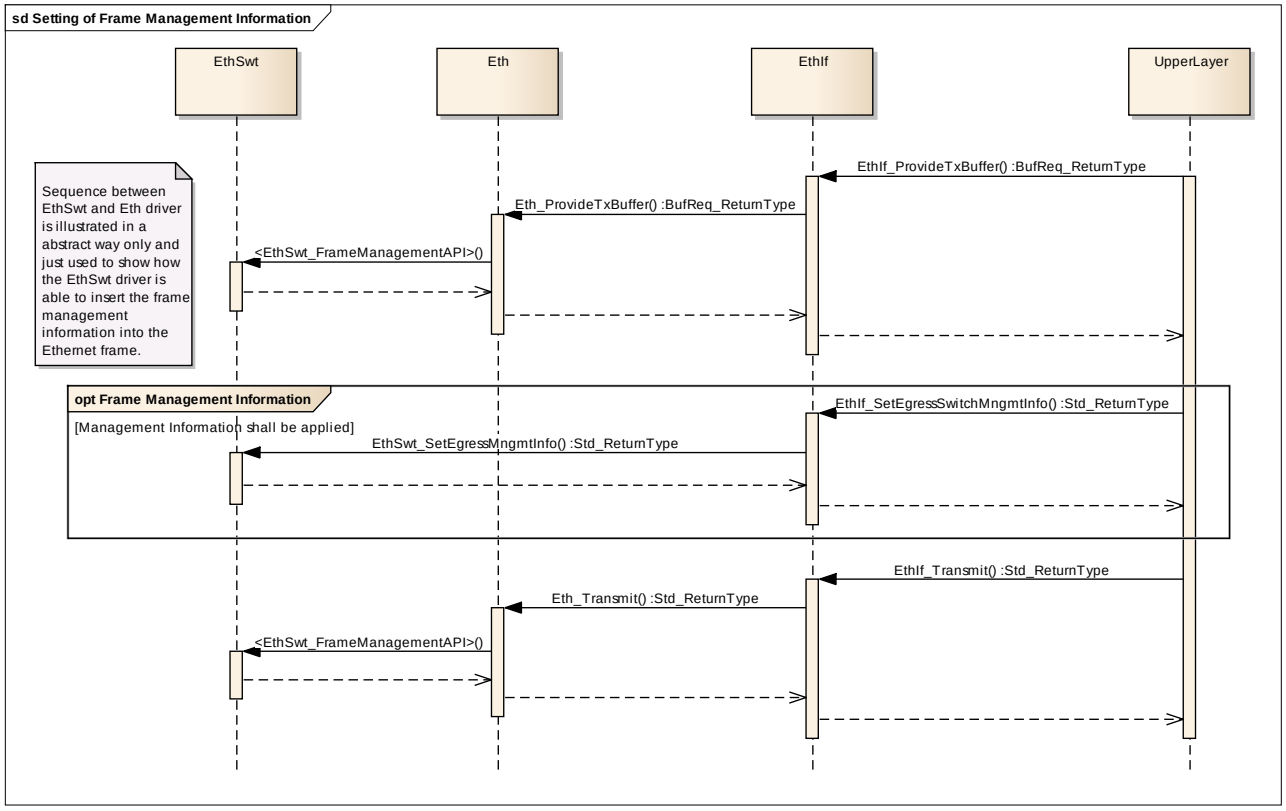

Figure 3-10 Sequence Diagram: Setting Frame Management Information

To provide management information for an Ethernet frame the upper layer has to call `EthIf_SetEgressSwitchMngmtInfo()` between the allocation of the Ethernet frame buffer (call of `EthIf_ProvideTxBuffer()`) and the actual transmission trigger (call of `EthIf_Transmit()`). The frame itself is identified by a tuple consisting of the EthIf controller index and the buffer index retrieved during `EthIf_ProvideTxBuffer()`.

Within the context of ‘Set Egress Switch Management Information’ the Ethernet Switch driver will prepare the frame (if management information must be inserted into it) or configure the Ethernet Switch (if management information must be applied on register level).

3.12 Ethernet Switch Time Synchronization

This section describes the ability to retrieve time stamps for Ethernet frames received and transmitted with the help of the management capabilities of Ethernet switches.

Note

This feature is only operable if Ethernet Switch Frame Management is used. So only Ethernet frames that are managed by the Ethernet Switch are able to be timestamped.

However it is possible that the Ethernet Switch hardware isn’t capable of taking time stamps for any kind of Ethernet frames but only for special frames like PTP frames.

3.12.1 Notification about time stamps for a frame transmitted

Users of the Ethernet Interface are able to get notified about the ingress and egress time stamp taken at the Ethernet Switch for managed Ethernet frames transmitted from the Host-CPU.

Therefore the user has to implement callout functions (see 5.8.2.2 and 5.8.2.3) that must be provided during configuration. To request the time stamps for an Ethernet frame during runtime the user has to call `EthIf_SwitchEnableEgressTimeStamp()` during buffer provision and frame transmission context. If transmission has finished and the underlying driver has retrieve the time stamps the Ethernet Interface will notify the user with the help of the previously noted callouts.

The following sequence diagram shows the transmission process, the request for timestamping and the notifications about the retrieved time stamps for an Ethernet frame on transmission.

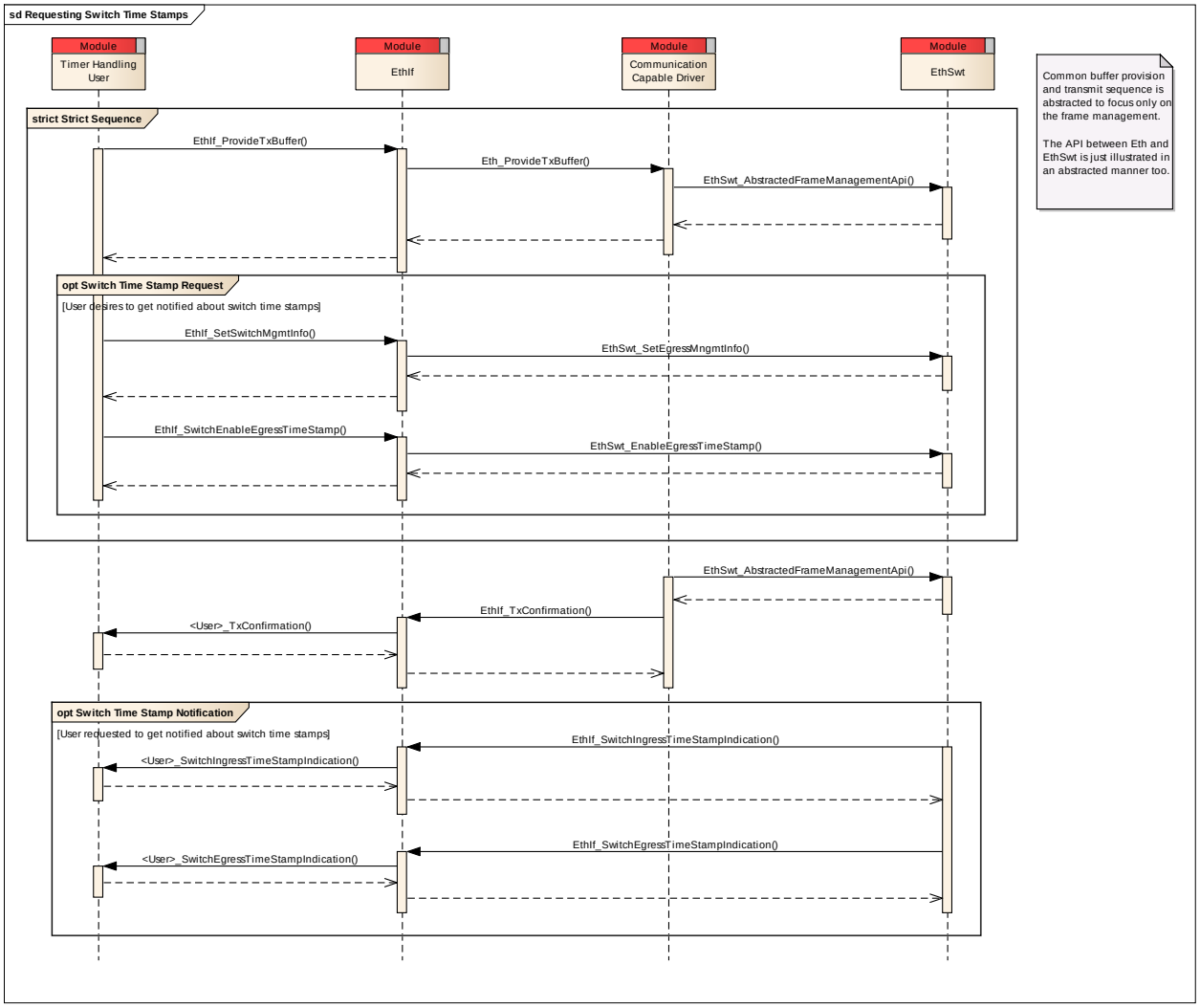

Figure 3-11 Sequence Diagram: Notification about switch time stamps for a frame transmitted

3.12.2 Notification about time stamps for a frame received

Users of the Ethernet Interface are able to get notified about the ingress and egress time stamp taken at the Ethernet Switch for managed Ethernet frames received on the Host-CPU.

Therefore the user has to implement callout functions (see 5.8.2.2 and 5.8.2.3) that must be provided during configuration. Each Ethernet frame that is handled like mentioned in section 3.11 and the hardware is able to take time stamps for results in having the Ethernet Interface notifying the user with the help of the previously noted callouts about the available time stamps.

The following sequence diagram shows the reception process and the notification about the retrieved time stamps for an Ethernet frame on reception.

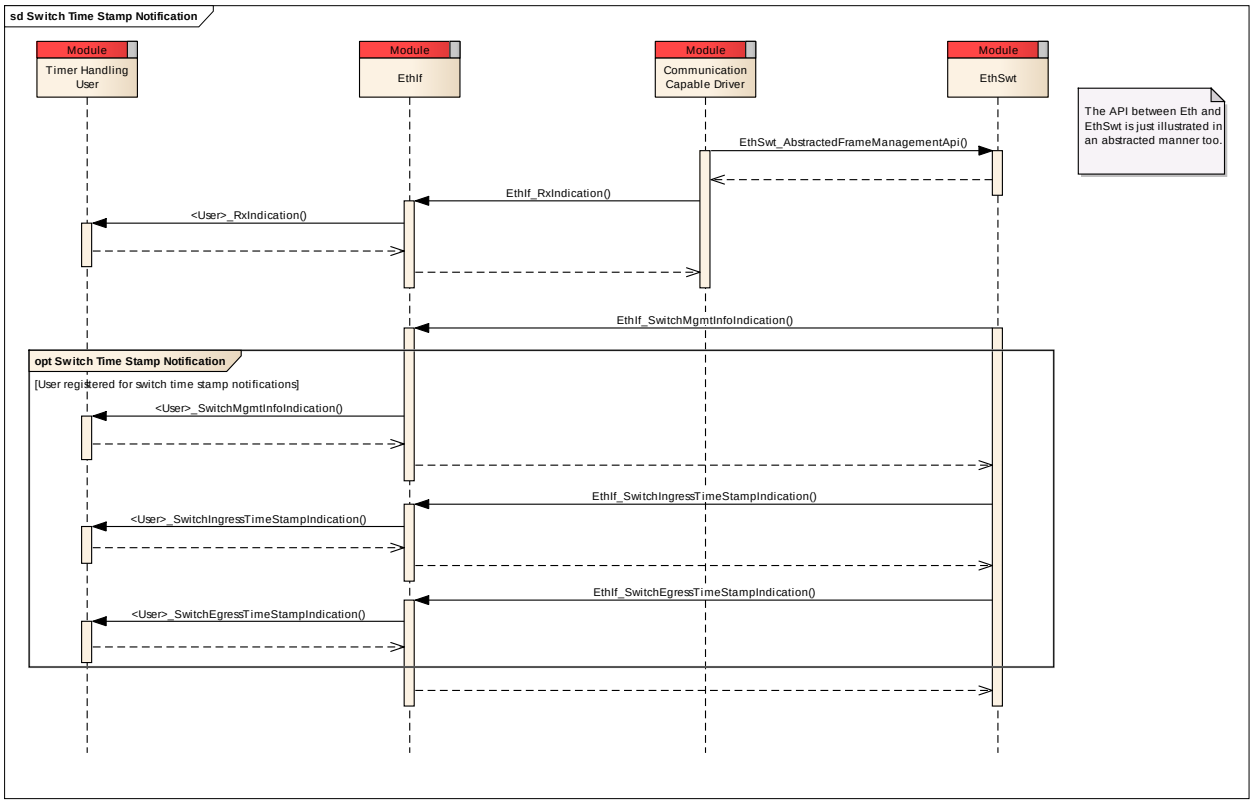

Figure 3-12 Sequence Diagram: Notification about switch time stamps for a frame received

3.13 Ethernet Switch Multicast to Port Assignment manipulation

Commonly multicasts are handled by Ethernet Switches like broadcast. They are flooded to all ports. However, in some special use-cases this behavior isn't feasible because a respective multicast should only be communicated on specific switch ports.

This feature allows changing this behavior from switching from the flooding mechanism to a dedicated communication of multicast on specific ports and vice versa by calling the API `EthIf_UpdateMCastPortAssignment()` with the respective information.

Note
The feature is implicitly enabled in the Ethernet Interface if it is enabled in the used Ethernet Switch driver. The parameter in the driver is called `EthSwtUpdateMCastPortAssignmentApi`.

3.14 Ethernet Firewall Support

The Ethernet Interface allows interfacing with the Vector specific BSW EthFw. This module implements a firewall. For further details on capabilities of the firewall and its configuration please see [6].

If the firewall is used each packet triggered for transmission or received is passed to it for transmission- or reception-check. The EthIf will only send or receive the frame if the firewall allows the EthIf to do so. Otherwise the frame will be silently discarded.

Note
The Ethernet firewall support in EthIf is enabled as soon as the EthFw selects at least one `EthIfController` with its configuration parameter `EthFwControllerRef`.

3.15 Ethernet Tx/Rx Statistics

The Ethernet interface allows retrieving transmission and reception statistics on an EthIf controller (VLAN) level.

Therefore one can utilize the APIs `EthIf_GetTxStats()` and `EthIf_GetRxStats()`. The counters that can be retrieved with respect to the EthIf controller are provided by the types `EthIf_TxStatsType` and `EthIf_RxStatsType`. For types and API please see 5.1 and 5.2.

Note
When retrieving the statistics they are cleared. So if the statistics shall be tracked over time one has to memorize the overall statistics and accumulate the latest statistics by himself.

The statistics have a limited value range. If they aren't retrieved and therefore cleared before this limit is reached they will be set to a special value (see 5.1 EthIf_TxStatsType/EthIf_RxStatsType for details) indicating the overflow and aren't updated anymore.

3.16 Error Handling

3.16.1 Default Error Tracing

Default errors are reported to DET using the service `Det_ReportError()` (specified in [3]), if this feature is enabled in configuration tool.

The reported Ethernet Interface ID is 65.

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

Service ID	Service ID Define	Service
0x01	ETHIF_SID_INIT	EthIf_Init()
0x02	ETHIF_SID_CONTROLLER_INIT	EthIf_ControllerInit()
0x03	ETHIF_SID_SET_CONTROLLER_MODE	EthIf_SetControllerMode()
0x04	ETHIF_SID_GET_CONTROLLER_MODE	EthIf_GetControllerMode()
0x05	ETHIF_SID_TRANSCEIVER_INIT	EthIf_TransceiverInit()
0x06	ETHIF_SID_SET_TRANSCEIVER_MODE	EthIf_SetTransceiverMode()
0x07	ETHIF_SID_GET_TRANSCEIVER_MODE	EthIf_GetTransceiverMode()
0x08	ETHIF_SID_GET_PHYS_ADDR	EthIf_GetPhysAddr()
0x09	ETHIF_SID_PROVIDE_TX_BUFFER	EthIf_ProvideTxBuffer()
0x0A	ETHIF_SID_TRANSMIT	EthIf_Transmit()
0x0B	ETHIF_SID_GET_VERSION_INFO	EthIf_GetVersionInfo()
0x0C	ETHIF_SID_UPDATE_PHYS_ADDR_FILTER	EthIf_UpdatePhysAddrFilter()
0x0D	ETHIF_SID_SET_PHYS_ADDR	EthIf_SetPhysAddr()
0x0E	ETHIF_SID_RELEASE_RX_BUFFER	EthIf_ReleaseRxBuffer()
0x0F	ETHIF_SID_PROVIDE_EXT_TX_BUFFER	EthIf_ProvideExtTxBuffer()
0x10	ETHIF_SID_RX_INDICATION	EthIf_RxInidcation()
0x11	ETHIF_SID_TX_CONFIRMATION	EthIf_TxConfirmation()
0x20	ETHIF_SID_MAIN_FUNCTION_RX	EthIf_MainFunctionRx()
0x21	ETHIF_SID_MAIN_FUNCTION_TX	EthIf_MainFunctionTx()
0x22	ETHIF_SID_GET_CURRENT_TIME	EthIf_GetCurrentTime()
0x23	ETHIF_SID_ENABLE_EGRESS_TIMESTAMP	EthIf_EnableEgressTimestamp()
0x24	ETHIF_SID_GET_EGRESS_TIMESTAMP	EthIf_GetEgressTimestamp()
0x25	ETHIF_SID_GET_INGRESS_TIMESTAMP	EthIf_GetIngressTimestamp()
0x26	ETHIF_SID_SET_CORRECTION_TIME	EthIf_SetCorrectionTime()
0x27	ETHIF_SID_SET_GLOBAL_TIME	EthIf_SetGlobalTime()
0x28	ETHIF_SID_GET_PORT_MAC_ADDR	EthIf_GetPortMacAddr()
0x29	ETHIF_SID_GET_ARL_TABLE	EthIf_GetArlTable()
0x2A	ETHIF_SID_GET_BUFFER_LEVEL	EthIf_GetBufferLevel()
0x2B	ETHIF_SID_GET_DROP_COUNT	EthIf_GetDropCount()
0x2C	ETHIF_SID_STORE_CONFIGURATION	EthIf_StoreConfiguration()

Service ID	Service ID Define	Service
0x2D	ETHIF_SID_RESET_CONFIGURATION	EthIf_ResetConfiguration()
0x2E	ETHIF_SID_SET_TRANSCEIVER_WAKEUP_MODE	EthIf_SetTransceiverWakeupMode()
0x2F	ETHIF_SID_GET_TRANSCEIVER_WAKEUP_MODE	EthIf_GetTransceiverWakeupMode()
0x30	ETHIF_SID_CHECK_WAKEUP	EthIf_CheckWakeup()
0x40	ETHIF_SID_GET_TX_HEADER_PTR	EthIf_GetTxHeaderPtr()
0x41	ETHIF_SID_GET_RX_HEADER_PTR	EthIf_GetRxHeaderPtr()
0x60	ETHIF_SID_SET_BANDWIDTH_LIMIT	EthIf_SetBandwidthLimit()
0x61	ETHIF_SID_GET_BANDWIDTH_LIMIT	EthIf_GetBandwidthLimit()
0x80	ETHIF_SID_SET_SWITCH_MGMT_INFO	EthIf_SetSwitchMgmtInfo()
0x81	ETHIF_SID_SWITCH_ENABLE_EGRESS_TIME_STAMP	EthIf_SwitchEnableEgressTimeStamp()
0x82	ETHIF_SID_SWITCH_MGMT_INFO_INDICATION	EthIf_SwitchMgmtInfoIndicaiton()
0x83	ETHIF_SID_SWITCH_EGRESS_TIME_STAMP_INDICATION	EthIf_SwitchEgressTimeStampIndication()
0x84	ETHIF_SID_SWITCH_INGRESS_TIME_STAMP_INDICATION	EthIf_SwitchIngressTimeStampIndication()
0xA0	ETHIF_SID_UPDATE_MCAT_PORT_ASSIGNMENT	EthIf_UpdateMCastPortAssignment()
0xB0	ETHIF_SID_GET_TX_STATS	EthIf_GetTxStats()
0xB1	ETHIF_SID_GET_RX_STATS	EthIf_GetRxStats()

Table 3-8 Mapping of service IDs to services

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

Error Code	Description
0x00	ETHIF_E_NO_ERROR No error occured
0x01	ETHIF_E_INV_CTRL_IDX API service was called with invalid controller index
0x02	ETHIF_E_INV_TRCV_IDX API service was called with invalid transceiver index
0x03	ETHIF_E_INV_PORT_GROUP_IDX API service was called with invalid port group index
0x04	ETHIF_E_NOT_INITIALIZED API service used without module initialization
0x05	ETHIF_E_INV_PARAM_POINTER API service used with invalid pointer parameter (NULL_PTR)
0x06	ETHIF_E_INV_PARAM API service used with invalid value for parameter
0x07	ETHIF_E_INIT_FAILED The service EthIf_Init() was called with an invalid configuration

Error Code	Description
0x08	ETHIF_E_INV_SWITCH_IDX API service was called with invalid switch index
0x09	ETHIF_E_INV_DRIVER_API_CALL API service can't be redirected to driver due to either configuration related lack or unsupported API
0x0A	ETHIF_E_INV_STATE API service processing leads to an invalid state of the module
0xFF	ETHIF_E_INTERNAL_ERROR Internal error occurred

Table 3-9 Errors reported to DET

3.16.2 Production Code Error Reporting

Production code related errors are reported to DEM using the service `Dem_ReportErrorStatus()` (specified in [4]).

Error Code	Description
None	-

Table 3-10 Errors reported to DEM

4 Integration

This chapter gives necessary information for the integration of the Ethernet Interface into an application environment of an ECU.

4.1 Scope of Delivery

Depending on the delivery type of the Ethernet Interface the static files described in chapter 4.1.1 or 4.1.2 are delivered. In both case the files described in 4.1.3 are delivered.

4.1.1 Static Files (Source Code Delivery)

The static files are not to be modified.

File Name	Description
EthIf.c	Implementation
EthIf_Time.c	Precision-Time-Protocol (PTP) implementation
EthIf.h	API declaration
EthIf_Time.h	Precision-Time-Protocol (PTP) API declaration
EthIf_Types.h	Data types declaration
EthIf_Cbk.h	API call-back declaration
EthIf_Priv.h	Component local macro and variable declaration
EthIf_CfgAccess_Int.h	Configuration access abstraction
EthIf_Switch.c	Ethernet Switch driver abstraction layer API implementation
EthIf_Switch.h	Ethernet Switch driver abstraction layer API declaration
EthIf_Switch_Cbk.h	EthIf API declaration provided to the Ethernet Switch driver
EthIf_ExtndTrafficHndl.h	API declaration of the extended traffic handling features
EthIf_ExtndTrafficHndl.c	Implementation of the extended traffic handling features
EthIf_ZeroCopy.h	API declaration of the zero copy features
EthIf_ZeroCopy.c	Implementation of the zero copy features

Table 4-1 Static files (source code delivery)

Do not edit manually
The static files must not be edited manually!

4.1.2 Static Files (Object Code Delivery)

The static files are not to be modified.

File Name	Description
libEthIf.a	Implementation
EthIf.h	API declaration
EthIf_Time.h	Precision-Time-Protocol (PTP) API declaration

File Name	Description
EthIf_Types.h	Data types declaration
EthIf_Cbk.h	API call-back declaration
EthIf_Switch.h	Ethernet Switch driver abstraction layer API declaration
EthIf_Switch_Cbk.h	EthIf API declaration provided to the Ethernet Switch driver
EthIf_ZeroCopy.h	API declaration of the zero copy features

Table 4-2 Static files (object code delivery)

Do not edit manually
The static files must not be edited manually!

4.1.3 Dynamic Files

The dynamic files can be modified.

File Name	Description
EthIf_Cfg.h	Pre-compile time parameter configuration
EthIf_Lcfg.h	Link-time parameter configuration declaration
EthIf_Lcfg.c	Link-time parameter configuration
EthIf_HwTypes.h	Header file providing access to the driver specific types header files.
EthIf_EthCtrlCfg.h	Header file containing declarations of data structures used to redirect calls to the underlying Ethernet Controller.
EthIf_EthCtrlCfg.c	Source file containing data structures used to redirect calls to the underlying Ethernet Controller driver.
EthIf_EthTrcvCfg.h	Header file containing defines used to redirect calls to the underlying Ethernet Transceiver driver in case there is only one driver used by EthIf.
EthIf_EthTrcvCfg.c	Source file containing data structures used to redirect calls to the underlying Ethernet Transceiver driver in case there are multiple drivers used by EthIf.
EthIf_EthSwtCfg.h	Header file containing defines used to redirect calls to the underlying Ethernet Switch driver in case there is only one driver used by EthIf.
EthIf_EthSwtCfg.c	Source file containing data structures used to redirect calls to the underlying Ethernet Switch driver in case there are multiple drivers used by EthIf.

Table 4-3 Dynamic files

Do not edit manually
The dynamic files must not be edited manually but be generated with the configuration tool to guarantee a valid configuration of the component!

4.2 Compiler Abstraction and Memory Mapping

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

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

Memory Mapping Sections \ Compiler Abstraction Definitions	ETHIF_CONST	ETHIF_VAR	ETHIF_CODE
ETHIF_START_SEC_CONST_UNSPECIFIED	■		
ETHIF_START_SEC_CONST_32BIT	■		
ETHIF_START_SEC_CONST_16BIT	■		
ETHIF_START_SEC_CONST_8BIT	■		
ETHIF_START_SEC_VAR_NOINIT_UNSPECIFIED		■	
ETHIF_START_SEC_VAR_NOINIT_32BIT		■	
ETHIF_START_SEC_VAR_NOINIT_16BIT		■	
ETHIF_START_SEC_VAR_NOINIT_8BIT		■	
ETHIF_START_SEC_VAR_ZERO_INIT_UNSPECIFIED		■	
ETHIF_START_SEC_CODE			■

Table 4-4 Compiler abstraction and memory mapping

4.3 Exclusive Areas

This section describes the exclusive areas utilized by the Ethernet Interface to ensure data consistency. All exclusive areas are listed in Table 4-5.

Note
For better readability the prefix `ETHIF_EXCLUSIVE_AREA` of the exclusive area names were removed and must be considered during integration.
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.

Exclusive Area	Description
`CTRL_INIT`	This exclusive area ensures the data consistency of the mode management related data structures during initialization. The length of the exclusive are is kept as short as possible by just protecting the access and modification of a semaphore managed by EthIf. Therefore no internal function calls have to be expected within the exclusive area.
`SET_CTRL_MODE`	This exclusive area ensures the data consistency of the mode management related data structures during mode change. The length of the exclusive are is kept as short as possible by just protecting the access and modification of a semaphore managed by EthIf. Therefore no internal function calls have to be expected within the exclusive area.
`TX_MIRROR_ELEMENT`	This exclusive area ensures the data consistency of the traffic mirroring related data structures.
`MGMT_RX_CTXT_POOL`	This exclusive area ensures the data consistency of the switch frame management related data structures during reception.
`SWT_TIME_STAMP_TX_CTXT_POOL`	This exclusive area ensures the data consistency of the switch time stamping related data structures during transmission.
`SWT_TIME_STAMP_RX_CTXT_POOL`	This exclusive area ensures the data consistency of the switch time stamping related data structures during reception.
`RXTX_STATS`	This exclusive area ensures consistency of the statistic counters provided for an EthIf-controller.

Table 4-5 Exclusive areas

5 API Description

5.1 Type Definitions

Type Name	C-Type	Description	Value Range
EthIf_ConfigType	void	Defines the Ethernet Interface configuration type	NULL_PTR Pre-compile or link-time configuration
			Pointer Pointer to Post-build configuration (Not supported)
EthIf_StateType	uint8	Defines all possible Ethernet Interface states	ETHIF_STATE_UNINIT Ethernet Interface not initialized
			ETHIF_STATE_INIT Ethernet Interface initialized

Table 5-1 Type definitions

EthIf_FrameHdrType

This structure contains prepared information of the MAC layer header of an Ethernet frame.

Struct Element Name	C-Type	Description	Value Range
DstMacAddrPtr	uint8*	Pointer to the destination MAC address of the Ethernet frame	Pointer Location of a byte array with 6 elements containing the parts of a MAC address. Element 0 contains the most significant part of the address and Element 5 the least significant part.
SrcMacAddrPtr	uint8*	Pointer to the source MAC address of the Ethernet frame	Pointer Location of a byte array with 6 elements containing the parts of a MAC address. Element 0 contains the most significant part of the address and Element 5 the least significant part.
EtherType	Eth_FrameType	EtherType of the Ethernet frame (VLAN – if present – already removed)	0x0000 – 0xFFFF without 0x8100 (VLAN-EtherType)
VlanId	uint16	VLAN ID of the Ethernet frame	0 – 4094 Frame is VLAN-tagged
			ETHIF_INV_VLAN_ID Frame isn't VLAN-tagged
Priority	uint8	VLAN Priority (PCP)	0 – 7 Frame is VLAN-tagged

			unspecified Frame isn't VLAN-tagged

Table 5-2 EthIf_FrameHdrType

EthIf_TxStatsType

This structure contains transmission statistic counters related to an EthIf-controller.

Struct Element Name	C-Type	Description	Value Range
NumTxPkts	uint32	Number of transmitted Ethernet frames	0 .. 4.294.967.293
			ETHIF_RXTX_STATS_COUNTER_OVERFLOW_VAL Counter has overflown since last retrieval.
			ETHIF_RXTX_STATS_INV_COUNTER_VAL Counter isn't available.
NumTxBytes	uint32	Number of transmitted bytes	0 .. 4.294.967.293
			ETHIF_RXTX_STATS_COUNTER_OVERFLOW_VAL Counter has overflown since last retrieval.
			ETHIF_RXTX_STATS_INV_COUNTER_VAL Counter isn't available.

Table 5-3 EthIf_TxStatsType

EthIf_RxStatsType

This structure contains reception statistic counters related to an EthIf-controller.

Struct Element Name	C-Type	Description	Value Range
NumRxPkts	uint32	Number of transmitted Ethernet frames	0 .. 4.294.967.293
			ETHIF_RXTX_STATS_COUNTER_OVERFLOW_VAL Counter has overflown since last retrieval.
			ETHIF_RXTX_STATS_INV_COUNTER_VAL Counter isn't available.
NumRxBytes	uint32	Number of transmitted bytes	0 .. 4.294.967.293
			ETHIF_RXTX_STATS_COUNTER_OVERFLOW_VAL Counter has overflown since last retrieval.
			ETHIF_RXTX_STATS_INV_COUNTER_VAL Counter isn't available.

Table 5-4 EthIf_RxStatsType

5.2 API Table

This section contains the description of the functions of the common API of the Ethernet Interface.

5.2.1 EthIf_InitMemory

Prototype	
`void EthIf_InitMemory (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Function for *_INIT_*-variable initialization.	
Particularities and Limitations	
Module is uninitialized. 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.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-5 EthIf_InitMemory

5.2.2 EthIf_Init

Prototype	
`void EthIf_Init (const EthIf_ConfigType *CfgPtr)`	
Parameter	
CfgPtr [in]	Configuration structure for initializing the module
Return code	
void	none
Functional Description	
Initializes the EthIf module.	
Particularities and Limitations	
Specification of module initialization CREQ-111162 SPEC-2393574 SPEC-2393566 > Interrupts are disabled.Module is uninitialized.EthIf_InitMemory has been called unless	

EthIf_ModuleInitialized is initialized by start-up code. Function initializes the module EthIf. It initializes all variables and sets the module state to initialized.
Call context
> TASK > This function is Synchronous > This function is Non-Reentrant

Table 5-6 EthIf_Init

5.2.3 EthIf_ControllerInit

Prototype	
`Std_ReturnType` **`EthIf_ControllerInit`** `(uint8 CtrlIdx, uint8 CfgIdx)`	
Parameter	
CtrlIdx [in]	EthIf controller index
CfgIdx [in]	Configuration index
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters or call interrupted pending operation
Functional Description	
Initializes a EthIf controller.	
Particularities and Limitations	
Module and drivers have been initialized Function initializes the EthIf controller addressed and redirects the call to the underlying hardware drivers mapped to the EthIf controller.	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-7 EthIf_ControllerInit

5.2.4 EthIf_SetControllerMode

Prototype	
`Std_ReturnType` **`EthIf_SetControllerMode`** `(uint8 CtrlIdx, Eth_ModeType CtrlMode)`	
Parameter	
CtrlIdx [in]	EthIf controller index
CtrlMode [in]	Mode that shall be applied: ETH_MODE_DOWN - shut down the EthIf controller ETH_MODE_ACTIVE - activate the EthIf controller

Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters or call interrupted pending operation
Functional Description	
Modifies the EthIf controller mode.	
Particularities and Limitations	
> Module and drivers have been initializedEthIf_ControllerInit() was called for the respective EthIf controller before Function alters the EthIf controller mode and redirects the call to the underlying hardware drivers mapped to the EthIf controller.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-8 EthIf_SetControllerMode

5.2.5 EthIf_GetControllerMode

Prototype	
`Std_ReturnType` **`EthIf_GetControllerMode`** `(uint8 CtrlIdx, Eth_ModeType *CtrlModePtr)`	
Parameter	
CtrlIdx [in]	EthIf controller index
CtrlModePtr [out]	Retrieved mode: ETH_MODE_DOWN - EthIf controller is turned of ETH_MODE_ACTIVE - EthIf controller is active
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the EthIf controller mode.	
Particularities and Limitations	
Module and drivers have been initialized Function retrieves the current EthIf controller mode.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-9 EthIf_GetControllerMode

5.2.6 EthIf_GetPhysAddr

Prototype	
`void EthIf_GetPhysAddr (uint8 CtrlIdx, uint8 *PhysAddrPtr)`	
Parameter	
CtrlIdx [in]	EthIf controller index
PhysAddrPtr [out]	MAC address retrieved
Return code	
void	none
Functional Description	
Retrieves the MAC address related to the EthIf controller.	
Particularities and Limitations	
Module and drivers have been initialized Function retrieves the MAC address that is used as source MAC address by the Ethernet controller mapped to the EthIf controller.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-10 EthIf_GetPhysAddr

5.2.7 EthIf_SetPhysAddr

Prototype	
`void EthIf_SetPhysAddr (uint8 CtrlIdx, const uint8 *PhysAddrPtr)`	
Parameter	
CtrlIdx [in]	EthIf controller index
PhysAddrPtr [in]	MAC address to use as source MAC address
Return code	
void	none
Functional Description	
Sets the MAC address related to the EthIf controller.	
Particularities and Limitations	
Module and drivers have been initialized Function alters the MAC address that is used as source MAC address by the Ethernet controller mapped to the EthIf controller.	
Call context	
> ANY	

> This function is Synchronous > This function is Reentrant

Table 5-11 EthIf_SetPhysAddr

5.2.8 EthIf_UpdatePhysAddrFilter

Prototype	
`Std_ReturnType` **`EthIf_UpdatePhysAddrFilter`** `(uint8 CtrlIdx, const uint8 *PhysAddrPtr, Eth_FilterActionType Action)`	
Parameter	
CtrlIdx [in]	EthIf controller index
PhysAddrPtr [in]	MAC address that shall be added/removed
Action [in]	Action that shall be applied on the MAC address filter: ETH_ADD_TO_FILTER - adapt filter to be able to receive frames with the given MAC address as destination MAC address ETH_REMOVE_FROM_FILTER - adapt filter to prevent reception of frames with the given MAC address as destination MAC address
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Modifies the receive MAC address filter related to the EthIf Controller.	
Particularities and Limitations	
Module and drivers have been initialized Function modifies the receive MAC address filter of the Ethernet controller mapped to the EthIf controller by adding/removing the MAC address to/from the receive MAC address filter.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-12 EthIf_UpdatePhysAddrFilter

5.2.9 EthIf_ProvideTxBuffer

Prototype	
`BufReq_ReturnType` **`EthIf_ProvideTxBuffer`** `(uint8 CtrlIdx, Eth_FrameType FrameType, uint8 Priority, uint8 *BufIdxPtr, Eth_DataType **BufPtr, uint16 *LenBytePtr)`	
Parameter	
CtrlIdx [in]	EthIf controller index
FrameType [in]	EtherType to insert into the Ethernet frame header

Priority [in]	Priority of the Ethernet frame, which is coded into the PCP of the IEEE802.3Q VLAN tag. If EthIf controller represents a physical data connection the priority is ignored.
BufIdxPtr [out]	Index to identify the acquired buffer
BufPtr [out]	Buffer the payload can be written to
LenBytePtr [in,out]	Buffer length: [in] - Length in byte needed for the payload, which shall be transmitted [out] - Length of the buffer that is provided in byte (has at least the size of the requested length needed for the payload)
Return code	
BufReq_ReturnType	BUFREQ_OK - success
	BUFREQ_E_NOT_OK - function has been called with invalid parameters
	BUFREQ_E_BUSY - all buffers are in use
	BUFREQ_E_OVFL - requested length is too large
Functional Description	
Provides a transmission buffer for an Ethernet frame.	
Particularities and Limitations	
Module and drivers have been initialized Function allows to acquire a buffer where a upper layer is able to insert the payload for the Ethernet frame.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-13 EthIf_ProvideTxBuffer

5.2.10 EthIf_VTransmit

Prototype
`Std_ReturnType EthIf_VTransmit (uint8 CtrlIdx, uint8 BufIdx, Eth_FrameType FrameType, boolean TxConfirmation, uint16 LenByte, uint8 *DstMacAddrPtr, uint8 *SrcMacAddrPtr)`

Parameter	
CtrlIdx [in]	EthIf controller index
BufIdx [in]	Index to identify the buffer for frame transmission
FrameType [in]	EtherType to insert into the Ethernet frame header
TxConfirmation [in]	Request for a transmission confirmation: FALSE - no confirmation desired TRUE - confirmation desired
LenByte [in]	Payload length to be transmitted
DstMacAddrPtr [in]	Destination MAC address
SrcMacAddrPtr [in]	Source MAC address: MAC address as defined by IEEE802.3 - using this MAC address as source MAC address NULL_PTR - using the Ethernet controllers MAC address as source MAC address

Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Triggers transmission of an Ethernet frame with a given source MAC address.	
Particularities and Limitations	
> Module and drivers have been initializedEthIf controller was set in ACTIVE state by EthIf_SetControllerMode()Buffer to be transmitted must be previously acquired by EthIf_ProvideTxBuffer()/EthIf_ProvideExtTxBuffer()	
Function triggers the transmission of an Ethernet frame identified by the buffer and using the provided MAC address as source MAC address of the Ethernet frame.	
Call context	
> ANY	
> This function is Synchronous	
> This function is Reentrant	

Table 5-14 EthIf_VTransmit

5.2.11 EthIf_Transmit

Prototype	
`Std_ReturnType EthIf_Transmit (uint8 CtrlIdx, uint8 BufIdx, Eth_FrameType FrameType, boolean TxConfirmation, uint16 LenByte, uint8 *PhysAddrPtr)`	
Parameter	
CtrlIdx [in]	EthIf controller index
BufIdx [in]	Index to identify the buffer for frame transmission
FrameType [in]	EtherType to insert into the Ethernet frame header
TxConfirmation [in]	Request for a transmission confirmation: FALSE - no confirmation desired TRUE - confirmation desired
LenByte [in]	Payload length to be transmitted
PhysAddrPtr [in]	Destination MAC address
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Triggers transmission of an Ethernet frame with the Ethernet controllers source MAC address.	
Particularities and Limitations	
> Module and drivers have been initializedEthIf controller was set in ACTIVE state by EthIf_SetControllerMode()Buffer to be transmitted must be previously acquired by EthIf_ProvideTxBuffer()/EthIf_ProvideExtTxBuffer()	
Function triggers the transmission of an Ethernet frame identified by the buffer and using the MAC address	

of the Ethernet controller as source MAC address of the Ethernet frame.
Call context
> ANY > This function is Synchronous > This function is Reentrant

Table 5-15 EthIf_Transmit

5.2.12 EthIf_SwitchPortGroupRequestMode

Prototype	
`Std_ReturnType` **`EthIf_SwitchPortGroupRequestMode`** `(EthIf_SwitchPortGroupIdxType PortGroupIdx, EthTrcv_ModeType PortMode)`	
Parameter	
PortGroupIdx [in]	Index of the port group within the context of the Ethernet Interface
PortMode [in]	Request for the EthIfSwtPortGroup ETHTRCV_MODE_DOWN - disable the port group ETHTRCV_MODE_ACTIVE - enable the port group
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - port group mode could not be changed
Functional Description	
Requests a mode for the EthIfSwtPortGroup.	
Particularities and Limitations	
Module and drivers have been initialized Function requests a mode for the EthIfSwtPortGroup. The call shall be forwarded to EthSwt by calling EthSwt_SetSwitchPortMode for all EthSwtPorts referenced by the port group.	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 5-16 EthIf_SwitchPortGroupRequestMode

5.2.13 EthIf_SetTransceiverWakeupMode

Prototype	
`Std_ReturnType` **`EthIf_SetTransceiverWakeupMode`** `(EcuM_WakeupSourceType WakeupSource, EthTrcv_WakeupModeType WakeupMode)`	
Parameter	
WakeupSource [in]	EcuM wakeup source
WakeupMode [in]	Wakeup mode to set

Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Changes the wakeup mode of the related hardware driver.	
Particularities and Limitations	
Module and drivers have been initialized Function allows to change the wakeup mode of the related hardware driver by redirecting the call depending on the passed EcuM wakeup source.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-17 EthIf_SetTransceiverWakeupMode

5.2.14 EthIf_GetTransceiverWakeupMode

Prototype	
`Std_ReturnType EthIf_GetTransceiverWakeupMode (EcuM_WakeupSourceType WakeupSource, EthTrcv_WakeupModeType *WakeupModePtr)`	
Parameter	
WakeupSource [in]	EcuM wakeup source
WakeupModePtr [out]	Pointer pointing to variable where the wakeup mode is stored to
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the wakeup mode of the related hardware driver.	
Particularities and Limitations	
Module and drivers have been initialized Function allows to retrieve the wakeup mode of the related hardware driver by redirecting the call depending on the passed EcuM wakeup source.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-18 EthIf_GetTransceiverWakeupMode

5.2.15 EthIf_CheckWakeup

Prototype	
`void EthIf_CheckWakeup (EcuM_WakeupSourceType WakeupSource)`	
Parameter	
WakeupSource [in]	EcuM wakeup source
Return code	
void	none
Functional Description	
Initiates the wakeup check.	
Particularities and Limitations	
Module and drivers have been initialized Function allows to initiate the wakeup check of the related hardware driver by redirecting the call depending on the passed EcuM wakeup source.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-19 EthIf_CheckWakeup

5.2.16 EthIf_VSetTransceiverWakeupMode

Prototype	
`Std_ReturnType EthIf_VSetTransceiverWakeupMode (uint8 CtrlIdx, EthTrcv_WakeupModeType WakeupMode)`	
Parameter	
CtrlIdx [in]	EthIf controller index
WakeupMode [in]	Wakeup mode to set
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Changes the wakeup mode of the related hardware drivers.	
Particularities and Limitations	
Module and drivers have been initialized Function allows to change the wakeup mode of the related hardware drivers by redirecting the call depending on the passed EcuM wakeup source.	
Call context	
> ANY > This function is Synchronous	

> This function is Reentrant

Table 5-20 EthIf_VSetTransceiverWakeupMode

5.2.17 EthIf_VGetTransceiverWakeupMode

Prototype	
`Std_ReturnType` **`EthIf_VGetTransceiverWakeupMode`** `(uint8 CtrlIdx, EthTrcv_WakeupModeType *WakeupModePtr)`	
Parameter	
CtrlIdx [in]	EthIf controller index
WakeupModePtr [out]	Pointer pointing to variable where the wakeup mode is stored to
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the wakeup mode of the related hardware driver.	
Particularities and Limitations	
Module and drivers have been initialized Function allows to retrieve the wakeup mode of the related hardware driver by redirecting the call depending on the passed EcuM wakeup source.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-21 EthIf_VGetTransceiverWakeupMode

5.2.18 EthIf_VCheckWakeup

Prototype	
`void` **`EthIf_VCheckWakeup`** `(uint8 CtrlIdx)`	
Parameter	
CtrlIdx [in]	EthIf controller index
Return code	
void	none
Functional Description	
Initiates the wakeup check.	
Particularities and Limitations	
Module and drivers have been initialized	

Function allows to initiate the wakeup check of the related hardware drivers by redirecting the call depending on the passed EthIf controller index.
Call context
> ANY > This function is Synchronous > This function is Reentrant

Table 5-22 EthIf_VCheckWakeup

5.2.19 EthIf_SetBandwidthLimit

Prototype	
`Std_ReturnType` **`EthIf_SetBandwidthLimit`** `(uint8 CtrlIdx, uint8 QueuePrio, uint32 BandwidthLimit)`	
Parameter	
CtrlIdx [in]	EthIf virtual controller index
QueuePrio [in]	Traffic queue index
BandwidthLimit [in]	New bandwidth limit [bit/s]
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Manipulates the maximum bandwidth of a traffic queue.	
Particularities and Limitations	
Module and drivers have been initialized Function allows to manipulate the maximum amount of bandwidth the indexed traffic queue is allowed to acquire.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-23 EthIf_SetBandwidthLimit

5.2.20 EthIf_GetBandwidthLimit

Prototype	
`Std_ReturnType` **`EthIf_GetBandwidthLimit`** `(uint8 CtrlIdx, uint8 QueuePrio, uint32 *BandwidthLimitPtr)`	
Parameter	
CtrlIdx [in]	EthIf virtual controller index

QueuePrio [in]	Traffic queue index
BandwidthLimitPtr [out]	Current bandwidth limit [bit/s]
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the current maximum bandwidth of a traffic queue.	
Particularities and Limitations	
Module and drivers have been initialized Function allows to retrieve the maximum amount of bandwidth the indexed traffic queue is allowed to acquire currently.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-24 EthIf_GetBandwidthLimit

5.2.21 EthIf_GetTxStats

Prototype	
`Std_ReturnType` **`EthIf_GetTxStats`** `(uint8 CtrlIdx, Eth_TxStatsType *EthTxStats, EthIf_TxStatsType *EthIfTxStats)`	
Parameter	
CtrlIdx [in]	EthIf controller identifier
EthTxStats [out]	Transmission statistics of the respective Eth-controller
EthIfTxtSats [out]	Transmission statistics of the EthIf-controller
Return code	
Std_ReturnType	E_NOT_OK - Wrong parameters or Eth-driver call isn't successful
Std_ReturnType	E_OK - Statistics could be retrieved
Functional Description	
Retrieves the transmission statistic counters.	
Particularities and Limitations	
> Module is initiliazedRespective EthIf controller is initialized Function redirects call to the Eth-driver to retrieve the transmission statistic counters defined by AUTOSAR (see Eth-driver for more details). EthIf extends these statistics by additional counters for the respective EthIf-controller. The counters provided by EthIf within the EthIf_TxStatsType are cleared on read.	
Call context	
> TASK > This function is Synchronous	

> This function is Non-Reentrant

Table 5-25 EthIf_GetTxStats

5.2.22 EthIf_GetRxStats

Prototype	
`Std_ReturnType EthIf_GetRxStats (uint8 CtrlIdx, Eth_RxStatsType *EthRxStats, EthIf_RxStatsType *EthIfRxStats)`	
Parameter	
CtrlIdx [in]	EthIf controller identifier
EthRxStats [out]	Reception statistics of the respective Eth-controller
EthIfRxtSats [out]	Reception statistics of the EthIf-controller
Return code	
Std_ReturnType	E_NOT_OK - Wrong parameters or Eth-driver call isn't successful
Std_ReturnType	E_OK - Statistics could be retrieved
Functional Description	
Retrieves the reception statistic counters.	
Particularities and Limitations	
> Module is initiliazedRespective EthIf controller is initialized Function redirects call to the Eth-driver to retrieve the reception statistic counters defined by AUTOSAR (see Eth-driver for more details). EthIf extends these statistics by additional counters for the respective EthIf-controller. The counters provided by EthIf within the EthIf_RxStatsType are cleared on read.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-26 EthIf_GetRxStats

5.2.23 EthIf_GetVersionInfo

Prototype	
`void EthIf_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	
Reception Main Function.	

Particularities and Limitations
> nonenonenonenone Main function to handle Ethernet frame reception in polling mode.
Call context
> TASK > TASK > TASK > ANY > This function is Synchronous > This function is Non-Reentrant

Table 5-27 EthIf_GetVersionInfo

5.3 Main Function API Table

This section contains the main functions provided by EthIf.

5.3.1 EthIf_MainFunctionRx

Prototype	
`void` **`EthIf_MainFunctionRx`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Reception Main Function.	
Particularities and Limitations	
none Main function to handle Ethernet frame reception in polling mode.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-28 EthIf_MainFunctionRx

5.3.2 EthIf_MainFunctionTx

Prototype	
`void` **`EthIf_MainFunctionTx`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Transmission confirmation Main Function.	
Particularities and Limitations	
none Main function to handle Ethernet frame transmission confirmation in polling mode.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-29 EthIf_MainFunctionTx

5.3.3 EthIf_MainFunctionState

Prototype	
`void` **`EthIf_MainFunctionState`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Link state supervision Main Function.	
Particularities and Limitations	
none Main function to monitor link state changes of the managed hardware elements.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-30 EthIf_MainFunctionState

5.4 Time API Table

This section contains the description of the functions of the Time API of the Ethernet Interface.

5.4.1 EthIf_GetCurrentTime

Prototype	
`Std_ReturnType` **`EthIf_GetCurrentTime`** `(uint8 CtrlIdx, Eth_TimeStampQualType *timeQualPtr, Eth_TimeStampType *timeStampPtr)`	
Parameter	
CtrlIdx [in]	EthIf Controller index
timeQualPtr [out]	Pointer to the buffer the quality of the time retrieved shall be stored to
timeStampPtr [out]	Pointer to the buffer the current time shall be stored to
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the current time of the Ethernet controllers timer module.	
Particularities and Limitations	
Module and drivers have been initialized Function redirects the call to the Ethernet controller driver to retrieve the current time of the controllers timer module.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-31 EthIf_GetCurrentTime

5.4.2 EthIf_SetGlobalTime

Prototype	
`Std_ReturnType` **`EthIf_SetGlobalTime`** `(uint8 CtrlIdx, const Eth_TimeStampType *TimestampPtr)`	
Parameter	
CtrlIdx [in]	EthIf Controller index
TimestampPtr [in]	Pointer to the buffer the current time shall be stored to
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters

Functional Description
Sets the current time of the Ethernet controllers timer module.
Particularities and Limitations
Module and drivers have been initialized Function redirects the call to the Ethernet controller driver to set the current time of the controllers timer module to a specific value.
Call context
> ANY > This function is Synchronous > This function is Reentrant

Table 5-32 EthIf_SetGlobalTime

5.4.3 EthIf_SetCorrectionTime

Prototype	
`Std_ReturnType` **`EthIf_SetCorrectionTime`** `(uint8 CtrlIdx, const Eth_TimediffType *OffsetTimePtr, const Eth_RateRatioType *RateRatioPtr)`	
Parameter	
CtrlIdx [in]	EthIf Controller index
OffsetTimePtr [in]	Offset to correct the time
RateRatioPtr [in]	Ratio used to de-/accelerate the time
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Corrects the time of the Ethernet controllers timer module.	
Particularities and Limitations	
Module and drivers have been initialized Function redirects the call to the Ethernet controller driver to correct the time of the timer module by a offset and/or de-/accelerate the time.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-33 EthIf_SetCorrectionTime

5.4.4 EthIf_EnableEgressTimestamp

Prototype	
`Std_ReturnType` **`EthIf_EnableEgressTimestamp`** `(uint8 CtrlIdx, uint8 BufIdx)`	
Parameter	
CtrlIdx [in]	EthIf Controller index
BufIdx [in]	Buffer index identifying the frame
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Enable timestamping of a frame in Ethernet controller.	
Particularities and Limitations	
Module and drivers have been initialized Function redirects the call to the Ethernet controller driver to enable egress timestamping of a frame.	
Call context	
> ANY - call only allowed between EthIf_ProvideTxBuffer()/EthIf_ProvideExtTxBuffer() and EthIf_Transmit() for a specific frame > This function is Synchronous > This function is Reentrant	

Table 5-34 EthIf_EnableEgressTimestamp

5.4.5 EthIf_GetEgressTimestamp

Prototype	
`Std_ReturnType` **`EthIf_GetEgressTimestamp`** `(uint8 CtrlIdx, uint8 BufIdx, Eth_TimeStampType *TimestampPtr, Eth_TimestampQualityType *TimestampQualityPtr)`	
Parameter	
CtrlIdx [in]	EthIf Controller index
BufIdx [in]	Buffer index identifying the frame
TimestampPtr [out]	Pointer to buffer for timestamp storage
TimestampQualityPtr [out]	Pointer to buffer for timestamp quality storage
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieve the egress timestamp of a frame.	
Particularities and Limitations	
Module and drivers have been initialized	

Function redirects the call to the Ethernet controller driver to retrieve the egress timestamp of a frame.
Call context
> TASK|ISR1|ISR2 - call only allowed in EthIf_TxConfirmation() > This function is Synchronous > This function is Reentrant

Table 5-35 EthIf_GetEgressTimestamp

5.4.6 EthIf_GetIngressTimestamp

Prototype	
`Std_ReturnType EthIf_GetIngressTimestamp (uint8 CtrlIdx, Eth_DataType *DataPtr, Eth_TimeStampType *TimestampPtr, Eth_TimestampQualityType *TimestampQualityPtr)`	
Parameter	
CtrlIdx [in]	EthIf Controller index
DataPtr [in]	Pointer to the frame payload for identifying the frame
TimestampPtr [out]	Pointer to buffer for timestamp storage
TimestampQualityPtr [out]	Pointer to buffer for timestamp quality storage
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieve the ingress timestamp of a frame.	
Particularities and Limitations	
Module and drivers have been initialized Function redirects the call to the Ethernet controller driver to retrieve the ingress timestamp of a frame.	
Call context	
> TASK|ISR1|ISR2 - call only allowed in EthIf_RxIndication() > This function is Synchronous > This function is Reentrant	

Table 5-36 EthIf_GetIngressTimestamp

5.5 Ethernet Switch Abstraction API Table

This section contains the description of the functions of the Ethernet Switch Abstraction API of the Ethernet Interface.

5.5.1 EthIf_GetPortMacAddr

Prototype	
`Std_ReturnType EthIf_GetPortMacAddr (const uint8 *MacAddrPtr, EthSwt_SwitchIdxType *SwitchIdxPtr, EthSwt_PortIdxType *PortIdxPtr)`	
Parameter	
MacAddrPtr [in]	MAC address to be queried
SwitchIdxPtr [out]	Index of the switch instance the corresponding frame was received on
PortIdxPtr [out]	Index of the port the corresponding frame was received on
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters or no port information found
Functional Description	
Retrieves the switch instance and port a MAC address is assigned to.	
Particularities and Limitations	
Module and drivers have been initialized Function allows to retrieve the switch instance and port a Ethernet frame with a source MAC address matching the passed MAC address was received on.	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 5-37 EthIf_GetPortMacAddr

5.5.2 EthIf_GetArlTable

Prototype	
`Std_ReturnType EthIf_GetArlTable (EthSwt_SwitchIdxType SwitchIdx, uint32 *LenPtr, EthSwt_MacVlanType *ArlTablePtr)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
in/out] [in]	LenPtr in: Size of the passed buffer the entries shall be written to out: Number of entries written into buffer
ArlTablePtr [out]	Pointer to the buffer the data shall be written to
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the complete address resolution table.	

Particularities and Limitations
Module and drivers have been initialized
Function allows to retrieve the valid entries of the address resolution table of a switch instance.
Call context
> ANY
> This function is Synchronous
> This function is Non-Reentrant

Table 5-38 EthIf_GetArlTable

5.5.3 EthIf_GetBufferLevel

Prototype	
`Std_ReturnType` **`EthIf_GetBufferLevel`** `(EthSwt_SwitchIdxType SwitchIdx, EthSwt_BufferLevelType *SwitchBufferLevelPtr)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
SwitchBufferLevelPtr [out]	The interpretation of this value is switch dependent
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Reads the buffer level of the currently used buffer of the switch.	
Particularities and Limitations	
Module and drivers have been initialized	
Function reads the buffer level of the currently used buffer of the switch.	
Call context	
> ANY	
> This function is Synchronous	
> This function is Non-Reentrant	

Table 5-39 EthIf_GetBufferLevel

5.5.4 EthIf_GetDropCount

Prototype	
`Std_ReturnType` **`EthIf_GetDropCount`** `(EthSwt_SwitchIdxType SwitchIdx, uint16 *LenPtr, uint32 *DropCountPtr)`	
Parameter	
SwitchIdx [in]	Index of the switch instance

LenPtr [in,out]	[in] - Size of the passed buffer the drop counts shall be written to [out] - Number of drop counts written into buffer
DropCountPtr [out]	Pointer to the buffer the data shall be written to
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the drop counts according to the AUTOSAR SWS.	
Particularities and Limitations	
Module and drivers have been initialized Function allows to retrieve the drop counts specified by the AUTOSAR SWS. Each count is the sum of the drop count of all ports.	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 5-40 EthIf_GetDropCount

5.5.5 EthIf_StoreConfiguration

Prototype	
`Std_ReturnType` **`EthIf_StoreConfiguration`** `(EthSwt_SwitchIdxType SwitchIdx)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Calls EthSwt_StoreConfiguration() API of the related EthSwt-driver.	
Particularities and Limitations	
Module and drivers have been initialized Function calls the EthSwt_StoreConfiguration() API of the related EthSwt-driver. Behavior depends on the implementation of the driver. Commonly the latest MAC/Port table retrieved out of the address resolution table of the switch is stored in NV RAM.	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 5-41 EthIf_StoreConfiguration

5.5.6 EthIf_ResetConfiguration

Prototype	
`Std_ReturnType` **`EthIf_ResetConfiguration`** `(EthSwt_SwitchIdxType SwitchIdx)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Calls EthSwt_ResetConfiguration() API of the related EthSwt-driver.	
Particularities and Limitations	
Module and drivers have been initialized Function calls the EthSwt_ResetConfiguration() API of the related EthSwt-driver. Behavior depends on the implementation of the driver. Commonly the MAC/Port table previously stored in NV RAM triggered by EthIf_StoreConfiguration() is invalidated and switching behavior with regard to MACs and VLANs is reset to initial (as defined by static configuration) behavior.	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 5-42 EthIf_ResetConfiguration

5.5.7 EthIf_SetSwitchMgmtInfo

Prototype	
`Std_ReturnType` **`EthIf_SetSwitchMgmtInfo`** `(uint8 EthIfCtrlIdx, uint8 BufIdx, const EthSwt_MgmtInfoType *MgmtInfo)`	
Parameter	
EthIfCtrlIdx [in]	Index of the EthIf controller
BufIdx [in]	Index of the Ethernet Tx buffer retrieved during EthIf_ProvideTxBuffer()
MgmtInfo [in]	Switch Management information
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Sets management information for a frame identified by the EthIf controller and the Ethernet buffer index.	
Particularities and Limitations	
Module and drivers have been initialized and buffer has to be acquired with EthIf_ProvideTxBuffer()	

Function allows to apply special treatment for an Ethernet frame. The frame is identified by the EthIf controller and the Ethernet buffer index. This function can only be called between a EthIf_ProvideTxBuffer() and EthIf_Transmit().
Call context
> ANY - call only allowed between EthIf_ProvideTxBuffer()/EthIf_ProvideExtTxBuffer() and EthIf_Transmit() for a specific frame > This function is Synchronous > This function is Non-Reentrant

Table 5-43 EthIf_SetSwitchMgmtInfo

5.5.8 EthIf_SwitchEnableEgressTimeStamp

Prototype	
`Std_ReturnType` **`EthIf_SwitchEnableEgressTimeStamp`** `(uint8 EthIfCtrlIdx, uint8 BufIdx, const EthSwt_MgmtInfoType *MgmtInfo)`	
Parameter	
EthIfCtrlIdx [in]	Index of the EthIf controller
BufIdx [in]	Index of the Ethernet Tx buffer retrieved during EthIf_ProvideTxBuffer()
MgmtInfo [in]	Switch Management information
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Enables time stamping within the Ethernet switch for an Ethernet frame.	
Particularities and Limitations	
Module and drivers have been initialized and buffer has to be acquired with EthIf_ProvideTxBuffer() Function enables time stamping for an Ethernet frame identified by the buffer index on the port described by the management information.	
Call context	
> ANY - call only allowed between EthIf_ProvideTxBuffer()/EthIf_ProvideExtTxBuffer() and EthIf_Transmit() for a specific frame > This function is Synchronous > This function is Non-Reentrant	

Table 5-44 EthIf_SwitchEnableEgressTimeStamp

5.5.9 EthIf_SwitchUpdateMCastPortAssignment

Prototype
`Std_ReturnType` **`EthIf_SwitchUpdateMCastPortAssignment`** `(uint8 SwitchIdx, uint8 PortIdx, const uint8 *MCastAddr, EthSwt_MCastPortAssignActionType Action)`

Parameter	
SwitchIdx [in]	Index of the EthIf switch
PortIdx [in]	Index of the Ethernet Switch Port
MCastAddr [in]	Pointer to the multicast address
Action [in]	Action that shall be applied - ETHSWT_MCAST_PORT_ASSIGN_ACTION_ADD: Request passing of multicast on the port - ETHSWT_MCAST_PORT_ASSIGN_ACTION_REMOVE: Request removal of multicast on the port
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Updates the multicast assignment for a specific port.	
Particularities and Limitations	
Module and drivers have been initialized Function updates the multicast assignment for a specific Ethernet switch port.	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 5-45 EthIf_SwitchUpdateMCastPortAssignment

5.6 Zero Copy API Table

This section contains the description of the functions of the Zero Copy API of the Ethernet Interface.

5.6.1 EthIf_ProvideExtTxBuffer

Prototype
`BufReq_ReturnType` **`EthIf_ProvideExtTxBuffer`** `(uint8 CtrlIdx, Eth_FrameType FrameType, uint8 Priority, uint8 *BufIdxPtr, Eth_DataType **BufPtr, uint16 *LenBytePtr)`
Parameter
CtrlIdx [in]	EthIf controller index
FrameType [in]	EtherType to insert into the Ethernet frame header

Priority [in]	Priority of the Ethernet frame, which is coded into the PCP of the IEEE802.1Q VLAN tag. If EthIf controller represents a physical data connection the priority is ignored.
BufIdxPtr [out]	Index to identify the external buffer in context of EthIf
BufPtr [in,out]	Buffer pointer: [in] - Location of the buffer provided externally [out] - Location where payload can be written to
LenBytePtr [in,out]	Buffer length: [in] - Length of the buffer in byte [out] - Length of the buffer reduced by Ethernet header and, dependent on EthIf controller, by VLAN tag size
Return code	
BufReq_ReturnType	BUFREQ_OK - success
	BUFREQ_E_NOT_OK - function has been called with invalid parameters
	BUFREQ_E_BUSY - maximum amount of external buffers that can be handled reached
	BUFREQ_E_OVFL - provided buffer is to small to hold the Ethernet header and, dependent on EthIf controller, the VLAN tag
Functional Description	
Provides an external transmission buffer for an Ethernet frame.	
Particularities and Limitations	
Module and drivers have been initialized Function allows to provide an external buffer for an Ethernet frame transmission.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-46 EthIf_ProvideExtTxBuffer

5.6.2 EthIf_ReleaseRxBuffer

Prototype	
`Std_ReturnType` **`EthIf_ReleaseRxBuffer`** `(uint8 CtrlIdx, Eth_DataType *BufPtr)`	
Parameter	
CtrlIdx [in]	EthIf controller index
BufPtr [in]	Pointer to the buffer to be released
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Releases an reception buffer.	

Particularities and Limitations
Module and drivers have been initialized
Function releases an reception buffer and allows the underlying Ethernet driver to reuse it for further receptions.
Call context
> ANY > This function is Synchronous > This function is Reentrant

Table 5-47 EthIf_ReleaseRxBuffer

5.6.3 EthIf_GetTxHeaderPtr

Prototype	
`Std_ReturnType` **`EthIf_GetTxHeaderPtr`** `(uint8 CtrlIdx, uint8 BufIdx, Eth_DataType **BufPtr, uint16 *LenBytePtr)`	
Parameter	
CtrlIdx [in]	EthIf controller index
BufIdx [in]	Index to identify the buffer the Ethernet header location shall be retrieved for
BufPtr [out]	Location of the Ethernet header
LenBytPtr [out]	Length of the Ethernet header in byte: 14 byte - non VLAN enabled EthIf controller 18 byte - VLAN enable EthIf controller
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the location of the Ethernet header for a given transmission buffer.	
Particularities and Limitations	
> Module and drivers have been initializedBuffer to retrieve the Ethernet header location for must be previously acquired by EthIf_ProvideTxBuffer()/EthIf_ProvideExtTxBuffer()	
Function retrieves the location of the Ethernet header for a given transmission buffer.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-48 EthIf_GetTxHeaderPtr

5.6.4 EthIf_GetRxHeaderPtr

Prototype	
`Std_ReturnType` **`EthIf_GetRxHeaderPtr`** `(uint8 CtrlIdx, Eth_DataType **BufPtr, uint16 *LenBytePtr)`	
Parameter	
CtrlIdx [in]	EthIf controller index
BufPtr [in,out]	Pointer to reception buffer: [in] - Location of the payload of the Ethernet frame within the reception buffer [out] - Location of the Ethernet header
LenBytPtr [out]	Length of the Ethernet header in byte: 14 byte - non VLAN enabled EthIf controller 18 byte - VLAN enable EthIf controller
Return code	
Std_ReturnType	E_OK - success
Std_ReturnType	E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the location of the Ethernet header for a given reception buffer.	
Particularities and Limitations	
Module and drivers have been initialized Function retrieves the location of the Ethernet header for a given reception buffer.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 5-49 EthIf_GetRxHeaderPtr

5.7 Services used by Ethernet Interface

In the following table services provided by other components, which are used by the Ethernet Interface are listed.

Note
The need for availability of the services depends on configuration settings.

For details about prototype and functionality refer to the documentation of the providing component.

Component	API
Det	▶ Det_ReportError()
Dem	▶ Dem_ReportErrorStatus()
Eth	▶ Eth_ControllerInit() ▶ Eth_SetControllerMode() ▶ Eth_GetControllerMode() ▶ Eth_GetPhysAddr() ▶ Eth_SetPhysAddr() ▶ Eth_UpdatePhysAddrFilter() ▶ Eth_ProvideTxBuffer() ▶ Eth_ProvideExtTxBuffer() ▶ Eth_ReleaseRxBuffer() ▶ Eth_GetTxHeaderPtr() ▶ Eth_GetRxHeaderPtr() ▶ Eth_Transmit() ▶ Eth_VTransmit() ▶ Eth_Receive() ▶ Eth_TxConfirmation() ▶ Eth_GetGlobalTime() ▶ Eth_SetGlobalTime() ▶ Eth_SetCorrectionTime() ▶ Eth_EnableEgressTimestamp() ▶ Eth_GetEgressTimestamp() ▶ Eth_GetIngressTimestamp() ▶ Eth_SetBandwidthLimit() ▶ Eth_GetBandwidthLimit()
EthTrcv	▶ EthTrcv_TransceiverInit() ▶ EthTrcv_SetTransceiverMode() ▶ EthTrcv_GetTransceiverMode() ▶ EthTrcv_StartAutoNegotiation() ▶ EthTrcv_GetLinkState() ▶ EthTrcv_GetBaudRate() ▶ EthTrcv_GetDuplexMode() ▶ EthTrcv_SetTransceiverWakeupMode() ▶ EthTrcv_GetTransceiverWakeupMode() ▶ EthTrcv_CheckWakeup()
EthSwt	▶ EthSwt_SwitchInit() ▶ EthSwt_SetSwitchPortMode() ▶ EthSwt_GetSwitchPortMode() ▶ EthSwt_StartSwitchPortAutoNegotiation() ▶ EthSwt_GetLinkState() ▶ EthSwt_GetBaudRate()

Component	API
	▶ EthSwt_GetDuplexMode() ▶ EthSwt_GetPortMacAddr() ▶ EthSwt_GetArlTable() ▶ EthSwt_GetBufferLevel() ▶ EthSwt_GetDropCount() ▶ EthSwt_EnableVlan() ▶ EthSwt_StoreConfiguration() ▶ EthSwt_ResetConfiguration() ▶ EthSwt_SetMacLearningMode() ▶ EthSwt_GetMacLearningMode() ▶ EthSwt_SetSwitchMgmtInfo() ▶ EthSwt_EnableEgressTimeStamp() ▶ EthSwt_UpdateMCastPortAssignment()
BswM	▶ BswM_EthIf_PortGroupLinkStateChg()
EthFw	▶ EthFw_IsFrameRxAllowed() ▶ EthFw_IsFrameTxAllowed()

Table 5-50 Services used by the Ethernet Interface

5.8 Call-back Functions

Following sections describe the callback functions the Ethernet Interface provides.

5.8.1 Common Call-back Functions

This section contains the description of the call-back functions of the common API of Ethernet Interface.

5.8.1.1 EthIf_RxIndication

Prototype	
`void EthIf_RxIndication (uint8 CtrlIdx, Eth_FrameType FrameType, boolean IsBroadcast, uint8 *PhysAddrPtr, Eth_DataType *DataPtr, uint16 LenByte)`	
Parameter	
CtrlIdx [in]	Ethernet controller index
FrameType [in]	EtherType the Ethernet frame is related to
IsBroadcast [in]	Broadcast indication: FALSE - frame isn't a broadcast frame TRUE - frame is a broadcast frame
PhysAddrPtr [in]	Source MAC address of the Ethernet frame
DataPtr [out]	Location of the Ethernet frame payload (no VLAN tag considered)
LenByte [in]	Length of the Ethernet frame payload (no VLAN tag considered)
Return code	
void	none

Functional Description
Notifies the EthIf about a received Ethernet frame
Particularities and Limitations
Module has been initialized Functions takes the given Ethernet frame data, analysis the Ethernet header for VLAN and information and decides whether to drop the frame or pass it to a known EthIf user.
Call context
> TASK|ISR1|ISR2 > This function is Synchronous > This function is Reentrant

Table 5-51 EthIf_RxIndication

5.8.1.2 EthIf_TxConfirmation

Prototype	
`void EthIf_TxConfirmation (uint8 CtrlIdx, uint8 BufIdx)`	
Parameter	
CtrlIdx [in]	Ethernet controller index
BufIdx [in]	Index of the buffer the transmission is confirmed for
Return code	
void	none
Functional Description	
Notifies the EthIf about the transmission of a Ethernet frame	
Particularities and Limitations	
Module has been initialized Function handles the confirmation of an Ethernet frame transmission and passes it to the respective EthIf user.	
Call context	
> interrupt or task level > TASK|ISR1|ISR2 > This function is Synchronous > This function is Reentrant	

Table 5-52 EthIf_TxConfirmation

5.8.2 Ethernet Switch Abstraction Call-Back Functions

This section contains the description of the call-back functions of the Ethernet Switch Abstraction API of Ethernet Interface.

5.8.2.1 EthIf_SwitchMgmtInfoIndication

Prototype	
`void EthIf_SwitchMgmtInfoIndication (uint8 CtrlIdx, uint8 *DataPtr, const EthSwt_MgmtInfoType *MgmtInfo)`	
Parameter	
CtrlIdx [in]	Index of the EthIf controller
DataPtr [in]	Pointer to identify the frame context
MgmtInfo [in]	Switch management information
Return code	
void	none
Functional Description	
Notifies the EthIf about switch management information related to a received Ethernet frame	
Particularities and Limitations	
This function allows to provide switch management information for an received Ethernet frame to the EthIf.	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 5-53 EthIf_SwitchMgmtInfoIndication

5.8.2.2 EthIf_SwitchEgressTimeStampIndication

Prototype	
`void EthIf_SwitchEgressTimeStampIndication (uint8 CtrlIdx, uint8 *DataPtr, const EthSwt_MgmtInfoType *MgmtInfo, const Eth_TimeStampType *timeStamp)`	
Parameter	
CtrlIdx [in]	Index of the EthIf controller
DataPtr [in]	Pointer to identify the frame context
MngmtInfo [in]	Switch management information
timeStamp [in]	Port Egress Time stamp
Return code	
void	none
Functional Description	
Notifies the EthIf about time stamping information related to an Ethernet frame transmitted	
Particularities and Limitations	
This function allows to provide time stamping information for an Ethernet frame transmitted at a switch port to the EthIf.	

Call context
> ANY > This function is Synchronous > This function is Non-Reentrant

Table 5-54 EthIf_SwitchEgressTimeStampIndication

5.8.2.3 EthIf_SwitchIngressTimeStampIndication

Prototype	
`void EthIf_SwitchIngressTimeStampIndication (uint8 CtrlIdx, uint8 *DataPtr, const EthSwt_MgmtInfoType *MgmtInfo, const Eth_TimeStampType *timeStamp)`	
Parameter	
CtrlIdx [in]	Index of the EthIf controller
DataPtr [in]	Pointer to identify the frame context
MngmtInfo [in]	Switch management information
timeStamp [in]	Port Ingress Time stamp
Return code	
void	none
Functional Description	
Notifies the EthIf about time stamping information related to an Ethernet frame received	
Particularities and Limitations	
This function allows to provide time stamping information for an Ethernet frame received at a switch port to the EthIf.	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 5-55 EthIf_SwitchIngressTimeStampIndication

5.9 Configurable Interfaces

5.9.1 Notifications

At its configurable interfaces the Ethernet Interface defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the Ethernet Interface but can be performed at configuration time. The function prototypes that can be used for the configuration have to match the appropriate function prototype signatures, which are described in the following sub-chapters.

5.9.1.1 <User>_RxIndication

Prototype	
`void` **`<User>_RxIndication`** `( uint8 EthIfCtrlIdx, Eth_FrameType FrameType, boolean IsBroadcast, uint8 *PhysAddrPtr, uint8 *DataPtr, uint16 LenByte )`	
Parameter	
`EthIfCtrlIdx [in]`	EthIf Controller index the Ethernet frame was received on
`FrameType [in]`	EtherType of the Ethernet frame
`IsBoradcast [in]`	Infromation about if the frame was a MAC broadcast ▶ FALSE: Frame is a Unicast or Multicast frame ▶ TRUE: Frame is a Broadcast frame
`PhysAddrPtr [in]`	Pointer pointing to the location where the source MAC address is stored (length is 6 byte)
`DataPtr [in]`	Pointer pointing to the Ethernet frames payload
`LenByte [in]`	Length of the Ethernet frames payload in byte
Return code	
`Void`	No return value
Functional Description	
Notification function informing about the reception of an Ethernet frame.	
Particularities and Limitations	
Call context	
> interrupt or task context > called if an Ethernet frame is received	

Table 5-56 <User>_RxIndication

5.9.1.2 <User>_TxConfirmation

Prototype	
`void` **`<User>_TxConfirmation`** `( uint8 EthIfCtrlIdx, uint8 BufferIdx )`	
Parameter	
`EthIfCtrlIdx [in]`	EthIf Controller index the Ethernet frame was received on
`BufferIdx [in]`	Buffer index to identify the Ethernet frame previously triggered for transmission
Return code	
`Void`	No return value
Functional Description	
Notification function informing about the finished transmission of an Ethernet frame.	
Particularities and Limitations	

Call context
> interrupt or task context > called if an Ethernet frame was transmitted and TxConfirmation was enabled during EthIf_Transmit()

Table 5-57 <User>_TxConfirmation

5.9.1.3 <User>_TrcvLinkStateChg

Prototype	
`void` **`<User>_TrcvLinkStateChg`** `( uint8 EthIfCtrlIdx, EthTrcv_LinkStateType TrcvLinkState )`	
Parameter	
`EthIfCtrlIdx [in]`	EthIf Controller index the Ethernet frame was received on
`TrcvLinkState [in]`	Link state indicated ▶ ETHTRCV_LINK_STATE_DOWN: No link anymore ▶ ETHTRCV_LINK_STATE_ACTIVE: Link established
Return code	
`Void`	No return value
Functional Description	
Notification function informing about a link state change on EthIf Controller level.	
Particularities and Limitations	
Call context	
> interrupt or task context > called if an EthIf Controller changes its link state	

Table 5-58 <User>_TrcvLinkStateChg

5.9.1.4 <User>_SwitchMgmtInfoIndication

Prototype	
`void` **`<User>_SwitchMgmtInfoIndication`** `( uint8 EthIfCtrlIdx, const EthSwt_MgmtInfoType *MgmtInfo )`	
Parameter	
`EthIfCtrlIdx [in]`	EthIf Controller index the Ethernet frame was received on
`MgmtInfo [in]`	Management information like switch port index the frame was received on
Return code	
`Void`	No return value
Functional Description	
Notification function informing about the frame management information of an Ethernet frame.	

Particularities and Limitations
Ethernet Switch driver must support management capability
Call context
> interrupt or task context

Table 5-59 <User>_SwitchMgmtInfoIndication

5.9.1.5 <User>_EgressTimeStampIndication

Prototype	
`void <User>_EgressTimeStampIndication ( uint8 EthIfCtrlIdx, const EthSwt_MgmtInfoType *MgmtInfo, const Eth_TimeStampType *TimeStamp )`	
Parameter	
`EthIfCtrlIdx [in]`	EthIf Controller index the Ethernet frame was received on
`MgmtInfo [in]`	Management information containing the port where the frame has egressed at the switch.
`TimeStamp[in]`	Time Stamp taken for the frame as it egressed the switch.
Return code	
`Void`	No return value
Functional Description	
Notification function informing about the availability of the egress time stamp of a frame as it egressed at a switch port.	
Particularities and Limitations	
Ethernet Switch driver must support time stamping capability	
Call context	
> interrupt or task context	

Table 5-60 <User>_EgressTimeStampIndication

5.9.1.6 <User>_IngressTimeStampIndication

Prototype	
`void <User>_IngressTimeStampIndication ( uint8 EthIfCtrlIdx, const EthSwt_MgmtInfoType *MgmtInfo, const Eth_TimeStampType *TimeStamp )`	
Parameter	
`EthIfCtrlIdx [in]`	EthIf Controller index the Ethernet frame was received on
`MgmtInfo [in]`	Management information containing the port where the frame has arrived at the switch.
`TimeStamp[in]`	Time Stamp taken for the frame as it arrived the switch.
Return code	
`Void`	No return value

Functional Description
Notification function informing about the availability of the ingress time stamp of a frame as it arrived at a switch port.
Particularities and Limitations
Ethernet Switch driver must support time stamping capability
Call context
> interrupt or task context

Table 5-61 <User>_IngressTimeStampIndication

6 Configuration

The configuration of Ethernet Interface can be carried out by one of the following configuration variants:

> Pre-compile time configuration

6.1 Configuration with DaVinci Configurator Pro

The Ethernet Interface is configured with the help of the configuration 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.

6.1.1 Wakeup Support

This section describes how to configure the fundamental configuration elements needed for wakeup support. However the scope is only on the Ethernet interface related configuration elements. For configuration of the elements of the other modules involved in the wakeup procedure please refer to the corresponding Technical Reference.

General Wakeup Support

To enable the wakeup support for the Ethernet interface it must be enabled globally. The parameter doing so is located in the general configuration container.

Following screenshot marks the parameter to adapt.

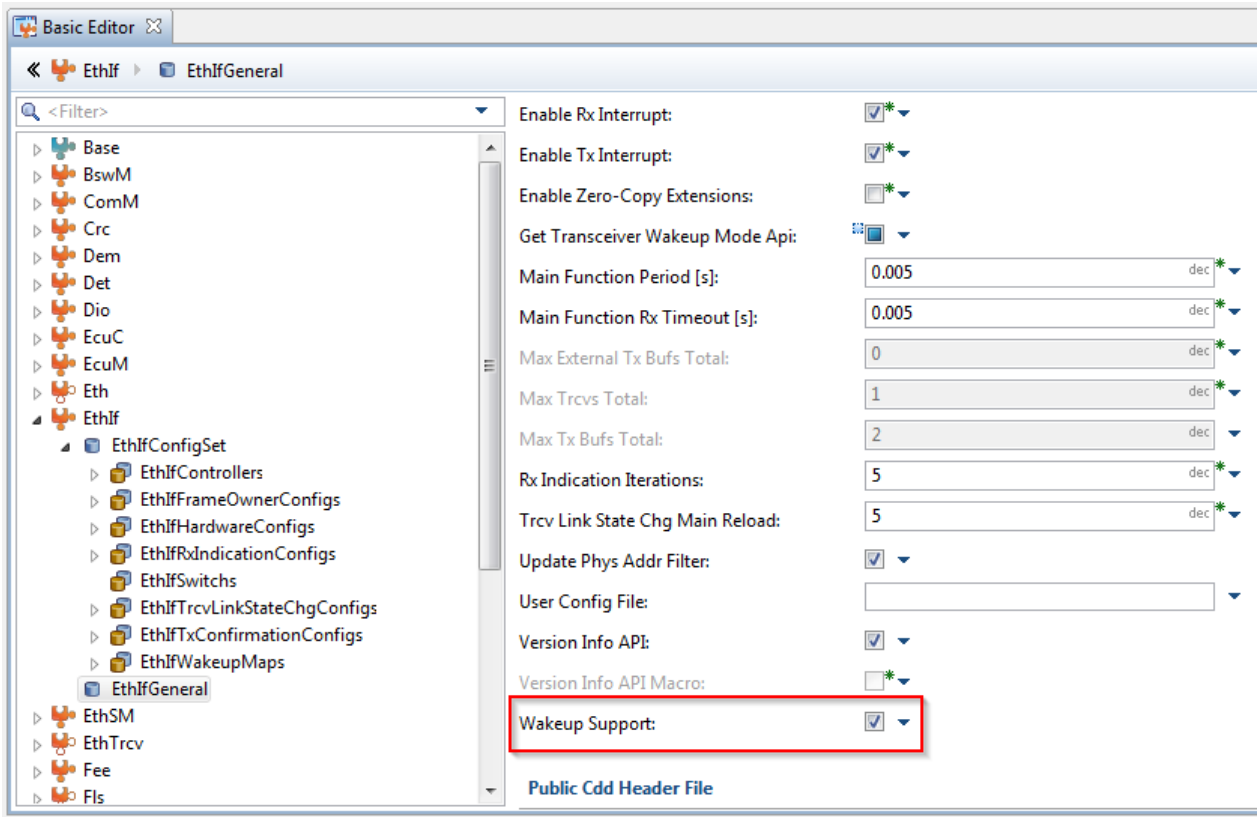

Figure 6-1 General Wakeup Support Configuration

Wakeup Map

The wakeup map allows associating an EcuM Wakeup Reason with an Ethernet transceiver the transceiver driver shall perform the wakeup detection for.

As seen in the following screenshot one has just to create a wakeup map container and select the mentioned elements to achieve this mapping.

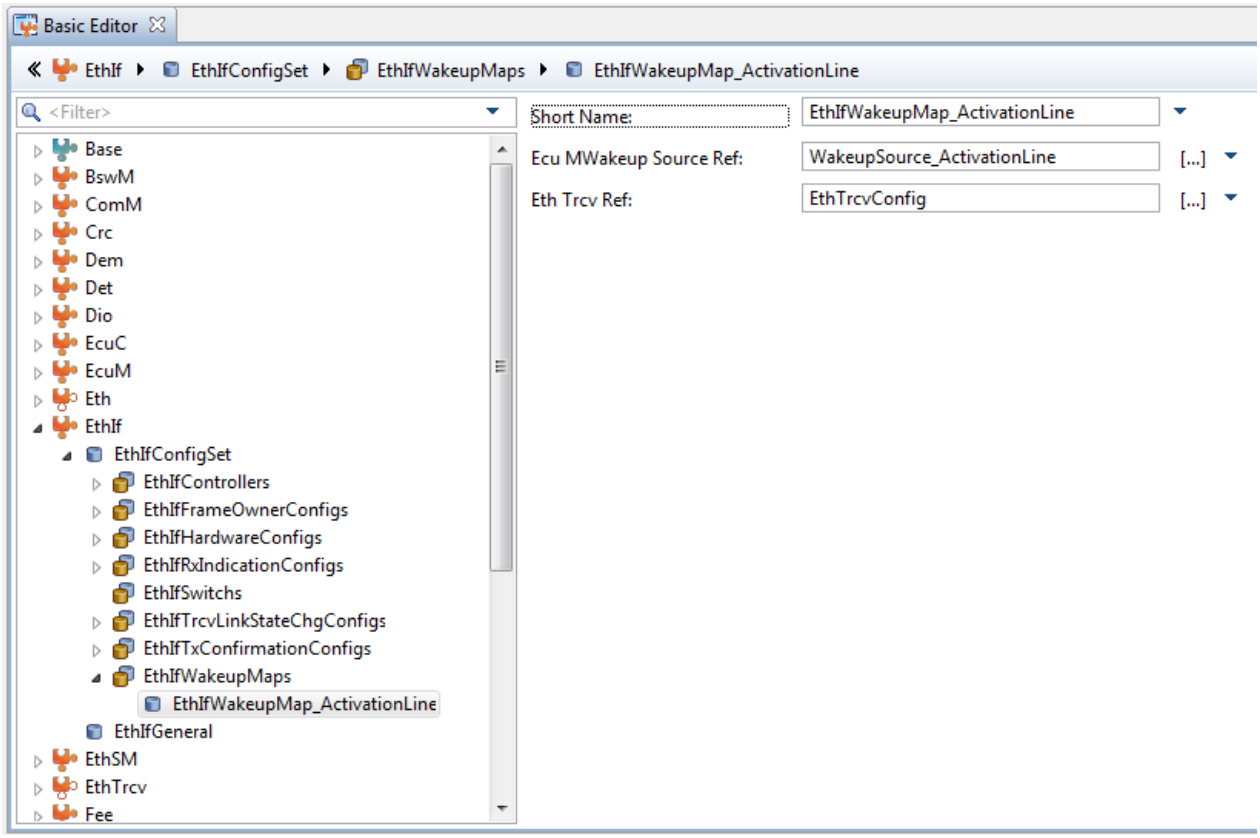

Figure 6-2 Wakeup Map Configuration

6.1.2 Extended Traffic Handling

This section describes how to configure the configuration elements for the extended traffic handling features “Traffic Mirroring” and “Traffic Gateway”.

6.1.2.1 Traffic Mirroring

Traffic Mirroring allows to mirror received and transmitted Frames for EthIf controllers. This section describes how to configure the feature.

For a detailed description of the mechanism refer to 3.9.1.

To mirror receive/transmit traffic of an EthIf controller the feature must be enabled be creating the `EthIfExtendedTrafficHandling` and its choice container `EthIfTrafficMirroring` (achieved by using the “Create sub container” and “Choose” options in the context menu of the respective parent container).

This container contains a collection of so called `EthIfMirrorMaps`. The mirror maps represent the configuration of one mirroring destination, which is an Ethernet controller. This mirroring destination is selected by `EthIfDestEthCtrlRef`. The traffic to be mirrored is parted into receive and transmit traffic. To force traffic mirroring for an EthIf

controller select it by referencing it in either `EthIfRxSourceEthIfCtrlRef` (for mirroring received traffic on the selected EthIf controller) or `EthIfTxSourceEthIfCtrlRef` (for mirroring transmit traffic on the selected EthIf controller).

The following example shows a configuration where both receive and transmit traffic of the EthIf controller “EthIfController_Ctrl0” is mirrored to the Ethernet controller “ENET0”.

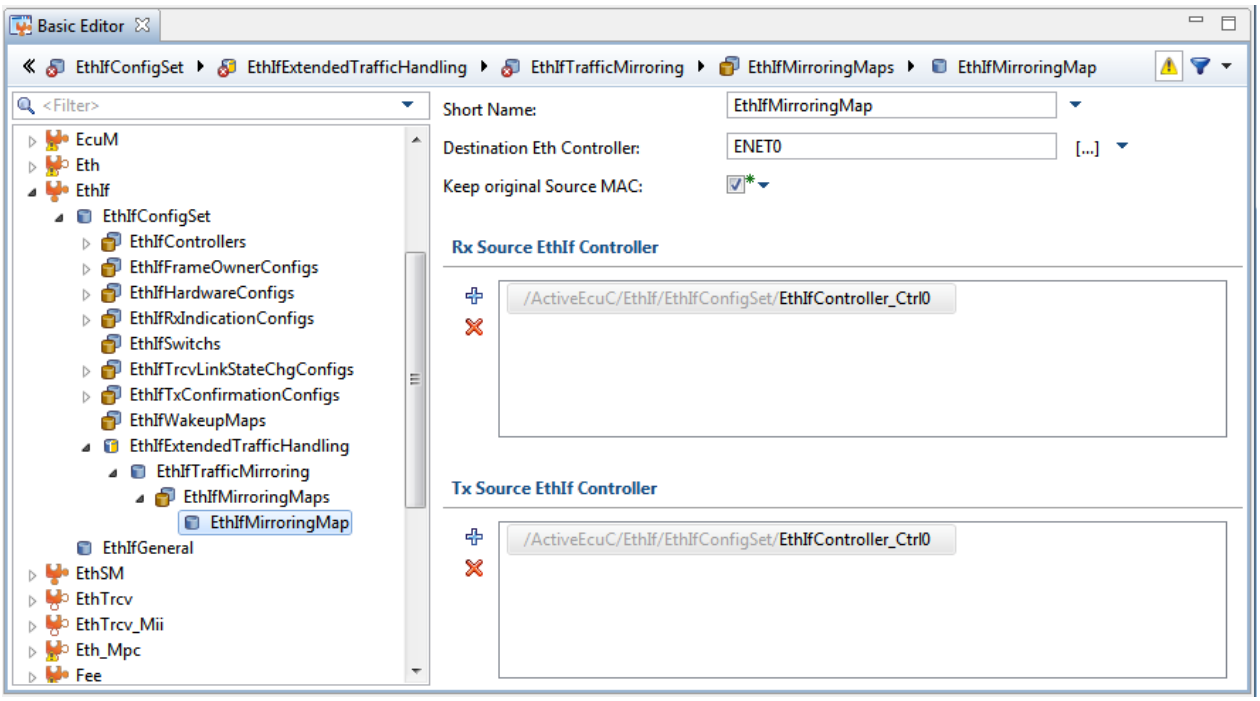

Figure 6-3 Traffic Mirroring – Mirroring Map configuration

6.1.2.2 Traffic Gateway

Traffic Gateway allows to route traffic from one EthIf controller to another without passing the traffic to EthIf upper layers.

For a detailed description of the mechanism please refer to 3.9.2.

To gateway traffic from one EthIf controller to another the feature must be enabled be creating the `EthIfExtendedTrafficHandling` and its choice container `EthIfTrafficGateway` (achieved by using the “Create sub container” and “Choose” options in the context menu of the respective parent container).

This container contains a collection of so called `EthIfTrafficGatewayRoutes`. The routes represent one traffic gateway route configuration. To involve an EthIf controller into a gateway route it must be selected by `EthIfRouteEthIfCtrlRef`.

The following example shows a Traffic gateway route which routes traffic from EthIf controller “EthIfController_Ctrl0” to EthIf controller “EthIfController_Ctrl1” and vice versa.

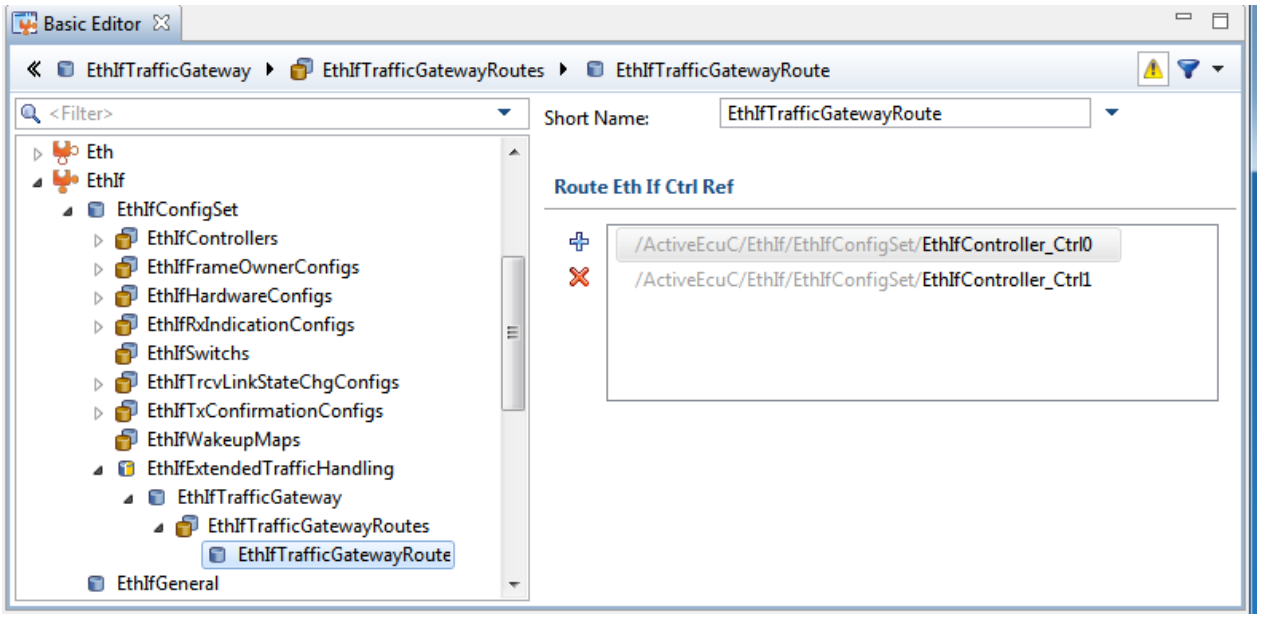

Figure 6-4 Traffic Gateway – Traffic Gateway Route configuration

To exclude frames from the routing mechanism and pass it to an EthIf upper layer a black list can be configured. The black list contains MAC addresses which are compared to the source MAC address of a frame. If a match occurs the frame will be excluded from traffic routing and handled like a normal frame (passing it to an EthIf upper layer if appropriate).

The following example shows the configuration of the black list for a media conversion use-case (100BaseTx <-> PLC). The MAC addresses on the black list are related to the QCA7000 and its driver. 08:00:00:00:00:08 (PLC MAC address of QCA7000 EthTrcv driver), 00:B0:52:00:00:01 (configuration MAC address of the QCA7000 chip) and 02:00:00:00:00:02 (Ethernet MAC address of QCA7000 Eth driver). This configuration allows managing the QCA7000 although the used EthIf controller is involved in a traffic gateway.

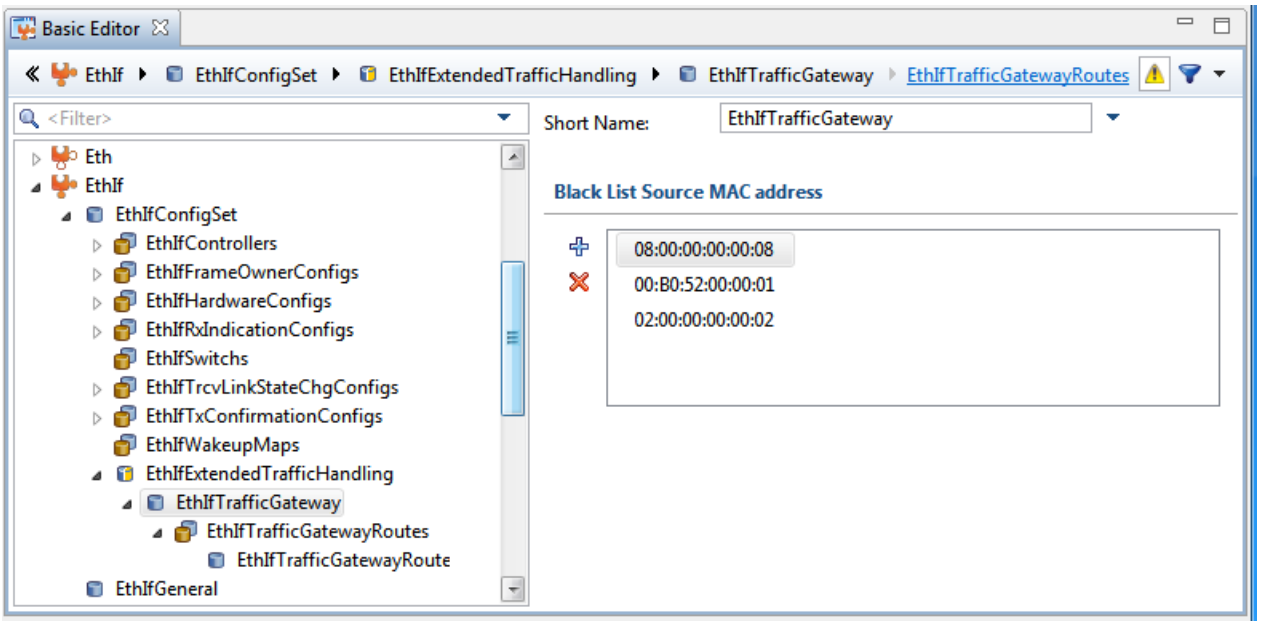

Figure 6-5 Traffic Gateway – Black List configuration

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
DaVinci Configurator Pro	Configuration tool for MICROSAR components
Frame Management Information	Information like switch port and switch instance related to a Ethernet frame and provided by an Ethernet switch.
Hardware Element	Abstract description for Ethernet Controller, Ethernet Transceiver, Ethernet Switch and Ethernet Switch Port.
Host-CPU	MCU running the Ethernet Switch driver controlling the Ethernet Switch(es) connected through a management interface (e.g. SPI) to the MCU.
Physical Layer Element	Hardware element, which is able to report a link state (e.g. Ethernet Transceiver, Ethernet Switch Port).
Platform	Hardware including Host and Communication Controller (might also be integrated in Host) on which the communication stack is implemented.
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 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 7-1 Glossary

7.2 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
EcuM	ECU Manager
Eth	Ethernet Controller Driver
EthFw	Ethernet Firewall (Vector specific BSW)
EthIf	Ethernet Interface
EthTrcv	Ethernet Transceiver Driver
EthSwt	Ethernet Switch Driver
FQTSS	Forwarding and Queuing Enhancements for Time-Sensitive Streams

HIS	Hersteller Initiative Software
ICU	Input Capture Unit
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PTP	Precision Time Protocol
QoS	Quality of Service
RTE	Runtime Environment
SRP	Stream Reservation Protocol
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector-informatik.com