

# MICROSAR Ethernet Driver

## Technical Reference

Core

Version 1.0.0

Authors	David Röder
Status	Released

# 1 Document Information

## 1.1 History

Author	Date	Version	Remarks
David Röder	2016-12-16	1.00.00	Initial Creation

## 1.2 Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_EthernetDriver.pdf	4.1.1
[2]	AUTOSAR	AUTOSAR_SWS_DET.pdf	4.1.1
[3]	AUTOSAR	AUTOSAR_SWS_DEM.pdf	4.1.1
[4]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[5]	Vector	TechnicalReference_Eth_<Platform>.pdf	see delivery
[6]	AUTOSAR	AUTOSAR_SWS_NVRAMManager.pdf	4.1.1
[7]	Vector	TechnicalReference_Asr4Rtm.pdf	2.02.00

## 1.3 Scope of the Document

This technical reference describes the aspects of the Ethernet Driver basis software that are common for all supported platforms, thus it is called 'Core Technical Reference'. It is complemented by the platform dependent lower layer technical reference [5] which is also contained in your delivery.

Please refer to your Release Notes to get a detailed description of the platform (host, compiler and controller) your Vector Ethernet Bundle has been configured for.



### Caution

We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector's release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

**Note**

This document describes the common aspects that apply for all Ethernet Driver modules provided by Vector, which are independent of the used hardware platform. For this reason the corresponding infixes of APIs, macros, parameters etc. are replaced by <Platform> or, if uppercase is used, <PLATFORM>.

The infix applied to a hardware-specific Ethernet Driver module is mentioned in the according hardware-specific supplement to this technical reference.

## Contents

<b>1</b>	<b>Document Information .....</b>	<b>2</b>
1.1	History .....	2
1.2	Reference Documents .....	2
1.3	Scope of the Document.....	2
<b>2</b>	<b>Component History .....</b>	<b>9</b>
<b>3</b>	<b>Introduction.....</b>	<b>10</b>
3.1	Architecture Overview .....	11
<b>4</b>	<b>Functional Description .....</b>	<b>12</b>
4.1	Features .....	12
4.1.1	Deviations .....	12
4.1.2	Additions/ Extensions.....	12
4.1.3	Platform Dependent Features .....	13
4.2	Initialization .....	13
4.2.1	High-Level Initialization .....	13
4.2.2	Low-Level Initialization .....	13
4.3	States .....	13
4.4	Main Functions .....	14
4.5	Error Handling.....	14
4.5.1	Development Error Reporting.....	14
4.5.2	Production Code Error Reporting .....	15
4.6	MAC Address.....	15
4.6.1	NV-RAM.....	15
4.7	Hardware Cancellation.....	16
4.8	Runtime Measurement.....	16
4.9	Ethernet Switch Frame Management.....	17
4.10	Pre-/Post-ControllerInit User-Functions.....	17
4.11	Platform Dependent Features .....	18
4.11.1	Precision Time Protocol Support .....	18
4.11.1.1	Timer .....	18
4.11.1.2	Timer Manipulation .....	18
4.11.1.3	Global Time Retrieving/Setting.....	18
4.11.1.4	Ingress Time Stamping .....	19
4.11.1.5	Egress Time Stamping .....	19
4.11.2	Quality of Service Support.....	21
4.11.2.1	Interrupt Configuration .....	22
4.11.3	FQTSS Support (Traffic Shaping).....	22

4.11.4	Receive Buffer Segmentation.....	22
4.11.5	Checksum Offloading Support.....	23
<b>5</b>	<b>Integration.....</b>	<b>24</b>
5.1	Scope of Delivery.....	24
5.1.1	Static Files .....	24
5.1.2	Dynamic Files .....	24
5.2	Compiler Abstraction and Memory Mapping.....	25
5.3	Critical Sections .....	26
5.4	Core Specific Memory Mapping of Buffers and Descriptors .....	26
5.5	Interrupts .....	26
5.6	Multi Controller Support .....	26
<b>6</b>	<b>API Description.....</b>	<b>27</b>
6.1	Type Definitions .....	27
6.2	Structure Definitions.....	28
6.2.1	Eth_TimeStampType.....	28
6.3	API Table .....	29
6.3.1	Eth_30_<Platform>_InitMemory.....	29
6.3.2	Eth_30_<Platform>_Init .....	29
6.3.3	Eth_30_<Platform>_ControllerInit .....	30
6.3.4	Eth_30_<Platform>_SetControllerMode.....	31
6.3.5	Eth_30_<Platform>_GetControllerMode .....	31
6.3.6	Eth_30_<Platform>_GetPhysAddr .....	32
6.3.7	Eth_30_<Platform>_SetPhysAddr.....	32
6.3.8	Eth_30_<Platform>_WriteMii .....	33
6.3.9	Eth_30_<Platform>_ReadMii .....	33
6.3.10	Eth_30_<Platform>_GetCounterState.....	34
6.3.11	Eth_30_<Platform>_ProvideTxBuffer.....	34
6.3.12	Eth_30_<Platform>_Transmit.....	35
6.3.13	Eth_30_<Platform>_Receive .....	36
6.3.14	Eth_30_<Platform>_VTransmit .....	36
6.3.15	Eth_30_<Platform>_TxConfirmation .....	37
6.3.16	Eth_30_<Platform>_SetBandwidthLimit .....	38
6.3.17	Eth_30_<Platform>_GetBandwidthLimit.....	38
6.3.18	Eth_30_<Platform>_GetVersionInfo.....	39
6.3.19	Eth_30_<Platform>_MainFunction .....	39
6.3.20	Eth_30_<Platform>_GetGlobalTime.....	40
6.3.21	Eth_30_<Platform>_SetGlobalTime .....	40
6.3.22	Eth_30_<Platform>_SetCorrectionTime .....	41
6.3.23	Eth_30_<Platform>_EnableEgressTimestamp .....	41

6.3.24	Eth_30_<Platform>_GetEgressTimestamp .....	42
6.3.25	Eth_30_<Platform>_GetIngressTimestamp.....	42
6.4	Services used by Ethernet Driver .....	43
6.5	Callback Functions.....	43
<b>7</b>	<b>Configuration.....</b>	<b>44</b>
7.1	Configuration Variants.....	44
7.1.1	General Ethernet Driver Configuration .....	44
7.1.2	Platform Dependent Ethernet Driver Configuration .....	48
<b>8</b>	<b>Glossary and Abbreviations .....</b>	<b>49</b>
8.1	Glossary .....	49
8.2	Abbreviations .....	49
<b>9</b>	<b>Contact.....</b>	<b>50</b>

## Illustrations

Figure 3-1	AUTOSAR 4.x Architecture Overview .....	11
Figure 3-2	Interfaces to adjacent modules of the Ethernet Driver .....	11
Figure 4-1	Ingress Time-Stamping Sequence Diagram .....	19
Figure 4-2	Egress Time-Stamping Sequence Diagram .....	20
Figure 4-3	QoS traffic class configuration .....	21

## Tables

Table 2-1	Component history .....	9
Table 4-1	Supported AUTOSAR standard conform features .....	12
Table 4-2	Not supported AUTOSAR standard conform features .....	12
Table 4-3	Features provided beyond the AUTOSAR standard .....	12
Table 4-4	Platform dependent Ethernet features .....	13
Table 4-5	Lower-Level-Initialization .....	13
Table 4-6	Service IDs .....	14
Table 4-7	Errors reported to DET .....	15
Table 4-8	Errors reported to DEM .....	15
Table 4-9	Runtime Measurement points .....	16
Table 4-10	Traffic class to ring priority mapping .....	21
Table 5-1	Static files .....	24
Table 5-2	Generated files .....	24
Table 5-3	Compiler abstraction and memory mapping .....	25
Table 6-1	Type definitions .....	28
Table 6-2	Eth_TimeStampType Structure .....	29
Table 6-3	Eth_30_<Platform>_InitMemory .....	29
Table 6-4	Eth_30_<Platform>_Init .....	30
Table 6-5	Eth_30_<Platform>_ControllerInit .....	30
Table 6-6	Eth_30_<Platform>_SetControllerMode .....	31
Table 6-7	Eth_30_<Platform>_GetControllerMode .....	31
Table 6-8	Eth_30_<Platform>_GetPhysAddr .....	32
Table 6-9	Eth_30_<Platform>_SetPhysAddr .....	32
Table 6-10	Eth_30_<Platform>_WriteMii .....	33
Table 6-11	Eth_30_<Platform>_ReadMii .....	34
Table 6-12	Eth_30_<Platform>_GetCounterState .....	34
Table 6-13	Eth_30_<Platform>_ProvideTxBuffer .....	35
Table 6-14	Eth_30_<Platform>_Transmit .....	36
Table 6-15	Eth_30_<Platform>_Receive .....	36
Table 6-16	Eth_30_<Platform>_VTransmit .....	37
Table 6-17	Eth_30_<Platform>_TxConfirmation .....	37
Table 6-18	Eth_30_<Platform>_SetBandwidthLimit .....	38
Table 6-19	Eth_30_<Platform>_GetBandwidthLimit .....	39
Table 6-20	Eth_30_<Platform>_GetVersionInfo .....	39
Table 6-21	Eth_30_<Platform>_MainFunction .....	40
Table 6-22	Eth_30_<Platform>_GetGlobalTime .....	40
Table 6-23	Eth_30_<Platform>_SetGlobalTime .....	41
Table 6-24	Eth_30_<Platform>_SetCorrectionTime .....	41
Table 6-25	Eth_30_<Platform>_EnableEgressTimestamp .....	42
Table 6-26	Eth_30_<Platform>_GetEgressTimestamp .....	42
Table 6-27	Eth_30_<Platform>_GetIngressTimestamp .....	43
Table 6-28	Services used by the Ethernet Driver .....	43
Table 7-1	General Ethernet Configuration .....	48

Table 8-1	Glossary .....	49
Table 8-2	Abbreviations.....	49



## 2 Component History

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

Component Version	New Features
01.00.00	Initial Component Release

Table 2-1 Component history

### 3 Introduction

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

<b>Supported AUTOSAR Release*:</b>	4.1.1	
<b>Supported Configuration Variants:</b>	pre-compile	
<b>Vendor ID:</b>	ETH_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
<b>Module ID:</b>	ETH_MODULE_ID	88 decimal (according to ref. [4])

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

The Ethernet Driver provides hardware independent access to control connected or integrated Controllers in a generic way. It offers the functionality to control the mode of operation of the Controllers as well as to determine their current state, e.g. if events like bus errors occur.

The controller itself is a hardware device, which receives and transmits Ethernet frames over the Media Independent Interface (MII).

### 3.1 Architecture Overview

Figure 3-1 shows where the Ethernet Driver is located in the AUTOSAR architecture.

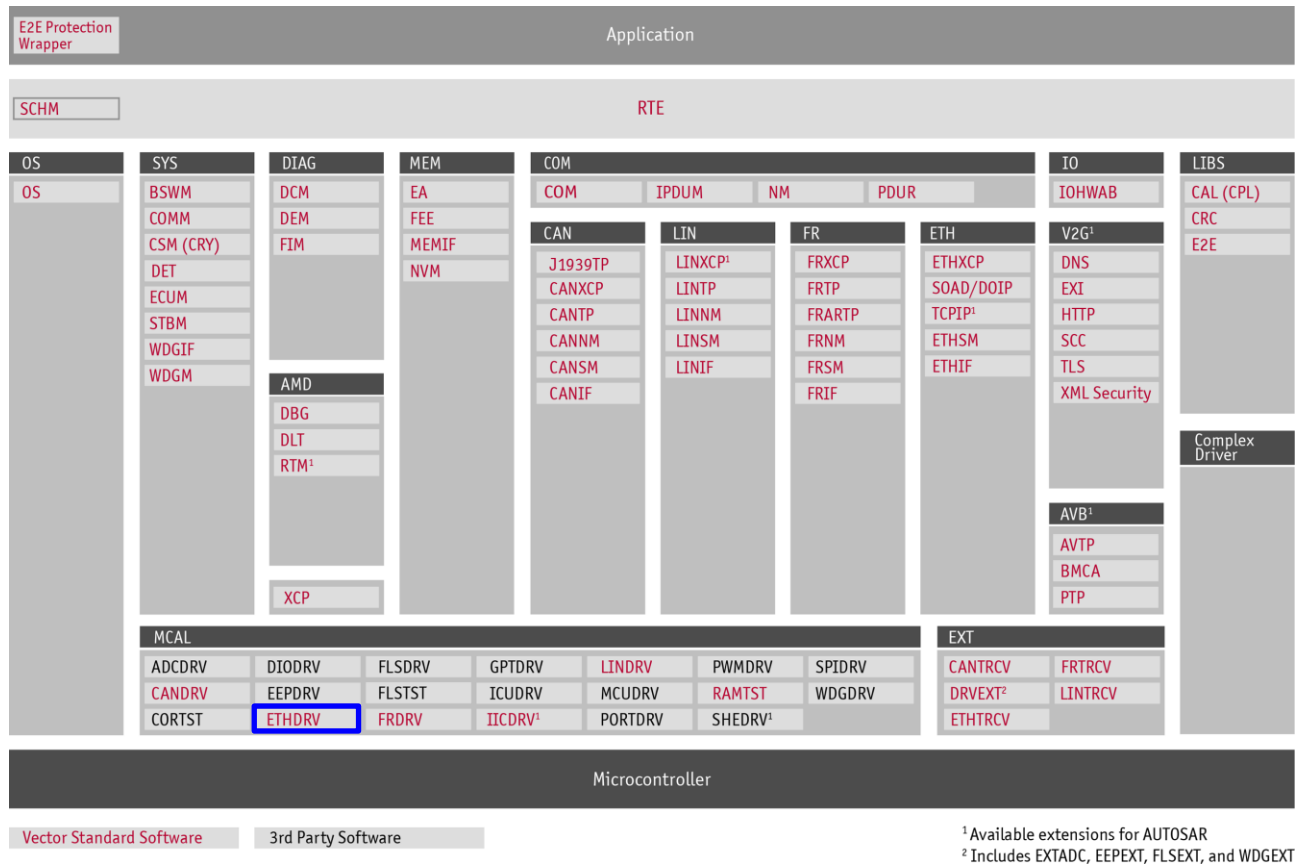


Figure 3-1 AUTOSAR 4.x Architecture Overview

Figure 3-2 shows the interfaces to adjacent modules of the Ethernet Driver.

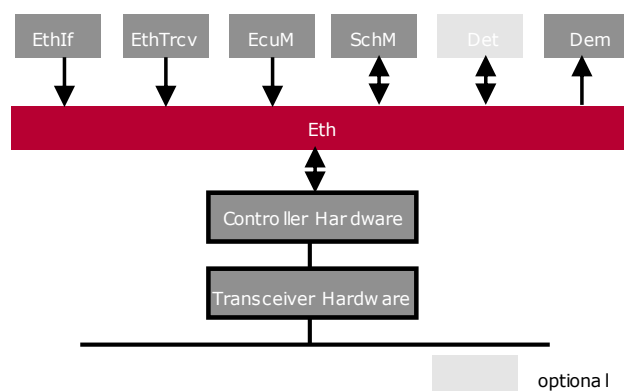


Figure 3-2 Interfaces to adjacent modules of the Ethernet Driver

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

## 4 Functional Description

### 4.1 Features

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

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

- > Table 4-1 Supported AUTOSAR standard conform features
- > Table 4-2 Not supported AUTOSAR standard conform features

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

- > Table 4-3 Features provided beyond the AUTOSAR standard
- > Table 4-4 Platform dependent Ethernet features

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Retrieve version info
Report development errors to DET
Report production errors to DEM

Table 4-1 Supported AUTOSAR standard conform features

#### 4.1.1 Deviations

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

Not Supported AUTOSAR Standard Conform Features
none

Table 4-2 Not supported AUTOSAR standard conform features

#### 4.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Extended Polling Mode (Receive Polling)
NvM MAC Address Storage
Extended Buffer Configuration
Runtime Measurement
Flexible Interrupt configuration
Pre-/Post-ControllerInit User-Callouts

Table 4-3 Features provided beyond the AUTOSAR standard

### 4.1.3 Platform Dependent Features

Table 4-4 lists all hardware dependent features provided by Vector. Please refer to the lower layer supplement to this technical reference [5] for details, if any features of this list are available for your delivery.

Platform dependent Ethernet features
Receive Buffer Segmentation
Precision Time Protocol Support
Quality of Service Support
FQTSS Support (Traffic Shaping)
Checksum Offloading Support

Table 4-4 Platform dependent Ethernet features

## 4.2 Initialization

### 4.2.1 High-Level Initialization

The Ethernet Driver is initialized by calling the `Eth_30_<Platform>_InitMemory` and `Eth_30_<Platform>_Init` services with the configuration as parameter in the named order.

The controller itself is initialized by calling the `Eth_30_<Platform>_ControllerInit` service with the corresponding index for each controller. Usually, this is done by the `EthIf` component.

### 4.2.2 Low-Level Initialization

The Ethernet Driver relies on the proper configuration of the MCU, which must be done by the MCAL modules. Table 4-5 contains information about the modules involved and what preconditions must be provided by them for proper functionality of the Ethernet Controller.

AUTOSAR MCAL module	Preconditions
Port	Initialization of the MCU pins for the MII interface to the Ethernet Transceiver/Ethernet Switch Port.
Mcu	Initialization of the clocks provided to the Ethernet Controller.

Table 4-5 Lower-Level-Initialization

## 4.3 States

The controller should be set to a defined state during initialization by upper layer software component (Ethernet Interface). Otherwise the initial state is undefined.

## 4.4 Main Functions

If it is enabled in the configuration and implemented in the hardware depended lower layer part of the Ethernet Driver, the module provides a Main Function. Please Refer to [1] and [5] for further information.

## 4.5 Error Handling

### 4.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `ETH_30_<PLATFORM>_DEV_ERROR_DETECT==STD_ON`).

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

The reported Ethernet Driver ID is 88.

The reported service IDs identify the services which are described in [2]. Table 4-6 presents the service IDs and the related services:

Service ID	Service
0x01	ETH_API_ID_INIT
0x02	ETH_API_ID_CONTROLLER_INIT
0x03	ETH_API_ID_SET_CONTROLLER_MODE
0x04	ETH_API_ID_GET_CONTROLLER_MODE
0x05	ETH_API_ID_WRITE_MII
0x06	ETH_API_ID_READ_MII
0x07	ETH_API_ID_GET_COUNTER_STATE
0x08	ETH_API_ID_GET_PHYS_ADDR
0x09	ETH_API_ID_SET_PHYS_ADDR
0x0A	ETH_API_ID_UPDATE_PHYS_ADDR_FILTER
0x0B	ETH_API_ID_PROVIDE_TX_BUFFER
0x0C	ETH_API_ID_TRANSMIT
0x0D	ETH_API_ID_RECEIVE
0x0E	ETH_API_ID_TX_CONFIRMATION
0x0F	ETH_API_ID_GET_VERSION_INFO

Table 4-6 Service IDs

The errors reported to DET are described in Table 4-7:

Error Code	Description
0x00	ETH_E_NO_ERROR No error occurred
0x01	ETH_E_INV_CTRL_IDX The Ethernet Driver was called with an invalid Controller

Error Code		Description
		Index
0x02	ETH_E_NOT_INITIALIZED	An Ethernet Driver service was called without initializing the module first by calling <code>Eth_Init</code>
0x03	ETH_E_INV_POINTER	An Ethernet Driver service was called with a zero pointer as parameter
0x04	ETH_E_INV_PARAM	An Ethernet Driver service was called with an invalid parameter
0x05	ETH_E_INV_CONFIG	The Ethernet Driver configuration is invalid
0x06	ETH_E_INV_MODE	The Ethernet Driver mode is invalid
0x07	ETH_E_FRAMES_LOST	Ethernet Frame lost

Table 4-7 Errors reported to DET

## 4.5.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3].

The errors reported to DEM are described in Table 4-8:

Error Code	Description
ETH_E_ACCESS	Accessing the controller failed

Table 4-8 Errors reported to DEM

## 4.6 MAC Address

The MAC address of a controller may be configured to a default value via the MAC address configuration parameter within the configuration tool. Additionally, the user may call the API `Eth_30_<Platform>_SetPhysAddr` to change the MAC address at runtime.

In case the user does not provide a default address within the configuration tool, the address is undefined after ECU startup. Therefore it is essential that the user sets the MAC at runtime before any Ethernet communication starts!

The default behavior of `Eth_30_<Platform>_SetPhysAddr` is non-persistent. The user must newly provide the MAC address after an ECU restart (and in case no default is configured within the configuration tool). An extended, persistent behavior can be enabled via the Vector feature “Enable MAC address write access” discussed in the next chapter.

### 4.6.1 NV-RAM

The feature “Enable MAC address write access” enables NvM support for the API `Eth_30_<Platform>_SetPhysAddr`. In this case the MAC address gets written into a NV-RAM block and is loaded at controller initialization. This ensures that the MAC address is persistent even after a system restart.

The NV-RAM block for a MAC must have a length of 6 bytes. The initial value is configured via the MAC address configuration parameter within the configuration tool. The NV-RAM

block must be managed by the AUTOSAR NV-Manager (NvM) and is addressed by a NvM block descriptor (refer to [6]).

The NV-RAM blocks must be processed during the `NvM_ReadAll()` and `NvM_WriteAll()` function calls.

The symbols for the ROM and RAM mirrors are listed below

- > ROM default block:  
`Eth_30_<Platform>_VPhysSrcAddrRomDefault_<CtrlIdx>`
- > RAM mirror: `Eth_30_<Platform>_VPhysSrcAddr_<CtrlIdx>`

where `<CtrlIdx>` is the index of the controller.

#### 4.7 Hardware Cancellation

The hardware cancellation feature can be applied for different hardware operations by setting the corresponding cycle number. While performing these operations a counter is incremented in a loop in every cycle until either the operation is finished or the configured maximum value of the cycle number has been reached. In this case the operation has failed. The configurable maximum cycle counter values are:

- > Controller Reset Loop Cycles: Specifies the number of loop cycles after which the controller reset timeout occurs.
- > Controller MII Loop Cycles: Specifies the number of loop cycles after which the timeout for MII accesses occurs.
- > Controller Loop Cycles: Specifies the number of loop cycles after which other hardware dependent loop timeouts occur.

#### 4.8 Runtime Measurement

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

This description only shows the Ethernet drivers specific aspect of the runtime measurement. For a more detailed description of the runtime measurement feature and the Rtm module providing the service see [7].

Dependent on the Ethernet controller core integrated on the derivative used and the features enabled, more or less measurement points are provided.

Measurement Point	Description
<code>Eth_30_&lt;Platform&gt;_ControllerInit</code>	Measures the runtime of the API <code>Eth_30_&lt;Platform&gt;_ControllerInit()</code> . The API is used for controller initialization during runtime.

Table 4-9 Runtime Measurement points



**Note**

The runtime measurement is enabled by setting `Runtime Measurement Support` in the general settings of the Ethernet driver.

## 4.9 Ethernet Switch Frame Management

Some Ethernet Switches provide a functionality called frame management. This feature makes it possible to transfer Meta data related to an Ethernet frame over the MII interface. This Meta data is either embedded into the Ethernet Frame itself or transferred as a consecutive Ethernet frame. To allow the Ethernet Driver to process the Ethernet frames without the knowledge which kind of mechanism is used, an interaction between Ethernet Driver and Ethernet Switch Driver is needed.

For a more detailed description of the interface please refer to the Ethernet Switch Drivers technical reference.

**Note**

The feature is implicitly activated if a Ethernet Switch Driver is contained in the configuration and uses the Ethernet Controller as frame management interface.

## 4.10 Pre-/Post-ControllerInit User-Functions

The driver allows to inject integration code in the processing of the `Eth_30_<Platform>_ControllerInit()` API.

This integration code is either executed before the actual code of the `Eth_30_<Platform>_ControllerInit()` (Pre-ControllerInit User-Function), after the actual code (Post-ControllerInit User-Function) or both.

The functions can be used to integrate derivative specific code that must be executed for a proper operation of the Ethernet Controller. Such code could be setting registers in port or clock modules, which are needed to configure the xMII interface between the Ethernet Controller and the Ethernet PHY properly.

The User-Functions will be generated into the `Eth_30_<Platform>_Lcfg.c` file and contain a so called User-Block-Comment where integration code can be provided, which is persisted and not overridden if the `Eth_30_<Platform>_Lcfg.c` file is generated again.

The function names are `Eth_30_<Platform>_PreControllerInitCallout()` and `Eth_30_<Platform>_PostControllerInitCallout()`.

## 4.11 Platform Dependent Features

The following sections introduce Ethernet features which are contained in the software provided by Vector but are not supported on every hardware platform. Please refer to the lower layer part of the technical reference [5] to find out if a specific feature is supported in your delivery.

### 4.11.1 Precision Time Protocol Support

Precision Time Protocol (PTP) allows retrieving timestamps of Ethernet frames on transmission and reception.

**Note**

The feature is enabled by setting 'Enable PTP' in the Ethernet Controller configuration in DaVinci Configurator.

#### 4.11.1.1 Timer

For the support of PTP the Ethernet Controller core implements a timer which maintains the local time. This time base is used to timestamp the frames. Additionally to the time stamping feature the timer can be manipulated by multiple APIs, which will be described in the following sections.

**Note**

For proper operation of the timer an external clock must be provided. Please refer to the peripheral clock configuration chapter of the Technical Reference provided for the microcontroller derivative used.

The clock frequency provided additionally must be defined in the controller configuration in DaVinci Configurator by setting the parameter 'PTP Reference Clock Frequency'.

#### 4.11.1.2 Timer Manipulation

The local time can be manipulated to be approximated to a global time negotiated between the nodes in the Ethernet network by a time protocol like PTP.

The API `Eth_30_<Platform>_SetCorrectionTime()` allows this manipulation. It enables the upper layer to correct the time by an offset or slow/speed the clock.

#### 4.11.1.3 Global Time Retrieving/Setting

An upper layer is able to retrieve and set the global time. Therefor the APIs

`Eth_30_<Platform>_GetGlobalTime()` and

`Eth_30_<Platform>_SetGlobalTime()` are provided.



## Note

These APIs can be called without the restriction to the Rx Indication or TX Confirmation context.

#### 4.11.1.4 Ingress Time Stamping

All incoming Ethernet Frames are time stamped by the ENET Ethernet Controller Core. The upper layer is able to retrieve this timestamp by calling the API

Eth\_30\_&lt;Platform&gt;\_GetIngressTimestamp().



## Caution

The API `Eth_30_<Platform>_GetIngressTimestamp()` must be called in the Rx Indication context to retrieve a proper timestamp for the received Ethernet frame. Otherwise the API will return with the negative return value `E_NOT_OK` and the timestamp is invalid.

Figure 4-1 illustrates the sequence with a GPtp upper layer:

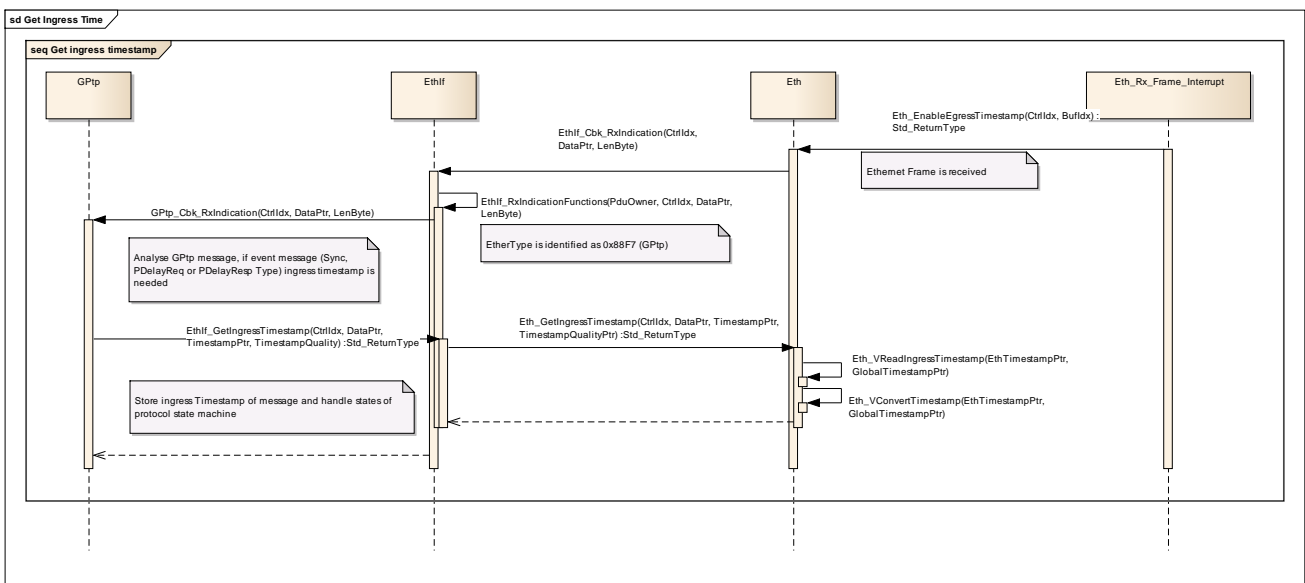


Figure 4-1 Ingress Time-Stamping Sequence Diagram

#### 4.11.1.5 Egress Time Stamping

The time stamping of outgoing Ethernet frames must be triggered by the upper layer itself.

To achieve the time stamping the upper layer must obtain a transmit buffer by calling `Eth_30_<Platform>_ProvideTxBuffer()`. Afterwards the buffer provided is locked and time stamping for this buffer can be enabled by calling `Eth_30_<Platform>_EnableEgressTimestamp()`. The transmission now can be trigger with the API `Eth_30_<Platform>_Transmit()`.

The time stamp itself is retrieved on transmission indication by calling `Eth_30_<Platform>_GetEgressTimestamp()`.



### Caution

The API `Eth_30_<Platform>_GetEgressTimestamp()` must be called in the TX Confirmation context to retrieve a proper timestamp for the transmitted Ethernet Frame. Otherwise the API will return with the negative return value `E_NOT_OK` and the timestamp is invalid.



### Caution

On transmit of an Ethernet frame to be time-stamped a TX Confirmation must be requested by the upper layer.

Figure 4-2 illustrates the sequence with a GPtp upper layer:

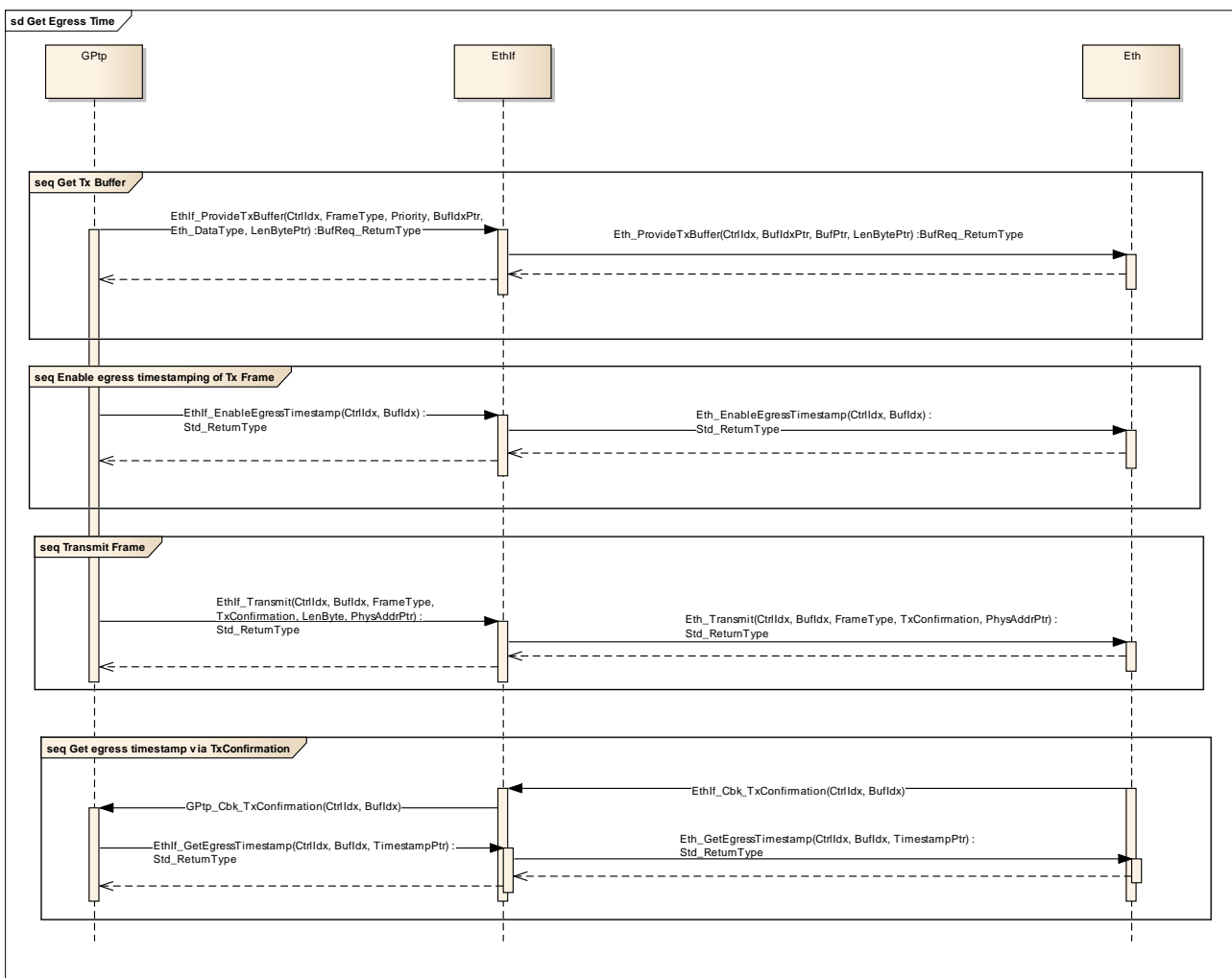


Figure 4-2 Egress Time-Stamping Sequence Diagram

### 4.11.2 Quality of Service Support

Quality of Service (QoS) allows prioritizing the Ethernet traffic dependent on the priority of a VLAN tagged frame.



**Note**

Quality of Service can only be applied on VLAN tagged traffic because it relies on the priority encoded in the VLAN TCI (Tag Control Information).

The number of supported traffic classes can be found in the hardware dependent part of the technical reference for your delivery [5]. The following configuration example provides three traffic classes: The best effort class-, class-1- and class-2-traffic, where best effort has the lowest priority and class 2 has the highest priority.

The different classes can be identified in the configuration tool by the parameter `Priority` (see Figure 4-3) and Table 4-10.

Traffic Class	Priority
Best effort	0
Class 1	1
Class 2	2

Table 4-10 Traffic class to ring priority mapping

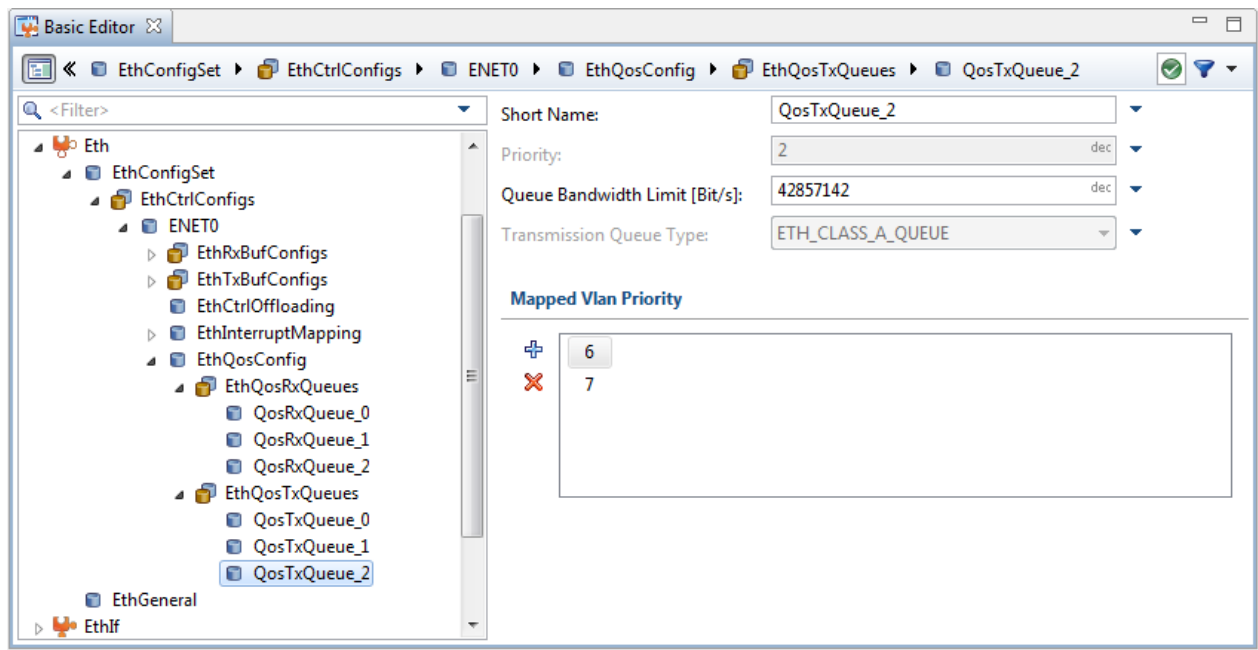


Figure 4-3 QoS traffic class configuration

Like mentioned before, the traffic classification is done with the help of the priority encoded by the VLAN tag. To achieve this, the VLAN priority is used during configuration time to assign traffic to a specific traffic class. This can be done for the Transmission- and for the Reception-Path as well. In the screenshot, illustrated by Figure 4-3, for example, traffic class 2 on Transmission-Path has assigned all frames sent with the VLAN priority 6 and 7.

#### 4.11.2.1 Interrupt Configuration

Each traffic class of the QoS feature has its own receive and transmit interrupts assigned. If interrupts are enabled these handlers must be configured according to the section 7.1.2 Platform Dependent Ethernet Driver Configuration.

**Note**

The interrupt handler for the best effort traffic class is equal to the common interrupt handler used in non QoS operation.

#### 4.11.3 FQTSS Support (Traffic Shaping)

The FQTSS (Forwarding and Queuing Enhancements for Time-Sensitive Streams) support allows applying traffic shaping for Ethernet traffic on the Transmission-Path. The mechanism is based on the QoS traffic classes described in the section 4.11.2.

The traffic classes are assigned to a traffic shaper, which defines an idle slope to shape the Ethernet traffic. The driver allows configuring the shapers in a static way during configuration time by providing the desired bandwidth in [bit/s].

Additionally the current bandwidth limit can be manipulated or retrieved during runtime by using the APIs `Eth_30_<Platform>_SetBandwidthLimit()` and `Eth_30_<Platform>_GetBandwidthLimit()`.

#### 4.11.4 Receive Buffer Segmentation

Some Ethernet MACs provide the ability to receive one frame using multiple descriptors connected to small buffer. When this feature is available and enabled the configuration allows configuring segment sizes different to the maximum frame size, which can be received. The MAC uses as much descriptors per frame as necessary to receive the whole frame.

The implementation of this mechanism is transparent for the upper layers.

Enabling Receive Buffer Segmentation can be activated for a more efficient usage of RAM, which is needed in systems which need to receive large frames only in rare cases, but mainly receive small frames. Instead of having a pool of large buffers available which cause a lot of wasted memory for small frames, the usage of memory scales with the amount of used segments per frame.

Considering an example of three frames with a size of 128 Bytes and one frame with a size of 1500 Bytes, without Receive Buffer Segmentation  $4 \times 1500 = 6000$  Bytes of receive buffers are needed. Using Receive Buffer Segmentation only  $15 \times 128 = 1920$  Bytes of receive buffers are needed.

#### 4.11.5 Checksum Offloading Support

The Checksum Offloading support allows delegating the calculation of the following checksums to the Ethernet controller:

- ▶ ICMPv4
- ▶ IPv4
- ▶ UDP encapsulated in IPv4
- ▶ TCP encapsulated in IPv4
- ▶ UDP encapsulated in IPv6
- ▶ TCP encapsulated in IPv6

Checksum offloading can be enabled during configuration time. The lower layer technical reference [5] provides more information on the respective configuration steps, if the feature is available for the delivered platform.

## 5 Integration

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

### 5.1 Scope of Delivery

The delivery of the Ethernet Driver always contains the files which are described in the chapters 5.1.1 and 5.1.2. The lower layer technical reference [5] addresses additional files that might be included in the delivery and are not mentioned here.

#### 5.1.1 Static Files

File Name	Description
Eth_30_<Platform>.c	Core Implementation
Eth_30_<Platform>.h	Core API declaration
Eth_30_<Platform>_Priv.h	Component local macro and variable declaration
Eth_30_<Platform>_LL.c	Lower layer Implementation
Eth_30_<Platform>_LL.h	Lower layer API declaration

Table 5-1 Static files



#### Do not edit manually

Static source code files must not be edited manually!

#### 5.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator PRO.

File Name	Description
Eth_30_<Platform>_Cfg.h	Pre-compile time parameter configuration
Eth_30_<Platform>_Lcfg.c	Link-time parameter configuration
Eth_30_<Platform>_Lcfg.h	Link-time parameter configuration declaration
Eth_30_<Platform>_DataStructures.h	Data structures declaration

Table 5-2 Generated files



**Do not edit manually**

Generated source code files must not be edited manually but can be adjusted according to the configuration elements in the DaVinci Configurator Pro tool!

## 5.2 Compiler Abstraction and Memory Mapping

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

Table 5-3 contains the memory section names and the compiler abstraction definitions which are defined for the Ethernet Driver and illustrates their assignment among each other.

Memory Mapping Sections	Compiler Abstraction Definitions		
	ETH_30_<PLATFORM>_START_SEC_CONST	ETH_30_<PLATFORM>_START_SEC_VAR	ETH_30_<PLATFORM>_START_SEC_CODE
ETH_30_<PLATFORM>_START_SEC_CONST_UNSPECIFIED	■		
ETH_30_<PLATFORM>_START_SEC_CONST_32BIT	■		
ETH_30_<PLATFORM>_START_SEC_CONST_16BIT	■		
ETH_30_<PLATFORM>_START_SEC_CONST_8BIT	■		
ETH_30_<PLATFORM>_START_SEC_VAR_NOINIT_UNSPECIFIED		■	
ETH_30_<PLATFORM>_START_SEC_VAR_NOINIT_32BIT		■	
ETH_30_<PLATFORM>_START_SEC_VAR_NOINIT_16BIT		■	
ETH_30_<PLATFORM>_START_SEC_VAR_NOINIT_8BIT		■	
ETH_30_<PLATFORM>_START_SEC_VAR_NOINIT_BUFFER		■	
ETH_30_<PLATFORM>_START_SEC_VAR_NOINIT_DESCRIPTOR		■	
ETH_30_<PLATFORM>_START_SEC_VAR_NOINIT_BOOLEAN		■	
ETH_30_<PLATFORM>_START_SEC_CODE			■

Table 5-3 Compiler abstraction and memory mapping

### 5.3 Critical Sections

To ensure data consistency and a correct function of the Ethernet Driver the exclusive areas `ETH_30_<PLATFORM>_BEGIN_CRITICAL_SECTION_0` and

`ETH_30_<PLATFORM>_BEGIN_CRITICAL_SECTION_1` have to be provided.

Considering the timing behavior of your system (e.g. depending on the CPU load of your system, priorities and interruptibility of interrupts and OS tasks and their jitter and delay times) the integrator has to choose and configure a critical section solution in such way that it is ensured that the API functions do not interrupt each other.

It is recommended to use OS Resources for both exclusive areas instead of `SuspendAllInterrupts()` and `ResumeAllInterrupts()`. Suspending all interrupts may lead to a bad system performance.

### 5.4 Core Specific Memory Mapping of Buffers and Descriptors

The Ethernet buffers and descriptors must be mapped with the AUTOSAR memory mapping feature. For this purpose the two memory sections

`ETH_30_<PLATFORM>_START_SEC_VAR_NOINIT_BUFFER` and

`ETH_30_<PLATFORM>_START_SEC_VAR_NOINIT_DESCRIPTOR` are available.

These sections must be aligned according to a specific restriction dependent on the Ethernet Controller Core contained on the used derivative. Refer to [5] for further information.

### 5.5 Interrupts

The Ethernet driver supports the configuration of an interrupt mapping.

In case the interrupts are enabled and an OS is running, there are two different interrupt categories, provided by the OS. The Ethernet driver supports both Category 1 and Category 2 interrupts. Additionally the interrupt operation without an OS is possible.

For usage of the interrupt service routines and their configuration please refer to the lower layer technical reference [5].

Due to support for different derivatives a generic interrupt service routine configuration is provided.

### 5.6 Multi Controller Support

The multi controller support allows using any number of Ethernet controllers integrated on the hardware platform used. Please refer to the lower layer technical reference [5] for possible hardware specific constraints.

For more information or for information about derivatives included in the technical reference [5], please refer to the corresponding reference manual of the derivative used.

## 6 API Description

For an interfaces overview please see Figure 3-1.

### 6.1 Type Definitions

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

Type Name	C-Type	Description	Value Range
Eth_ConfigType	void	Controller configuration	NULL_PTR Ctrl uses Link-time configuration
Eth_ReturnType	uint8	Return type of Eth_30_<Platform> _ReadMii and Eth_30_<Platform> _WriteMii APIs	ETH_OK Success ETH_E_NOT_OK General failure ETH_E_NO_ACCESS Ethernet hardware access failure
Eth_ModeType	uint8	Defines all possible controller modes	ETH_MODE_DOWN Controller inactive ETH_MODE_ACTIVE Normal operation mode ETH_TX_STATE_NOT_TRANSMITTED Frame not yet transmitted
Eth_FrameType	uint16	Ethernet Frame Type	0x0000 - 0xFFFF Any Ethernet frame type
Eth_DataType	uint32	Defines the Ethernet data type	0x00000000 - 0xFFFFFFFF User data
Eth_RxStatusType	uint8	Out parameter in Eth_30_<Platform> _Receive to indicate that a frame has been received and if so, whether more frames are available or frames got lost	ETH_RECEIVED Frame received, no further frames available ETH_NOT_RECEIVED Frame received, no further frames available ETH_RECEIVED_MORE_DATA_AVAILABLE Frame received, more frames available ETH_RECEIVED_FRAMES_LOST Frame received, some

Type Name	C-Type	Description	Value Range
			frames lost
Eth_FilterActionType	void	Describes the action to be taken for the MAC address given to API Eth_30_<Platform>_UpdatePhysAddrFilter	ETH_ADD_TO_FILTER Add MAC to the filter, meaning allow reception  ETH_REMOVE_FROM_FILTER Remove MAC from the filter, meaning reception is blocked in the lower layer
Eth_StateType	uint8	Defines all possible controller Driver states	ETH_STATE_UNINIT Ethernet Controller Driver not initialized  ETH_STATE_INIT Ethernet Controller Driver initialized  ETH_STATE_ACTIVE Ethernet Controller Driver enabled  ETH_STATE_DOWN Ethernet Controller Driver disabled
Eth_TimestampQualityType	uint8	Defines the quality of a PTP timestamp	ETH_TIMESTAMP_VALID Returned timestamp is valid  ETH_TIMESTAMP_INVALID Returned timestamp is invalid, e.g. because of an hardware error  ETH_TIMESTAMP_UNCERTAIN Timestamp validity is uncertain. This value is not used by the Tricore Ethernet Driver!
Eth_RateRatioType	float32	Defines a ratio rate that is used to correct the time drift for the internal PTP clock	According to IEEE 754 single-precision binary floating-point format.

Table 6-1 Type definitions

## 6.2 Structure Definitions

### 6.2.1 Eth\_TimeStampType

Defines a timestamp, according to IEEE 1588-2008 (PTP version 2)

Struct Element Name	C-Type	Description	Value Range
nanoseconds	uint32	Nanoseconds part of time	[0 ; 999.999.999d ] [0x00 ; 0x3B9A_C9ff ]
seconds	uint32	32 LSB of the 48 Bit seconds part of time	[0 ; 2 <sup>32</sup> ]
secondsHi	uint16	16 MSB of the 48 Bit seconds part of time	[0 ; 2 <sup>16</sup> ]

Table 6-2 Eth\_TimeStampType Structure

## 6.3 API Table

### 6.3.1 Eth\_30\_<Platform>\_InitMemory


Prototype	
void <b>Eth_30_&lt;Platform&gt;_InitMemory</b> (void)	
Parameter	
void	
Return code	
void	void
Functional Description	
This function initializes global variables. It has to be called before any other calls to the Eth API.	
Particularities and Limitations	
NOT Re-entrant, synchronous	
	<b>Caution</b> Has to be called before usage of the module
Call context	
> Initialization	

Table 6-3 Eth\_30\_<Platform>\_InitMemory

### 6.3.2 Eth\_30\_<Platform>\_Init

Prototype	
void <b>Eth_30_&lt;Platform&gt;_Init</b> (const Eth_ConfigType *CfgPtr)	
Parameter	
CfgPtr [in]	Pointer to post-build configuration or null pointer


Return code	
void	void
Functional Description	
This API call stores the start address of the post build time configuration of the Ethernet Controller driver, resets all transceivers controlled by the driver and may be used to initialize the data structures.	
Particularities and Limitations	
Re-entrant, synchronous	
	<b>Caution</b> Has to be called before usage of the module
Call context	
> Initialization	

Table 6-4 Eth\_30\_<Platform>\_Init

### 6.3.3 Eth\_30\_<Platform>\_ControllerInit


Prototype	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_ControllerInit</b> (uint8 CtrlIdx, uint8 CfgIdx)	
Parameter	
CtrlIdx [in]	Controller index
CfgIdx [in]	Configuration index
Return code	
Std_ReturnType	> E_OK : Controller configured > E_NOT_OK : Controller configuration failed
Functional Description	
This API call of a specific Eth driver initializes the Ethernet Driver with index CtrlIdx. The following actions are performed: - Configuration of low level parameters - Initialization of descriptors.	
Particularities and Limitations	
- Re-entrant, synchronous	
	<b>Caution</b> Has to be called before usage of the module
Call context	
> Initialization	

Table 6-5 Eth\_30\_<Platform>\_ControllerInit

### 6.3.4 Eth\_30\_<Platform>\_SetControllerMode

Prototype	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_SetControllerMode</b> (uint8 CtrlIdx, Eth_ModeType CtrlMode)	
Parameter	
CtrlIdx [in]	Controller index
CtrlMode [in]	Operation mode
Return code	
Std_ReturnType	<ul style="list-style-type: none"> <li>&gt; E_OK : Controller mode changed</li> <li>&gt; E_NOT_OK : Controller mode change failed</li> </ul>
Functional Description	
Set controller mode.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>- Re-entrant, synchronous</li> <li>- If API optimization is enabled, parameter CtrlIdx is void</li> </ul>	
Call context	
<ul style="list-style-type: none"> <li>&gt; Interrupt or task level</li> </ul>	

Table 6-6 Eth\_30\_&lt;Platform&gt;\_SetControllerMode

### 6.3.5 Eth\_30\_<Platform>\_GetControllerMode

Prototype	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_GetControllerMode</b> (uint8 CtrlIdx, Eth_ModeType *CtrlModePtr)	
Parameter	
CtrlIdx [in]	Controller index
CtrlModePtr [out]	Operation mode
Return code	
Std_ReturnType	<ul style="list-style-type: none"> <li>&gt; E_OK : Controller mode evaluated</li> <li>&gt; E_NOT_OK : Controller mode evaluation failed</li> </ul>
Functional Description	
Get controller mode.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>- Re-entrant, synchronous</li> </ul>	
Call context	
<ul style="list-style-type: none"> <li>&gt; Interrupt or task level</li> </ul>	

Table 6-7 Eth\_30\_&lt;Platform&gt;\_GetControllerMode

### 6.3.6 Eth\_30\_<Platform>\_GetPhysAddr

Prototype	
void <b>Eth_30_&lt;Platform&gt;_GetPhysAddr</b> (uint8 CtrlIdx, uint8 *PhysAddrPtr)	
Parameter	
CtrlIdx [in]	Controller index
PhysAddrPtr [out]	Physical address as byte array
Return code	
void	void
Functional Description	
Get physical address (MAC address).	
Particularities and Limitations	
- Re-entrant, synchronous	
Call context	
> Interrupt or task level	

Table 6-8 Eth\_30\_<Platform>\_GetPhysAddr

### 6.3.7 Eth\_30\_<Platform>\_SetPhysAddr

Prototype	
void <b>Eth_30_&lt;Platform&gt;_SetPhysAddr</b> (uint8 CtrlIdx, const uint8 *PhysAddrPtr)	
Parameter	
CtrlIdx [in]	Controller index
PhysAddrPtr [in]	Pointer to the physical address
Return code	
void	void
Functional Description	
Sets the physical source address used by the indexed controller. If "MAC Write Access" is enabled the function also persistently writes the MAC address into NVM and sets the address on next ControllerInit.	
Particularities and Limitations	
- Re-entrant, synchronous	
Call context	
> Interrupt or task level	

Table 6-9 Eth\_30\_<Platform>\_SetPhysAddr



### 6.3.8 Eth\_30\_<Platform>\_WriteMii

Prototype	
Eth_ReturnType <b>Eth_30_&lt;Platform&gt;_WriteMii</b> (uint8 CtrlIdx, uint8 TrcvIdx, uint8 RegIdx, uint16 RegVal)	
Parameter	
CtrlIdx [in]	Controller index
PhysAddrPtr [in]	Pointer to memory containing the physical source address (MAC address) in network byte order.
Eth_30_<Platform>_FilterActionType [in]	Add or remove the address from the Ethernet controllers filter
CtrlIdx [in]	Controller index
TrcvIdx [in]	Transceiver index (MII address)
RegIdx [in]	Transceiver register index
RegVal [in]	Transceiver register value
Return code	
Eth_ReturnType	<ul style="list-style-type: none"> <li>&gt; E_OK : Filter was successfully changed</li> <li>&gt; E_NOT_OK : Filter could not be changed</li> </ul>
Eth_ReturnType	<ul style="list-style-type: none"> <li>&gt; ETH_OK : MII register written</li> <li>&gt; ETH_E_NOT_OK : MII register write failure</li> <li>&gt; ETH_E_NO_ACCESS : Ethernet transceiver access failure</li> </ul>
Functional Description	
Updated the physical source address to/from the indexed controller filter.	
Particularities and Limitations	
- Re-entrant, synchronous	
Call context	
<ul style="list-style-type: none"> <li>&gt; Interrupt or task level</li> <li>&gt; Interrupt or task level</li> </ul>	

Table 6-10 Eth\_30\_&lt;Platform&gt;\_WriteMii

### 6.3.9 Eth\_30\_<Platform>\_ReadMii

Prototype	
Eth_ReturnType <b>Eth_30_&lt;Platform&gt;_ReadMii</b> (uint8 CtrlIdx, uint8 TrcvIdx, uint8 RegIdx, uint16 *RegValPtr)	
Parameter	
CtrlIdx [in]	Controller index
TrcvIdx [in]	Transceiver index (MII address)
RegIdx [in]	Transceiver register index

RegValPtr [out]	Pointer for transceiver register value
<b>Return code</b>	
Eth_ReturnType	<ul style="list-style-type: none"> <li>&gt; ETH_OK : MII register read</li> <li>&gt; ETH_E_NOT_OK : MII register read failure</li> <li>&gt; ETH_E_NO_ACCESS : Ethernet transceiver access failure</li> </ul>
<b>Functional Description</b>	
Read transceiver register via MII.	
<b>Particularities and Limitations</b>	
- Re-entrant, synchronous	
<b>Call context</b>	
> Interrupt or task level	

Table 6-11 Eth\_30\_&lt;Platform&gt;\_ReadMii

### 6.3.10 Eth\_30\_<Platform>\_GetCounterState

<b>Prototype</b>	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_GetCounterState</b> (uint8 CtrlIdx, uint16 CtrOffs, uint32 *CtrValPtr)	
<b>Parameter</b>	
CtrlIdx [in]	Controller index
CtrlCtrOffs [in]	Counter offset into the Mac Management Counter block
CtrValPtr [out]	Counter value
<b>Return code</b>	
Std_ReturnType	<ul style="list-style-type: none"> <li>&gt; E_NOT_OK : Error</li> <li>&gt; E_OK : Success</li> </ul>
<b>Functional Description</b>	
Returns a MAC management counter value.	
<b>Particularities and Limitations</b>	
- Re-entrant, synchronous	
<b>Call context</b>	
> Interrupt or task level	

Table 6-12 Eth\_30\_&lt;Platform&gt;\_GetCounterState

### 6.3.11 Eth\_30\_<Platform>\_ProvideTxBuffer

<b>Prototype</b>	
BufReq_ReturnType <b>Eth_30_&lt;Platform&gt;_ProvideTxBuffer</b> (uint8 CtrlIdx, uint8 *BufIdxPtr, Eth_DataType **BufPtr, uint16 *LenBytePtr)	

Parameter	
CtrlIdx [in]	Controller index
BufIdxPtr [out]	Buffer index
BufPtr [out]	Pointer to buffer area
LenBytePtr [out]	<p>LenBytePtr is an in/out parameter.</p> <p>[in] The requested buffer length. The requested length needs to be the Ethernet header length + payload length.</p> <p>[out] The actual buffer length reduced by Ethernet header length. The Ethernet header is written by <code>Eth_30_&lt;Platform&gt;_Transmit</code>. The actual buffer length is equal to or greater than the requested payload length, as long as the return value is <code>BUFREQ_OK</code>.</p>
Return code	
BufReq_ReturnType	<ul style="list-style-type: none"> <li>&gt; <code>BUFREQ_OK</code> : Buffer locked and ready to use</li> <li>&gt; <code>BUFREQ_E_NOT_OK</code> : Development error check failed</li> <li>&gt; <code>BUFREQ_E_BUSY</code> : All buffers in use. Try later</li> <li>&gt; <code>BUFREQ_E_OVFL</code> : Requested buffer is too large</li> </ul>
Functional Description	
<p>Provide a buffer for frame transmission. The buffer is locked until <code>Eth_30_&lt;Platform&gt;_TxConfirmation</code> is called by interrupt. Alternatively the user may call <code>Eth_30_&lt;Platform&gt;_Transmit</code> with <code>LenByte=0</code> to release the buffer.</p>	
Particularities and Limitations	
- Re-entrant, synchronous	
Call context	
> Interrupt or task level	

Table 6-13 `Eth_30_<Platform>_ProvideTxBuffer`

### 6.3.12 `Eth_30_<Platform>_Transmit`

Prototype	
<code>Std_ReturnType Eth_30_&lt;Platform&gt;_Transmit (uint8 CtrlIdx, uint8 BufIdx, Eth_FrameType FrameType, boolean TxConfirmation, uint16 LenByte, const uint8 *PhysAddrPtr)</code>	
Parameter	
CtrlIdx [in]	Controller index
BufIdx [in]	Buffer index
FrameType [in]	Ethernet frame type, according to type field of IEEE802.3
TxConfirmation [in]	True if a transmit confirmation is desired, otherwise false
LenByte [in]	Payload length (no Ethernet header length included)
PhysAddrPtr [in]	Destination MAC address as byte array.

Return code	
Std_ReturnType	<ul style="list-style-type: none"> <li>&gt; E_NOT_OK : Frame transmission not successful</li> <li>&gt; E_OK : Frame handed over to transmission ring buffer</li> </ul>
Functional Description	
Transmit the locked buffer provided by <code>Eth_30_&lt;Platform&gt;_ProvideTxBuffer</code> and identified by <code>BufIdx</code> .	
Particularities and Limitations	
- Re-entrant, asynchronous	
Call context	
> Interrupt or task level	

Table 6-14 `Eth_30_<Platform>_Transmit`

### 6.3.13 `Eth_30_<Platform>_Receive`

Prototype	
void <b>Eth_30_&lt;Platform&gt;_Receive</b> (uint8 CtrlIdx, Eth_RxStatusType *RxStatusPtr)	
Parameter	
CtrlIdx [in]	Controller index
RxStatusPtr [out]	Indicates whether a frame has been received and if so, whether more frames are available or frames got lost
Return code	
void	void
Functional Description	
Calls the reception callback of all fully received Ethernet frames.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>- NOT Re-entrant, synchronous</li> <li>- If API optimization is enabled, parameter <code>CtrlIdx</code> is void</li> <li>- <code>RxStatusPtr</code> is in interrupt mode unused because information is not used by interrupt handler</li> <li>- When interrupt mode is enabled this function must not be called except from the interrupt handler</li> </ul>	
Call context	
> Interrupt or task level	

Table 6-15 `Eth_30_<Platform>_Receive`

### 6.3.14 `Eth_30_<Platform>_VTransmit`

Prototype	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_VTransmit</b> (uint8 CtrlIdx, uint8 BufIdx, Eth_FrameType FrameType, boolean TxConfirmation, uint16 LenByte, const uint8 *PhysAddrPtrDst, const uint8 *PhysAddrPtrSrc)	

Parameter	
CtrlIdx [in]	Controller index
BufIdx [in]	Buffer index
FrameType [in]	Ethernet frame type, according to type field of IEEE802.3
TxConfirmation [in]	True if a transmit confirmation is desired, otherwise false
LenByte [in]	Payload length (no Ethernet header length included)
PhysAddrPtrDst [in]	Destination MAC address as byte array.
PhysAddrPtrSrc [in]	Source MAC address as byte array.
Return code	
Std_ReturnType	<ul style="list-style-type: none"> <li>&gt; E_NOT_OK : Frame transmission not successful</li> <li>&gt; E_OK : Frame handed over to transmission ring buffer</li> </ul>
Functional Description	
Transmit the locked buffer provided by <code>Eth_30_&lt;Platform&gt;_ProvideTxBuffer</code> and identified by <code>BufIdx</code> .	
Particularities and Limitations	
- Re-entrant, asynchronous	
Call context	
> Interrupt or task level	

Table 6-16 Eth\_30\_&lt;Platform&gt;\_VTransmit

### 6.3.15 Eth\_30\_<Platform>\_TxConfirmation

Prototype	
void <b>Eth_30_&lt;Platform&gt;_TxConfirmation</b> (uint8 CtrlIdx)	
Parameter	
CtrlIdx [in]	Controller index
Return code	
void	void
Functional Description	
Unlocks the buffers of fully transmitted frames.	
Particularities and Limitations	
<ul style="list-style-type: none"> <li>- NOT Re-entrant, synchronous</li> <li>- When interrupt mode is enabled this function must not be called except from the interrupt handler</li> </ul>	
Call context	
> Interrupt or task level	

Table 6-17 Eth\_30\_&lt;Platform&gt;\_TxConfirmation

### 6.3.16 Eth\_30\_<Platform>\_SetBandwidthLimit

Prototype	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_SetBandwidthLimit</b> (uint8 CtrlIdx, uint8 QueuePrio, uint32 BandwidthLimit)	
Parameter	
CtrlIdx [in]	Controller Index
QueuePrio [in]	Queue Priority
BandwidthLimit [in]	Bandwidth limit which shall be assigned for the Tx queue (in [bits/s])
Return code	
Std_ReturnType	<ul style="list-style-type: none"> <li>&gt; E_NOT_OK : New bandwidth limit could not be set.</li> <li>&gt; E_OK : New bandwidth limit set.</li> </ul>
Functional Description	
Reconfigures the bandwidth limit set for a transmission queue.	
Particularities and Limitations	
Input parameter must not be NULL	
Call context	
<ul style="list-style-type: none"> <li>&gt; Interrupt or task level</li> <li>&gt; This function is Synchronous</li> <li>&gt; This function is Reentrant</li> <li>&gt; Availability: Configuration parameter "Traffic Shaping" must be set to "dynamic traffic shaping"</li> </ul>	

Table 6-18 Eth\_30\_<Platform>\_SetBandwidthLimit

### 6.3.17 Eth\_30\_<Platform>\_GetBandwidthLimit

Prototype	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_GetBandwidthLimit</b> (uint8 CtrlIdx, uint8 QueuePrio, uint32 *BandwidthLimitPtr)	
Parameter	
CtrlIdx [in]	Controller Index
QueuePrio [in]	Queue Priority
BandwidthLimitPtr [out]	Pointer to where to store the currently configured bandwidth limit (in [bit/s])
Return code	
Std_ReturnType	<ul style="list-style-type: none"> <li>&gt; E_NOT_OK : Current bandwidth limit could not be retrieved.</li> <li>&gt; E_OK : Current bandwidth limit retrieved.</li> </ul>
Functional Description	
Retrieves the currently configured bandwidth limit of a transmission queue.	
Particularities and Limitations	
Input parameter BandwidthLimitPtr must not be a NULL_PTR	

Call context
<ul style="list-style-type: none"> <li>&gt; Interrupt or task level</li> <li>&gt; This function is Synchronous</li> <li>&gt; This function is Non-Reentrant</li> <li>&gt; Availability: Configuration parameter "Traffic Shaping" must be set to "dynamic traffic shaping"</li> </ul>

Table 6-19 Eth\_30\_&lt;Platform&gt;\_GetBandwidthLimit

### 6.3.18 Eth\_30\_<Platform>\_GetVersionInfo

Prototype	
void <b>Eth_30_&lt;Platform&gt;_GetVersionInfo</b> (Std_VersionInfoType *VersionInfoPtr)	
Parameter	
CtrlIdx [in]	Controller index
VersionInfoPtr [out]	Returns the following version information: <ul style="list-style-type: none"> <li>&gt; Vendor ID</li> <li>&gt; Module ID</li> <li>&gt; Software major version</li> <li>&gt; Software minor version</li> <li>&gt; Software patch version</li> </ul>
Return code	
void	none
Functional Description	
Get driver version	
Particularities and Limitations	
Re-entrant, synchronous	
Call context	
<ul style="list-style-type: none"> <li>&gt; Interrupt or task level</li> </ul>	

Table 6-20 Eth\_30\_&lt;Platform&gt;\_GetVersionInfo

### 6.3.19 Eth\_30\_<Platform>\_MainFunction

Prototype	
void <b>Eth_30_&lt;Platform&gt;_MainFunction</b> (void)	
Parameter	
void	void
Return code	
void	void

Functional Description
Ethernet driver Mainfunction for processing descriptors which could not be finished in interrupt context.
Particularities and Limitations
NOT Re-entrant, synchronous
Call context
> Task

Table 6-21 Eth\_30\_&lt;Platform&gt;\_MainFunction

### 6.3.20 Eth\_30\_<Platform>\_GetGlobalTime

Prototype	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_GetGlobalTime</b> (ETH_PTP_VCTRLIDX_FIRST Eth_TimeStampType *TimestampPtr, Eth_TimeStampQualityType *TimestampQualityPtr)	
Parameter	
CtrlIdx [in]	Controller index
TimestampPtr [out]	Pointer to a timestamp of type Eth_TimeStampType
TimestampQualityPtr [out]	Pointer to the timestamp quality of type Eth_TimeStampQualityType
Return code	
Std_ReturnType	> E_OK : Call successful. Timestamp and TimestampQuality are valid > E_NOT_OK : Obtaining global time failed
Functional Description	
Returns the global time of the controllers timesync submodule.	
Particularities and Limitations	
- Re-entrant, synchronous	
Call context	
> Interrupt or task level	

Table 6-22 Eth\_30\_&lt;Platform&gt;\_GetGlobalTime

### 6.3.21 Eth\_30\_<Platform>\_SetGlobalTime

Prototype	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_SetGlobalTime</b> (ETH_PTP_VCTRLIDX_FIRST const Eth_TimeStampType *TimestampPtr)	
Parameter	
CtrlIdx [in]	Controller index
TimestampPtr [out]	Pointer to a timestamp of type Eth_TimeStampType
Return code	
Std_ReturnType	> E_OK : Time successfully set > E_NOT_OK : Failed to set time



Functional Description
Resets the global time of the controllers PTP submodule to a specified value.
Particularities and Limitations
- Re-entrant, synchronous
Call context
> Interrupt or task level

Table 6-23 Eth\_30\_&lt;Platform&gt;\_SetGlobalTime

### 6.3.22 Eth\_30\_<Platform>\_SetCorrectionTime

Prototype	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_SetCorrectionTime</b> (ETH_PTP_VCTRLIDX_FIRST const Eth_TimediffType *OffsetTimePtr, const Eth_RateRatioType *RateRatioPtr)	
Parameter	
CtrIdx [in]	Controller index
OffsetTimePtr [in]	Pointer to a time difference object of type sint32
RateRatioPtr [in]	Pointer to a rate ratio object of type Eth_RateRatioType
Return code	
Std_ReturnType	> E_OK : Time successfully corrected > E_NOT_OK : Call failed. Try again later
Functional Description	
Corrects the global time by an offset and a rate ratio.	
Particularities and Limitations	
- Re-entrant, synchronous	
Call context	
> Interrupt or task level	

Table 6-24 Eth\_30\_&lt;Platform&gt;\_SetCorrectionTime

### 6.3.23 Eth\_30\_<Platform>\_EnableEgressTimestamp

Prototype	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_EnableEgressTimestamp</b> (ETH_PTP_VCTRLIDX_FIRST uint8 BufIdx)	
Parameter	
CtrIdx [in]	Controller index
BufIdx [in]	Index associated with the buffer containing the frame to be timestamped.
Return code	
Std_ReturnType	> E_OK : Egress timestamping successfully enabled

	> E_NOT_OK : DET check failed or buffer identified with BufIdx not locked.
<b>Functional Description</b>	
Enables timestamping for a frame that is going to be transmitted.	
<b>Particularities and Limitations</b>	
- Re-entrant, synchronous	
<b>Call context</b>	
> Interrupt or task level	

Table 6-25 Eth\_30\_&lt;Platform&gt;\_EnableEgressTimestamp

### 6.3.24 Eth\_30\_<Platform>\_GetEgressTimestamp

<b>Prototype</b>	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_GetEgressTimestamp</b> (ETH_PTP_VCTRLIDX_FIRST uint8 BufIdx, Eth_TimeStampType *TimestampPtr, Eth_TimestampQualityType *TimestampQualityPtr)	
<b>Parameter</b>	
CtrlIdx [in]	Controller index
BufIdx [in]	Index associated with the buffer containing the frame that was timestamped
TimestampPtr [out]	Pointer to a timestamp of type Eth_TimeStampType
TimestampQualityPtr [out]	Pointer to the timestamp quality of type Eth_TimestampQualityType
<b>Return code</b>	
Std_ReturnType	> E_OK : Egress timestamp successfully obtained > E_NOT_OK : DET check failed or wrong call context
<b>Functional Description</b>	
Returns the egress timestamp of the frame identified by BufIdx.	
<b>Particularities and Limitations</b>	
- NOT Re-entrant, synchronous	
<b>Call context</b>	
> Context of Eth_TxConfirmation. All other call contexts are invalid.	

Table 6-26 Eth\_30\_&lt;Platform&gt;\_GetEgressTimestamp

### 6.3.25 Eth\_30\_<Platform>\_GetIngressTimestamp

<b>Prototype</b>	
Std_ReturnType <b>Eth_30_&lt;Platform&gt;_GetIngressTimestamp</b> (ETH_PTP_VCTRLIDX_FIRST Eth_DataType *DataPtr, Eth_TimeStampType *TimestampPtr, Eth_TimestampQualityType *TimestampQualityPtr)	
<b>Parameter</b>	
CtrlIdx [in]	Controller index

DataPtr [in]	Data portion of the frame that was timestamped
TimestampPtr [out]	Pointer to a timestamp of type Eth_TimestampType
TimestampQualityPtr [out]	Pointer to the timestamp quality of type Eth_TimestampQualityType
<b>Return code</b>	
Std_ReturnType	<ul style="list-style-type: none"> <li>&gt; E_OK : Ingress timestamp successfully obtained</li> <li>&gt; E_NOT_OK : DET check failed or wrong call context</li> </ul>
<b>Functional Description</b>	
Returns the ingress timestamp of the frame identified by BufIdx.	
<b>Particularities and Limitations</b>	
- NOT Re-entrant, synchronous	
<b>Call context</b>	
> Context of Eth_RxIndication. All other call contexts are invalid.	

Table 6-27 Eth\_30\_<Platform>\_GetIngressTimestamp

## 6.4 Services used by Ethernet Driver

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

Component	API
DET	Det_ReportError
DEM	Dem_SetEventStatus
ETHIF	EthIf_RxIndication
ETHIF	EthIf_TxConfirmation

Table 6-28 Services used by the Ethernet Driver

## 6.5 Callback Functions

The Ethernet Driver does not provide callback functions.

## 7 Configuration

The Ethernet Driver is configured with the help of the following Vector tools:

> DaVinci Configurator PRO

### 7.1 Configuration Variants

The Ethernet Driver supports the configuration variants

> VARIANT-PRE-COMPILE

The configuration classes of the Ethernet Driver parameters depend on the supported configuration variants. For their definitions please see the Eth\_30\_<Platform>\_bswmd.arxml file.

#### 7.1.1 General Ethernet Driver Configuration

Attribute Name	Value Type	Values (Default value is typed bold)	Description
Dev Error Detect	Boolean	<b>ON</b> OFF	This parameter switches the Development Error Detection and Notification ON or OFF.
Enable Explicit Buffer Mapping	Boolean	ON <b>OFF</b>	This parameter defines if RX and TX Ethernet buffers of different controllers and descriptor queues can be mapped to different memory locations. The final mapping is done via AUTOSAR Memory Map, whereas the user must implement the buffer specific memory sections within the memory map by himself.
Enable Rx Interrupt	Boolean	<b>ON</b> OFF	This parameter defines if the receive interrupt is used.
Enable Tx Interrupt	Boolean	<b>ON</b> OFF	This parameter defines if the transmit interrupt is used.
Index	Integer	<b>0</b> -255	Specifies the Instance Id of this module instance. If only one instance is present it shall have the Id 0.
Mainfunction Period	Integer	<b>0</b> -n	This parameter specifies the period for the Mainfunction call of the Ethernet driver. The Mainfunction is used for processing descriptors, which could not be finished in interrupt context.
Max Ctrls Supported	Integer	0-255	This value specifies the maximum number of controller configurations per

Attribute Name	Value Type	Values (Default value is typed bold)	Description
			driver.
Update Phys Addr Filter	Boolean	ON OFF	Enables/Disables optional API <code>Eth_30_&lt;Platform&gt;_UpdatePhysAddrFilter</code> .
Os Isr Category	Enumeration	CATEGORY_1 <b>CATEGORY_2</b> NO_OS_SUPPORT	<p>This parameter specifies how the interrupt routines will be handled.</p> <p>If CATEGORY_1 or CATEGORY_2 is selected the Os will handle the interrupts according to the selected category.</p> <p>For interrupt handling without support from Os the value NO_OS_SUPPORT must be selected.</p>
Runtime Measurement Support	Boolean	ON <b>OFF</b>	<p>If enabled runtime measurement points are added to the BSW module code and are added to the module configuration of MICROSAR RTM. The available measurement points can be seen and configured in the RTM module configuration. Availability of the MICROSAR RTM module is a prerequisite to make use of the runtime measurement functionality.</p> <p>Runtime measurement should be disabled in production code as measurement points may inflict additional runtime.</p>
Enable Zero Copy Extensions	Boolean	ON <b>OFF</b>	<p>This parameter defines if the Zero-Copy Extensions for reception buffers and transmission buffers are enabled.</p> <p>Impact on handling of:</p> <p>Receive Buffers: The Rx Buffers are not implicitly released after the <code>RxIndication()</code> to the upper layer by the driver itself anymore but must be explicitly released by the Upper Layer. This mechanism allows to keep buffers for explicit usage until the <code>EthIf_ReleaseRxBuffer()</code> API is called.</p> <p>Transmission Buffers: In addition to using internal Tx Buffers provided by the driver during <code>ProvideTxBuffer()</code> a User is able to</p>

Attribute Name	Value Type	Values (Default value is typed bold)	Description
			inject external buffers used for transmission with the call to <code>ProvideExtTxBuffer()</code> .
Enable Header Access API	Boolean	ON <b>OFF</b>	<p>This parameter defines if APIs are provided to allow access to the Ethernet frame header.</p> <p>Usually the Ethernet frame header is cut off during reception or is assembled during transmission by the Ethernet driver. To read the header information during reception or to overwrite the header that is assembled by the driver during transmission these APIs can be used.</p>
Zero Copy Buffer Start Offset	Integer	0-	<p>Configure the buffer start offset for zero copy extension.</p> <p>This offset will be added to each buffer start. The additional space can be used for transmission on other bus systems. It will not be used for external provided buffers.</p> <p>Note: The size of the offset must be a multiple of the expected controller alignment size. Value 0 is allowed to disable the offset.</p>
Use Peripheral Access API	Boolean	ON <b>OFF</b>	<p>This parameter defines if the Peripheral Access APIs are used for accessing protected registers related to the Ethernet driver.</p> <p>Depending on the derivative some Ethernet driver related register may be only accessibly in a Privileged Mode. If the Ethernet driver is not running in a necessary Privileged Mode then the OS has to support the register access by Peripheral Access APIs.</p>
Runtime Environment	Enumeration	ETH_AUTOSAR_OS ETH_GREENHILLS _INTEGRITY	<p>This parameter defines the runtime environment the Ethernet driver is executed on.</p> <p>ETH_AUTOSAR_OS: The driver is executed in a common AUTOSAR OS environment.</p>

Attribute Name	Value Type	Values (Default value is typed bold)	Description
			ETH_GREENHILLS_INTEGRITY: The driver is executed in a Greenhills INTEGRITY OS environment.
Additional Includes	String		This parameter defines an additional file to be included in the <code>Eth_30_&lt;Platform&gt;_Lcfg.h</code> .
Version Info API	Boolean	ON OFF	This parameter enables/disables the function <code>Eth_30_&lt;Platform&gt;_GetVersionInfo()</code> to get major, minor and patch version information.
User Config File	String		Reference to an external text file that will be included during generation to <code>Eth_30_&lt;Platform&gt;_Cfg.h</code> .  The content of the user configuration file will be added at the end of the generated module configuration file and allows altering or extending the generated code.  Caution: User configuration files can cause the software module to malfunction and must only be used with great care!
Physical Address (MAC-Address)	String		This parameter defines the Controller Physical Hardware Address (MAC-Address). The input format has to match <code>xx:xx:xx:xx:xx:xx</code> , where x stands for a hex value between 0 and F.
Enable MAC-Address Write Access	Boolean	ON OFF	This parameter defines if access to an NvM block containing the MAC address and the corresponding API is enabled.
Ctrl Enable PTP	Boolean	ON OFF	This parameter defines if Precision-Time-Protocol support is enabled or not. PTP is mostly used in combination with Audio Video Bridging (AVB).
PHY Management Interface	Enumeration	ETH_MII_MODE ETH_RMI_MODE ETH_GMII_MODE ETH_RGMII_MODE ETH_SGMII_MODE	This parameter defines the management interface type used to connect PHY and MAC.  Please refer to the derivative's reference manual and the board schematic to retrieve the correct management interface type.
Pre Controller Init Callout	Boolean	ON OFF	This parameter defines if a callout is generated, which can be used to add special startup handling before <code>Eth_30_&lt;Platform&gt;_ControllerInit</code> is executed.

Attribute Name	Value Type	Values (Default value is typed bold)	Description
Post Controller Init Callout	Boolean	ON <b>OFF</b>	This parameter defines if a callout is generated, which can be used to add special startup handling after Eth_30_<Platform>_ControllerInit is executed.
Peripheral Region OS Identifier	Function Name		Unique C Identifier which is used by the Ethernet Controller driver for accessing the OS configured peripheral region.
Controller Reset Loop Cycles	Integer	1-1000000	This value specifies the number of loop cycles after which a controller reset timeout occurs.
Controller Loop Cycles	Integer	1-1000000	This value specifies the number of loop cycles after which hardware dependent loop timeouts occur.
Controller MII Loop Cycles	Integer	1-1000000	This value specifies the number of loop cycles after which timeouts for MII accesses occur.

Table 7-1 General Ethernet Configuration

## 7.1.2 Platform Dependent Ethernet Driver Configuration

Please refer to the hardware specific technical reference [5] for these configuration steps.



## 8 Glossary and Abbreviations

### 8.1 Glossary

Term	Description
DaVinci Configurator PRO	Generation tool for MICROSAR components

Table 8-1 Glossary

### 8.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MII	Media Independent Interface
PSPORT	Provide Port
RSPORT	Require Port
RTE	Runtime Environment
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 8-2 Abbreviations

## 9 Contact

Visit our website for more information on

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

[www.vector.com](http://www.vector.com)