MICROSAR Classic EthIf

Technical Reference

Version 15.3.0

Status	Released

Contents

Document

Version

Author	Date	Version	Remarks
...	...	...	...
vistoc	2024-01-04	15.01.00	Introduced Extend AUTOSAR Forward Compatibility for all callbacks used by Eth
baubeck, visdep	2024-02-13	15.02.00	> Migrated TechRef to RST > Performed Cyber Security Analysis > Removed description about the deprecated feature FIREWALL.
visdep	2024-03-22	15.03.00	Performed the AUTOSAR fulfilment analysis against 23-11 (4.9.0).

Reference

No.	Source	Title	Version
[1]	AUTOSAR	Specification of Ethernet Interface	R4.1.1
[2]	AUTOSAR	Specification of Ethernet Interface	R4.2.1
[3]	AUTOSAR	Specification of Ethernet Interface	R4.4.0
[4]	AUTOSAR	Specification of Ethernet Interface	R19-11 (4.5.0)
[5]	AUTOSAR	Specification of Ethernet Interface	R20-11 (4.6.0)
[6]	Vector	Basic Software Module Description (BSWMD)	see delivery
[7]	AUTOSAR	Specification of Ethernet Interface	R22-11 (4.8.0)
[8]	AUTOSAR	Specification of Ethernet Interface	R23-11 (4.9.0)

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.

1 Introduction

This document describes the functionality, integration, API and configuration of the module EthIf.

The EthIf is specified in AUTOSAR. Please note that the module is not compliant with one particular AUTOSAR version. Instead a mix of different AUTOSAR versions *[1] [2] [3] [4] [5] [7] [8]*.

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.

1.1 Architecture Overview

These are the interfaces of the module:

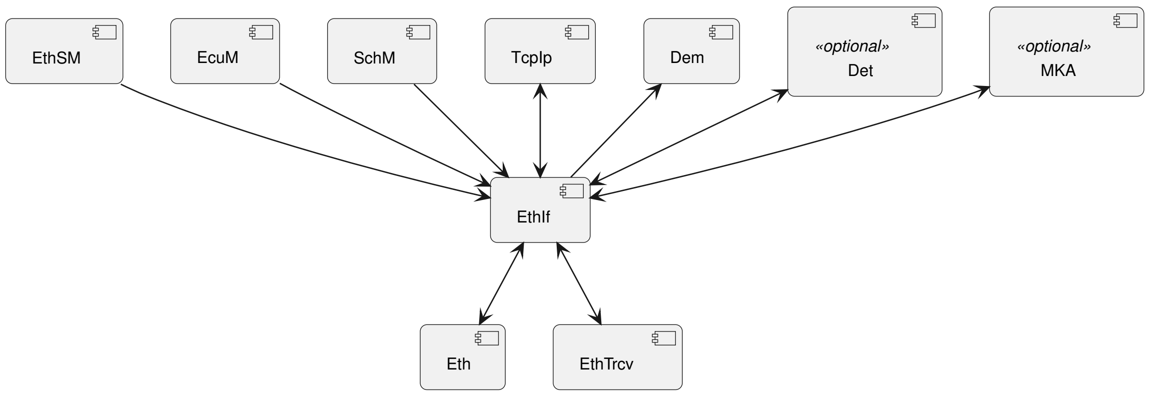


Figure 1.1 Module Architectural Environment

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

2 Functional Description

2.1 Features

2.1.1 Supported Autosar Features

The following features specified in *[8]* are supported. For detailed description and/or limitations of a specific feature (if any), please refer the related subsection below.

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 To DET
Retrieval of Measurement Data
Signal Quality Monitoring
Post-Build Selectable (MICROSAR Classic Identity Manager)
Security Events Reporting To the IdsM
Cable Diagnostic Handling for Transceiver[1]
Cable Diagnostic Handling for Switch Port
Transceiver MACsec
Switch MACsec

Table 2.1 Supported AUTOSAR Standard Conform Features

[1] The module with the software version before 16.02.00 supports only synchronous cable diagnostic handling for transceiver. From the software version 16.02.00 and onwards, both synchronous and

2.1.1.1 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.

The following figure shows the abstract virtual controller and its specializations in form of a class diagram.

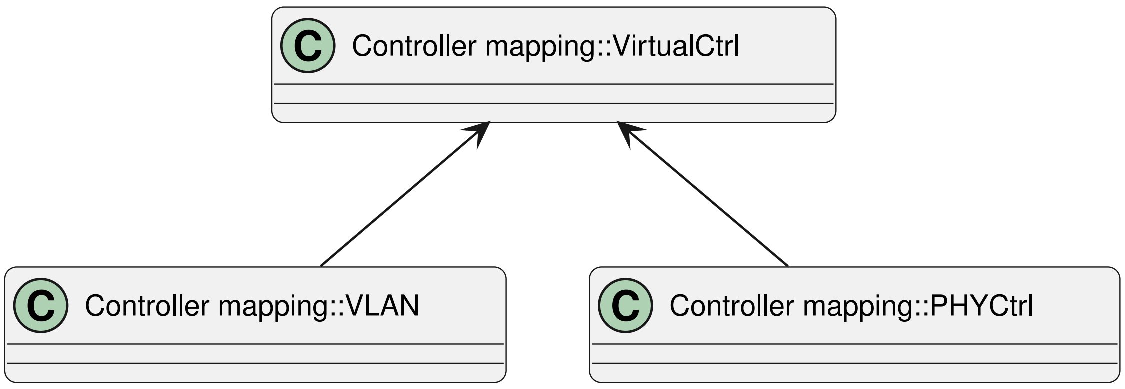


Figure 2.1 Virtual Controllers

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 can 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.

asynchronous cable diagnostic handling for transceiver is supported, please refer to the related subsection under Functional Description.

Note
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.

2.1.1.2 Ethernet Signal Quality Monitoring

The signal quality monitoring is done within `EthIf_MainFunctionState()` where the link observation is done. The polling period of signal quality is configurable and an integral multiple of `EthIfMainFunctionStatePeriod`.

The signal quality is stored in Ethernet interface as current, highest (updated dependent on current signal quality) and lowest (updated dependent on current signal quality). These measured signal quality values for an Ethernet transceiver or an Ethernet switch port is retrievable and clearable with the following APIs:

- `EthIf_GetTrcvSignalQuality()`
- `EthIf_ClearTrcvSignalQuality()`
- `EthIf_GetSwitchPortSignalQuality()`
- `EthIf_ClearSwitchPortSignalQuality()`

2.1.1.3 Security Event Reporting

Security events are reported to the IdsM using the service `IdsM_SetSecurityEvent()` if the security events reporting is enabled and if the security events are defined in the IdsM (specified in *[5]*).

Security Event	Description
`ETHIF_SEV_DROPPED_UNKNOWN_ETHERTYPE_DATAGRAM`	An ethernet datagram was dropped due the EtherType is not known.
`ETHIF_SEV_DROPPED_VLAN_DOUBLE_TAG_DATAGRAM`	An ethernet datagram was dropped due to double VLAN tag.
`ETHIF_SEV_DROPPED_INVALID_CTRLIDX_DATAGRAM`	An ethernet datagram was dropped due to an invalid CrtlIdx/VLAN .
`ETHIF_SEV_ETH_MAC_COLLISION`	Ethernet datagram was dropped because local MAC was same as source MAC in an incoming frame.

Table 2.2 Errors Reported To IdsM

2.1.1.4 Cable Diagnostic Handling for Transceiver

The module supports both asynchronous and synchronous cable diagnostic handling for transceiver.

The **asynchronous cable diagnostic** handling for transceiver is supported from the component module version 16.02.00 and onwards.

Before version 16.02.00, the module only supports the **synchronous cable diagnostic** handling for transceiver.

Since version 16.02.00, the synchronous cable diagnostic runs in **compatibility mode**.

Caution
If the used Ethernet Transceiver does not support asynchronous cable diagnostic handling, the component Ethernet Interface (Irrespective of version) can only perform synchronous cable diagnostic.

2.1.1.4.1 Asynchronous Cable Diagnostic Handling for Transceiver

In asynchronous cable diagnostic handling for transceiver, the user should firstly call the API `EthIf_RunCableDiagnostic()` to prepare the results. This API call will return `E_OK` if the running of cable diagnostic is successful. After that, to retrieve the results, the `EthIf_GetCableDiagnosticsResult()` shall be polled by the user as long `ETHTRCV_CABLEDIAG_PENDING` is returned (until the cable diagnostic is finished) and the user can use the diagnostic result.

Note
The asynchronous cable diagnostic handling for transceiver is supported from the component Ethernet Interface version 16.02.00 and onwards.

Note
The asynchronous cable diagnostic handling for transceiver can only be used if this feature is supported and activated in the used Ethernet Transceiver. If this feature is supported by Ethernet Transceiver, it can be activated by enabling the configuration switch `EthTrcvRunCableDiagnosticApi` in the Ethernet Transceiver configuration using Davinci Configurator Classic.

2.1.1.4.2 Synchronous Cable Diagnostic Handling for Transceiver

In synchronous cable diagnostic handling for transceiver, the user will call the API `EthIf_GetCableDiagnosticsResult()` to trigger the cable diagnostic and wait until this API returns, when the cable diagnostic is finished.

Note
By default the module with version before 16.02.00 supports only synchronous cable diagnostic handling for transceiver. From version 16.02.00 and onwards, the synchronous cable diagnostic handling for transceiver can run only in compatibility mode.

2.1.1.4.3 Synchronous Cable Diagnostic Handling in Compatibility Mode

This mode is used for synchronous cable diagnostic handling from module version 16.02.00 and onwards. To run the synchronous cable diagnostic handling for transceiver in compatibility mode, the feature "run cable diagnostic" should be deactivated in the used Ethernet Transceiver. This can be done by disabling the configuration switch `EthTrcvRunCableDiagnosticApi` in the Ethernet Transceiver configuration using Davinci Configurator Classic.

2.1.1.5 Cable Diagnostic Handling for Switch Port

The module supports cable diagnostic handling for switch port. This feature can be enabled by enabling the configuration switch `EthSwtEnableCableDiagnosticApi` in the Ethernet Switch configuration using Davinci Configurator Classic (If supported by the Ethernet Switch used).

Both asynchronous and synchronous cable diagnostic handling for switch port are supported depending on the configuration of the used Ethernet Switch and Ethernet Transceiver.

Caution
If the Ethernet Transceiver (which is connected to the used Ethernet Switch Port) does not support asynchronous cable diagnostic handling, the Ethernet Interface (Irrespective of version) can only perform synchronous cable diagnostic.

2.1.1.5.1 Asynchronous Cable Diagnostic Handling for Switch Port

In asynchronous cable diagnostic handling for switch port, the user should firstly call the API `EthIf_RunPortCableDiagnostic()` to prepare the results. This API call will return `E_OK` if the running of cable diagnostic is successful. After that, to retrieve the results, the `EthIf_GetPortCableDiagnosticsResult()` shall be polled by the user as long `ETHTRCV_CABLEDIAG_PENDING` is returned (until the cable diagnostic is finished) and the user can use the diagnostic result.

Note
The asynchronous cable diagnostic handling for switch port can only be used if this feature is supported and activated in the Ethernet Transceiver which is connected to the used Ethernet Switch Port. If this feature is supported by this Ethernet Transceiver, it can be activated by enabling the configuration switch `EthTrcvRunCableDiagnosticApi` in the Ethernet Transceiver configuration using Davinci Configurator Classic.

2.1.1.5.2 Synchronous Cable Diagnostic Handling for Switch Port

In synchronous cable diagnostic handling for switch port, the user will call the API `EthIf_GetPortCableDiagnosticsResult()` to trigger the cable diagnostic and wait until this API returns, when the cable diagnostic is finished.

Note
The synchronous cable diagnostic handling for switch port will be used if asynchronous cable diagnostic handling is not supported or activated in the Ethernet Transceiver which is connected to the used Ethernet Switch Port.

2.1.2 Additions/Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Multicast To Switch Port Assignment Manipulation (feature must also be supported by Ethernet switch 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)
Zero-Copy Extension - Reception Buffer Locking Mechanism Based on VLAN and Frame Type (feature must also be supported by Ethernet driver)
Zero-Copy Extension - External Transmission Buffer Provision (feature must also be supported by Ethernet driver)
Extended Traffic Handling - Ethernet Frame Mirroring Functionality on Ethernet Interface Controller Level (feature must also be supported by Ethernet driver)
Extended Traffic Handling - Ethernet Frame Gateway Functionality on Ethernet Interface Controller Level (feature must also be supported by Ethernet driver)

continues on next page

Table 2.3 – continued from previous page

Features Provided Beyond The AUTOSAR Standard
Header Access APIs - Retrieval of the Ethernet Header Information for Reception and Transmission Buffers (feature must also be supported by Ethernet driver)
Ethernet Rx/Tx Statistics
IEEE802.1Qbv Support for Ethernet Switch (feature must also be supported by Ethernet switch driver)
Enabling and Disabling of VLANs on Ethernet Switch Ports
Lax Link Aggregation
Multi-Core Environment Support
Wakeup Support
Parameter Checking

Table 2.3 Features Provided Beyond The AUTOSAR Standard

For a detailed description of the Additions/Extensions please refer to the related subsection below.

2.1.2.1 Ethernet Switch Multicast to Port Assignment Manipulation

Commonly multicast 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 EthSwt driver is called `EthSwtUpdateMCastPortAssignmentApi`.

2.1.2.2 Ethernet Switch Frame Management

For specific protocols like the PTP protocol that must be treated by an Ethernet Switch in a managed way. Additional information to an Ethernet frame either must be applied on transmission or must be retrieved after reception of the frame.

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

On transmission side the module provides the abstraction of the Ethernet Switch driver API `EthIf_SetSwitchMgmtInfo()` that can be used between an `EthIf_ProvideTxBuffer()` and `EthIf_Transmit()` to apply such information to an Ethernet frame.

On reception side the module provides the abstraction of the Ethernet Switch driver API `EthIf_GetRxMgmtObject()` that allows to gain access to a shared object of type `EthSwt_MgmtObjectType` provided by the Ethernet Switch driver that contains such information. The API must be used in context of `<User>_RxIndication()` triggered for the frame one is interested in.

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.

2.1.2.3 Ethernet Switch Time Synchronization

As an extension of the Ethernet Switch Frame Management feature to retrieve data like the switch port an Ethernet frame was received on. This feature provides the ability to request and retrieve ingress and egress timestamps taken on an Ethernet switch port for specific frames.

On transmission side the module provides the abstraction of the Ethernet Switch driver API `EthIf_SwitchEnableTimestamping()`. This can be used between `EthIf_ProvideTxBuffer()` and `EthIf_Transmit()` to request timestamping for an Ethernet frame. When the transmission of the frame gets confirmed by `<User>_TxConfirmation()` the API `EthIf_GetTxMgmtObject()` allows to gain access to a shared object of type `EthSwt_MgmtObjectType` provided by the Ethernet Switch driver that contains the timestamping information.

On reception side module provides the abstraction of the Ethernet Switch driver API `EthIf_GetRxMgmtObject()`. This allows to gain access to a shared object of type `EthSwt_MgmtObjectType` provided by the Ethernet Switch driver that contains the timestamping information. The API must be used in context of `<User>_RxIndication()` triggered for the frame one is interested in.

Note
This feature is only operable if Ethernet Switch Frame Management is used. So only Ethernet frames that are managed by the Ethernet Switch can be timestamped. However, it is possible that the Ethernet Switch hardware can't take timestamps for any kind of Ethernet frames but only for special frames like PTP frames.

2.1.2.4 Zero-Copy Extension

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 the Technical Reference of the respective driver for more about the support.

2.1.2.4.1 Feature 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.

2.1.2.4.2 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 Davinci Configurator Classic.

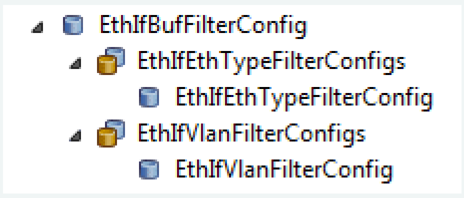

EthIfBufFilterConfig
EthIfEthTypeFilterConfigs
EthIfEthTypeFilterConfig
EthIfVlanFilterConfigs
EthIfVlanFilterConfig

Figure 2.2 Screenshot of an EtherType and a VLAN buffer filter container

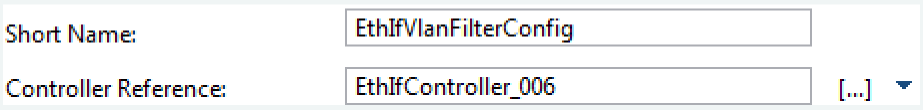

Short Name: EthIfVlanFilterConfig
Controller Reference: EthIfController_006

Figure 2.3 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.

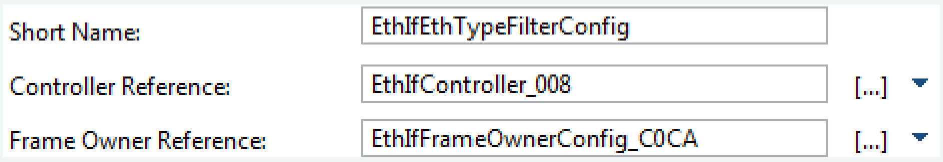

Short Name: EthIfEthTypeFilterConfig
Controller Reference: EthIfController_008
Frame Owner Reference: EthIfFrameOwnerConfig_C0CA

Figure 2.4 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 can 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 starvation. In these situations, the unlocking strategy must be reworked or the amount of receive buffers must be increased.

2.1.2.4.3 External Transmission Buffers

Further external transmission buffers need to be announced to the Ethernet Driver by configuration. The integrator must specify the maximum number 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:

Step By Step

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 (number of bytes needed for Destination-/Source-MAC, EtherType 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.

Please refer to chapter API Description to see the detailed signature of the functions.

2.1.2.5 Extended Traffic Handling

The module 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 the related section under Configuration.

2.1.2.5.1 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 module 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.

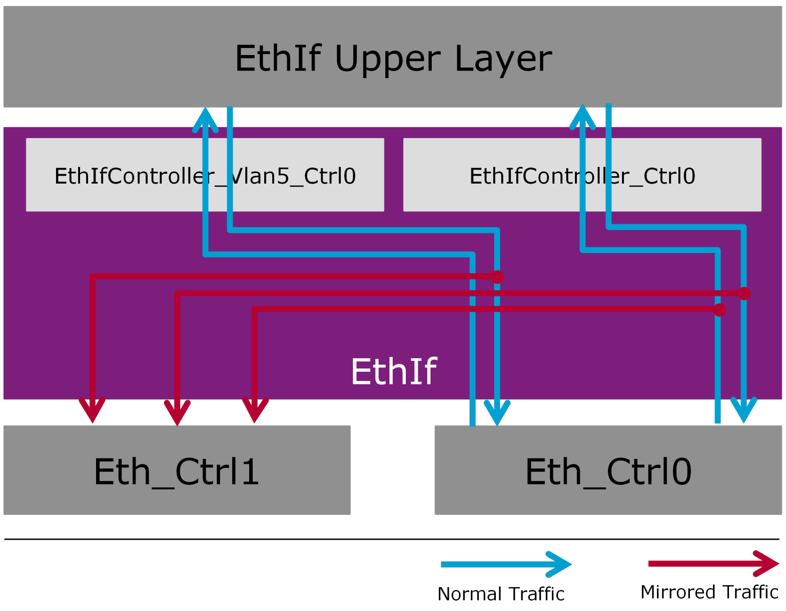


Figure 2.5 Traffic Mirroring

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.

2.1.2.5.2 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.

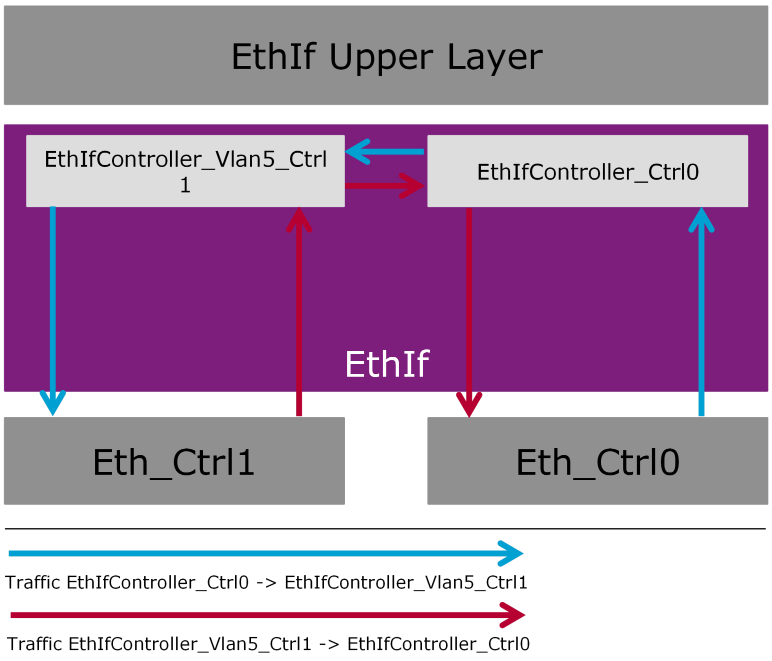

EthIf Upper Layer
EthIfController_Vlan5_Ctrl 1
EthIfController_Ctrl0
EthIf
Eth_Ctrl1
Eth_Ctrl0
Traffic EthIfController_Ctrl0 -> EthIfController_Vlan5_Ctrl1
Traffic EthIfController_Vlan5_Ctrl1 -> EthIfController_Ctrl0

Figure 2.6 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`.

`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.

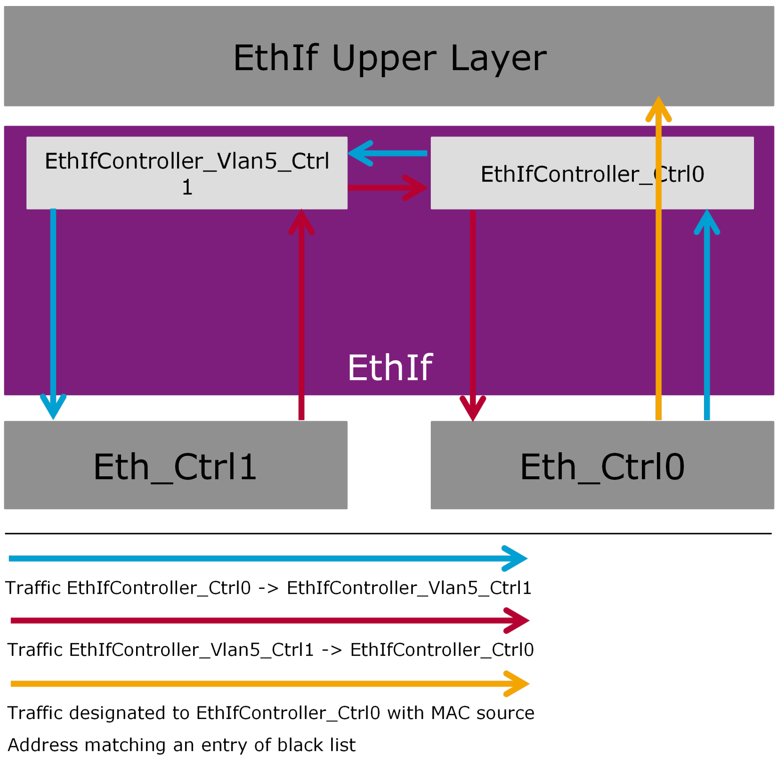

EthIf Upper Layer
EthIfController_Vlan5_Ctrl
1
EthIfController_Ctrl0
EthIf
Eth_Ctrl1
Eth_Ctrl0
Traffic EthIfController_Ctrl0 -> EthIfController_Vlan5_Ctrl1
Traffic EthIfController_Vlan5_Ctrl1 -> EthIfController_Ctrl0
Traffic designated to EthIfController_Ctrl0 with MAC source
Address matching an entry of black list

Figure 2.7 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.

2.1.2.6 Header Access APIs

The Header 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.
`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.

Table 2.4 Header Access APIs

Please refer to chapter API Description to see the detailed signature of the functions.

2.1.2.7 Ethernet Tx/Rx Statistics

The module 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 the chapter API Description.

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 API description of `EthIf_TxStatsType/EthIf_RxStatsType` for details) indicating the overflow and aren't updated anymore.

2.1.2.8 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 EthIf Controller doesn't control a Switch Port Group.

The standard mechanism for link aggregation considers a link as a combination of the link state of all physical layer components, assigned to an EthIf Controller. This implies that both the associated Ethernet Transceiver and the Ethernet Switch Port Group (which includes all the Host/Uplink ports in the Port Group and at least one non-role port, if present) must have established a physical link.

Note
A Switch Port Group must always contain the switch Host Port and any ports that are used to reach the Host Port from any of the other ports in the Port Group, the so-called "Uplink Ports". These required ports get added to the Port Group automatically.

When Lax Link Aggregation feature is 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. With this the minimal communication path is ensured (by an established link on at least two Ethernet Switch Ports of the Switch Port Group irrespective of their role) and 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`.

2.1.2.9 Multi-Core Environment Support

The module allows to utilize the processing power of a multi-core system by handling the reception and transmission of Ethernet frames on dedicated cores. The exact mapping of TX/RX frame processing to a dedicated core is based on the VLAN ID of an Ethernet frame. Therefore, each EthIf Controller with a unique VLAN ID requires the mapping to an EcuC Partition which is related to one core. This is done by setting the parameter `EthIfEcucPartitionRef` of an EthIf Controller in the Davinci Configurator Classic.

All other EthIf processing activities beside of TX/RX frame processing are always done only by the main core. The configuration of the whole Ethernet stack must be consistent regarding the TX/RX frame processing for a certain VLAN ID / EthIf Controller by a dedicated core. This has to be considered for the used Ethernet Driver and any EthIf Upper Layer module.

Caution
Multi-core environment support is only available if the hardware platform and the used Ethernet Controller driver are both supporting the multi-core use case!

Caution
Multi-core environment support is considered as disabled if all the partition references are pointing to the same partition!

2.1.2.10 Wakeup Support

The module 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 module 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 module 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 to the the related section under Configuration.

2.1.2.11 Parameter Checking

The internal parameter checks of the API functions can be en-/disabled separately. The AUTOSAR standard requires en-/disabling of the complete parameter checking only.

Note
The internal parameter checks of the API functions can be en-/disabled through the parameter `/MICROSAR/EthIf/EthIfGeneral/EthIfDevErrorDetect` of Davinci Configurator Classic.

2.2 Error Handling

2.2.1 Development Error Reporting

The module supports development error detection and reporting using the AUTOSAR Det. If enabled, these errors are reported using the service `Det_ReportError()`.

The reported service IDs and error IDs are contained and documented in the module header file.

2.2.2 Production Code Error Reporting

The module does not support production error detection and reporting.

2.3 Compliance

These features from AUTOSAR R23-11 *[8]* have deviations

- Transceiver Mode Management
- Public Interfaces, Callback Functions and Schedulable Functions
- Development Errors
- Initialization
- Imported Datatypes
- Switch Port Group
- Controller Mode Management
- Function Prototypes
- Configuration Parameters
- Transceiver Wakeup Mode
- Transceiver Signal Quality
- TxConfirmation
- MKA

Please see the following sections for details about the deviations.

2.3.1 Transceiver Mode Management Deviations

In contradiction to AUTOSAR standard 23-11 the APIs `EthIf_GetTransceiverMode()`, `EthIf_TransceiverGetLinkState()`, `EthIf_TransceiverGetBaudRate()`, `EthIf_TransceiverGetDuplexMode()` 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. This controller serves as an abstraction for the tuple composed of the underlying hardware elements, such as the 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 API `EthIf_GetControllerMode()`.

Additionally, `EthTrcv_SetTransceiverMode()` is called from `EthIf_SetControllerMode()` with `ETH_MODE_ACTIVE` and `ETH_MODE_DOWN` in their respective cases. This approach is used instead of calling it in the context of `EthIf_CtrlModeIndication()`, as this function is implemented as an empty function." The reason for that is to guarantee the compatibility by using the 3rd party AUTOSAR conformed Ethernet Controller Driver, which calls `EthIf_CtrlModeIndication()`. In `EthIf_CtrlModeIndication()`, there is no call to EthSM, which is specified in AUTOSAR.

2.3.2 Unprovided Public Interfaces, Callback Functions and Schedulable Functions Deviations

Not all Public Interfaces, Callback functions and Schedulable functions specified in AUTOSAR Standard 23-11 are Implemented. Refer the chapter API Description for more information about the services provided by the module.

2.3.3 Development Errors Deviations

The following table lists Development Error IDs which has deviation relative to AUTOSAR.

Development Error	Corresponding ID
ETHIF_E_INV_SWT_IDX	0x08u
ETHIF_E_INV_PORT_GROUP_IDX	0x03u
ETHIF_E_UNINIT	0x04u
ETHIF_E_PARAM_POINTER	0x05u
ETHIF_E_INV_PARAM	0x06u
ETHIF_E_INIT_FAILED	0x07u
ETHIF_E_INV_PORT_IDX	Not implemented

Table 2.5 Deviation in Development Error IDs relative to AUTOSAR R23-11

The following table lists Development Errors that are implemented in addition to the ones specified by AUTOSAR.

Development Error	Corresponding ID
ETHIF_E_INV_DRIVER_API_CALL	0x09u
ETHIF_E_INV_STATE	0x0Au
ETHIF_TRAFFIC_GATEWAY_E_ MANIPULATE_PROMISCUOUS_MODE	0x10u
ETHIF_TRAFFIC_GATEWAY_E_NO_BUFFER	0x11u

continues on next page

Table 2.6 – continued from previous page

Development Error	Corresponding ID

Table 2.6 Additional Development Errors

2.3.4 Initialization Deviations

In contradiction with AUTOSAR standard 23-11, the initialization of the module is performed through the `EthIf_ControllerInit()` and `EthIf_Init()` functions. The function `EthIf_ControllerInit()` is responsible for the initialization of the Ethernet controller. `EthIf_Init()` is responsible for putting the Ethernet interface module in the initialized state and link time configuration. The `EthIf_Init()` function must be called before `EthIf_ControllerInit()`.

2.3.5 Imported Datatypes Deviations

In contradiction with AUTOSAR standard 23-11, all Eth types are not present. Other datatype deviations are listed in the following table.

Datatype	Deviation
EthTrcv_BaudRateType	Defined in "EthTrcv_GeneralTypes.h"
EthTrcv_CableDiagResultType	Defined in "EthTrcv_GeneralTypes.h"
EthTrcv_DuplexModeType	Defined in "EthTrcv_GeneralTypes.h"
EthTrcv_LinkStateType	Defined in "EthTrcv_GeneralTypes.h"
EthTrcv_PhyLoopbackModeType	Defined in "EthTrcv_GeneralTypes.h"
EthTrcv_PhyTestModeType	Defined in "EthTrcv_GeneralTypes.h"
EthTrcv_PhyTxModeType	Defined in "EthTrcv_GeneralTypes.h"
EthTrcv_WakeupModeType	Defined in "EthTrcv_GeneralTypes.h"
Eth_MacVlanType	Not implemented
Eth_TxErrorCounterValuesType	Not implemented
Eth_TimeStampQualType	Implemented as "Eth_TimeStampQualityType"

Table 2.7 Imported Datatypes Deviation

2.3.6 Switch Port Group Deviations

In contradiction with AUTOSAR standard 23-11, EthIf does not delay the shutdown of an `EthIfPhysController` referencing an `EthIfSwitch` until the ports of the switch are in down state. There is no timer used when `EthIf_SwitchPortRequestMode()` is called with `ETH_MODE_DOWN`.

In addition, if the Lax Link Aggregation feature is enabled, port groups are only considered Active when the number of ports (independent of role) with link state Active is bigger than the threshold.

2.3.7 Controller Mode Management Deviations

In contradiction with AUTOSAR standard 23-11, there are no calls to `EthSwt_PortLinkStateRequest()` from the function `EthIf_SetControllerMode()` either in the `ETH_MODE_ACTIVE` case or the `ETH_MODE_DOWN` case. Furthermore, there are no calls made to `EthTrcv_TransceiverLinkStateRequest()` either in the `ETH_MODE_ACTIVE` case or the `ETH_MODE_DOWN` case.

Moreover, the symbols `ETHTRCV_MODE_ACTIVE` and `ETHTRCV_MODE_DOWN` are used instead of `ETH_MODE_ACTIVE` and `ETH_MODE_DOWN` respectively in the service `EthIf_SetControllerMode()` to represent the mode of the Ethernet controller driver.

2.3.8 Configuration Parameters Deviations

In contradiction with AUTOSAR standard 23-11, there are differences in the configuration parameters name, range, multiplicity, along with extra parameters and unimplemented parameters. For detailed information about the configuration parameters please check the BSWMD *[6]*.

Some parameters are implemented in an indirect way by checking configurations in other module components. For example, the service `EthIf_GetPortMacAdrr()` is switched on and off by the preprocessor switch `ETHIF_GET_PORT_MAC_ADDR_API`. This symbol is generated if the configuration parameter EthSwtGetPortMacAddrApi belonging to the Ethernet switch module is set to on in the Davinci configurator Classic. The following table lists the services which are switched on and off in an indirect way.

Services without a configuration switch
`EthIf_GetPortMacAddr()`
`EthIf_GetArlTable()`
`EthIf_StoreConfiguration()`
`EthIf_ResetConfiguration()`

Table 2.8 Services without a configuration switch

2.3.9 Transceiver Wakeup Mode Deviations

In contradiction with AUTOSAR standard 23-11, the transceiver index (TrcvIdx) is not used as an input argument in the service `EthIf_SetTransceiverWakeupMode()` because it is not uniquely identifiable by the EthIf. Instead, the wakeup source is used as a parameter. For more information about the service `EthIF_SetTransceiverWakeupMode()`, please check the chapter API description.

Also, in the service `EthIf_SetTransceiverWakeupMode()` the current state is not checked before forwarding the call to `EthTrcv_SetTransceiverWakeupMode()`.

2.3.10 Transceiver Signal Quality Deviations

In contradiction with AUTOSAR standard 23-11, `EthIf_ClearTrcvSignalQuality()` does not forward a call to `EthTrcv_ClearSignalQuality()`. Instead, the signal quality data is cleared in a data structure in the EthIf.

2.3.11 TxConfirmation Deviations

The API `EthIf_TxConfirmation()` is implemented according to multiple AUTOSAR standards, which can be selected by the configuration parameter `EthIfEnableAutosarForwardCompatibility`. For AUTOSAR versions >=4.3.0 the switch shall be set to true. Otherwise, it shall be set to false. With the parameter being enabled the signature of the function is changed to also include the parameter `Result`. Deviating to the AUTOSAR version >=4.3.0 definition the parameter `Result` is not forwarded to the upper layer, but gets ignored within the Ethernet Interface, as the MICROSAR upper layer of the Ethernet Interface do not make use of the information.

2.3.12 MKA Deviations

In contradiction with AUTOSAR standard 23-11, the EthIf accesses multiple Vector-specific BSWMD parameter within the MKA module, resulting in incompatibility with third-party MKA modules. Only the Vector MKA module can be used with EthIf if MACsec is enabled.

2.4 States

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

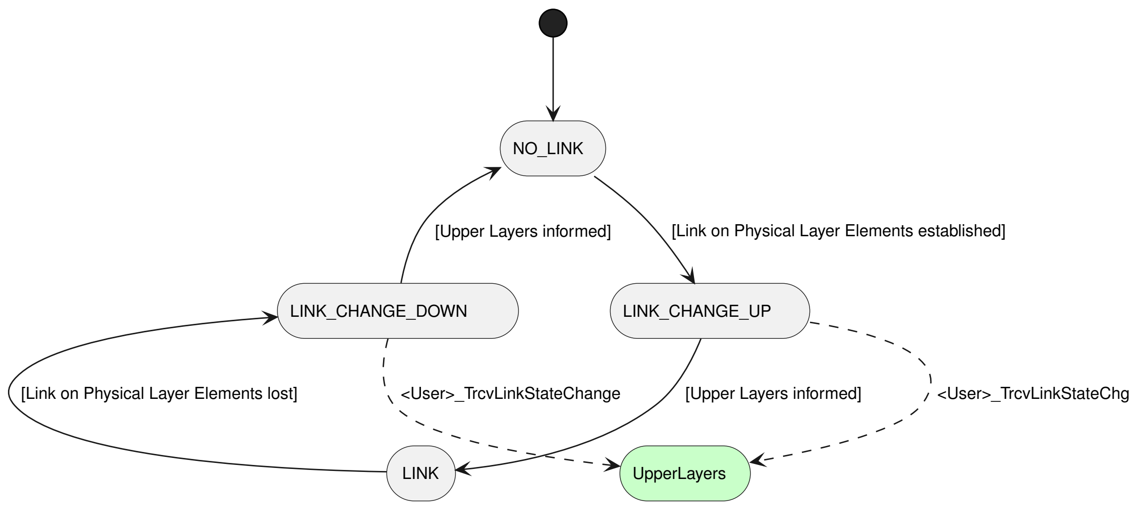


Figure 2.8 EthIf Controller Link State Machine

Initially the EthIf Controller is in state `NO_LINK`. If all 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_CHANGE_UP` and from state `LINK` to `LINK_CHANGE_DOWN` can be influenced by the feature 'Lax Link Aggregation' described in the related section under Functional Description.

3 Integration

Caution
Please note that Vector's Ethernet Interface only supports MICROSAR Classic Ethernet Controller, Switch, and Transceiver Drivers.

3.1 Embedded Implementation

The module contains these source code files. Some files are generated by the configuration tool DaVinci Configurator Classic.

File Name	Description	Integration Tasks
EthIf.c	Implementation of public API of EthIf not related to a sub-module.	-
EthIf.h	Header file to access the public API of EthIf.	-
EthIf_Cbk.h	Header file to access the public call back API of EthIf.	-
EthIf_Int.h	Private header file of EthIf.	-
EthIf_Types.h	Header file to access the public types of EthIf.	-
EthIf_<Sub>.c	Implementation of the public API of a EthIf sub-module.	-
EthIf_<Sub>.h	Header file to access the public API of a EthIf sub-module.	-
EthIf_<Sub>_Cbk.h	Header file to access the public call-back API of a EthIf sub-module.	-
EthIf_<Sub>_Int.c	Implementation of the private API of a EthIf sub-module.	-
EthIf_<Sub>_Int.h	Header file to access the private API of a EthIf sub-module.	-
EthIf_<Sub>_Types.h	Header file to access the public types of a EthIf sub-module.	-
EthIf_<Sub>_Types_Int.h	Header file to access the private types of a EthIf sub-module.	-
EthIf_Cfg.h	Generated file that contains pre-compile time parameter configuration.	-
EthIf_Lcfg.h	Generated file that contains link-time parameter configuration declaration.	-
EthIf_Lcfg.c	Generated file that contains link-time parameter configuration.	-

continues on next page

Table 3.1 – continued from previous page

File Name	Description	Integration Tasks
EthIf_HwTypes.h	Generated file that contains the interface to access the driver specific types header files.	-
EthIf_GenTypes.h	Generated file that contains the interface to access the types of the generated data.	-

Table 3.1 Source Code Files

3.2 Initialization

The module is initialized by calling the `EthIf_Init()` service with the respective configuration pointer pointing to the pre-compile-configuration due to Pre-Compile-Configuration-Variant only support.

Note
When using the module in a Davinci Configurator Classic project with the MICROSAR Classic stack the initialization functions and respective configuration pointers are autonomously registered and configured and no additional steps must be taken.

Caution
If start-up code doesn't initialize the RAM and therefore some data isn't set to a predefined value the module 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 Main Functions

The module has multiple main functions. It is dependent on receive and transmit mode (polling or interrupt) whether they exist or not. Following table lists all possible main functions and describes which processing is done by them.

Note
The main functions aren't declared by the module itself but declaration is provided by the SchM during generation with Davinci Configurator Classic.

`EthIf_MainFunctionState()`

continues on next page

Table 3.2 – continued from previous page

Existence	`EthIfEnableMainFunctionState` is activated.
Timing	Scheduled according to the period provided by configuration parameter `EthIfMainFunctionPeriod` and `EthIfTrcvLinkStateChgMainReload`. `Period = EthIfMainFunctionPeriod * EthIfTrcvLinkStateChgMainReload`
Description	Processes the link state observation of the connected transceivers/switch-ports and propagates changes to the upper layers. Processes the signal quality monitoring of the connected transceivers/switch-ports with `Period = EthIfSignalQualityCheckPeriod`, which is an integral multiple of the period of `EthIf_MainFunctionState()`.

Table 3.2 EthIf_MainFunctionState Description

`EthIf_MainFunctionRx()`	
Existence	`EthIfEnableRxInterrupt` is deactivated.
Timing	Scheduled according the period provided by configuration parameter `EthIfMainFunctionPeriod`.
Description	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 3.3 EthIf_MainFunctionRx Description

`EthIf_MainFunctionTx()`	
Existence	`EthIfEnableTxInterrupt` is deactivated.
Timing	Scheduled according to the period provided by configuration parameter `EthIfMainFunctionPeriod`.
Description	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 and the signal quality monitoring normally performed by the state main function are done here. The link state observation is performed each `n`-th run of the main function where `n` is the value of `EthIfTrcvLinkStateChgMainReload`. The signal quality monitoring is performed each `m`-th run of the main function where `m` is an integral multiple of `n`. If `EthIfPortStartupActiveTime` is active then corresponding counter is decremented. Once the counter reaches zero (timer expired) the timer is deactivated and `EthIf_DeactivateAllPortGroups()` is called.

Table 3.4 EthIf_MainFunctionTx Description

3.4 Synchronization

3.4.1 Critical Sections

To protect internal data structures against unwanted modification, the module uses critical sections for blocking concurrent access.

Note
For better readability the prefix *ETHIF_EXCLUSIVE_AREA* of the exclusive area names was removed and must be considered during integration.
When using DaVinci Configurator Classic and the respective MICROSAR Classic 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 must 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 area is kept as short as possible by just protecting the access and modification of a semaphore managed by EthIf. Therefore, no internal function calls must be expected within the exclusive area.
`TX_MIRROR_ELEMENT`	This exclusive area ensures the data consistency of the traffic mirroring related data structures.
`RXTX_STATS`	This exclusive area ensures consistency of the statistic counters provided for an EthIf-controller or on EthIf global level (measurement data).
`SIGNAL_QUALITY`	This exclusive area ensures consistency of the transceiver and switch port signal quality statistic counters.

Table 3.5 Exclusive Areas

3.5 Memory Sections

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 module and illustrates their assignment among each other.

Compiler Abstraction Definitions / Memory Mapping Sections	ETHIF_CONST	ETHIF_VAR	ETHIF_CODE
ETHIF_START_SEC_CONST_UNSPECIFIED	x		
ETHIF_START_SEC_CONST_32	x		
ETHIF_START_SEC_CONST_16	x		
ETHIF_START_SEC_CONST_8	x		
ETHIF_START_SEC_PBCFG	x		
ETHIF_START_SEC_PBCFG_ROOT	x		
ETHIF_START_SEC_VAR_NO_INIT_UNSPECIFIED		x	
ETHIF_START_SEC_VAR_NO_INIT_32		x	
ETHIF_START_SEC_VAR_NO_INIT_16		x	
ETHIF_START_SEC_VAR_NO_INIT_8		x	
ETHIF_START_SEC_VAR_ZERO_INIT_UNSPECIFIED		x	
ETHIF_START_SEC_CODE			x
ETHIF_START_SEC_CODE_ISR			x

3.6 Callouts

The module has the following possible callout functions. Please refer to the related section under API Description for more information.

```
void <User>_RxIndication (uint8 EthIfCtrlIdx,
                          Eth_FrameType FrameType,
                          boolean IsBroadcast,
                          uint8 * PhysAddrPtr,
                          uint8 * DataPtr,
                          uint16 LenByte)
{
   /* do something with the received frame */
}
```

Listing 3.1 <User>_RxIndication Template

```
void <User>_TxConfirmation (uint8 EthIfCtrlIdx,
                            uint8 BufferIdx)
{
   /* handle successful transmission of frame */
}
```

Listing 3.2 <User>_TxConfirmation Template

```
void <User>_TrcvLinkStateChg (uint8 EthIfCtrlIdx,
                              EthTrcv_LinkStateType TrcvLinkState)
{
   /* handle transceiver link state change */
}
```

Listing 3.3 <User>_TrcvLinkStateChg Template

3.7 Error Reporting Det

The module has these properties used for the error reporting to the Det.

> `ETHIF_MODULE_ID` = 65

> `ETHIF_VENDOR_ID` = 30

3.8 Delivery Migration

This section provides information about migration steps that must be performed manually when updating your MICROSAR Classic delivery to a newer release.

Target MICROSAR Classic Release	Migration Steps
MSR4-R20 () (EthIf, 9.00.00)	- Delete EthIf_EthCtrlCfg.h, EthIf_EthCtrlCfg.c in GenData folder of the DaVinci project - Delete EthIf_EthTrcvCfg.h, EthIf_EthTrcvCfg.c in GenData folder of the DaVinci project - Delete EthIf_EthSwtCfg.h, EthIf_EthSwtCfg.c in GenData folder of the DaVinci project

Table 3.7 Delivery Migration

3.9 API Usage

This section provides information about things that needs to be considered when using APIs mentioned in this section.

3.9.1 EthIf_SwitchPortGroupRequestMode API

This API shall be used for Activation/Deactivation of the switch port groups that are handled by BswM in synchronous manner.

When the activation of a port group is requested, if any of the ports associated with the requested port group is in state Down then an activation request will be triggered to the corresponding switch and a port mode counter will be incremented on successful activation. If is already activated, then only the port mode counters are incremented for the corresponding ports.

When the deactivation of the port group is requested, if any of the ports associated with the requested port group is in state Active, then the port mode counters are decremented for the corresponding port. Deactivation is triggered only when the port mode counter reaches zero, meaning that the last active user of the port also requested deactivation.

Hence, the activation and deactivation should be done in a synchronous manner. Ideally, each user should request the activation once and deactivation once. If a user is requesting for activation more than once for whatever reason, then they must request the deactivation for each activation request explicitly.

Caution
If the activation and deactivation are not done in a synchronous manned, this leads to an inconsistent port group state and unexpected behavior.

Caution
It should be made sure that at any given point of time, total number of active activations requests (number of activation requests - number of deactivation requests) should not exceed 255 (Size of `EthIf_SwitchPortGroupIdxType`). Exceeding this leads to DET errors and unexpected behavior. Hence, it is highly recommended the user of the switch port group does not request for an already activated switch port group index.

4 API Description

4.1 Provided Functions

The module provides these functions.

Service functions are implemented by the module and are typically called by upper-layer modules. They are declared in `*.h`.

4.1.1 EthIf_MainFunctionState

Prototype	
void **EthIf_MainFunctionState**( )	
Parameters	
None	
Return code	
void	
Functional Description	
Link state supervision Main Function. Main function to monitor link state changes of the managed hardware elements.	
Options	
Callcontext: TASK Reentrant: This function is non-reentrant Synchronous: TRUE	

4.1.2 EthIf_MacSecUpdateSecY

Prototype	
Std_ReturnType **EthIf_MacSecUpdateSecY**(uint8 CtrlIdx, const Mka_MacSecConfigType * MACsecCfgPtr, uint64 TxSci)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
MACsecCfgPtr [in]	Pointer to the structure to configure a MACsec Entity (SecY)
TxSci [in]	Secure Channel Identifier for the MACsec's Transmission Secure channel
Return code	
Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted
Functional Description	

continues on next page

Table 4.2 – continued from previous page

Requests to update the SecY/PAC of the PHY with the provided parameters. Requests the Ethernet Transceiver to update the SecY/PAC of the PHY with the provided parameters. A Transmission Secure Channel with the provided SCI shall be configured during the first call. A pointer to a MACsec Basic Parameters Configuration file shall be provided to create the Secure Channel.
Options
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: FALSE

4.1.3 EthIf_MacSecUpdateSecYNotification

Prototype	
void **EthIf_MacSecUpdateSecYNotification**(uint8 CtrlIdx)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
Return code	
void	
Functional Description	
Informs MKA module that requested Security Entity (SecY) update has been performed.	
Options	
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: TRUE	

4.1.4 EthIf_MacSecInitRxSc

Prototype	
Std_ReturnType **EthIf_MacSecInitRxSc**(uint8 CtrlIdx, uint64 Sci)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
Sci [in]	Secure Channel Identifier for the MACsec's Reception Secure channel
Return code	

continues on next page

Table 4.4 – continued from previous page

Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted
Functional Description	
Requests to configure a Reception Secure Channel. Requests the Ethernet Transceiver Driver to configure a Reception Secure Channel for the given Secure Channel Identifier.	
Options	
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: TRUE	

4.1.5 EthIf_MacSecResetRxSc

Prototype	
Std_ReturnType **EthIf_MacSecResetRxSc**(uint8 CtrlIdx, uint64 Sci)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
Sci [in]	Secure Channel Identifier for the MACsec's Reception Secure channel
Return code	
Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted
Functional Description	
Requests to reset to default the MACsec values of the Reception Secure Channel. Requests the Ethernet Transceiver Driver to reset to default the MACsec values of the Reception Secure Channel for the given Secure Channel Identifier.	
Options	
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: TRUE	

4.1.6 EthIf_MacSecAddTxSa

Prototype	
Std_ReturnType **EthIf_MacSecAddTxSa**(uint8 CtrlIdx, uint8 An, uint64 NextPn, uint32 Ssci, const Mka_SakKeyPtrType * KeysPtr, boolean Active)	
Parameters	

continues on next page

Table 4.6 – continued from previous page

CtrlIdx [in]	EthIf Controller Index
An [in]	Association Number to use in the MACsec's transmission secure association
NextPn [in]	Next accepted Packet Number in the MACsec's transmission secure association
Ssci [in]	Short Secure Channel Identifier used in the MACsec's transmission secure association
KeysPtr [in]	Pointer to the SAKs Key (and needed Key information) to use in the MACsec's transmission secure association
Active [in]	Boolean to enable/disable the MACsec's transmission secure association
Return code	
Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted
Functional Description	
Requests to create a Transmission Secure Association. Requests the Ethernet Transceiver Driver to create a Transmission Secure Association in the Transceiver. The Short Secure Channel Identifier is included to support XPN configurations.	
Options	
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: FALSE	

4.1.7 EthIf_MacSecAddTxSaNotification

Prototype	
void **EthIf_MacSecAddTxSaNotification**(uint8 CtrlIdx)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
Return code	
void	
Functional Description	
Informs MKA module that requested Tx Secure Association has been created.	
Options	

continues on next page

Table 4.7 – continued from previous page

Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: TRUE

4.1.8 EthIf_MacSecUpdateTxSa

Prototype	
Std_ReturnType **EthIf_MacSecUpdateTxSa**(uint8 CtrlIdx, uint8 An, uint64 NextPn, boolean Active)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
An [in]	Association Number to use in the MACsec's transmission secure association
NextPn [in]	Next accepted Packet Number in the MACsec's transmission secure association
Active [in]	Boolean to enable/disable the MACsec's transmission secure association
Return code	
Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted
Functional Description	
Requests to update the Transmission Secure Association. Requests the Ethernet Transceiver Driver to update the Transmission Secure Association with the given Packet Number. The Active parameter is included to change the specified AN status.	
Options	
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: TRUE	

4.1.9 EthIf_MacSecDeleteTxSa

Prototype	
Std_ReturnType **EthIf_MacSecDeleteTxSa**(uint8 CtrlIdx, uint8 An)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
An [in]	Association Number to use in the MACsec's transmission secure association
Return code	

continues on next page

Table 4.9 – continued from previous page

Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted
Functional Description	
Requests to remove the Transmission Secure Association. Requests the Ethernet Transceiver Driver to remove the Transmission Secure Association identified by the provided Association Number.	
Options	
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: TRUE	

4.1.10 EthIf_MacSecAddRxSa

Prototype	
Std_ReturnType **EthIf_MacSecAddRxSa**(uint8 CtrlIdx, uint8 An, uint64 LowestPn, uint32 Ssci, const Mka_SakKeyPtrType * KeysPtr, boolean Active)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
An [in]	Association Number to use in the MACsec's reception secure association
LowestPn [in]	Lowest accepted Packet Number in the MACsec's reception secure association
Ssci [in]	Short Secure Channel Identifier used in the MACsec's reception secure association
KeysPtr [in]	Pointer to the SAKs Key (and needed Key information) to use in the MACsec's reception secure association
Active [in]	Boolean to enable/disable the MACsec's reception secure association
Return code	
Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted
Functional Description	
Requests to create a Reception Secure Association. Requests the Ethernet Transceiver Driver to create a Reception Secure Association in the Transceiver. The Short Secure Channel Identifier is included to support XPN configurations.	
Options	

continues on next page

Table 4.10 – continued from previous page

Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: FALSE

4.1.11 EthIf_MacSecAddRxSaNotification

Prototype	
void **EthIf_MacSecAddRxSaNotification**(uint8 CtrlIdx)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
Return code	
void	
Functional Description	
Informs MKA module that requested Rx Secure Association has been created.	
Options	
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: TRUE	

4.1.12 EthIf_MacSecUpdateRxSa

Prototype	
Std_ReturnType **EthIf_MacSecUpdateRxSa**(uint8 CtrlIdx, uint8 An, uint64 LowestPn, boolean Active)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
An [in]	Association Number to use in the MACsec's reception secure association
LowestPn [in]	Lowest accepted Packet Number in the MACsec's reception secure association
Active [in]	Boolean to enable/disable the MACsec's reception secure association
Return code	
Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted
Functional Description	

continues on next page

Table 4.12 – continued from previous page

Requests to update the Reception Secure Association. Requests the Ethernet Transceiver Driver to update the Reception Secure Association with the given Packet Number. The Active parameter is included to change the specified AN status.
Options
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: TRUE

4.1.13 EthIf_MacSecDeleteRxSa

Prototype	
Std_ReturnType **EthIf_MacSecDeleteRxSa**(uint8 CtrlIdx, uint8 An)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
An [in]	Association Number to use in the MACsec's reception secure association
Return code	
Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted
Functional Description	
Requests to remove the Reception Secure Association. Requests the Ethernet Transceiver Driver to remove the Reception Secure Association identified by the provided Association Number.	
Options	
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: TRUE	

4.1.14 EthIf_MacSecGetTxSaNextPn

Prototype	
Std_ReturnType **EthIf_MacSecGetTxSaNextPn**(uint8 CtrlIdx, uint8 An, uint64 * NextPnPtr)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
An [in]	Association Number to use in the MACsec's reception secure association
NextPnPtr [out]	Pointer to the Next Packet Number read out from the MACsec Entity (SecY)

continues on next page

Table 4.14 – continued from previous page

Return code	
Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted
Functional Description	
Requests to return the Packet Number of Transmission Secure Association. Requests the Ethernet Transceiver Driver to return the Packet Number that is used for the next packet in the given Transmission Secure Association.	
Options	
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: TRUE	

4.1.15 EthIf_MacSecGetMacSecStats

Prototype	
Std_ReturnType **EthIf_MacSecGetMacSecStats**(uint8 CtrlIdx)	
Parameters	
CtrlIdx [in]	EthIf Controller Index
Return code	
Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted
Functional Description	
Requests the Ethernet Transceiver Driver to provide MACsec statistics.	
Options	
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: FALSE	

4.1.16 EthIf_MacSecSetControlledPortEnabled

Prototype	
Std_ReturnType **EthIf_MacSecSetControlledPortEnabled**(uint8 CtrlIdx, boolean ControlledPortEnabled)	
Parameters	
CtrlIdx [in]	EthIf Controller Index

continues on next page

Table 4.16 – continued from previous page

ControlledPortEnabled [in]	Boolean to activate the Controlled Port of the PAE
Return code	
Std_ReturnType	E_OK The request has been accepted E_NOT_OK The request has not been accepted
Functional Description	
Requests to set the Controlled Port enabled parameter of a Port Access Entity (PAE).	
Options	
Callcontext: ANY Reentrant: TRUE for different CtrlIdx, FALSE for the same CtrlIdx Synchronous: TRUE	

4.1.17 EthIf_GetTxStats

Prototype	
Std_ReturnType **EthIf_GetTxStats**(uint8 CtrlIdx, Eth_TxStatsType * EthTxStats, EthIf_TxStatsType * EthIfTxStats)	
Parameters	
CtrlIdx [in]	EthIf controller identifier
EthTxStats [out]	Transmission statistics of the respective Eth-controller
EthIfTxStats	
Return code	
Std_ReturnType	E_OK - Statistics could be retrieved E_NOT_OK - Wrong parameters or Eth-driver call isn't successful
Functional Description	
Retrieves the transmission statistic counters. 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.	
Particularities and Limitations	
Particularities: Respective EthIf controller is initialized	
Options	

continues on next page

Table 4.17 – continued from previous page

Callcontext: TASK Reentrant: FALSE Synchronous: TRUE

4.1.18 EthIf_GetRxStats

Prototype	
Std_ReturnType **EthIf_GetRxStats**(uint8 CtrlIdx, Eth_RxStatsType * EthRxStats, EthIf_RxStatsType * EthIfRxStats)	
Parameters	
CtrlIdx [in]	EthIf controller identifier
EthRxStats [out]	Reception statistics of the respective Eth-controller
EthIfRxStats	
Return code	
Std_ReturnType	E_NOT_OK - Wrong parameters or Eth-driver call isn't successful E_OK - Statistics could be retrieved
Functional Description	
Retrieves the reception statistic counters. 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.	
Particularities and Limitations	
Particularities: Respective EthIf controller is initialized	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.1.19 EthIf_GetAndResetMeasurementData

Prototype
Std_ReturnType **EthIf_GetAndResetMeasurementData**(EthIf_MeasurementIdxType MeasurementIdx, boolean MeasurementResetNeeded, uint32 * MeasurementDataPtr)
Parameters

continues on next page

Table 4.19 – continued from previous page

MeasurementIdx [in]	Data index of measurement data ETHIF_MEAS_ALL - Index only to be used to reset all MeasurementIdx's at once ETHIF_MEAS_DROP_CTRLIDX - Index of dropped datagrams caused by invalid CtrlIdx/VLAN ETHIF_MEAS_VLAN_DOUBLE_TAGGED - Index of dropped datagrams caused by double VLAN tag ETHIF_MEAS_UNKNOWN_ETHERTYPE - Index of dropped datagrams caused by unknown EtherType
MeasurementResetNeeded [in]	Flag to trigger a reset of the measurement data TRUE - Request a reset of measurement data FALSE - Do not request a reset of measurement data
MeasurementDataPtr [out]	Reference to data buffer, where to copy measurement data
Return code	
Std_ReturnType	E_NOT_OK - failed E_OK - successful
Functional Description	
Retrieves and resets detailed measurement data. Function allows to read and reset detailed measurement data for diagnostic purposes. Getting all MeasurementIdx's at once is not supported. ETHIF_MEAS_ALL shall only be used to reset all MeasurementIdx's at once. A NULL_PTR shall be provided for MeasurementDataPtr in this case. The statistics are all protected from a counter overrun by not incrementing the related statistics counter value any more once the counter reached the maximum value of the related data type.	
Particularities and Limitations	
Particularities: EthIf is initialized	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.1.20 EthIf_ControllerInit

Prototype	
Std_ReturnType **EthIf_ControllerInit**(uint8 CtrlIdx, uint8 CfgIdx)	
Parameters	
CtrlIdx [in]	EthIf controller that shall be initialized

continues on next page

Table 4.20 – continued from previous page

CfgIdx [in]	Configuration index
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters or call interrupted pending operation
Functional Description	
Initializes a EthIf controller Function initializes the EthIf controller addressed and redirects the call to the underlying hardware drivers mapped to the EthIf controller.	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.1.21 EthIf_SetControllerMode

Prototype	
Std_ReturnType **EthIf_SetControllerMode**(uint8 CtrlIdx, Eth_ModeType CtrlMode)	
Parameters	
CtrlIdx [in]	EthIf controller the mode shall be changed for
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 E_NOT_OK - function has been called with invalid parameters or call interrupted pending operation
Functional Description	
Modifies the EthIf controller mode Function alters the EthIf controller mode and redirects the call to the underlying hardware drivers mapped to the EthIf controller.	
Particularities and Limitations	
Particularities: EthIf_ControllerInit() was called for the respective EthIf controller before	
Options	
Callcontext: TASK Reentrant: TRUE Synchronous: TRUE	

4.1.22 EthIf_GetControllerMode

Prototype	
Std_ReturnType **EthIf_GetControllerMode**(uint8 CtrlIdx, Eth_ModeType * CtrlModePtr)	
Parameters	
CtrlIdx [in]	EthIf controller index
CtrlModePtr [out]	Pointer to store the mode retrieved: ETH_MODE_DOWN - EthIf controller is turned of ETH_MODE_ACTIVE - EthIf controller is active
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the EthIf controller mode	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.23 EthIf_SwitchPortGroupRequestMode

Prototype	
Std_ReturnType **EthIf_SwitchPortGroupRequestMode**(EthIf_SwitchPortGroupIdxType PortGroupIdx, EthTrcv_ModeType PortMode)	
Parameters	
PortGroupIdx [in]	EthIf switch port group the mode shall be changed for
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 E_NOT_OK - port group mode could not be changed
Functional Description	

continues on next page

Table 4.23 – continued from previous page

Requests a mode for a EthIf switch port group
Options
Callcontext: ANY Reentrant: FALSE Synchronous: TRUE

4.1.24 EthIf_StartAllPorts

Prototype	
Std_ReturnType **EthIf_StartAllPorts**( )	
Parameters	
None	
Return code	
Std_ReturnType	E_OK - Request was accepted E_NOT_OK - Request was rejected
Functional Description	
Request to set all configured and affected EthSwtPorts to ETH_MODE_ACTIVE This function activates all inactive EthSwtPortGroups(Hence all mapped EthSwtPorts) and starts the EthIfPortStartupActiveTime timer.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: FALSE	

4.1.25 EthIf_SetTransceiverWakeupMode

Prototype	
Std_ReturnType **EthIf_SetTransceiverWakeupMode**(EcuM_WakeupSourceType WakeupSource, EthTrcv_WakeupModeType WakeupMode)	
Parameters	
WakeupSource [in]	EcuM wakeup source
WakeupMode [in]	Wakeup mode to set
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	

continues on next page

Table 4.25 – continued from previous page

Changes the wakeup mode of the related hardware driver Function allows to change the wakeup mode of the related hardware driver by redirecting the call depending on the passed EcuM wakeup source.
Options
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE

4.1.26 EthIf_GetTransceiverWakeupMode

Prototype	
Std_ReturnType **EthIf_GetTransceiverWakeupMode**(EcuM_WakeupSourceType WakeupSource, EthTrcv_WakeupModeType * WakeupModePtr)	
Parameters	
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 E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the wakeup mode of the related hardware driver Function allows to retrieve the wakeup mode of the related hardware driver by redirecting the call depending on the passed EcuM wakeup source.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.27 EthIf_CheckWakeup

Prototype	
void **EthIf_CheckWakeup**(EcuM_WakeupSourceType WakeupSource)	
Parameters	
WakeupSource [in]	EcuM wakeup source
Return code	
void	
Functional Description	

continues on next page

Table 4.27 – continued from previous page

Initiates the wakeup check Function allows to initiate the wakeup check of the related hardware driver by redirecting the call depending on the passed EcuM wakeup source.
Options
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE

4.1.28 EthIf_VSetTransceiverWakeupMode

Prototype	
Std_ReturnType **EthIf_VSetTransceiverWakeupMode**(uint8 CtrlIdx, EthTrcv_WakeupModeType WakeupMode)	
Parameters	
CtrlIdx [in]	EthIf controller index
WakeupMode [in]	Wakeup mode to set
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Changes the wakeup mode of the related hardware drivers Function allows to change the wakeup mode of the related hardware drivers by redirecting the call depending on the passed EcuM wakeup source.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.29 EthIf_VGetTransceiverWakeupMode

Prototype	
Std_ReturnType **EthIf_VGetTransceiverWakeupMode**(uint8 CtrlIdx, EthTrcv_WakeupModeType * WakeupModePtr)	
Parameters	
CtrlIdx [in]	EthIf controller index
WakeupModePtr [out]	Pointer pointing to variable where the wakeup mode is stored to
Return code	

continues on next page

Table 4.29 – continued from previous page

Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the wakeup mode of the related hardware driver Function allows to retrieve the wakeup mode of the related hardware driver by redirecting the call depending on the passed EcuM wakeup source.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.30 EthIf_VCheckWakeup

Prototype	
void **EthIf_VCheckWakeup**(uint8 CtrlIdx)	
Parameters	
CtrlIdx [in]	EthIf controller index
Return code	
void	
Functional Description	
Initiates the wakeup check Function allows to initiate the wakeup check of the related hardware drivers by redirecting the call depending on the passed EthIf controller index.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.31 EthIf_SetPhyTxMode

Prototype	
Std_ReturnType **EthIf_SetPhyTxMode**(uint8 TrcvIdx, EthTrcv_PhyTxModeType Mode)	
Parameters	
TrcvIdx [in]	Index of the transceiver within the context of the Ethernet Interface.
Mode [in]	Transmission mode to be activated.
Return code	

continues on next page

Table 4.31 – continued from previous page

Std_ReturnType	E_OK - The request has been accepted E_NOT_OK - The request has not been accepted
Functional Description	
Activates a given transmission mode of the related hardware driver Function allows to activate a given transmission mode of the related hardware driver by redirecting the call depending on the passed transceiver index.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.32 EthIf_GetCableDiagnosticsResult

Prototype	
Std_ReturnType **EthIf_GetCableDiagnosticsResult**(uint8 TrcvIdx, EthTrcv_CableDiagResultType * ResultPtr)	
Parameters	
TrcvIdx [in]	Index of the transceiver within the context of the Ethernet Interface.
ResultPtr [out]	Pointer to the location where the cable diagnostics result shall be stored.
Return code	
Std_ReturnType	E_OK - The request has been accepted E_NOT_OK - The request has not been accepted
Functional Description	
Retrieves the cable diagnostics result of a given transceiver. Function allows to retrieve the cable diagnostics result of the related hardware driver by redirecting the call depending on the passed transceiver index.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.33 EthIf_RunCableDiagnostic

Prototype	
Std_ReturnType **EthIf_RunCableDiagnostic**(uint8 TrcvIdx)	
Parameters	
TrcvIdx [in]	Index of the transceiver within the context of the Ethernet Interface.
Return code	
Std_ReturnType	E_OK - The trigger has been accepted E_NOT_OK - The trigger has not been accepted
Functional Description	
Trigger the cable diagnostics for the given Ethernet transceiver. Function triggers the cable diagnostics for the given Ethernet transceiver. This function is used for asynchronous handling of cable diagnostic, that means, the EthIf User calls this function (EthIf_RunCableDiagnostic) to trigger the cable diagnostic and then use another function (EthIf_RunCableDiagnosticsResult) to get the result of the cable diagnostic.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: FALSE	

4.1.34 EthIf_SetPhyTestMode

Prototype	
Std_ReturnType **EthIf_SetPhyTestMode**(uint8 TrcvIdx, EthTrcv_PhyTestModeType Mode)	
Parameters	
TrcvIdx [in]	Index of the transceiver within the context of the Ethernet Interface.
Mode [in]	Test mode to be activated.
Return code	
Std_ReturnType	E_OK - The request has been accepted E_NOT_OK - The request has not been accepted
Functional Description	
Activates a given test mode of the related hardware driver Function allows to activate a given test mode of the related hardware driver by redirecting the call depending on the passed transceiver index.	
Options	

continues on next page

Table 4.34 – continued from previous page

Callcontext: ANY Reentrant: TRUE Synchronous: TRUE

4.1.35 EthIf_GetTrcvSignalQuality

Prototype	
Std_ReturnType **EthIf_GetTrcvSignalQuality**(uint8 TrcvIdx, EthIf_SignalQualityResultType * ResultPtr)	
Parameters	
TrcvIdx [in]	Index of the transceiver within the context of the Ethernet Interface
ResultPtr [out]	Pointer to the memory where the signal quality in percent shall be stored
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - The signal quality not retrieved successfully
Functional Description	
Retrieves the signal quality of the link of the given Ethernet transceiver and update the measured signal quality stored in EthIf. Function allows to retrieve the signal quality of the link of the given Ethernet transceiver by redirecting the call depending on the passed transceiver index. The highest signal quality and the lowest signal quality will be updated depending on the currently retrieved signal quality. Function is reentrant for different TrcvIdx and non reentrant for the same TrcvIdx.	
Particularities and Limitations	
Particularities: EthIf is initialized	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.36 EthIf_ClearTrcvSignalQuality

Prototype	
Std_ReturnType **EthIf_ClearTrcvSignalQuality**(uint8 TrcvIdx)	
Parameters	
TrcvIdx [in]	Index of the transceiver within the context of the Ethernet Interface

continues on next page

Table 4.36 – continued from previous page

Return code	
Std_ReturnType	E_OK - success E_NOT_OK - The signal quality not cleared successfully
Functional Description	
Clear the stored signal quality of the link of the given Ethernet transceiver. Function clears in Ethernet interface stored signal quality of the link of the given Ethernet transceiver. Function is reentrant for different TrcvIdx and non reentrant for the same TrcvIdx.	
Particularities and Limitations	
Particularities: EthIf is initialized	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.37 EthIf_ProvideExtTxBuffer

Prototype	
BufReq_ReturnType **EthIf_ProvideExtTxBuffer**(uint8 CtrlIdx, Eth_FrameType FrameType, uint8 Priority, uint8 * BufIdxPtr, Eth_DataType ** BufPtr, uint16 * LenBytePtr)	
Parameters	
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	Buffer pointer: [in] - Location of the buffer provided externally [out] - Location where payload can be written to
LenBytePtr	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	

continues on next page

Table 4.37 – continued from previous page

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 Function allows to provide an external buffer for an Ethernet frame transmission.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.38 EthIf_ReleaseRxBuffer

Prototype	
Std_ReturnType **EthIf_ReleaseRxBuffer**(uint8 CtrlIdx, Eth_DataType * BufPtr)	
Parameters	
CtrlIdx [in]	EthIf controller index
BufPtr [in]	Pointer to the buffer to be released
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Releases an reception buffer Function releases an reception buffer and allows the underlying Ethernet driver to reuse it for further receptions.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.39 EthIf_RxIndication

Prototype	
void **EthIf_RxIndication**(uint8 CtrlIdx, Eth_FrameType FrameType, boolean IsBroadcast, const uint8 * PhysAddrPtr, const Eth_DataType * DataPtr, uint16 LenByte)	
Parameters	
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	
Functional Description	
Notifies the EthIf about a received Ethernet frame Functions takes the given Ethernet frame data, analysis the Ethernet header for VLAN and EtherType information and decides whether to drop the frame or pass it to a known EthIf user.	
Options	
Callcontext: TASK|ISR1|ISR2 Reentrant: TRUE Synchronous: TRUE	

4.1.40 EthIf_MainFunctionRx

Prototype	
void **EthIf_MainFunctionRx**( )	
Parameters	
None	
Return code	
void	
Functional Description	

continues on next page

Table 4.40 – continued from previous page

callcontext TASK **reentrant** This function is non-reentrant **synchronous** TRUE Reception Main Function. Main function to handle Ethernet frame reception in polling mode.

4.1.41 EthIf_GetPortMacAddr

Prototype	
Std_ReturnType **EthIf_GetPortMacAddr**(const uint8 * MacAddrPtr, uint8 * SwitchIdxPtr, uint8 * PortIdxPtr)	
Parameters	
MacAddrPtr [in]	MAC address to be queried for
SwitchIdxPtr [out]	Pointer to store the switch the port belongs to
PortIdxPtr [out]	Pointer to store the port
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters or no port information found
Functional Description	
Retrieves the switch port a MAC address is assigned to. Function allows to retrieve the switch port a Ethernet node with the given MAC address is connected to.	
Options	
Callcontext: ANY Reentrant: FALSE Synchronous: TRUE	

4.1.42 EthIf_GetArlTable

Prototype	
Std_ReturnType **EthIf_GetArlTable**(uint8 SwitchIdx, uint32 * LenPtr, EthSwt_MacVlanType * ArlTablePtr)	
Parameters	
SwitchIdx [in]	Switch the ARL table shall be retrieved for

continues on next page

Table 4.42 – continued from previous page

LenPtr	Pointer to length of the buffer passed: [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 E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the complete address resolution table. Function allows to retrieve the valid entries of the address resolution table of a switch instance.	
Options	
Callcontext: ANY Reentrant: FALSE Synchronous: TRUE	

4.1.43 EthIf_GetBufferLevel

Prototype	
Std_ReturnType **EthIf_GetBufferLevel**(uint8 SwitchIdx, EthSwt_BufferLevelType * SwitchBufferLevelPtr)	
Parameters	
SwitchIdx [in]	Switch the buffer level shall be retrieved for
SwitchBufferLevelPtr [out]	The interpretation of this value is switch dependent
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Reads the buffer level of the currently used buffer of the switch Function reads the buffer level of the currently used buffer of the switch.	
Particularities and Limitations	
Particularities: Module and drivers have been initialized	
Options	
Callcontext: ANY Reentrant: FALSE Synchronous: TRUE	

4.1.44 EthIf_GetDropCount

Prototype	
Std_ReturnType **EthIf_GetDropCount**(uint8 SwitchIdx, uint16 * LenPtr, uint32 * DropCountPtr)	
Parameters	
SwitchIdx [in]	Switch the drop counts shall be retrieved for
LenPtr	Pointer to the length of the buffer passed: [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 E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the drop counts according to the AUTOSAR SWS. Function allows to retrieve the drop counts specified by the AUTOSAR SWS. Each count is the sum of the drop count of all ports.	
Particularities and Limitations	
Particularities: Module and drivers have been initialized	
Options	
Callcontext: ANY Reentrant: FALSE Synchronous: TRUE	

4.1.45 EthIf_StoreConfiguration

Prototype	
Std_ReturnType **EthIf_StoreConfiguration**(uint8 SwitchIdx)	
Parameters	
SwitchIdx [in]	Switch the configuration shall be stored for
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Calls EthSwt_StoreConfiguration() API of the related EthSwt-driver. 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.	

continues on next page

Table 4.45 – continued from previous page

Particularities and Limitations
Particularities: Module and drivers have been initialized
Options
Callcontext: ANY Reentrant: FALSE Synchronous: TRUE

4.1.46 EthIf_ResetConfiguration

Prototype	
Std_ReturnType **EthIf_ResetConfiguration**(uint8 SwitchIdx)	
Parameters	
SwitchIdx [in]	Switch the configuration shall be invalidated for
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Calls EthSwt_ResetConfiguration() API of the related EthSwt-driver. 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.	
Particularities and Limitations	
Particularities: Module and drivers have been initialized	
Options	
Callcontext: ANY Reentrant: FALSE Synchronous: TRUE	

4.1.47 EthIf_SwitchUpdateMCastPortAssignment

Prototype	
Std_ReturnType **EthIf_SwitchUpdateMCastPortAssignment**(uint8 SwitchIdx, uint8 PortIdx, const uint8 * MCastAddr, EthSwt_MCastPortAssignActionType Action)	
Parameters	
SwitchIdx [in]	Index of the EthIf switch
PortIdx [in]	Index of the Ethernet Switch Port

continues on next page

Table 4.47 – continued from previous page

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 E_NOT_OK - function has been called with invalid parameters
Functional Description	
Updates the multicast assignment for a specific port. Function updates the multicast assignment for a specific Ethernet switch port.	
Options	
Callcontext: ANY Reentrant: FALSE Synchronous: TRUE	

4.1.48 EthIf_SwitchEnableVlan

Prototype	
Std_ReturnType **EthIf_SwitchEnableVlan**(uint8 SwitchIdx, uint8 PortIdx, uint16 VlanId, boolean Enable)	
Parameters	
SwitchIdx [in]	Index of the EthIf switch
PortIdx [in]	Index of the Ethernet Switch Port
VlanId [in]	VLAN identifier for which the configuration shall be performed
Enable [in]	Configuration that shall be applied: TRUE - VLAN-configuration enabled FALSE - VLAN-configuration disabled
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - VLAN configuration could not be performed
Functional Description	
Enables or disables a VLAN on a port. Function enables or disables a VLAN on a specific switch port.	
Options	

continues on next page

Table 4.48 – continued from previous page

Callcontext: ANY Reentrant: FALSE Synchronous: TRUE

4.1.49 EthIf_SetSwitchMgmtInfo

Prototype	
Std_ReturnType **EthIf_SetSwitchMgmtInfo**(uint8 CtrlIdx, Eth_BufIdxType BufIdx, EthSwt_MgmtInfoType * MgmtInfoPtr)	
Parameters	
CtrlIdx [in]	Index of the EthIf controller
BufIdx [in]	Index of the Ethernet Tx buffer retrieved during EthIf_ProvideTxBuffer()
MgmtInfoPtr [in]	Switch Management information
Return code	
Std_ReturnType	E_OK - success 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. 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(). **Note** Further context restrictions: Call only allowed between EthIf_ProvideTxBuffer()/EthIf_ProvideExtTxBuffer() and EthIf_Transmit() for a specific frame	
Particularities and Limitations	
Particularities: Buffer has to be acquired with EthIf_ProvideTxBuffer()	
Options	
Callcontext: ANY Reentrant: FALSE Synchronous: TRUE	

4.1.50 EthIf_GetRxMgmtObject

Prototype	
Std_ReturnType **EthIf_GetRxMgmtObject**(uint8 CtrlIdx, Eth_DataType * DataPtr, EthSwt_ MgmtObjectType ** MgmtObjectPtr)	
Parameters	
CtrlIdx [in]	EthIf controller the frame is assigned to
DataPtr [in]	Pointer to the frame payload for identifying the frame
MgmtObjectPtr [out]	Switch management object provided by the Ethernet switch driver
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function called with invalid parameters
Functional Description	
Provides access to a reception switch management object Function allows to gain access to the reception switch management object provided by the related Ethernet switch driver. This function can only be called in context of <User>_RxIndication(). **Note** Further restriction of context: Call only allowed in <User>_RxIndication()	
Options	
Callcontext: TASK|ISR1|ISR2 Reentrant: TRUE Synchronous: TRUE	

4.1.51 EthIf_GetTxMgmtObject

Prototype	
Std_ReturnType **EthIf_GetTxMgmtObject**(uint8 CtrlIdx, Eth_BufIdxType BufIdx, EthSwt_ MgmtObjectType ** MgmtObjectPtr)	
Parameters	
CtrlIdx [in]	Index of the EthIf controller
BufIdx [in]	Index of the Ethernet Tx buffer retrieved during EthIf_ProvideTxBuffer()
MgmtObjectPtr [out]	Switch management object provided by the Ethernet switch driver
Return code	

continues on next page

Table 4.51 – continued from previous page

Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Provides access to a transmission switch management object Function allows to gain access to the transmission switch management object provided by the related Ethernet switch driver. This function can only be called in context of <User>_ TxConfirmation() **Note** Further context restrictions: Call only allowed in <User>_TxConfirmation()	
Particularities and Limitations	
Particularities: Buffer has to be acquired with EthIf_ProvideTxBuffer()	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.52 EthIf_SwitchEnableTimeStamping

Prototype	
Std_ReturnType **EthIf_SwitchEnableTimeStamping**(uint8 CtrlIdx, Eth_BufIdxType BufIdx, EthSwt_MgmtInfoType * MgmtInfo)	
Parameters	
CtrlIdx [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 E_NOT_OK - function has been called with invalid parameters
Functional Description	

continues on next page

Table 4.52 – continued from previous page

Enables time stamping within the Ethernet switch for an Ethernet frame. Function enables time stamping for an Ethernet frame identified by the buffer index on the port described by the management information. **Note** Further context restriction: Call only allowed between EthIf_ProvideTxBuffer()/ EthIf_ProvideExtTxBuffer() and EthIf_Transmit() for a specific frame
Particularities and Limitations
Particularities: Buffer has to be acquired with EthIf_ProvideTxBuffer()
Options
Callcontext: ANY Reentrant: FALSE Synchronous: TRUE

4.1.53 EthIf_SwitchSetCorrectionTime

Prototype	
Std_ReturnType **EthIf_SwitchSetCorrectionTime**(uint8 EthIfSwtIdx, const Eth_TimeIntDiffType * OffsetPtr, const float64 * RateRatioPtr)	
Parameters	
EthIfSwtIdx [in]	Index of the EthIf switch
OffsetPtr [in]	Pointer to the offset the clock shall be corrected with. NULL_PTR if no offset correction shall be done.
RateRatioPtr [in]	Pointer to the rate the clock shall be corrected with. NULL_PTR if no rate correction shall be done.
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters or correction can't be applied because it is not in a range the hardware is able to handle
Functional Description	
Corrects the clock of the Ethernet switch. Function triggers the correction of the clock of the Ethernet switch either by an offset or by a rate or both.	
Options	

continues on next page

Table 4.53 – continued from previous page

Callcontext: ANY Reentrant: FALSE Synchronous: TRUE

4.1.54 EthIf_StartSwitchQbvSchedule

Prototype	
Std_ReturnType **EthIf_StartSwitchQbvSchedule**(uint8 EthIfSwtIdx)	
Parameters	
EthIfSwtIdx [in]	Index of the EthIf switch
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters or Ethernet switch isn't in a state allowing to start the schedule
Functional Description	
Starts the IEEE802.1Qbv schedule of the Ethernet switch. Function triggers the start of the IEEE802.1Qbv schedule of the Ethernet switch.	
Particularities and Limitations	
Particularities: The Ethernet switch must have been synchronized to the global time.	
Options	
Callcontext: ANY Reentrant: FALSE Synchronous: TRUE	

4.1.55 EthIf_StopSwitchQbvSchedule

Prototype	
Std_ReturnType **EthIf_StopSwitchQbvSchedule**(uint8 EthIfSwtIdx)	
Parameters	
EthIfSwtIdx [in]	Index of the EthIf switch
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters or Ethernet switch isn't in a state allowing to stop the schedule
Functional Description	

continues on next page

Table 4.55 – continued from previous page

Stops the IEEE802.1Qbv schedule of the Ethernet switch. Function triggers the stop of the IEEE802.1Qbv schedule of the Ethernet switch.
Particularities and Limitations
Particularities: The IEEE802.1Qbv schedule must have been started previously.
Options
Callcontext: ANY Reentrant: FALSE Synchronous: TRUE

4.1.56 EthIf_GetSwitchPortSignalQuality

Prototype	
Std_ReturnType **EthIf_GetSwitchPortSignalQuality**(uint8 SwitchIdx, uint8 SwitchPortIdx, EthIf_SignalQualityResultType * ResultPtr)	
Parameters	
SwitchIdx [in]	Index of the Ethernet switch within the context of the Ethernet Interface
SwitchPortIdx [in]	Index of the port at the addressed switch
ResultPtr [out]	Pointer to the memory where the signal quality in percent shall be stored.
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - The signal quality not retrieved successfully
Functional Description	
Retrieves the signal quality of the link of the given Ethernet switch port and update the measured signal quality stored in EthIf. Function allows to retrieve the signal quality of the link of the given Ethernet switch port by redirecting the call depending on the passed switch index and switch port index. The highest signal quality and the lowest signal quality will be updated depending on the currently retrieved signal quality. Function is reentrant for different Ethernet switch indexes and Ethernet Switch port indexes, non reentrant for the same SwitchPortIdx.	
Particularities and Limitations	
Particularities: EthIf is initialized	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.57 EthIf_ClearSwitchPortSignalQuality

Prototype	
Std_ReturnType **EthIf_ClearSwitchPortSignalQuality**(uint8 SwitchIdx, uint8 SwitchPortIdx)	
Parameters	
SwitchIdx [in]	Index of the Ethernet switch within the context of the Ethernet Interface
SwitchPortIdx [in]	Index of the port at the addressed switch
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - The signal quality not cleared successfully
Functional Description	
Clear the stored signal quality of the link of the given Ethernet switch port. Function clears the in Ethernet interface stored signal quality of the link of the given Ethernet switch port. Function is reentrant for different Ethernet switch indexes and Ethernet Switch port indexes, non reentrant for the same SwitchPortIdx.	
Particularities and Limitations	
Particularities: EthIf is initialized	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.58 EthIf_RunPortCableDiagnostic

Prototype	
Std_ReturnType **EthIf_RunPortCableDiagnostic**(uint8 SwitchIdx, uint8 SwitchPortIdx)	
Parameters	
SwitchIdx [in]	Index of the Ethernet switch within the context of the Ethernet Interface
SwitchPortIdx [in]	Index of the port at the addressed switch
Return code	
Std_ReturnType	E_OK - The trigger to run the cable diagnostic has been accepted E_NOT_OK - The trigger to run the cable diagnostic has not been accepted
Functional Description	

continues on next page

Table 4.58 – continued from previous page

Trigger the cable diagnostics for the given Ethernet switch port Function allows to retrieve the cable diagnostics result of the given Ethernet switch port. This function is used for asynchronous handling of cable diagnostic, that means, the EthIf User calls this function (EthIf_RunPortCableDiagnostic) to trigger the cable diagnostic and then use another function (EthIf_GetPortCableDiagnosticsResult) to get the result of the cable diagnostic.
Particularities and Limitations
Particularities: EthIf is initialized
Options
Callcontext: ANY Reentrant: TRUE Synchronous: FALSE

4.1.59 EthIf_GetPortCableDiagnosticsResult

Prototype	
Std_ReturnType **EthIf_GetPortCableDiagnosticsResult**(uint8 SwitchIdx, uint8 SwitchPortIdx, EthTrcv_CableDiagResultType * ResultPtr)	
Parameters	
SwitchIdx [in]	Index of the Ethernet switch within the context of the Ethernet Interface
SwitchPortIdx [in]	Index of the port at the addressed switch
ResultPtr [out]	Pointer to the memory where the cable diagnostics result shall be stored.
Return code	
Std_ReturnType	E_OK - The port cable diagnostic result for the indexed Ethernet switch port was obtained successfully. E_NOT_OK - the port cable diagnostic result for the indexed Ethernet port was not obtained successfully. (i.e. indexed Ethernet switch port is not available)
Functional Description	
Retrieves the cable diagnostics result of the given Ethernet switch port Function allows to retrieve the cable diagnostics result of the given Ethernet switch port by redirecting the call to the corresponding switch driver depending on the passed switch index and switch port index.	
Particularities and Limitations	
Particularities: EthIf is initialized	
Options	

continues on next page

Table 4.59 – continued from previous page

Callcontext: ANY Reentrant: TRUE Synchronous: TRUE

4.1.60 EthIf_GetPhysAddr

Prototype	
void **EthIf_GetPhysAddr**(uint8 CtrlIdx, uint8 * PhysAddrPtr)	
Parameters	
CtrlIdx [in]	EthIf controller the MAC shall be retrieved for
PhysAddrPtr [out]	Pointer to the buffer where MAC shall be stored in
Return code	
void	
Functional Description	
Retrieves the MAC address related to the EthIf controller Function retrieves the MAC address that is used as source MAC address by the Ethernet controller mapped to the EthIf controller.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.61 EthIf_SetPhysAddr

Prototype	
void **EthIf_SetPhysAddr**(uint8 CtrlIdx, const uint8 * PhysAddrPtr)	
Parameters	
CtrlIdx [in]	EthIf controller the MAC shall be set for
PhysAddrPtr [in]	Pointer to the MAC to set
Return code	
void	
Functional Description	
Sets the MAC address related to the EthIf controller Function alters the MAC address that is used as source MAC address by the Ethernet controller mapped to the EthIf controller.	
Options	

continues on next page

Table 4.61 – continued from previous page

Callcontext: ANY Reentrant: TRUE for different Controller Indices Synchronous: TRUE

4.1.62 EthIf_UpdatePhysAddrFilter

Prototype	
Std_ReturnType **EthIf_UpdatePhysAddrFilter**(uint8 CtrlIdx, const uint8 * PhysAddrPtr, Eth_FilterActionType Action)	
Parameters	
CtrlIdx [in]	EthIf controller the filter shall be adapted for
PhysAddrPtr [in]	MAC address that shall be added/removed from filter
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 - filter successfully adapted E_NOT_OK - no filter adaption performed
Functional Description	
Modifies the receive MAC address filter related to the EthIf Controller 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.	
Options	
Callcontext: ANY Reentrant: TRUE for different Controller Indices Synchronous: TRUE	

4.1.63 EthIf_GetCurrentTime

Prototype	
Std_ReturnType **EthIf_GetCurrentTime**(uint8 CtrlIdx, Eth_TimeStampQualType * timeQualPtr, Eth_TimeStampType * timeStampPtr)	
Parameters	
CtrlIdx [in]	EthIf controller the current time shall be retrieved from

continues on next page

Table 4.63 – continued from previous page

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 - current time successfully retrieved E_NOT_OK - current time not retrieved
Functional Description	
Retrieves the current time of the Ethernet controllers timer module Function redirects the call to the Ethernet controller driver to retrieve the current time of the controllers timer module.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.64 EthIf_EnableEgressTimestamp

Prototype	
Std_ReturnType **EthIf_EnableEgressTimestamp**(uint8 CtrlIdx, uint8 BufIdx)	
Parameters	
CtrlIdx [in]	EthIf controller the frame is assigned to
BufIdx [in]	Buffer used for the frame
Return code	
Std_ReturnType	E_OK - timestamping enabled successfully E_NOT_OK - timestamping couldn't be enabled
Functional Description	
Enables timestamping for a frame Function redirects the call to the Ethernet controller driver to enable egress timestamping of a frame. **Note** Further restriction of context: Call only allowed between EthIf_ProvideTxBuffer()/EthIf_ProvideExtTxBuffer() and EthIf_Transmit()/EthIf_VTransmit() for a specific frame (identified by the BufIdx)	
Options	

continues on next page

Table 4.64 – continued from previous page

Callcontext: ANY Reentrant: TRUE Synchronous: TRUE

4.1.65 EthIf_GetEgressTimestamp

Prototype	
Std_ReturnType **EthIf_GetEgressTimestamp**(uint8 CtrlIdx, uint8 BufIdx, Eth_TimeStampType * TimestampPtr, Eth_TimestampQualityType * TimestampQualityPtr)	
Parameters	
CtrlIdx [in]	EthIf controller the frame is assigned to
BufIdx [in]	Buffer used for the frame
TimestampPtr [out]	Pointer to buffer the timestamp shall be stored to
TimestampQualityPtr [out]	Pointer to buffer the quality shall be stored to
Return code	
Std_ReturnType	E_OK - timestamp successfully retrieved E_NOT_OK - timestamp couldn't be retrieved
Functional Description	
Retrieves the egress timestamp of a frame Function redirects the call to the Ethernet controller driver to retrieve the egress timestamp of a frame. **Note** Further restriction of context: Call only allowed in <User>_TxConfirmation()	
Options	
Callcontext: TASK|ISR1|ISR2 Reentrant: TRUE Synchronous: TRUE	

4.1.66 EthIf_GetIngressTimestamp

Prototype	
Std_ReturnType **EthIf_GetIngressTimestamp**(uint8 CtrlIdx, Eth_DataType * DataPtr, Eth_TimeStampType * TimestampPtr, Eth_TimestampQualityType * TimestampQualityPtr)	
Parameters	
CtrlIdx [in]	EthIf controller the frame is assigned to

continues on next page

Table 4.66 – continued from previous page

DataPtr [in]	Pointer to the frame payload for identifying the frame
TimestampPtr [out]	Pointer to buffer the timestamp shall be stored to
TimestampQualityPtr [out]	Pointer to buffer the quality shall be stored to
Return code	
Std_ReturnType	E_OK - timestamp successfully retrieved E_NOT_OK - timestamp couldn't be retrieved
Functional Description	
Retrieves the ingress timestamp of a frame Function redirects the call to the Ethernet controller driver to retrieve the ingress timestamp of a frame. **Note** Further restriction of context: Call only allowed in <User>_RxIndication()	
Options	
Callcontext: TASK|ISR1|ISR2 Reentrant: TRUE Synchronous: TRUE	

4.1.67 EthIf_GetTxHeaderPtr

Prototype	
Std_ReturnType **EthIf_GetTxHeaderPtr**(uint8 CtrlIdx, uint8 BufIdx, Eth_DataType ** BufPtr, uint16 * LenBytePtr)	
Parameters	
CtrlIdx [in]	EthIf controller the buffer is assigned to
BufIdx [in]	Index to identify the buffer the Ethernet header location shall be retrieved for
BufPtr [out]	Location of the Ethernet header
LenBytePtr	
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the location of the Ethernet header for a given transmission buffer Function retrieves the location of the Ethernet header for a given transmission buffer.	

continues on next page

Table 4.67 – continued from previous page

Particularities and Limitations
Particularities: Buffer to retrieve the Ethernet header location for must be previously acquired by EthIf_ ProvideTxBuffer()/EthIf_ProvideExtTxBuffer()
Options
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE

4.1.68 EthIf_GetRxHeaderPtr

Prototype	
Std_ReturnType **EthIf_GetRxHeaderPtr**(uint8 CtrlIdx, Eth_DataType ** BufPtr, uint16 * LenBytePtr)	
Parameters	
CtrlIdx [in]	EthIf controller the buffer is assigned to
BufPtr	Pointer to reception buffer: [in] - Location of the payload of the Ethernet frame within the reception buffer [out] - Location of the Ethernet header
LenBytePtr	
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the location of the Ethernet header for a given reception buffer Function retrieves the location of the Ethernet header for a given reception buffer. **Note** Further restriction of context: Call only allowed during <User>_RxIndication()	
Options	
Callcontext: TASK|ISR1|ISR2 Reentrant: TRUE Synchronous: TRUE	

4.1.69 EthIf_SetBandwidthLimit

Prototype	
Std_ReturnType **EthIf_SetBandwidthLimit**(uint8 CtrlIdx, uint8 QueuePrio, uint32 BandwidthLimit)	
Parameters	
CtrlIdx [in]	EthIf controller the bandwidth shall be set for
QueuePrio [in]	Transmission queue the bandwidth shall be set for
BandwidthLimit [in]	New bandwidth limit [bit/s]
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Manipulates the maximum bandwidth of a traffic queue. Function allows to manipulate the maximum amount of bandwidth the indexed traffic queue is allowed to acquire.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.70 EthIf_GetBandwidthLimit

Prototype	
Std_ReturnType **EthIf_GetBandwidthLimit**(uint8 CtrlIdx, uint8 QueuePrio, uint32 * BandwidthLimitPtr)	
Parameters	
CtrlIdx [in]	EthIf controller the bandwidth shall be retrieved from
QueuePrio [in]	Transmission queue the bandwidth shall be retrieved from
BandwidthLimitPtr [out]	Bandwidth limit retrieved [bit/s]
Return code	
Std_ReturnType	E_OK - success E_NOT_OK - function has been called with invalid parameters
Functional Description	
Retrieves the current maximum bandwidth of a traffic queue. Function allows to retrieve the maximum amount of bandwidth the indexed traffic queue is allowed to acquire currently.	

continues on next page

Table 4.70 – continued from previous page

Options
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE

4.1.71 EthIf_ProvideTxBuffer

Prototype	
BufReq_ReturnType **EthIf_ProvideTxBuffer**(uint8 CtrlIdx, Eth_FrameType FrameType, uint8 Priority, uint8 * BufIdxPtr, Eth_DataType ** BufPtr, uint16 * LenBytePtr)	
Parameters	
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	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 Function allows to acquire a buffer where a upper layer is able to insert the payload for the Ethernet frame.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.72 EthIf_VTransmit

Prototype	
Std_ReturnType **EthIf_VTransmit**(uint8 CtrlIdx, uint8 BufIdx, Eth_FrameType FrameType, boolean TxConfirmation, uint16 LenByte, uint8 * DstMacAddrPtr, uint8 * SrcMacAddrPtr)	
Parameters	
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 E_NOT_OK - function has been called with invalid parameters
Functional Description	
Triggers transmission of an Ethernet frame with a given source MAC address 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.	
Particularities and Limitations	
Particularities: Buffer to be transmitted must be previously acquired by EthIf_ProvideTxBuffer()/EthIf_ProvideExtTxBuffer()	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.73 EthIf_Transmit

Prototype	
Std_ReturnType **EthIf_Transmit**(uint8 CtrlIdx, uint8 BufIdx, Eth_FrameType FrameType, boolean TxConfirmation, uint16 LenByte, uint8 * PhysAddrPtr)	
Parameters	
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 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 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.	
Particularities and Limitations	
Particularities: Buffer to be transmitted must be previously acquired by EthIf_ProvideTxBuffer()/EthIf_ProvideExtTxBuffer()	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.74 EthIf_TxConfirmation

Prototype	
void **EthIf_TxConfirmation**(uint8 CtrlIdx, Eth_BufIdxType BufIdx, Std_ReturnType Result)	
Parameters	
CtrlIdx [in]	Ethernet controller index
BufIdx [in]	Index of the buffer the transmission is confirmed for

continues on next page

Table 4.74 – continued from previous page

Result [in]	Indication if transmission was successful
Return code	
void	
Functional Description	
Notifies the EthIf about the transmission of a Ethernet frame Function handles the confirmation of an Ethernet frame transmission and passes it to the respective EthIf user.	
Particularities and Limitations	
Particularities: ETHIF_ENABLE_AUTOSAR_FORWARD_COMPATIBILITY == STD_ON	
Options	
Callcontext: TASK|ISR1|ISR2 Reentrant: TRUE Synchronous: TRUE	

4.1.75 EthIf_MainFunctionTx

Prototype	
void **EthIf_MainFunctionTx**( )	
Parameters	
None	
Return code	
void	
Functional Description	
callcontext TASK **reentrant** This function is non-reentrant **synchronous** TRUE Transmission confirmation Main Function. Main function to handle Ethernet frame transmission confirmation in polling mode.	

4.1.76 EthIf_InitMemory

Prototype	
void **EthIf_InitMemory**( )	
Parameters	
None	
Return code	
void	
Functional Description	
Function for _*INIT*_-variable initialization Service to initialize module global variables at power up. This function initializes the variables in _*INIT*_ sections. Used in case they are not initialized by the startup code.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.1.77 EthIf_Init

Prototype	
void **EthIf_Init**(const EthIf_ConfigType * CfgPtr)	
Parameters	
CfgPtr [in]	Configuration structure for initializing the module
Return code	
void	
Functional Description	
Initializes the EthIf module Function initializes the module EthIf. It initializes all variables and sets the module state to initialized. It must be called prior to using any other service except GetVersionInfo.	
Particularities and Limitations	
Particularities: Interrupts are disabled.	
Options	
Callcontext: TASK Reentrant: FALSE Synchronous: TRUE	

4.1.78 EthIf_GetVersionInfo

Prototype	
void **EthIf_GetVersionInfo**(Std_VersionInfoType * VersionInfoPtr)	
Parameters	
VersionInfoPtr [out]	Pointer to where to store the version information. Parameter must not be NULL.
Return code	
void	
Functional Description	
Returns the version information EthIf_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component.	
Options	
Callcontext: ANY Reentrant: TRUE Synchronous: TRUE	

4.1.79 <Up>_RxIndication

Prototype
void **<Up>_RxIndication**( )
Parameters
None
Return code
No return value (void)
Functional Description
The function notifies the upper layer about the Rx ethernet frame.

4.1.80 <Up>_TxConfirmation

Prototype
void **<Up>_TxConfirmation**( )
Parameters
None
Return code
No return value (void)
Functional Description

continues on next page

Table 4.80 – continued from previous page

The function notifies the upper layer about the completed transmission of the ethernet frame.

4.1.81 <Up>_TrcvLinkStateChg

Prototype
void **<Up>_TrcvLinkStateChg**( )
Parameters
None
Return code
No return value (void)
Functional Description
The function provides indication to the upper layer module about the change of the transceiver link state, affecting the specified controller.

4.1.82 EthIf_GetMacSecStatisticsNotification

Prototype
void **EthIf_GetMacSecStatisticsNotification**( )
Parameters
None
Return code
No return value (void)
Functional Description
The function notifies that the requested MACsec statistics are available. The request has been previously triggered by EthTrcv_MacSecGetMacSecStats().

4.2 Type Definitions

The types defined by the Ethernet Interface are described in this chapter.

Type Name	C-Type	Description	Value Range
EthIf_ConfigType	struct	Defines the Ethernet Interface configuration type	Pointer - Pointer to configuration

continues on next page

Table 4.83 – continued from previous page

Type Name	C-Type	Description	Value Range
EthIf_StateType	uint8	Defines all possible Ethernet Interface states	> `ETHIF_STATE_UNINIT` - Ethernet Interface not initialized > `ETHIF_STATE_INIT` - Ethernet Interface initialized
EthIf_MeasurementIdxType	uint8	Index to select specific measurement data	> `ETHIF_MEAS_DROP_CRTLIDX` - Measurement index of dropped datagrams caused by invalid CrtlIdx/VLAN > `ETHIF_MEAS_VLAN_DOUBLE_TAGGED` - Measurement index of dropped datagrams caused by double VLAN tag > `ETHIF_MEAS_UNKNOWN_ETHERTYPE` - Measurement index of dropped datagrams caused by unknown Ethertype > `ETHIF_MEAS_ALL` - Measurement index shall only be used to reset all MeasurementIdx's at once

Table 4.83 Type Definitions

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

Table 4.84 EthIf_FrameHdrType

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 4.85 EthIf_TxStatsType

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

Struct Element Name	C-Type	Description	Value Range
NumRxPkts	uint32	Number of received 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 received 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 4.86 EthIf_RxStatsType

Struct Element Name	C-Type	Description	Value Range
HighestSignalQuality	uint32	The highest signal quality of a link since last clear	0 .. 4.294.967.293
LowestSignalQuality	uint32	The lowest signal quality of a link since last clear	0 .. 4.294.967.293
ActualSignalQuality	uint32	The actual signal quality	0 .. 4.294.967.293

Table 4.87 EthIf_SignalQualityResultType

4.3 Callout Functions

At its configurable interfaces this module defines notifications that can be mapped to callout functions provided by other modules. The mapping is not statically defined by this module 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.

4.3.1 <User>_RxIndication

Prototype	
void **<User>_RxIndication**(uint8 EthIfCtrlIdx, Eth_FrameType FrameType, boolean IsBroadcast, uint8 * PhysAddrPtr, uint8 * DataPtr, uint16 LenByte)	
Parameters	
EthIfCtrlIdx [in]	EthIf Controller index the Ethernet frame was received on
FrameType [in]	EtherType of the Ethernet frame
IsBroadcast [in]	Information 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	
Functional Description	
Notification function informing about the reception of an Ethernet frame. Called if an Ethernet frame is received.	
Options	
Callcontext: TASK|ISR1|ISR2	

4.3.2 <User>_TxConfirmation

Prototype	
void **<User>_TxConfirmation**(uint8 EthIfCtrlIdx, uint8 BufferIdx)	
Parameters	
EthIfCtrlIdx [in]	EthIf Controller index the Ethernet frame was transmitted from
BufferIdx [in]	Buffer index to identify the Ethernet frame previously triggered for transmission
Return code	
void	
Functional Description	
Notification function informing about the finished transmission of an Ethernet frame. Called if an Ethernet frame was transmitted and TxConfirmation was enabled during EthIf_ Transmit().	
Options	
Callcontext: TASK|ISR1|ISR2	

4.3.3 <User>_TrcvLinkStateChg

Prototype	
void **<User>_TrcvLinkStateChg**(uint8 EthIfCtrlIdx, EthTrcv_LinkStateType TrcvLinkState)	
Parameters	
EthIfCtrlIdx [in]	EthIf Controller index the link state change happened on
TrcvLinkState [in]	Link state indicated - ETHTRCV_LINK_STATE_DOWN: No link anymore - ETHTRCV_LINK_STATE_ACTIVE: Link established
Return code	
void	
Functional Description	
Notification function informing about a link state change on EthIf Controller level. Called if an EthIf Controller changes its link state.	
Options	
Callcontext: TASK|ISR1|ISR2	

4.4 Required Functions

The module requires these functions from other modules. Refer to their documentation for API details.

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

Module	API
Det	> Det_ReportError
Dem	> Dem_ReportErrorStatus

continues on next page

Table 4.91 – continued from previous page

Module	API
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

continues on next page

Table 4.91 – continued from previous page

Module	API
EthTrcv	> EthTrcv_TransceiverInit > EthTrcv_SetTransceiverMode > EthTrcv_GetTransceiverMode > EthTrcv_StartAutoNegotiation > EthTrcv_GetLinkState > EthTrcv_GetBaudRate > EthTrcv_GetDuplexMode > EthTrcv_SetTransceiverWakeupMode > EthTrcv_GetTransceiverWakeupMode > EthTrcv_CheckWakeup > EthTrcv_MacSecUpdateSecY > EthTrcv_MacSecInitRxSc > EthTrcv_MacSecResetRxSc > EthTrcv_MacSecAddTxSa > EthTrcv_MacSecUpdateTxSa > EthTrcv_MacSecDeleteTxSa > EthTrcv_MacSecAddRxSa > EthTrcv_MacSecUpdateRxSa > EthTrcv_MacSecDeleteRxSa > EthTrcv_MacSecGetTxSaNextPn > EthTrcv_MacSecGetMacSecStats > EthTrcv_MacSecSetControlledPortEnabled

continues on next page

Table 4.91 – continued from previous page

Module	API
EthSwt	> EthSwt_SwitchInit > EthSwt_SetSwitchPortMode > EthSwt_GetSwitchPortMode > EthSwt_StartSwitchPortAutoNegotiation > EthSwt_GetLinkState > EthSwt_GetBaudRate > EthSwt_GetDuplexMode > EthSwt_GetPortMacAddr > EthSwt_GetArlTable > EthSwt_GetBufferLevel > EthSwt_GetDropCount > EthSwt_EnableVlan > EthSwt_StoreConfiguration > EthSwt_ResetConfiguration > EthSwt_SetMacLearningMode > EthSwt_GetMacLearningMode > EthSwt_SetSwitchMgmtInfo > EthSwt_PortEnableTimeStamp > EthSwt_UpdateMCastPortAssignment
BswM	> BswM_EthIf_PortGroupLinkStateChg
EthFw	> EthFw_IsFrameRxAllowed > EthFw_IsFrameTxAllowed

continues on next page

Table 4.91 – continued from previous page

Module	API
IdsM	> IdsM_SetSecurityEvent
MKA	> Mka_LinkStateChange > Mka_MacSecUpdateSecYNotification > Mka_MacSecAddTxSaNotification > Mka_MacSecAddRxSaNotification > Mka_GetMacSecStatisticsNotification

Table 4.91 Required Functions

4.5 Service Ports

This module does not provide any service ports.

5 Configuration

The functionalities of the module are configured with DaVinci Configurator Classic.

For details on configuration options refer to the module-specific description that is shown in DaVinci Configurator Classic.

5.1 Configuration Variants

This module supports these configuration variants:

- VARIANT-PRE-COMPILE
- VARIANT-POST-BUILD-SELECTABLE

The configuration classes of the parameters depend on the supported configuration variants. For their definitions please see the module-specific BSWMD ARXML file.

5.2 Configuration With Davinci Configurator Classic

This module is configured with the help of the configuration tool DaVinci Configurator Classic. 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 Classic.

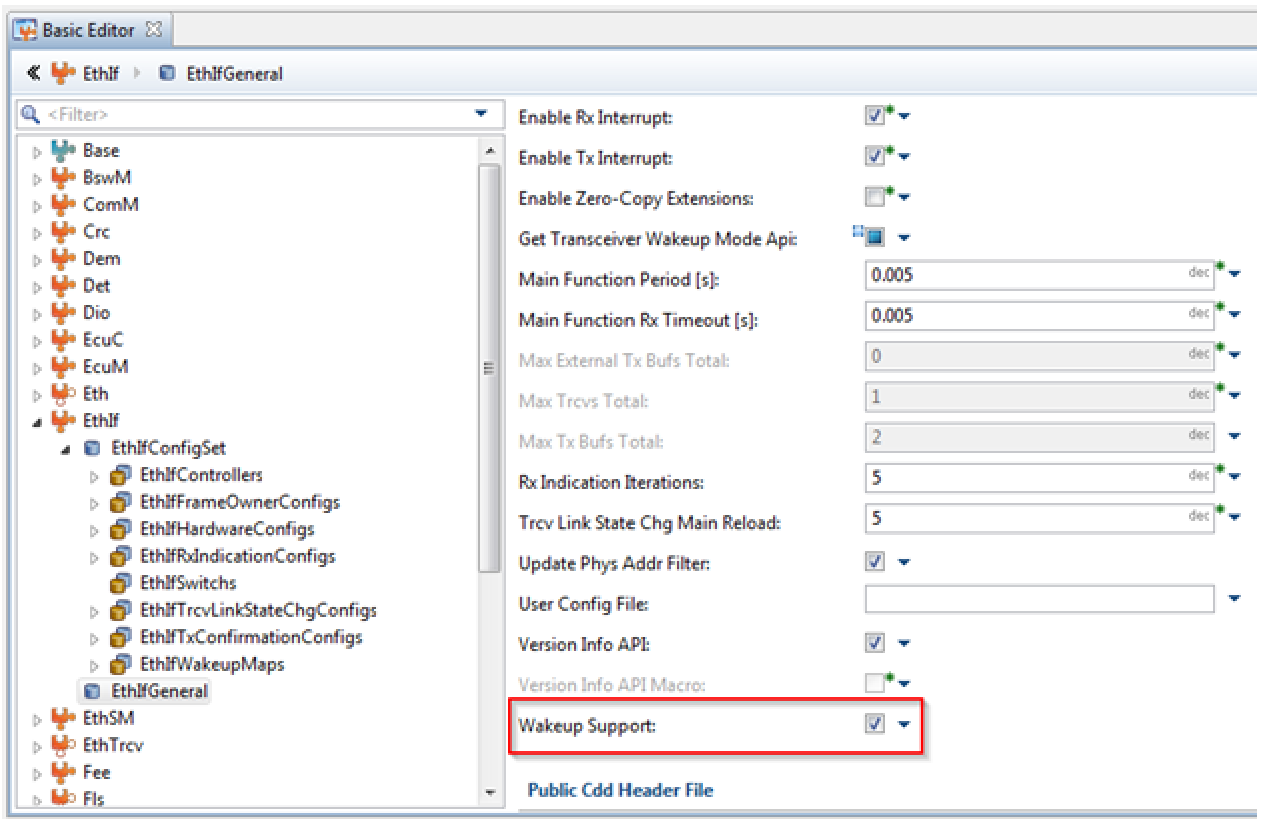
5.2.1 Wakeup Support Configuration

5.2.1.1 General 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.

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

Following screenshot marks the parameter to adapt.



Enable Rx Interrupt:	✓
Enable Tx Interrupt:	✓
Enable Zero-Copy Extensions:	
Get Transceiver Wakeup Mode Api:	■
Main Function Period [s]:	0.005 dec
Main Function Rx Timeout [s]:	0.005 dec
Max External Tx Bufs Total:	0 dec
Max Trcvs Total:	1 dec
Max Tx Bufs Total:	2 dec
Rx Indication Iterations:	5 dec
Trcv Link State Chg Main Reload:	5 dec
Update Phys Addr Filter:	✓
User Config File:	
Version Info API:	✓
Version Info API Macro:	
Wakeup Support:	✓

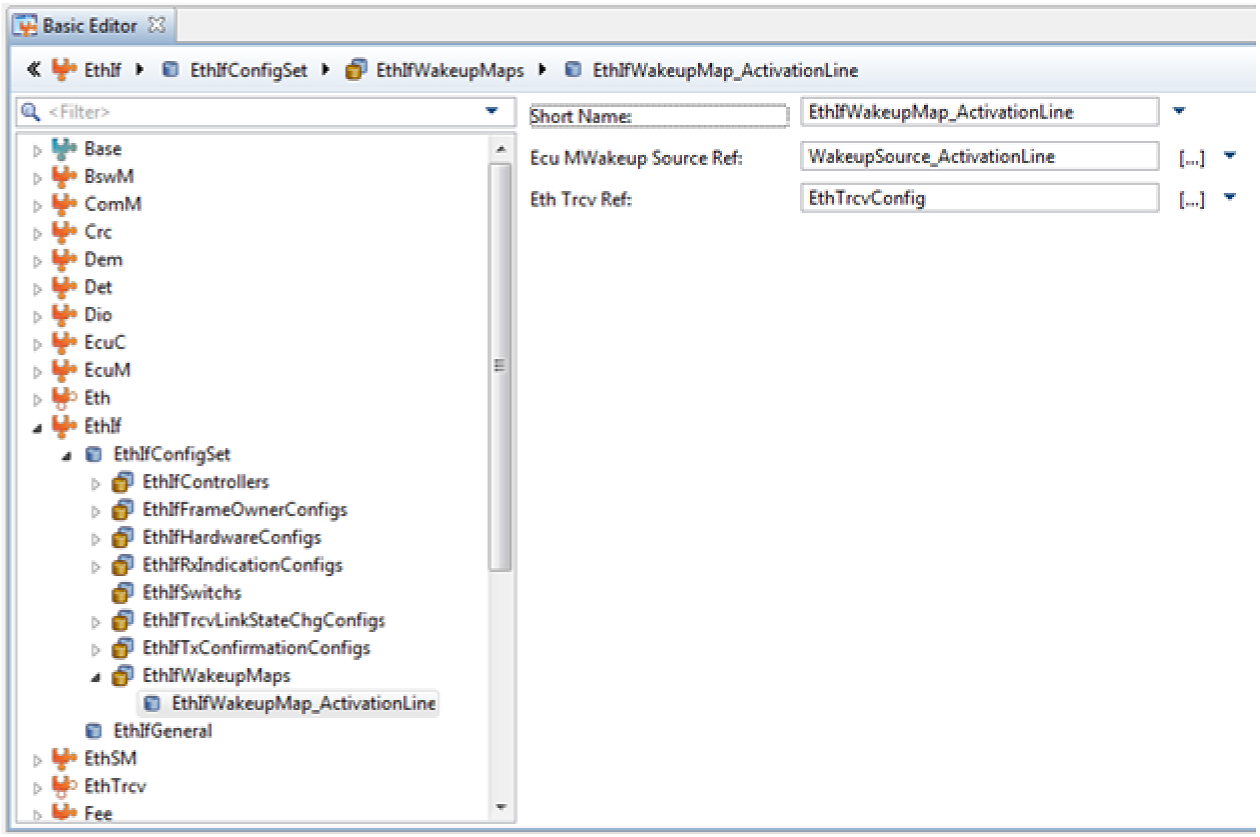
Figure 5.1 General Wakeup Support Configuration

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.

5.2.1.2 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.



Short Name: EthIfWakeupMap_ActivationLine
Ecu MWakeup Source Ref: WakeupSource_ActivationLine
Eth Trcv Ref: EthTrcvConfig

Figure 5.2 Wakeup Map Configuration

5.2.2 Extended Traffic Handling Configuration

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

5.2.2.1 Traffic Mirroring Configuration

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 the related chapter under Functional Description.

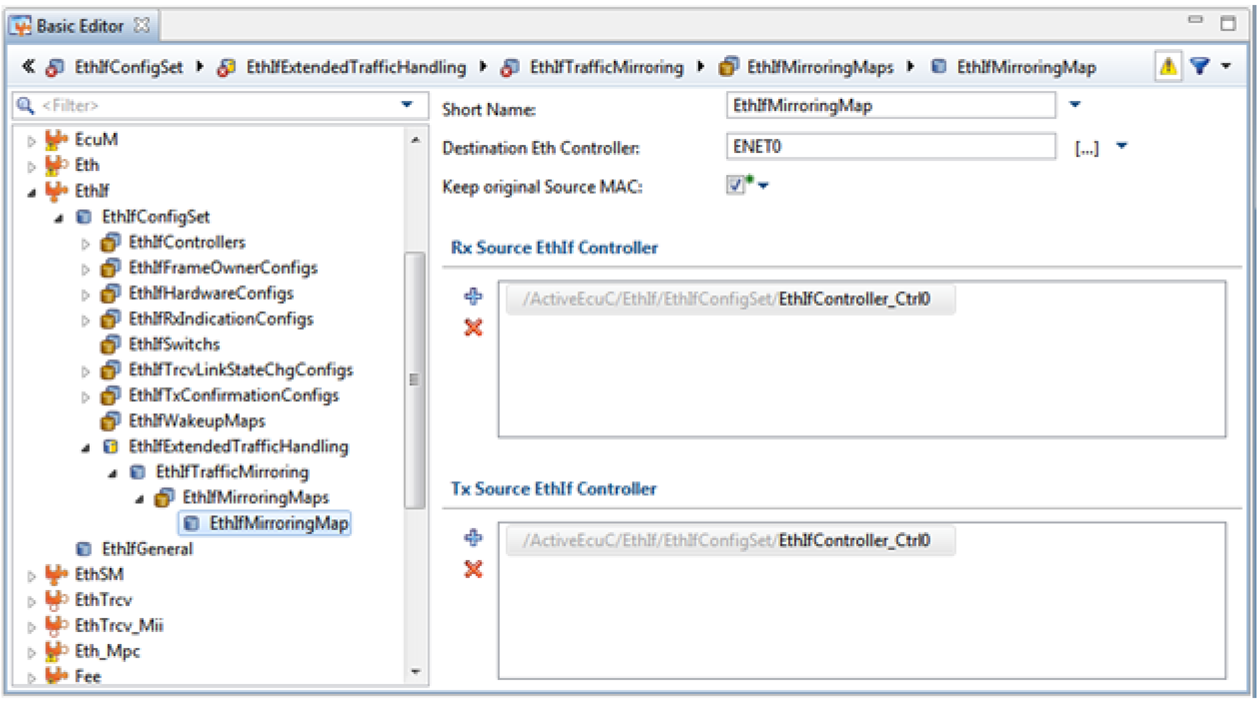
To mirror receive/transmit traffic of an EthIf controller the feature must be enabled by creating the `EthIfExtendedTrafficHandling` and its choice container `EthIfTrafficMirroring` This can be achieved by using the "Create sub container" and "Choose" options in the context menu of the respective parent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



Short Name: EthIfMirroringMap
Destination Eth Controller: ENET0
Keep original Source MAC: ✓

Rx Source EthIf Controller
/ActiveEcuC/EthIf/EthIfConfigSet/EthIfController_Ctrl0

Tx Source EthIf Controller
/ActiveEcuC/EthIf/EthIfConfigSet/EthIfController_Ctrl0

Figure 5.3 Traffic Mirroring - Mirroring Map Configuration

5.2.2.2 Traffic Gateway Configuration

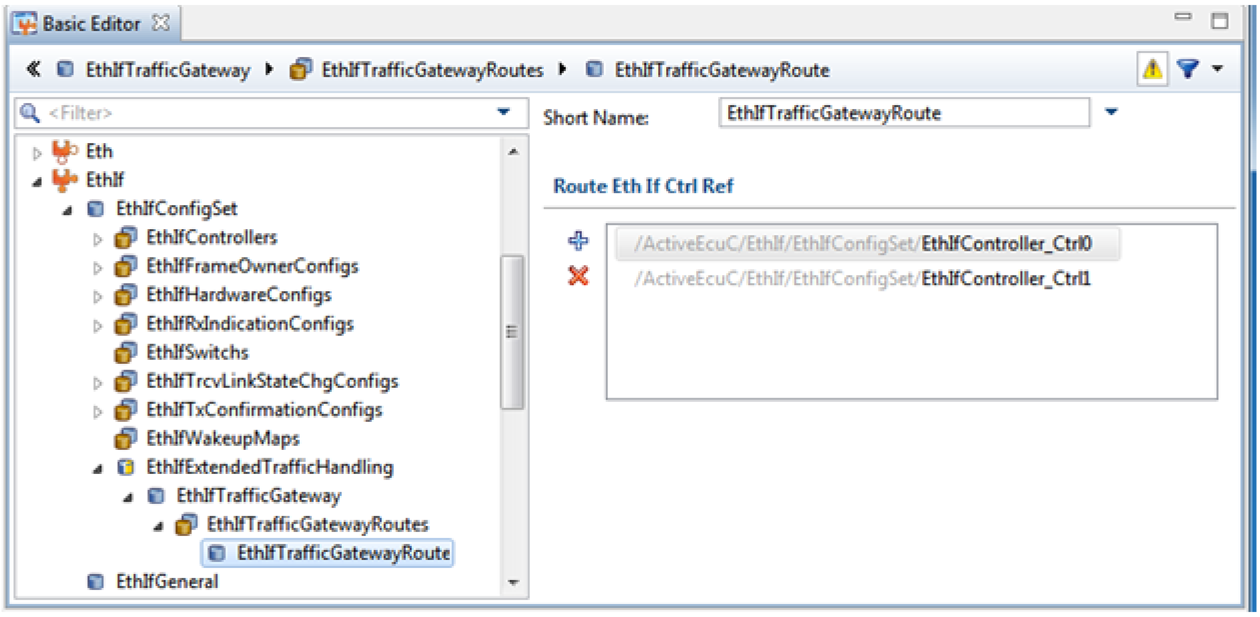
Traffic Gateway allows to route traffic from one EthIf controller to another without passing the traffic to EthIf upper layers.

For a detailed description of the mechanism please refer to the related chapter under Functional Description.

To gateway traffic from one EthIf controller to another the feature must be enabled by creating the `EthIfExtendedTrafficHandling` and its choice container `EthIfTrafficGateway`. This can be 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.



Short Name: EthIfTrafficGatewayRoute

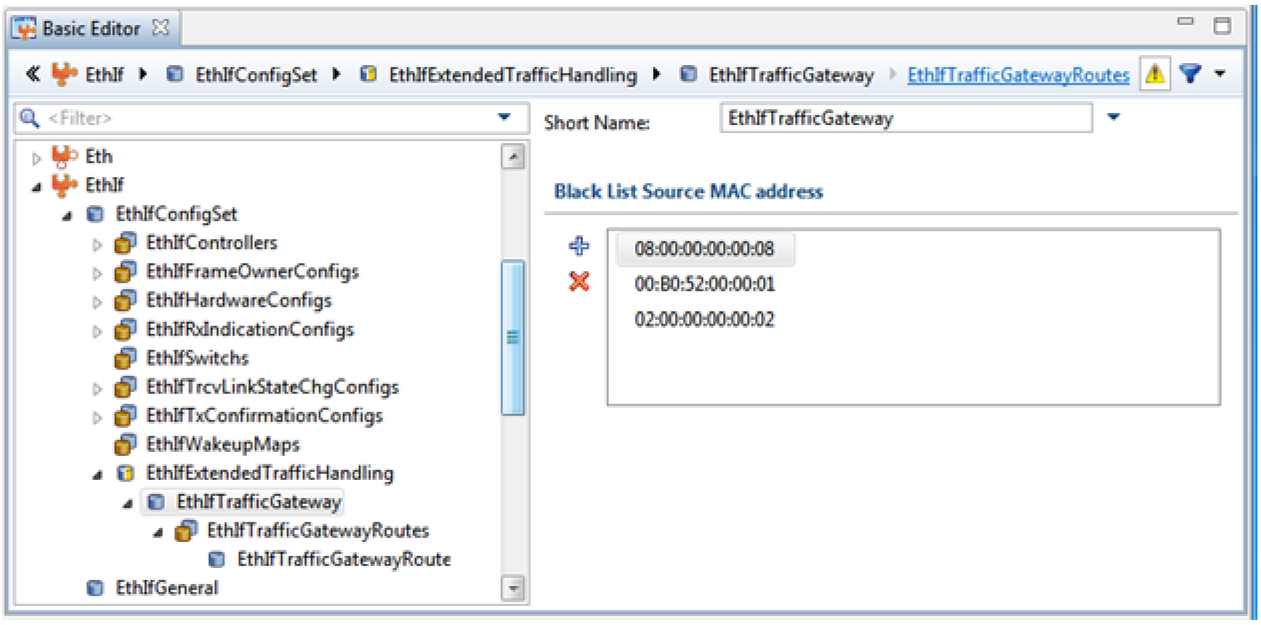
Route Eth If Ctrl Ref

/ActiveEcuC/EthIf/EthIfConfigSet/EthIfController_Ctrl0
/ActiveEcuC/EthIf/EthIfConfigSet/EthIfController_Ctrl1

Figure 5.4 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.


Basic Editor

EthIf ▸ EthIfConfigSet ▸ EthIfExtendedTrafficHandling ▸ EthIfTrafficGateway ▸ EthIfTrafficGatewayRoutes

- Eth
- EthIf
  - EthIfConfigSet
    - EthIfControllers
    - EthIfFrameOwnerConfigs
    - EthIfHardwareConfigs
    - EthIfRxIndicationConfigs
    - EthIfSwitchs
    - EthIfTrcvLinkStateChgConfigs
    - EthIfTxConfirmationConfigs
    - EthIfWakeupMaps
    - EthIfExtendedTrafficHandling
      - EthIfTrafficGateway
        - EthIfTrafficGatewayRoutes
          - EthIfTrafficGatewayRoute
  - EthIfGeneral

Short Name: EthIfTrafficGateway

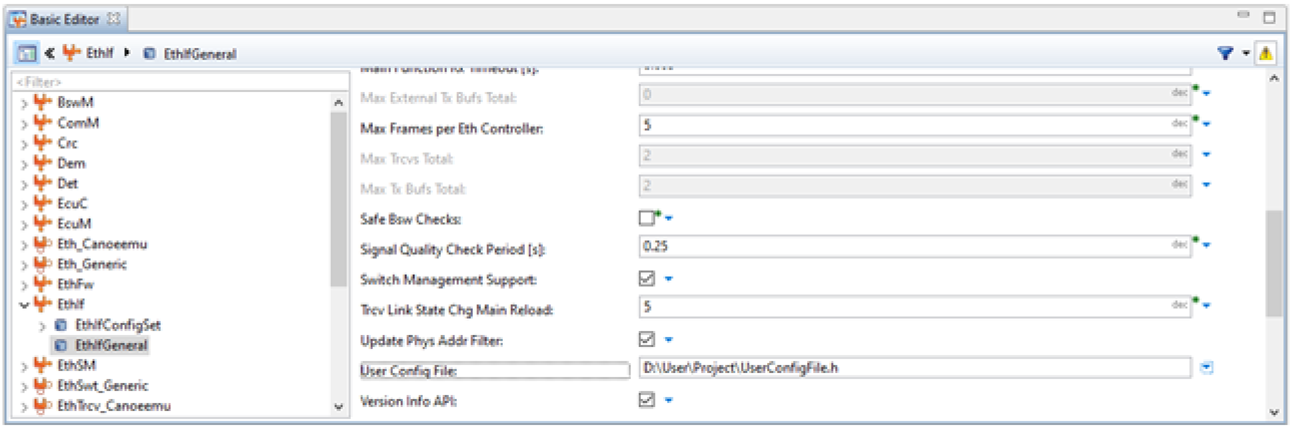
Black List Source MAC address

08:00:00:00:00:08
00:B0:52:00:00:01
02:00:00:00:00:02

Figure 5.5 Traffic Gateway - Black List Configuration

5.2.3 User Configuration File

This module offers the optional parameter `EthIfUserConfigFile` where the user can change or add their implementation at the end of the generated module configuration file **EthIf_Cfg.h**. This is done by including a user config file created by the user. The user must provide the full path of the user config file to `EthIfUserConfigFile` as shown in the following figure.


Basic Editor

EthIf ▸ EthIfGeneral

- BswM
- ComM
- Crc
- Dem
- Det
- EcuC
- EcuM
- Eth_Canoeemu
- Eth_Generic
- EthFw
- EthIf
  - EthIfConfigSet
  - EthIfGeneral
- EthSM
- EthSwt_Generic
- EthTrcv_Canoeemu

Max External Tx Bufs Total:	0
Max Frames per Eth Controller:	5
Max Trcvs Total:	2
Max Tx Bufs Total:	2
Safe Bsw Checks:	☐
Signal Quality Check Period [s]:	0.25
Switch Management Support:	☑
Trcv Link State Chg Main Reload:	5
Update Phys Addr Filter:	☑
User Config File:	D:\User\Project\UserConfigFile.h
Version Info API:	☑

Figure 5.6 User Config File

Caution
Changing or adding implementation to the configuration file generally should be avoided and should be used only in exceptional cases as a workaround for issues in a released product.

Caution
User configuration files can cause the software module to malfunction and must only be used with great care!

6 Cybersecurity

This chapter describes relevant information for a secure integration and configuration, so-called Cybersecurity Manual Instructions (CMI), of this component to fulfil identified Technical Cybersecurity Requirements (TCR). Additionally, functional security dependencies to other components are described.

This component implements the following TCRs:

- TCR-MSRC-GenerateSecurityEvents
- TCR-MSRC-SecCom

6.1 Configuration

6.1.1 CMI_EthIf_ActiveSecurityEventReporting

Technical Cybersecurity Requirements: TCR-MSRC-GenerateSecurityEvents

The user of MICROSAR shall activate the security event reporting via the MICROSAR configuration options.

Rationale: Security event reporting is only performed if activated.

6.2 Runtime Interfaces: BSW

TCR	Depends on Components(s)	Comment
TCR-MSRC-GenerateSecurityEvents	IdsM	TCR-MSRC-ManageSecurityEvents is required from IdsM
TCR-MSRC-SecCom	Mka, EthTrcv and EthSwt	

Table 6.1 Cybersecurity-relevant Dependencies for Runtime Interfaces

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
DaVinci Configurator Classic	Generation tool for MICROSAR Classic modules
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.
MACsec	MACsec is a network security standard that operates at the medium access control layer and defines connection less data confidentiality and integrity.
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.

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
AVB	Audio Video Bridge
BCD	Binary-Coded Decimal
BSW	Basis Software
BSWMD	Basic Software Module Description
CPU	Central Processing Unit
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
Eth	Ethernet Controller Driver
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
MAC	Media Access Control
MACsec	Media Access Control Security
MCU	Microcontroller Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MKA	MACsec key agreement
MSRC	MICROSAR Classic
NV	Non-Volatile
OS	Operating System
PCP	Priority Code Point
PLC	Programmable Logical Controller
PTP	Precision Time Protocol
RAM	Random Access Memory
RTE	Runtime Environment
SPEC	Specification

continues on next page

Table 7.2 – continued from previous page

Abbreviation	Description
SPI	Serial Peripheral Interface
SRP	Stream Reservation Protocol
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
VID	VLAN Identifier
VLAN	Virtual Local Area Network

Table 7.2 [Abbreviations]

8 Contact

Visit our website for more information on

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

www.vector.com