User Manual

for S32K14X ETH Driver

Document Number: UM2ETHASR4.2 Rev0002R1.0.2
Rev. 1.0

Contents

Section number **Title** **Page**

Chapter 1 Revision History

Chapter 2 Introduction

Chapter 3 Driver

Section number | Title | Page

Section number Title Page

Chapter 4
Tresos Configuration Plug-in

Chapter 1
Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
1.0	26/04/2019	NXP MCAL Team	Updated version for ASR 4.2.2S32K14XR1.0.2

Chapter 2
Introduction

This User Manual describes NXP Semiconductors AUTOSAR Ethernet (ETH) for S32K14X .

AUTOSAR ETH driver configuration parameters and deviations from the specification are described in ETH Driver chapter of this document. AUTOSAR ETH driver requirements and APIs are described in the AUTOSAR ETH driver software specification document.

2.1 Supported Derivatives

The software described in this document is intented to be used with the following microcontroller devices of NXP Semiconductors .

Table 2-1. S32K14X Derivatives

NXP Semiconductors	s32k148_lqfp144, s32k148_lqfp176, s32k148_mapbga100, s32k146_lqfp144, s32k146_lqfp100, s32k146_lqfp64, s32k146_mapbga100, s32k144_lqfp100, s32k144_lqfp64, s32k144_mapbga100, s32k142_lqfp100, s32k142_lqfp64, s32k118_lqfp48, s32k118_lqfp64, s32k142_lqfp48, s32k144_lqfp48, s32k148_lqfp100

All of the above microcontroller devices are collectively named as S32K14X .

2.2 Overview

AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership working to establish standards for software interfaces and software modules for automobile electronic control systems.

AUTOSAR

- paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness.
- is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation".
- is a key enabling technology to manage the growing electrics/electronics complexity. It aims to be prepared for the upcoming technologies and to improve cost-efficiency without making any compromise with respect to quality.
- facilitates the exchange and update of software and hardware over the service life of the vehicle.

2.3 About this Manual

This Technical Reference employs the following typographical conventions:

Boldface type: Bold is used for important terms, notes and warnings.

Italic font: Italic typeface is used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

Note

This is a note.

2.4 Acronyms and Definitions

Table 2-2. Acronyms and Definitions

Term	Definition
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ENET	Ethernet MAC (Ethernet Controller)
ETH	Ethernet
ETHIF	Ethernet Interface
FEC	Fast Ethernet Controlled (Ethernet Controller)
FIFO	First-In First-Out Reception Storage

Table continues on the next page...

Table 2-2. Acronyms and Definitions (continued)

Term	Definition
N/A	Not Applicable
MCU	Micro Controller Unit
MII	Media Independent Interface
RAM	Random Access Memory
RMII	Reduced Media Independent Interface
VLE	Variable Length Encoding

- The term "Ethernet Controller" is related to the hardware module providing the Ethernet functionality.
- The term "Ethernet Driver" is related to the software handling the Ethernet Controller.
- The term "Application" is used for the software utilizing the Ethernet Driver.

2.5 Reference List

Table 2-3. Reference List

#	Title	Version
1	Specification of ETH Driver	AUTOSAR Release 4.2.2
2	S32K14X Reference Manual	Reference Manual, Rev. 9, 9/2018
3	S32K142 Mask Set Errata for Mask 0N33V (0N33V)	30/11/2017
4	S32K144 Mask Set Errata for Mask 0N57U (0N57U)	30/11/2017
5	S32K146 Mask Set Errata for Mask 0N73V (0N73V)	30/11/2017
6	S32K148 Mask Set Errata for Mask 0N20V (0N20V)	25/10/2018
7	S32K118 Mask Set Errata for Mask 0N97V (0N97V)	07/01/2019

Chapter 3
Driver

3.1 Requirements

Requirements for this driver are detailed in the AUTOSAR 4.2 Rev0002ETH Driver Software Specification document (See Table Reference List).

3.2 Driver Design Summary

The Ethernet Driver controls the Ethernet Controller module of the S32K14X device. It provides the following features:

- Configuration and initialization of the Ethernet Controller
- Switching the Ethernet Controller on and off
- Reception and transmission of the Ethernet frames
- Access to the some Ethernet Controller counters (EtherStats and DropCounts)
- Access to the Ethernet Transceiver device registers through MII
- Ethernet Controller interrupt requests handling
- Half and full duplex operation support
- 10 Mbit/s and 100 Mbit/s MII operation support
- Timer synchronization over Ethernet (required PPP stack in upper layer).
- Hardware accelerator to add/verify checksum for IP package, protocol package (UDP, TCP).

3.3 Hardware Resources

The hardware configured by the Eth driver is the same between derivatives(see detail in chapter 2.1 of Integration Manual).

3.4 Deviations from Requirements

The driver deviates from the Autosar Ethernet Driver software specification in some places. The table Table 3-2 identifies the Autosar requirements that are not fully implemented, implemented differently, or out of scope for the Ethernet Driver. The table Table 3-1 provides the Status column description.

Table 3-1. Deviations Status Column Description

Term	Definition
N/A	Not Available
N/T	Not Testable
N/S	Out of Scope
N/F	Not Fully Implemented
N/I	Not Implemented

Table 3-2. ETH Deviations

Req.	Status	Description	Notes
SWS_Eth_ 00226	N/A	Service name: Eth_GetDropCount Syntax: Std_ReturnType Eth_GetDropCount(uint8 CtrlIdx, uint8 CountValues, uint32* DropCount) Service ID[hex]: 0x14 Sync/Async: Synchronous Reentrancy: Non Reentrant Parameters (in): CtrlIdx Index of the controller within the context of the Ethernet Driver Parameters (inout): CountValues In: Maximal number of values which can be written from DropCount. Out: Number of values which are returned in the DropCount list. Parameters (out): DropCount The interpretation of this list of values is hardware dependent Return value: Std_ReturnType E_OK: success E_NOT_OK: drop counter could not be obtained Description: Reads a list with drop counter values of the corresponding controller. The meaning of these values is hardware dependent. However, the list DropCount[] shall contain the following values in the given order, where the maximal possible value shall denote an invalid value, e.g. if this counter is not available: 1.) dropped packets due to buffer overrun 2.) dropped packets due to CRC errors 3.) number of undersize packets which were less than 64 octets long (excluding framing bits, but including FCS octets) and were otherwise will formed. (see IETF RFC 1757) 4.) number of oversize packets which are longer than 1518 octets (excluding framing bits, but including FCS octets) and were otherwise well formed. (see IETF RFC 1757) 5.) number of alignment errors, i.e. packets which	The prototype of this function changed upon request from CPR-MCAL-767.eth.

Table 3-2. ETH Deviations

Req.	Status	Description	Notes
		are received and are not an integral number of octets in length and do not pass the CRC. 6.) SQE test error according to IETF RFC1643 dot3StatsSQETestErrors 7.) The number of inbound packets which were chosen to be discarded even though no errors had been detected to prevent their being deliverable to a higher-layer protocol. One possible reason for discarding such a packet could be to free up buffer space. (see IETF RFC 2233 ifInDiscards) 8.) total number of erroneous inbound packets 9.) The number of outbound packets which were chosen to be discarded even though no errors had been detected to prevent their being transmitted. One possible reason for discarding such a packet could be to free up buffer space. (see IETF RFC 2233 ifOutDiscards) 10.) total number of erroneous outbound packets 11.) Single collision frames: A count of successfully transmitted frames on a particular interface for which transmission is inhibited by exactly one collision. (see IETF RFC1643 dot3StatsSingleCollisionFrames) 12.) Multiple collision frames: A count of successfully transmitted frames on a particular interface for which transmission is inhibited by more than one collision. (see IETF RFC1643 dot3StatsMultipleCollisionFrames) 13.) Number of deferred transmission: A count of frames for which the first transmission attempt on a particular interface is delayed because the medium is busy. (see IETF RFC1643 dot3StatsDeferredTransmissions) 14.) Number of late collisions: The number of times that a collision is detected on a particular interface later than 512 bit-times into the transmission of a packet. (see IETF RFC1643 dot3StatsLateCollisions) 15.) the following positions in the list can contain hardware dependent counter values	

Eth_VariantNo_PBcfg.c files will contain the definition for all parameters that are variant aware, independent of the configuration class that will be selected (PC, LT, PB). Eth_Cfg.c file will contain the definition for all parameters that are not variant aware.

3.5 Limitations

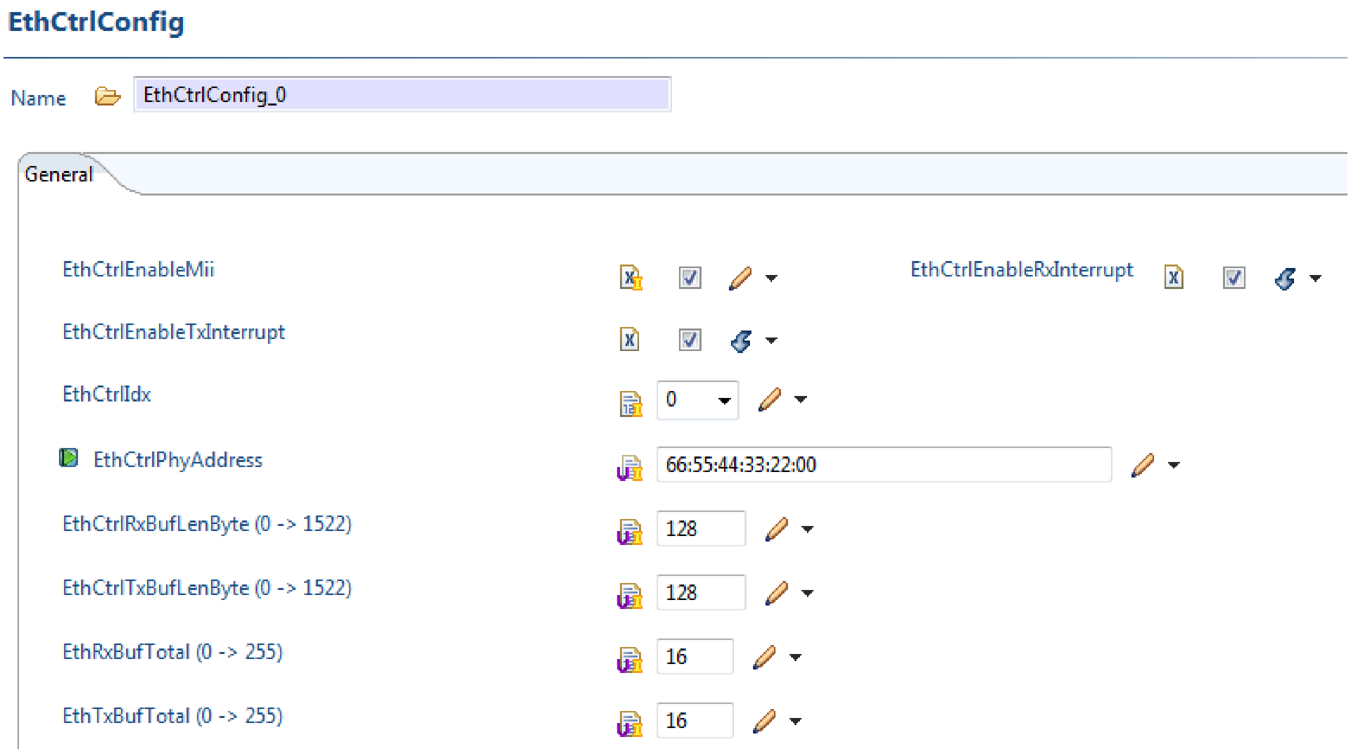
Ethernet Driver has the following limitations:

- Only one Ethernet Controller is supported since there is only one Ethernet controller available on the S32K14X device.

- The `Eth_ReadMii` function is blocking because it has to wait until MII transaction finishes to obtain requested data.
- The `Eth_WriteMii` function is blocking to avoid another MII transaction until the first one finishes.
- Received frames length must be less than or equal to value of *EthCtrlRxBufLenByte* if the `EthUseMultiBufferRxFrames` is not set. Otherwise, the maximum received frame length is set to 1500 bytes. Longer frames consume more than one receive buffer and later they are automatically discarded by this driver.
- Frame reception can be done either in the poll driven mode or in the interrupt driven mode but the mixture is not possible. Receive buffer processing is ignored by call of `Eth_Receive` when the receive interrupt is enabled.
- Frame transmission confirmation can be done either in the poll driven mode or in the interrupt driven mode but the mixture is not possible. Transmit confirmation is ignored by the function `Eth_TxConfirmation` however the transmit interrupt is enabled.
- User should configure at least 3 TxBuffer (EthMaxTXBuffersSupported) for ensure that driver work properly.

3.6 Driver Usage and Configuration tips

Spreading Eth frames over TX and RX multiple buffers:



EthCtrlConfig

Name: EthCtrlConfig_0

EthCtrlEnableMii ☑

EthCtrlEnableRxInterrupt ☑

EthCtrlEnableTxInterrupt ☑

EthCtrlIdx: 0

EthCtrlPhyAddress: 66:55:44:33:22:00

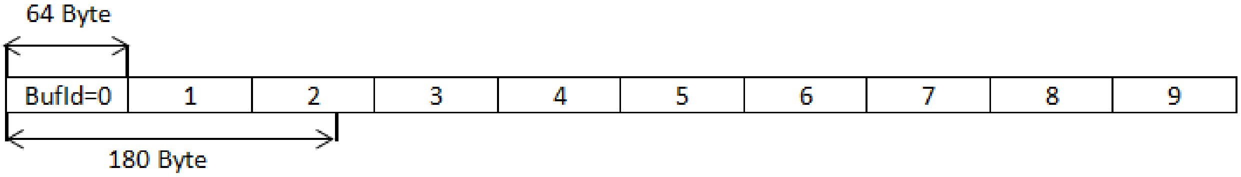
EthCtrlRxBufLenByte (0 -> 1522): 128

EthCtrlTxBufLenByte (0 -> 1522): 128

EthRxBufTotal (0 -> 255): 16

EthTxBufTotal (0 -> 255): 16

- Follow ASR, user can configure the number of buffers and the size for each buffer using: EthTxBufTotal, EthCtrlTxBufLenByte, EthRxBufTotal, EthCtrlRxBufLenByte.
- If user is using both long and short frames, it need to configure higher value for EthCtrlTxBufLenByte/ EthCtrlRxBufLenByte. But take notice that in this case, there will be wasting memory when transmitting short frames.

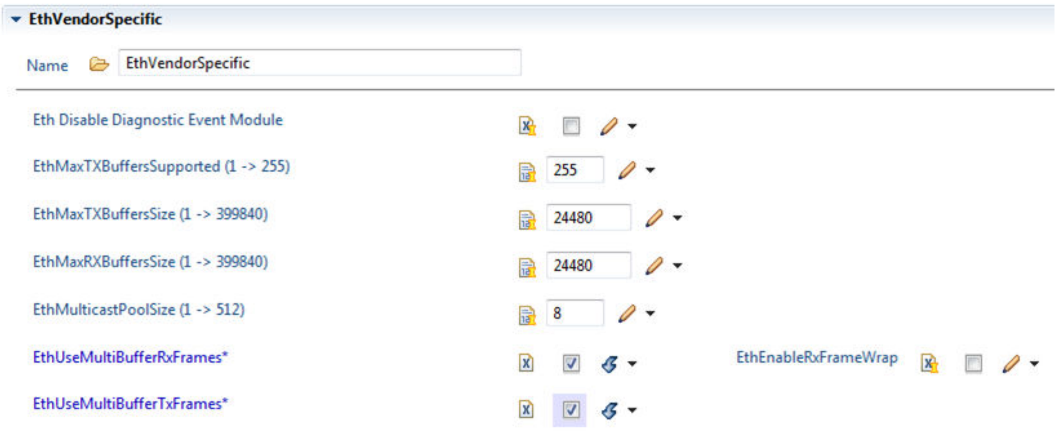


Example:

- Configuration of EthCtrolTxBufLenByte/ EthCtrolRxBufLenByte is 64 bytes.
- Follow ASR, we can only transmit/receive message with length <= 64 Bytes.
- By using enhance feature we could transmit/receive longer than 64. Ex: in the figure, transmitting a frame that have frame length 180 byte when configure the size equal to 64.

How to enable this feature:

- Select: EthUseMultiBufferTxFrames for transmit a frame in multi Tx buffer.
- Select: EthUseMultiBufferRxFrames for receive a frame which longer than a buffer size. However, in this case, if frame received near the wrap region is dropped because memory is not continuous. In order to receive in this case, user need to enable EthEnableRxFrameWrap.



EthVendorSpecific

Name EthVendorSpecific

Eth Disable Diagnostic Event Module □

EthMaxTXBuffersSupported (1 -> 255) 255

EthMaxTXBuffersSize (1 -> 399840) 24480

EthMaxRXBuffersSize (1 -> 399840) 24480

EthMulticastPoolSize (1 -> 512) 8

EthUseMultiBufferRxFrames* ☑ EthEnableRxFrameWrap □

EthUseMultiBufferTxFrames* ☑

3.6.1 Functional Description

3.6.1.1 Initialization

1. The `Eth_Init` function must be called before the ETH Driver can be used. It stores the access to the configuration for the subsequent API calls, configuring the Ethernet Controller and it changes the ETH Driver state from `ETH_STATE_UNINIT` to `ETH_STATE_INIT`.
2. The `Eth_SetControllerMode` with the argument `CtrlMode` set to `ETH_MODE_ACTIVE` will start operation of the Ethernet Controller.

The Ethernet Controller operation can be stopped by the `Eth_SetControllerMode` function call with the argument `CtrlMode` set to `ETH_MODE_DOWN`.

Note

Call of the `Eth_Init` function always stops and resets the Ethernet Controller. All data in the controller buffers are lost.

3.6.1.2 Transmission

The Ethernet driver provides transmit functionality by utilization of so-called transmit buffers. The application issues transmission of an Ethernet frame by calling the following sequence of functions:

1. The application has to reserve one transmit buffer for each Ethernet frame by calling the `Eth_ProvideTxBuffer` function. The desired payload length shall be passed in the `LenBytePtr` argument. This function locks the buffer, if it is available, and it returns its identifier (buffer index) in the `BufIdxPtr` argument. A pointer to memory space where to store payload data is returned in the `BufPtr` argument. The `LengthBytePtr` argument is loaded with actual size of the available memory space. It is on the application to handle a possible difference between the requested and granted length.
2. The application should put the payload data to the memory space pointed by the `BufPtr`.
3. The `Eth_Transmit` function can be called to issue Ethernet frame transmission after the buffer memory space is loaded with the payload data. Corresponding buffer index returned by the `Eth_ProvideTxBuffer` function should be passed in the `BufIdx` argument to identify buffer which shall be transmitted. The payload length (actual fill of the provided memory area) should be passed in the `LenByte` argument. Arguments `PhysAddrPtr` and `FrameType` are used to form the Ethernet frame header.
4. Each call of the `Eth_Transmit` function puts the buffer into transmission queue where it waits until the controller transmits all previously issued frames (i.e. buffers). Frame is transmitted after all previous buffer transmissions are finished.
5. The buffer is returned to the buffers pool after transmission is finished if the `Eth_Transmit` function argument `TxConfirmation` was set to `FALSE`. Otherwise it waits until

the `Eth_TxConfirmation` function is called which reports the buffer indexes of all already transmitted buffers via the ETHIF module callback `EthIf_TxConfirmation` call and then returns them into the buffers pool.

Note

Memory space of the configured *EthCtrlTxBufLenByte* size is always reserved for the frame payload regardless the requested buffer length (input value is ignored).

Note

Frames in the queue are lost if the controller is reconfigured by the `Eth_Init` function.

Note

Interrupt handling routine `Eth_TxIrqHdlr_0` works in the same way as the `Eth_TxConfirmation` function.

3.6.1.3 Reception

Reception of the Ethernet frames starts immediately after the initialization procedure, described in section Initialization, is completed.

Each received Ethernet frame is put into one (or more) receive buffer(s) where it waits until the `Eth_Receive` function is called. The `Eth_Receive` function discards all frames that do not fit into one receive buffer if `EthUseMultiBufferRxFrames` is not set or when the frame length greater than 1500.

When a MAC layer error (invalid CRC, wrong length, collision) is detected on frame, then the frame is automatically discarded. Depending on *EthDropInvalidMAC* configuration parameter, the frames are automatically discarded by hardware (*EthDropInvalidMAC* is enabled) or the frames are put into buffers and later discarded by driver (*EthDropInvalidMAC* is disabled).

Received frames that are not discarded are then reported to the application via the `EthIf_RxIndication` callback of the ETHIF module. The function `Eth_Receive` reports the first received frame and the value of the `RxStatusPtr` informs the application whether more received frames are available in the queue. Then the application can decide whether the `Eth_Receive` function shall be called again to obtain another frame.

Note

Interrupt handling routine `Eth_RxIrqHdlr_0` works in the same way as the `Eth_Receive` function but each received frame is reported in a single function call.

Note

All received frames are lost when the Ethernet Controller is reconfigured by the `Eth_Init` function.

Note

The zero padding will be kept and the CRC is terminated before the package forward to application layer.

CAUTION

It is forbidden to mix polling and interrupt mode i.e. to call both `Eth_Receive` and `Eth_RxIrqHdlr_0` in a single application.

The received frame payload is passed to the ETHIF module through the `DataPtr` argument which is a pointer to the received frame payload beginning. The argument `LenByte` is loaded with the frame payload length. The Ethertype is stored in the argument `FrameType` and a pointer to the frame source MAC address in the argument `PhysAddrPtr`. The argument `IsBroadcast` is set to `TRUE` if the received frame was sent to the broadcast address (FF:FF:FF:FF:FF:FF).

CAUTION

The received frame is no longer accessible after the callback function `EthIf_RxInidication` is finished. Therefore it must copy the received data and frame source address into another buffer if it shall be accessible later on.

3.6.1.4 MII Handling

The ETH Driver provides two functions to access MII, the `Eth_ReadMii` and `Eth_WriteMii`. Both functions take an argument `TrcvIdx`, which is the address of one connected Ethernet transceivers (there is only one connected in most cases), and `RegIdx`, which is the address of a register to be accessed.

Note

Both functions are blocking.

3.6.1.5 Interrupt Support

The ETH Driver provides interrupt handling routine for

- the transmit interrupt - `Eth_TxIrqHdlr_0/Eth_TxIrqHdlr_1`,
- the receive interrupt - `Eth_RxIrqHdlr_0/Eth_RxIrqHdlr_1` and
- *) the combined transmit-receive interrupt - `Eth_TxRxIrqHdlr_0/Eth_TxRxIrqHdlr_1`.

It is up to the application to assign these functions to the appropriate interrupt vectors. The `Eth_TxIrqHdlr_0/Eth_TxIrqHdlr_1` checks and reports all already transmitted frames (buffers) which have `TxConfirmation` set to true as the `Eth_TxConfirmation` function would do. The interrupt flag is also cleared by this function. The `Eth_RxIrqHdlr_0/Eth_RxIrqHdlr_1` function reports all error-free received frames. The interrupt flag is also cleared by this function. The `Eth_TxRxIrqHdlr_0/Eth_TxRxIrqHdlr_1` is a dispatcher for reception and transmission.

Note

*) `Eth_TxRxIrqHdlr_0/Eth_TxRxIrqHdlr_1` is suitable only for platforms without separated reception and transmission interrupts. Otherwise, it shall be disabled

Note

Other interrupt sources are always disabled by the `Eth_Init` function.

CAUTION

The function `Eth_Receive` shall not be called if the receive interrupt is enabled and the function `Eth_TxConfirmation` shall not be called if the transmit interrupt is enabled.

3.6.1.6 MAC Address Operations

The configured physical address (MAC address) of the Ethernet Controller can be read by the `Eth_GetPhysAddr` function. This address is configured by the `Eth_Init` function. The configured physical address can later be changed by the call of `Eth_SetPhysAddr` function.

CAUTION

Change of physical address by the call of `Eth_SetPhysAddr` function succeeds only if the controller is being configured in the ETH_MODE_DOWN mode.

When the API `Eth_UpdatePhysAddrFilter` is enabled, the controller is able to receive multicast frames. List of multicast addresses allowed for reception is managed through `Eth_UpdatePhysAddrFilter` function. Those addresses are stored in the multicast pool which size is defined by configuration parameter *EthMulticastPoolSize*.

3.6.1.7 Support Global time synchronization

The Ethernet Controller has internal timer which supports synchronization over Ethernet packets. Synchronization process follow IEEE 801.2AS. The synchronization required support from upper layer (PTP stack) to control transmit and receive message. The following functions are used to support this feature: `Eth_SetGlobalTime`, `Eth_SetCorrectionTime`, `Eth_GetIngressTimeStamp`, `Eth_GetEgressTimeStamp`, `Eth_EnableEgressTimeStamp`. This feature can be configure On/Off depend on the driver configuration.

3.6.1.8 Support hardware accelerator

The Ethernet Controller support hardware accelerator to add/verify checksum for IPv4, ICMP, TCP and UDP package. If this feature is enable, when transmit, the corresponding bits for the checksum should be set as zero.

3.6.1.9 Other

The Ethernet Controller mode (information whether controller is operational or stopped) can be obtained by calling the `Eth_GetControllerMode` function. The ETH Driver version information is returned by the `Eth_GetVersionInfo` function. It can be implemented as a macro depending on the driver configuration.

3.6.2 Configuration Parameters

The ETH Driver supports the Post-Build, Pre-Compile and Link-Time configuration variants. The following files are generated and compiled in all configuration variants: Eth_Cfg.h, Eth_Cfg.c, Eth_variant_PBcfg.c.

3.6.2.1 Pre-Compile Configuration Parameters

The Pre-Compile parameters and their possible values and their meanings are described in the following text. The Pre-Compile parameters are implemented as the preprocessor defines.

The following configuration parameters in the Eth_Cfg.h file are always Pre-Compile regardless of the configuration variant.

Table 3-3. IMPLEMENTATION-CONFIG-VARIANT

Description	Selects the chosen configuration variant. The ETH Driver uses this value to optimize and select the configuration parameters accesses. The value of this parameter affects all other configuration parameters therefore it is not sufficient to change the source code representation but the whole configuration must be re-generated to change the configuration variant

Table continues on the next page...

Table 3-3. IMPLEMENTATION-CONFIG-VARIANT (continued)

Class	Autosar Parameter
Range	VariantPreCompile, VariantLinkTime, VariantPostBuild
Default	VariantLinkTime
Source File	Eth_Cfg.h
Source Representation	`#define ETH_CONFIG_VARIANT VARIANT_PRE_COMPILE`

Table 3-4. EthMaxCtrlsSupported

Description	Configures the number of Ethernet Controllers supported by the ETH Driver.
Class	Autosar Parameter
Range	1..255 Only value 1 can be selected.
Default	1
Source File	Eth_Cfg.h
Source Representation	`#define ETH_MAXCTRLS_SUPPORTED 1U`

Table 3-5. EthVersionInfoApi

Description	Enables or disables compilation of the `Eth_GetVersionInfo` API function. This function is not affected by the *EthVersionInfoApiMacro* configuration parameter value.
Class	Autosar Parameter
Range	STD_ON, STD_OFF
Default	STD_ON
Source File	Eth_Cfg.h
Source Representation	`#define ETH_VERSION_INFO_API STD_ON`

Table 3-6. EthVersionInfoApiMacro

Description	Selects between a real code or a macro implementation of the `Eth_GetVersionInfo` API function. This value is ignored if the *EthVersionInfoApi* parameter is set to STD_OFF because the `Eth_GetVersionInfo` API function is not compiled.
Class	Autosar Parameter
Range	STD_ON, STD_OFF
Default	STD_OFF
Source File	Eth_Cfg.h
Source Representation	`#define ETH_VERSION_INFO_API_MACRO STD_OFF`

Table 3-7. EthUpdatePhysAddrFilter

Description	Enables or disables Eth_UpdatePhysAddFilter functionality.
Class	Autosar 4.1.1. Parameter
Range	STD_ON, STD_OFF
Default	STD_ON
Source File	Eth_Cfg.h
Source Representation	`#define ETH_UPDATE_PHYS_ADDR_FILTER STD_ON`

Table 3-8. EthCtrlEnableMii

Description	Enables or disables compilation of the `Eth_ReadMii` and `Eth_WriteMii` API functions. It can have value STD_OFF only when the value in all multiple configurations is STD_OFF because the code cannot be excluded from compilation when at least one multiple configuration could use it.
Class	Autosar Parameter
Range	STD_ON, STD_OFF
Default	STD_ON
Source File	Eth_Cfg.h
Source Representation	`#define ETH_CTRLENABLE_MII STD_ON`

Table 3-9. EthDevErrorDetect

Description	Enables or disables development error detection
Class	Autosar Parameter
Range	STD_ON, STD_OFF
Default	STD_ON
Source File	Eth_Cfg.h
Source Representation	`#define ETH_DEV_ERROR_DETECT STD_ON`

Table 3-10. EthConfigSet

Description	This is a multiple configuration container however the number of contained configurations is used as one of the configuration parameters. It limits the number of possible Post-Build configurations supported by the driver.
Class	Autosar Parameter
Range	1..255
Default	1
Source File	Eth_Cfg.h
Source Representation	`#define ETH_NUM_OF_CONFIGURATIONS 1U`

Table 3-11. EthIndex

Description	Specifies the driver instance. It is returned by the `Eth_GetVersionInfo` API function and passed to the `Det_ReportError` function.
Class	Autosar Parameter
Range	0..254 (but only value 0 makes sense because there is always only one ETH Driver)
Default	0
Source File	Eth_Cfg.h
Source Representation	`#define ETH_DRIVER_INSTANCE 0U`

Table 3-12. EthMaxTXBuffersSupported

Description	Limits the number of the transmit buffers supported by the ETH Driver. Each transmit buffer requires one byte of the ETH Driver internal memory even if it is not used. This configuration parameter allows a memory size optimization. The ETH Driver allocates memory space only for *EthMaxTXBuffersSupported* buffers when only small number of buffers is required.
Class	Implementation Specific Parameter
Range	1..255
Default	1
Source File	Eth_Cfg.h
Source Representation	`#define ETHTXBUFNUM 255U`

Table 3-13. EthMulticastPoolSize

Description	Defines the size of pool for multicast address handling. Via Eth_UpdatePhysAddrFilter are defined multicast addreses to be enabled for receive. This addresses are stored in the pool which size is defined by this parameter. When count of addresses in pool reach this value, then for receive is used only "Hash Algorithm"
Class	Implementation Specific Parameter
Range	1..512
Default	15
Source File	Eth_Cfg.h
Source Representation	`#define ETH_MULTICAST_POOL_SIZE 15U`

Table 3-14. EthUseMultiBufferRxFrames

Description	Enables or disables reception of frames spread over multiple RX buffers. When the feature is turned off the multi-buffer frames (i.e. frames which do not fit into a single buffer) are being discarded.
Class	Implementation Specific Parameter
Range	STD_ON, STD_OFF

Table continues on the next page...

Table 3-14. EthUseMultiBufferRxFrames (continued)

Default	STD_OFF
Source File	Eth_Cfg.h
Source Representation	`#define ETH_USE_MULTIBUFFER_RX_FRAMES STD_ON`

Table 3-15. EthEnableRxFrameWrap

Description	Enables or disables support of multi-buffer frames wrapped over the receive buffer ring boundary. When this feature is disabled then the wrapped frames are being discarded. Parameter is accessible and considered during generation only when value of EthUseMultiBufferRxFrames is set to 'true'.
Class	Implementation Specific Parameter
Range	STD_ON, STD_OFF
Default	STD_OFF
Source File	Eth_Cfg.h
Source Representation	`#define ETH_USE_RX_FRAMES_WRAP STD_ON`

Table 3-16. EthEnableUserModeSupport

Description	When this parameter is enabled, the Eth module will adapt to run from User Mode with the following measure : configuring REG_PROT for Eth Controllers so that the registers under protection can be accessed from user mode by setting UAA bit in REG_PROT_GCR to 1, using 'call trusted function' stubs for all internal function calls that access registers requiring supervisor mode, for more information and availability on this platform, please see chapter "User ModeSupport" in IM.
Class	Implementation Specific Parameter
Range	STD_ON, STD_OFF
Default	STD_OFF
Source File	Eth_Cfg.h
Source Representation	`#define ETH_ENABLE_USER_MODE_SUPPORT STD_ON`

Table 3-17. EthUseMultiBufferTxFrames

Description	Enables or disables support of spreading Eth frames over multiple TX buffers. When this feature is turned off it is not possible to send frame longer than single TX buffer.
Class	Implementation Specific Parameter
Range	STD_ON, STD_OFF
Default	STD_OFF
Source File	Eth_Cfg.h
Source Representation	`#define ETH_USE_MULTIBUFFER_TX_FRAMES STD_ON`

Table 3-18. EthGetDropCountApi

Description	Enables / Disables Eth_GetDropCount API.
Class	Autosar Parameterr
Range	STD_ON, STD_OFF
Default	STD_OFF
Source File	Eth_Cfg.h
Source Representation	`#define ETH_GETDROPCOUNTAPI STD_ON`

Table 3-19. EthGetEtherStatsApi

Description	Enables / Disables Eth_GetEtherStats API.
Class	Autosar Parameterr
Range	STD_ON, STD_OFF
Default	STD_OFF
Source File	Eth_Cfg.h
Source Representation	`#define ETH_GET_ETHERSTATS_API STD_ON`

Table 3-20. EthGlobalTimeSupport

Description	Enables/Disables the GlobalTime APIs used amongst others by Global Time Synchronization over Ethernet.
Class	Autosar Parameterr
Range	STD_ON, STD_OFF
Default	STD_OFF
Source File	Eth_Cfg.h
Source Representation	`#define ETH_GLOBALTIME_SUPPORT STD_ON`

Table 3-21. EthCtrlEnableOffloadChecksumIPv4

Description	Enables/Disables hardware accelerator to do checksum for IPv4 packets.
Class	Autosar Parameterr
Range	STD_ON, STD_OFF
Default	STD_OFF
Source File	Eth_Cfg.h
Source Representation	`#define ETH_ENABLE_OFFLOAD_CRC_IPV4 STD_ON`

Table 3-22. EthCtrlEnableOffloadChecksumICMP

Description	Enables/Disables hardware accelerator to do checksum for ICMP packets.
Class	Autosar Parameterr
Range	STD_ON, STD_OFF
Default	STD_OFF
Source File	Eth_Cfg.h
Source Representation	`#define ETH_ENABLE_OFFLOAD_CRC_ICMP STD_ON`

Table 3-23. EthCtrlEnableOffloadChecksumTCP

Description	Enables/Disables hardware accelerator to do checksum for TCP packets.
Class	Autosar Parameterr
Range	STD_ON, STD_OFF
Default	STD_OFF
Source File	Eth_Cfg.h
Source Representation	`#define ETH_ENABLE_OFFLOAD_CRC_TCP STD_ON`

Table 3-24. EthCtrlEnableOffloadChecksumUDP

Description	Enables/Disables hardware accelerator to do checksum for UDP packets.
Class	Autosar Parameterr
Range	STD_ON, STD_OFF
Default	STD_OFF
Source File	Eth_Cfg.h
Source Representation	`#define ETH_ENABLE_OFFLOAD_CRC_UDP STD_ON`

Table 3-25. EthMaxTXBuffersSize

Description	Define maximum size (in Bytes) of all TX buffers includes buffer descriptors (32 bytes for each TX buffer) and buffer data (sized of each defined by EthCtrlTxBufLenByte). Please take care this parameter carefully to optimize the memory used by the dirvers.
Class	Vendor Specific Parameterr
Range	0, 399840
Default	24480
Source File	Eth_Cfg.h
Source Representation	`#define ETH_TX_BUF_MEM_SIZE 24480`

Table 3-26. EthMaxRXBuffersSize

Description	Define maximum size (in Bytes) of all RX buffers includes buffer descriptors (32 bytes for each RX buffer) and buffer data (sized of each defined by EthCtrlRxBufLenByte). Please take care this parameter carefully to optimize the memory used by the dirvers.
Class	Vendor Specific Parameterr
Range	0, 399840
Default	24480
Source File	Eth_Cfg.h
Source Representation	`#define ETH_RX_BUF_MEM_SIZE 24480`

The following parameters configuration are present in the Eth_Cfg.c file to specify the parameters which is precompile for all Variant.

Table 3-27. EthCtrlIdx

Description	Defines the controller index within the ETHIF module context. It is passed to the `EthIf_RxIndication` and `EthIf_TxConfirmation` callbacks.
Class	Autosar Parameter
Range	0..255
Default	0
Source File	Eth_Cfg.c

Source Representation	`static CONST(Eth_StaticCtrlCfgType, ETH_APPL_CONST)`
	`Eth_StaticEthCtrlConfig_0 =`
	`{`
	`    ((VAR(boolean, AUTOMATIC))TRUE),`
	`    ((VAR(boolean, AUTOMATIC))TRUE),`
	`    ((VAR(boolean, AUTOMATIC))TRUE),`
	`    0U``,`
	`    (VAR(uint32, AUTOMATIC))(&Eth_TxBuffers[0][0]),`
	`    (VAR(uint32, AUTOMATIC))(&Eth_RxBuffers[0][0])`
	`};`

Table 3-28. EthCtrlEnableRxInterrupt

Description	Enables or disables the interrupt request generation when a frame has been received.
Class	Autosar Parameter
Range	TRUE, FALSE
Default	FALSE
Source File	Eth_Cfg.c

Source Representation	`static CONST(Eth_StaticCtrlCfgType, ETH_APPL_CONST)`
	`Eth_StaticEthCtrlConfig_0 =`
	`{`
	`    ((VAR(boolean, AUTOMATIC))TRUE)``,`
	`    ((VAR(boolean, AUTOMATIC))TRUE),`
	`    ((VAR(boolean, AUTOMATIC))TRUE),`
	`    0U,`
	`    (VAR(uint32, AUTOMATIC))(&Eth_TxBuffers[0][0]),`
	`    (VAR(uint32, AUTOMATIC))(&Eth_RxBuffers[0][0])`
	`};`

Table 3-29. EthCtrlEnableTxInterrupt

Description	Enables or disables the interrupt request generation when a frame has been transmitted.
Class	Autosar Parameter
Range	TRUE, FALSE
Default	FALSE
Source File	Eth_Cfg.c

Source Representation	`static CONST(Eth_StaticCtrlCfgType, ETH_APPL_CONST)`
	`Eth_StaticEthCtrlConfig_0 =`
	`{`
	`    ((VAR(boolean, AUTOMATIC))TRUE),`
	`    ((VAR(boolean, AUTOMATIC))TRUE)``,`
	`    ((VAR(boolean, AUTOMATIC))TRUE),`
	`    0U,`
	`    (VAR(uint32, AUTOMATIC))(&Eth_TxBuffers[0][0]),`
	`    (VAR(uint32, AUTOMATIC))(&Eth_RxBuffers[0][0])`
	`};`

Table 3-30. EthCtrlSupportMDIO

Description	For some platforms which support multicontroller, there might be possibility that a controller does not support MII.
Class	Vendor Specific Parameter
Range	TRUE, FALSE
Default	FALSE
Source File	Eth_Cfg.c

Source Representation	`static CONST(Eth_StaticCtrlCfgType, ETH_APPL_CONST)`
	`Eth_StaticEthCtrlConfig_0 =`
	`{`
	`    ((VAR(boolean, AUTOMATIC))TRUE),`
	`    ((VAR(boolean, AUTOMATIC))TRUE),`
	`    ((VAR(boolean, AUTOMATIC))TRUE)``,`
	`    0U,`
	`    (VAR(uint32, AUTOMATIC))(&Eth_TxBuffers[0][0]),`
	`    (VAR(uint32, AUTOMATIC))(&Eth_RxBuffers[0][0])`
	`};`

3.6.2.2 Post-Build Configuration Parameters

The Post-Build parameters are placed into the Eth_>Variant<PBcfg.c

All configuration parameters for each variant will be allocated in different files with suffix >Variant<

Table 3-31. EthCtrlRxBufLenByte

Description	Specifies the maximal length of the received Ethernet frame in bytes. This value in is computed from the desired payload length by adding 18 bytes and rounding up to a multiple of 16 bytes.

Table continues on the next page...

Table 3-31. EthCtrlRxBufLenByte (continued)

Class	Autosar Parameter
Range	0..1522 However a value less than 64 bytes does not make sense because of the minimal Ethernet Frame length requirement (payload length is then 46 bytes). A value greater than 1518 does not also make sense because the maximal payload length of the Ethernet frame is 1500 bytes. It is strongly recommended to use values from the range 64..1518.
Default	64 (payload length 46+18 = 64, rounding-up to multiple of 16 results in 64)
Source File	Eth_PBcfg.c

Source Representation	`static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)`
	`EthConfigSet_EthCtrlConfig_0 =`
	`{`
	`    0x66554433U,`
	`    0x22008808U,`
	`    0x00000104U,`
	`    (((VAR(uint32, AUTOMATIC))64U)<<16U) | 0x40004005U,`
	`    (((VAR(uint32, AUTOMATIC))32U)<<1U) | ((VAR(uint32,`
	`AUTOMATIC))0U<<8U),`
	`    0x00000040U,`
	`    0x00000040U,`
	`    12U,`
	`#if STD_ON == ETH_DEM_EVENT_DETECT`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_ACCESS },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_CRC },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_ALIGNMENT },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_LATECOLLISION },`
	`#endif /* ETH_DEM_EVENT_DETECT */`
	`#if (STD_ON == ETH_GLOBALTIME_SUPPORT)`
	`    ((VAR(uint32, AUTOMATIC))1000000U),`
	`#endif`
	`    `**`64U`**`,`
	`    64U,`
	`    255U,`
	`    255U,`
	`};`

Table 3-32. EthCtrlTxBufLenByte

Description	Specifies the maximal length of the transmitted Ethernet frame in bytes. This value is computed from the desired payload length by adding 14 bytes and rounding-up to a multiple of 4.
Class	Autosar Parameter
Range	0..1522 However a payload length of 0 bytes does not make sense. All frames with payload length less than 46 bytes are automatically padded by the controller. It is impossible to

Table continues on the next page...

Table 3-32. EthCtrlTxBufLenByte (continued)

<table>
<tr><td></td><td>transmit a frame with the payload length greater than 1500 bytes. It is strongly recommended to use values from the range 1..1518.</td></tr>
<tr><td><b>Default</b></td><td>64 (payload length 50+14=64, rounding-up to multiple of 4 results in 64)</td></tr>
<tr><td><b>Source File</b></td><td>Eth_PBcfg.c</td></tr>
<tr><td><b>Source Representation</b></td><td><pre>
static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)
EthConfigSet_EthCtrlConfig_0 =
{
    0x66554433U,
    0x22008808U,
    0x00000104U,
    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,
    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,
AUTOMATIC))0U>>8U),
    0x00000040U,
    0x00000040U,
    12U,
#if STD_ON == ETH_DEM_EVENT_DETECT
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_ACCESS },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_CRC },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_ALIGNMENT },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_LATECOLLISION },
#endif /* ETH_DEM_EVENT_DETECT */
#if (STD_ON == ETH_GLOBALTIME_SUPPORT)
    ((VAR(uint32, AUTOMATIC))1000000U),
#endif
    64U,
    64U,
    255U,
    255U,
};
</pre></td></tr>
</table>

Note

The *EthCtrlRxBufLenByte* and the *EthCtrlTxBufLenByte* values are lengths of the used receiver respective transmit buffer therefore they are not equal to the frame payload lengths.

Table 3-33. EthRxBufTotal

Description	Configures the number of the available receive buffers. Each received Ethernet frame occupies one receive buffer. The frames with payload longer than *EthCtrlRxBufLenByte* consume more receive buffers.
Class	Autosar Parameter

Table continues on the next page...

Table 3-33. EthRxBufTotal (continued)

Range	0..255 However the value of 0 does not make sense. It is strongly recommended to configure at least two receive buffers.
Default	16
Source File	Eth_PBcfg.c

Source Representation	`static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)`
	`EthConfigSet_EthCtrlConfig_0 =`
	`{`
	`    0x66554433U,`
	`    0x22008808U,`
	`    0x00000104U,`
	`    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,`
	`    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,`
	`AUTOMATIC))0U>>8U),`
	`    0x00000040U,`
	`    0x00000040U,`
	`    12U,`
	`#if STD_ON == ETH_DEM_EVENT_DETECT`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_ACCESS },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_CRC },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_ALIGNMENT },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_LATECOLLISION },`
	`#endif /* ETH_DEM_EVENT_DETECT */`
	`#if (STD_ON == ETH_GLOBALTIME_SUPPORT)`
	`    ((VAR(uint32, AUTOMATIC))1000000U),`
	`#endif`
	`    64U,`
	`    64U,`
	`    `**`255U`**`,`
	`    255U,`
	`};`

Table 3-34. EthTxBufTotal

Description	Configures the number of the available transmit buffers. Each transmitted Ethernet frame occupies exactly one transmit buffer.
Class	Autosar Parameter
Range	0..255 However the value of 0 does not make sense. It is strongly recommended to configure at least one transmit buffer. The upper boundary is limited by the value of *EthMaxTXBuffersSupported*. Note that at least 3 transmit buffers are needed to avoid the hardware bug described in the errata e19475.
Default	16
Source File	Eth_PBcfg.c

Table continues on the next page...

Table 3-34. EthTxBufTotal (continued)

Source Representation	static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)
	EthConfigSet_EthCtrlConfig_0 =
	{
	    0x66554433U,
	    0x22008808U,
	    0x00000104U,
	    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,
	    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,
	AUTOMATIC))0U>>8U),
	    0x00000040U,
	    0x00000040U,
	    12U,
	#if STD_ON == ETH_DEM_EVENT_DETECT
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ACCESS },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_CRC },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ALIGNMENT },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_LATECOLLISION },
	#endif /* ETH_DEM_EVENT_DETECT */
	#if (STD_ON == ETH_GLOBALTIME_SUPPORT)
	    ((VAR(uint32, AUTOMATIC))1000000U),
	#endif
	    64U,
	    64U,
	    255U,
	    255U,
	};

Table 3-35. EthPhyInterface

Description	Selects between MII_100Mbs and RMII_100Mbs mode. It reflects the current interface between the Ethernet Controller and the Ethernet PHY Transceiver. The MII_100Mbs value is represented by clearing the 8th least significant bit (with weight 8) in the `EthRCR` field in `Eth_CtrlCfgType` structure otherwise the RMII_100Mbs value is represented by setting the 8th least significant bit (with weight 8) in the `EthRCR` field in `Eth_CtrlCfgType` structure.
Class	Implementation Specific Parameter
Range	MII, RMII
Default	MII
Source File	Eth_PBcfg.c

Source Representation	static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)
	EthConfigSet_EthCtrlConfig_0 =
	{
	    0x66554433U,
	    0x22008808U,
	    0x00000104U,
	    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004**0**05U,

Table 3-35. EthPhyInterface

	(((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,
	AUTOMATIC))0U>>8U),
	    0x00000040U,
	    0x00000040U,
	    12U,
	#if STD_ON == ETH_DEM_EVENT_DETECT
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ACCESS },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_CRC },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ALIGNMENT },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_LATECOLLISION },
	#endif /* ETH_DEM_EVENT_DETECT */
	#if (STD_ON == ETH_GLOBALTIME_SUPPORT)
	    ((VAR(uint32, AUTOMATIC))1000000U),
	#endif
	    64U,
	    64U,
	    255U,
	    255U,
	};

Table 3-36. ETH_E_ACCESS

Description	Structure which consists of enablement or disablement of specific DEM event and reference of *DemEventParameter* which shall be issued when the error "Controller access failed" has occurred. A value of referenced *DemEventParameter*/*DemEventID* published by the DEM module via "Dem.h" file as a macro `ETH_E_ACCESS` is used when the code is generated.
Class	Autosar Parameter
Range	N/A (*STD_ON |STD_OFF for state, DemEventID* range is 1..65535 for id)
Default	N/A
Source File	Eth_PBcfg.c

Source Representation	static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)
	EthConfigSet_EthCtrlConfig_0 =
	{
	    0x66554433U,
	    0x22008808U,
	    0x00000104U,
	    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,
	    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,
	AUTOMATIC))0U>>8U),
	    0x00000040U,
	    0x00000040U,
	    12U,
	#if STD_ON == ETH_DEM_EVENT_DETECT
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ACCESS },
	    {(VAR(uint32, AUTOMATIC))STD_ON,

Table 3-36. ETH_E_ACCESS

	DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_CRC },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ALIGNMENT },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_LATECOLLISION },
	#endif /* ETH_DEM_EVENT_DETECT */
	#if (STD_ON == ETH_GLOBALTIME_SUPPORT)
	    ((VAR(uint32, AUTOMATIC))1000000U),
	#endif
	    64U,
	    64U,
	    255U,
	    255U,
	};

Table 3-37. EthCtrlPhyAddress

Description	Specifies the unique 48-bit physical address (MAC) of the controller in network byte order. The MAC address 66:55:44:33:22:11 is represented as 0x665544332211.
Class	Implementation Specific Parameter
Range	00:00:00:00:00:00..ff:ff:ff:ff:ff:ff
Default	66:55:44:33:22:11
Source File	Eth_PBcfg.c

Source Representation	static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)
	EthConfigSet_EthCtrlConfig_0 =
	{
	    0x66554433U,
	    0x22008808U,
	    0x00000104U,
	    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,
	    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,
	AUTOMATIC))0U>>8U),
	    0x00000040U,
	    0x00000040U,
	    12U,
	#if STD_ON == ETH_DEM_EVENT_DETECT
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ACCESS },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_CRC },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ALIGNMENT },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },

Table 3-37. EthCtrlPhyAddress

	{(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION }, {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_LATECOLLISION }, #endif /* ETH_DEM_EVENT_DETECT */ #if (STD_ON == ETH_GLOBALTIME_SUPPORT) ((VAR(uint32, AUTOMATIC))1000000U), #endif 64U, 64U, 255U, 255U, };

Table 3-38. EthFullDuplexEnable

Description	Enables or disables the full duplex operation. The Ethernet Controller ignores collision and carrier sense signals in the full duplex mode. The ENABLED value is represented by setting the 3rd least significant bit (with weight 4) in the `EthTCR` field and by clearing the 2nd least significant bit (with weight 2) in the `EthRCR` field in the `Eth_CtrlCfgType` structure instance.
Class	Implementation Specific Parameter
Range	ENABLE, DISABLE
Default	ENABLE
Source File	Eth_PBcfg.c
Source Representation	static CONST(Eth_CtrlCfgType, ETH_APPL_CONST) EthConfigSet_EthCtrlConfig_0 = { 0x66554433U, 0x22008808U, 0x00000104U, (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U, (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32, AUTOMATIC))0U>>8U), 0x00000040U, 0x00000040U, 12U, #if STD_ON == ETH_DEM_EVENT_DETECT {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_ACCESS }, {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST }, {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_CRC }, {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME }, {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME }, {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_ALIGNMENT }, {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_SINGLECOLLISION }, {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION }, {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_LATECOLLISION }, #endif /* ETH_DEM_EVENT_DETECT */ #if (STD_ON == ETH_GLOBALTIME_SUPPORT) ((VAR(uint32, AUTOMATIC))1000000U), #endif

Table 3-38. EthFullDuplexEnable

	64U, 64U, 255U, 255U, };

Table 3-39. EthEnableLoopbackMode

Description	Enables or disables the loopback mode operation. This mode is intended only for debugging purposes. The Ethernet Controller is instructed to perform the reception while transmitting. The ENABLE value is represented by clearing the 2nd least significant bit (with weight 2) in the `EthRCR` field in the `Eth_CtrlCfgType` structure instance. Note that the Ethernet Controller is instructed to perform reception while transmitting also if the full duplex mode is disabled.
Class	Implementation Specific Parameter
Range	ENABLE, DISABLE
Default	DISABLE
Source File	Eth_PBcfg.c

Source Representation	static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)
	EthConfigSet_EthCtrlConfig_0 =
	{
	    0x66554433U,
	    0x22008808U,
	    0x00000104U,
	    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,
	    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32, AUTOMATIC))0U>>8U),
	    0x00000040U,
	    0x00000040U,
	    12U,
	#if STD_ON == ETH_DEM_EVENT_DETECT
	    {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_ACCESS },
	    {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },
	    {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_CRC },
	    {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_ALIGNMENT },
	    {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON, DemConf_DemEventParameter_ETH_E_LATECOLLISION },
	#endif /* ETH_DEM_EVENT_DETECT */
	#if (STD_ON == ETH_GLOBALTIME_SUPPORT)
	    ((VAR(uint32, AUTOMATIC))1000000U),
	#endif
	    64U,
	    64U,
	    255U,
	    255U,
	};

Table 3-40. EthInternalLoopbackMode

Description	Enables or disables the usage of an internal loopback. Enabling the internal loopback connects the transmitter output to the receiver input and disables driving the Ethernet Controller outputs. No external device is then needed to test the controller. The external loopback can be created using a hardware loopback or by configuring the Ethernet transceiver to the loopback mode when the internal loopback is disabled. The loopback modes are intended for testing purposes only. The ENABLE value is represented by setting the least significant bit (with weight 1) in the `EthRCR` field in the `Eth_CtrlCfgType` structure instance.
Class	Implementation Specific Parameter
Range	ENABLE, DISABLE
Default	DISABLE
Source File	Eth_PBcfg.c

Source Representation	`static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)`
	`EthConfigSet_EthCtrlConfig_0 =`
	`{`
	`    0x66554433U,`
	`    0x22008808U,`
	`    0x00000104U,`
	`    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,`
	`    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,`
	`AUTOMATIC))0U>>8U),`
	`    0x00000040U,`
	`    0x00000040U,`
	`    12U,`
	`#if STD_ON == ETH_DEM_EVENT_DETECT`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_ACCESS },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_CRC },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_ALIGNMENT },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_LATECOLLISION },`
	`#endif /* ETH_DEM_EVENT_DETECT */`
	`#if (STD_ON == ETH_GLOBALTIME_SUPPORT)`
	`    ((VAR(uint32, AUTOMATIC))1000000U),`
	`#endif`
	`    64U,`
	`    64U,`
	`    255U,`
	`    255U,`
	`};`

Table 3-41. EthReceiveBroadcast

Description	Enables or disables the reception of the broadcast Ethernet frames (sent with destination MAC address ff:ff:ff:ff:ff:ff). The ENABLE value is represented by clearing the 5th least

Table continues on the next page...

Table 3-41. EthReceiveBroadcast (continued)

	significant bit (with weight 16) in the `EthRCR` field in the `Eth_CtrlCfgType` structure instance.
Class	Implementation Specific Parameter
Range	ENABLE, DISABLE
Default	ENABLE
Source File	Eth_PBcfg.c

Source Representation	`static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)`
	`EthConfigSet_EthCtrlConfig_0 =`
	`{`
	`    0x66554433U,`
	`    0x22008808U,`
	`    0x00000104U,`
	`    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,`
	`    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,`
	`AUTOMATIC))0U>>8U),`
	`    0x00000040U,`
	`    0x00000040U,`
	`    12U,`
	`#if STD_ON == ETH_DEM_EVENT_DETECT`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_ACCESS },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_CRC },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_ALIGNMENT },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },`
	`    {(VAR(uint32, AUTOMATIC))STD_ON,`
	`DemConf_DemEventParameter_ETH_E_LATECOLLISION },`
	`#endif /* ETH_DEM_EVENT_DETECT */`
	`#if (STD_ON == ETH_GLOBALTIME_SUPPORT)`
	`    ((VAR(uint32, AUTOMATIC))1000000U),`
	`#endif`
	`    64U,`
	`    64U,`
	`    255U,`
	`    255U,`
	`};`

Table 3-42. EthEnablePromiscuousMode

Description	Enables or disables the promiscuous mode which means that the Ethernet Controller receives all Ethernet frames regardless of the destination address. The ENABLE value is represented by setting the 4th least significant bit (with weight 8) in the `EthRCR` field in the `Eth_CtrlCfgType` structure instance.
Class	Implementation Specific Parameter
Range	ENABLE, DISABLE
Default	DISABLE
Source File	Eth_PBcfg.c

Table continues on the next page...

Table 3-42. EthEnablePromiscuousMode (continued)

Source Representation	static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)
	EthConfigSet_EthCtrlConfig_0 =
	{
	    0x66554433U,
	    0x22008808U,
	    0x00000104U,
	    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,
	    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,
	AUTOMATIC))0U>>8U),
	    0x00000040U,
	    0x00000040U,
	    12U,
	#if STD_ON == ETH_DEM_EVENT_DETECT
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ACCESS },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_CRC },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ALIGNMENT },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_LATECOLLISION },
	#endif /* ETH_DEM_EVENT_DETECT */
	#if (STD_ON == ETH_GLOBALTIME_SUPPORT)
	    ((VAR(uint32, AUTOMATIC))1000000U),
	#endif
	    64U,
	    64U,
	    255U,
	    255U,
	};

Table 3-43. EthMIISpeedControl

Description	Controls the frequency of the MDC signal of MII. The value of 0 disables the MDC signal generation. The MDC frequency is equal to Fsys/(EthMIISpeedControl * 4), where Fsys is a frequency of the system the bus clock. Note that 802.3 specification states that the MDC maximum frequency is 2,5 MHz.
Class	Implementation Specific Parameter
Range	0..63 The minimal value is equal to Fsys / 10 MHz
Default	32
Source File	Eth_PBcfg.c

Source Representation	static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)
	EthConfigSet_EthCtrlConfig_0 =
	{
	    0x66554433U,
	    0x22008808U,
	    0x00000104U,
	    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,
	    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,
	AUTOMATIC))0U>>8U),

Table 3-43. EthMIISpeedControl

	0x00000040U,
	    0x00000040U,
	    12U,
	#if STD_ON == ETH_DEM_EVENT_DETECT
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ACCESS },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_CRC },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ALIGNMENT },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_LATECOLLISION },
	#endif /* ETH_DEM_EVENT_DETECT */
	#if (STD_ON == ETH_GLOBALTIME_SUPPORT)
	    ((VAR(uint32, AUTOMATIC))1000000U),
	#endif
	    64U,
	    64U,
	    255U,
	    255U,
	};

Table 3-44. EthDropInvalidMAC

Description	Enables or disables discarding of frames received with MAC layer error like invalid CRC, incorrect length, collision and also frames that are longer than configured receive buffer. Detailed description can be found in section Reception. Note: if this parameter is on, frame payload length is automatically checked with Frametype field to discard frame in case of inconsitency.
Class	Implementation Specific Parameter
Range	ENABLE, DISABLE
Default	ENABLE
Source File	Eth_PBcfg.c

Source Representation	static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)
	EthConfigSet_EthCtrlConfig_0 =
	{
	    0x66554433U,
	    0x22008808U,
	    0x00000104U,
	    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,
	    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,
	AUTOMATIC))0U>>8U),
	    0x00000040U,
	    0x00000040U,
	    12U,
	#if STD_ON == ETH_DEM_EVENT_DETECT
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ACCESS },
	    {(VAR(uint32, AUTOMATIC))STD_ON,

Table 3-44. EthDropInvalidMAC

<table>
<tr><td></td><td>DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_CRC },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_ALIGNMENT },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_LATECOLLISION },
#endif /* ETH_DEM_EVENT_DETECT */
#if (STD_ON == ETH_GLOBALTIME_SUPPORT)
    ((VAR(uint32, AUTOMATIC))1000000U),
#endif
    64U,
    64U,
    255U,
    255U,
};</td></tr>
</table>

Table 3-45. EthInterPacketGap

<table>
<tr><td><b>Description</b></td><td>Configures minimal delay between two transmissions (two frames). The delay corresponds to transmit time of EthInterPacketGap bytes. Note that according to IEEE 802.3, section 4.4.2.3, minimal inter packet gap is 12.</td></tr>
<tr><td><b>Class</b></td><td>Implementation Specific Parameter</td></tr>
<tr><td><b>Range</b></td><td>8..27</td></tr>
<tr><td><b>Default</b></td><td>12</td></tr>
<tr><td><b>Source File</b></td><td>Eth_PBcfg.c</td></tr>
<tr><td><b>Source Representation</b></td><td>static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)
EthConfigSet_EthCtrlConfig_0 =
{
    0x66554433U,
    0x22008808U,
    0x00000104U,
    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,
    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,
AUTOMATIC))0U>>8U),
    0x00000040U,
    0x00000040U,
    <b>12U</b>,
#if STD_ON == ETH_DEM_EVENT_DETECT
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_ACCESS },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_CRC },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },
    {(VAR(uint32, AUTOMATIC))STD_ON,
DemConf_DemEventParameter_ETH_E_ALIGNMENT },
    {(VAR(uint32, AUTOMATIC))STD_ON,</td></tr>
</table>

Table 3-45. EthInterPacketGap

	DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_LATECOLLISION },
	#endif /* ETH_DEM_EVENT_DETECT */
	#if (STD_ON == ETH_GLOBALTIME_SUPPORT)
	    ((VAR(uint32, AUTOMATIC))1000000U),
	#endif
	    64U,
	    64U,
	    255U,
	    255U,
	};

Table 3-46. EthMDIOHoldTime

Description	IEEE802.3 clause 22 defines a minimum of 10 ns for the holdtime on the MDIO output. Depending on the host bus frequency, the setting may need to be increased. The hold time is (EthMDIOHoldTime+1) internal module clock cycles.
Class	Implementation Specific Parameter
Range	0..7
Default	0
Source File	Eth_PBcfg.c

Source Representation	static CONST(Eth_CtrlCfgType, ETH_APPL_CONST)
	EthConfigSet_EthCtrlConfig_0 =
	{
	    0x66554433U,
	    0x22008808U,
	    0x00000104U,
	    (((VAR(uint32, AUTOMATIC))64U)>>16U) | 0x40004005U,
	    (((VAR(uint32, AUTOMATIC))32U)>>1U) | ((VAR(uint32,
	AUTOMATIC))0U>>8U),
	    0x00000040U,
	    0x00000040U,
	    12U,
	#if STD_ON == ETH_DEM_EVENT_DETECT
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ACCESS },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_RX_FRAMES_LOST },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_CRC },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_UNDERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_OVERSIZEFRAME },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_ALIGNMENT },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_SINGLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_MULTIPLECOLLISION },
	    {(VAR(uint32, AUTOMATIC))STD_ON,
	DemConf_DemEventParameter_ETH_E_LATECOLLISION },
	#endif /* ETH_DEM_EVENT_DETECT */
	#if (STD_ON == ETH_GLOBALTIME_SUPPORT)
	    ((VAR(uint32, AUTOMATIC))1000000U),
	#endif
	    64U,

Table 3-46. EthMDIOHoldTime

	`    64U,`
	`    255U,`
	`    255U,`
	`};`

3.6.2.3 Link-Time Configuration Parameters

The Link-Time configuration parameters are placed into the Eth_Lcfg.c file however they are generated only when the Link-Time configuration variant is selected otherwise they are part of the Eth_PCcfg.h or Eth_PBcfg.c as described in the sections: Pre-Compile Configuration Parameters or Post-Build Configuration Parameters.

Note

> Post-Build and Link-Time parameters source representation is the same, only their referencing in the driver differs.

The Link-Time parameters and source representation is the same as described in the section Post-Build Configuration Parameters however they are placed into the Eth_Lcfg.c file.

3.6.2.4 Constant Parameters

There are parameters with generated constant value which cannot be changed (it is hardwired in the generator) however it is still possible to change the generated files. These parameters are not intended to be changed by the user however they are described here to avoid any confusion. All constant parameters are part of the Eth_Cfg.h file in a form of macros.

Table 3-47. ETH_RESET_WAIT_LOOP_COUNT

Description	The Ethernet Controller shall not be accessed in approximately 8 bus cycles after it has been reset. A loop which iterates *ETH_RESET_WAIT_LOOP_COUNT* times is used to delay the program execution.
Class	Implementation Specific Parameter
Value	30
Source File	Eth_Cfg.h
Source Representation	`#define ETH_RESET_WAIT_LOOP_COUNT 30U`

Table 3-48. ETH_INFINITE_LOOP_PROTECTION

Description	Each driver loop which could be possibly infinite is protected by a counter which is incremented in each of the loop iterations. The loop is claimed to be infinite and broken when the counter reaches the *ETH_INFINITE_LOOP_PROTECTION* value. The minimal required value is equal to (64 * *EthMIISpeedControl*).
Class	Implementation Specific Parameter
Value	4096
Source File	Eth_Cfg.h
Source Representation	`#define ETH_INFINITE_LOOP_PROTECTION 4096U`

3.6.2.5 Buffers Memory

The ETH Driver utilizes so-called buffers for the Ethernet frame storage. There are two buffer types, which are placed into the separate memory areas. Receive buffers are used for a frame reception and transmit buffers are used for a frame transmission.

Each receive buffer is *EthCtrlRxBufLenByte* bytes long and consists of:

- 14 bytes for the received Ethernet frame header,
- n bytes for the received Ethernet frame payload,
- 4 bytes for the received Ethernet frame CRC,
- 0 to 63 bytes of the alignment pad,

The length of the buffer must be evenly divisible by 64. Note that the buffer provided by Eth_ProvideRxBuffer to application is the payload only, so its length is at most *EthCtrlRxBufLenByte* - (14+4). Additionally there is one 32 bytes long buffer descriptor for each buffer. Parameter *EthRxBufTotal* specifies the number of available receive buffers.

Each transmit buffer is *EthCtrlTxBufLenByte* bytes long and consists of:

- 14 bytes for the Ethernet frame header
- n bytes for the received Ethernet frame payload,
- 0 to 63 bytes of the alignment pad,

The length of the buffer must be evenly divisible by 64. Note that the buffer provided by Eth_ProvideTxBuffer to application is the payload only, so its length is at most *EthCtrlTxBufLenByte* - 14. Additionally there is one 32 bytes long buffer descriptor for each buffer. Parameter *EthTxBufTotal* specifies the number of available transmit buffers.

3.6.2.6 Loopback modes

The loopback modes are modes where the receiver input is connected to the transmitter output creating a loop where all the transmitted data are routed into the receiver. The ETH Driver can be configured to two loopback modes - the internal and the external loopback mode.

The *EthEnableLoopbackMode* set to ENABLE ensures that the receiver is operational during the transmission which is requested by both loopback modes. Note that, in loopback mode, the receiver is active during transmission even if the half duplex mode is configured.

Setting the *EthInternalLoopbackMode* to ENABLE configures the Ethernet Controller to connect the transmitter output to the receiver input internally, without need of any external hardware. The Ethernet Controller also stops driving output pins. This is the internal loopback mode.

The external loopback mode requires setting of the *EthInternalLoopbackMode* to DISABLE and connecting the receiver input to the transmitter output by an external hardware e.g. the Ethernet transceiver configured to the loopback mode.

3.7 Runtime Errors

The Ethernet driver generates the following DEM errors at runtime:

Table 3-49. Compiled Configuration Files

Function	Error code	Condition triggering the error
Eth_MainFunction	ETH_E_ACCESS	Ethernet Controller Access Failure
Eth_MainFunction	ETH_E_RX_FRAMES_LOST	Lost frames are detected
Eth_MainFunction	ETH_E_CRC	Invalid CRC frames are detected
Eth_MainFunction	ETH_E_UNDERSIZEFRAME	Frames shorter than accepted are detected
Eth_MainFunction	ETH_E_OVERSIZEFRAME	Oversize frames are detected
Eth_MainFunction	ETH_E_ALIGNMENT	Invalid alignment frames are detected
Eth_MainFunction	ETH_E_SINGLECOLLISION	Frames with single collision are detected
Eth_MainFunction	ETH_E_MULTIPLECOLLISION	Frame with multiple collisions are detected
Eth_MainFunction	ETH_E_LATECOLLISION	Frames with late collisions are detected

3.8 Software specification

The following sections contains driver software specifications.

3.8.1 Define Reference

Constants supported by the driver are as per AUTOSAR ETH Driver software specification Version 4.2 Rev0002 .

3.8.1.1 ETH Counters

The ETH Driver provides access to the Ethernet Controller counter registers by the `Eth_GetCounterState` function. The counter is selected by passing one of the macros predefined in **Eth_Counters.h** as the `CtrOffs` function argument. Detailed description of the available counters can be found in the S32K14X Microcontroller Reference Manual [Reference List].

All counters are reset to 0 by the `Eth_ControllerInit` function call.

3.8.1.1.1 Define ENET_IEEE_R_ALIGN_ADDR16

Frames Received with Alignment Error.

Definition:`#define ENET_IEEE_R_ALIGN_ADDR16 0x02D4U`

3.8.1.1.2 Define ENET_IEEE_R_CRC_ADDR16

Frames Received with CRC Error.

Definition:`#define ENET_IEEE_R_CRC_ADDR16 0x02D0U`

3.8.1.1.3 Define ENET_IEEE_R_DROP_ADDR16

Count of frames not counted correctly.

Definition:`#define ENET_IEEE_R_DROP_ADDR16 0x02C8U`

3.8.1.1.4 Define ENET_IEEE_R_FDXFC_ADDR16

Flow Control Pause frames received.

Definition:`#define ENET_IEEE_R_FDXFC_ADDR16 0x02DCU`

3.8.1.1.5 Define ENET_IEEE_R_FRAME_OK_ADDR16

Frames Received OK.

Definition:`#define ENET_IEEE_R_FRAME_OK_ADDR16 0x02CCU`

3.8.1.1.6 Define ENET_IEEE_R_MACERR_ADDR16

Receive Fifo Overflow count.

Definition:`#define ENET_IEEE_R_MACERR_ADDR16 0x02D8U`

3.8.1.1.7 Define ENET_IEEE_R_OCTETS_OK_ADDR16

Octet count for Frames received without an error.

Definition:`#define ENET_IEEE_R_OCTETS_OK_ADDR16 0x02E0U`

3.8.1.1.8 Define ENET_IEEE_T_1COL_ADDR16

Frames Transmitted with Single Collision.

Definition:`#define ENET_IEEE_T_1COL_ADDR16 0x0250U`

3.8.1.1.9 Define ENET_IEEE_T_CSERR_ADDR16

Carrier Sense Errors count.

Definition:`#define ENET_IEEE_T_CSERR_ADDR16 0x0268U`

3.8.1.1.10 Define ENET_IEEE_T_DEF_ADDR16

Frames Transmitted after Deferral Delay.

Definition:`#define ENET_IEEE_T_DEF_ADDR16 0x0258U`

3.8.1.1.11 Define ENET_IEEE_T_DROP_ADDR16

Count of frames not counted correctly.

Definition:`#define ENET_IEEE_T_DROP_ADDR16 0x0248U`

3.8.1.1.12 Define ENET_IEEE_T_EXCOL_ADDR16

Frames Transmitted with Excessive Collisions.

Definition:`#define ENET_IEEE_T_EXCOL_ADDR16 0x0260U`

3.8.1.1.13 Define ENET_IEEE_T_FDXFC_ADDR16

Flow Control Pause frames transmitted count.

Definition:`#define ENET_IEEE_T_FDXFC_ADDR16 0x0270U`

3.8.1.1.14 Define ENET_IEEE_T_FRAME_OK_ADDR16

Frames Transmitted OK.

Definition:`#define ENET_IEEE_T_FRAME_OK_ADDR16 0x024CU`

3.8.1.1.15 Define ENET_IEEE_T_LCOL_ADDR16

Frames Transmitted with Late Collision.

Definition:`#define ENET_IEEE_T_LCOL_ADDR16 0x025CU`

3.8.1.1.16 Define ENET_IEEE_T_MACERR_ADDR16

Frames Transmitted with Tx FIFO Underrun.

Definition:`#define ENET_IEEE_T_MACERR_ADDR16 0x0264U`

3.8.1.1.17 Define ENET_IEEE_T_MCOL_ADDR16

Frames Transmitted with Multiple Collisions.

Definition:`#define ENET_IEEE_T_MCOL_ADDR16 0x0254U`

3.8.1.1.18 Define ENET_IEEE_T_OCTETS_OK_ADDR16

Octet count for Frames Transmitted w/o Error.

Definition:`#define ENET_IEEE_T_OCTETS_OK_ADDR16 0x0274U`

3.8.1.1.19 Define ENET_IEEE_T_SQE_ADDR16

SQE_TEST_ERROR receptions count.

Definition:`#define ENET_IEEE_T_SQE_ADDR16 0x026CU`

3.8.1.1.20 Define ENET_RMON_R_BC_PKT_ADDR16

Received Broadcast Packets count.

Definition:`#define ENET_RMON_R_BC_PKT_ADDR16 0x0288U`

3.8.1.1.21 Define ENET_RMON_R_CRC_ALIGN_ADDR16

Received Packets with CRC or Align error count.

Definition:`#define ENET_RMON_R_CRC_ALIGN_ADDR16 0x0290U`

3.8.1.1.22 Define ENET_RMON_R_FRAG_ADDR16

Received Packets < 64 bytes, bad CRC.

Definition:`#define ENET_RMON_R_FRAG_ADDR16 0x029CU`

3.8.1.1.23 Define ENET_RMON_R_JAB_ADDR16

Received Packets > MAX_FL bytes, bad CRC.

Definition:`#define ENET_RMON_R_JAB_ADDR16 0x02A0U`

3.8.1.1.24 Define ENET_RMON_R_MC_PKT_ADDR16

Received Multicast Packets count.

Definition:`#define ENET_RMON_R_MC_PKT_ADDR16 0x028CU`

3.8.1.1.25 Define ENET_RMON_R_OCTETS_ADDR16

Received octets count.

Definition:`#define ENET_RMON_R_OCTETS_ADDR16 0x02C4U`

3.8.1.1.26 Define ENET_RMON_R_OVERSIZE_ADDR16

Received Packets > MAX_FL bytes, good CRC.

Definition:`#define ENET_RMON_R_OVERSIZE_ADDR16 0x0298U`

3.8.1.1.27 Define ENET_RMON_R_P_GTE2048_ADDR16

Received packets with length greater than 2047 byte count.

Definition:`#define ENET_RMON_R_P_GTE2048_ADDR16 0x02C0U`

3.8.1.1.28 Define ENET_RMON_R_P1024TO2047_ADDR16

Received packets with length 1024 to 2047 byte count.

Definition:`#define ENET_RMON_R_P1024TO2047_ADDR16 0x02BCU`

3.8.1.1.29 Define ENET_RMON_R_P128TO255_ADDR16

Received packets with length 128 to 255 byte count.

Definition:`#define ENET_RMON_R_P128TO255_ADDR16 0x02B0U`

3.8.1.1.30 Define ENET_RMON_R_P256TO511_ADDR16

Received packets with length 256 to 511 byte count.

Definition:`#define ENET_RMON_R_P256TO511_ADDR16 0x02B4U`

3.8.1.1.31 Define ENET_RMON_R_P512TO1023_ADDR16

Received packets with length 512 to 1023 byte count.

Definition:`#define ENET_RMON_R_P512TO1023_ADDR16 0x02B8U`

3.8.1.1.32 Define ENET_RMON_R_P64_ADDR16

Received packets with length equal to 64 byte count.

Definition:`#define ENET_RMON_R_P64_ADDR16 0x02A8U`

3.8.1.1.33 Define ENET_RMON_R_P65TO127_ADDR16

Received packets with length 65 to 127 byte count.

Definition:`#define ENET_RMON_R_P65TO127_ADDR16 0x02ACU`

3.8.1.1.34 Define ENET_RMON_R_PACKETS_ADDR16

received Packets count

Definition:`#define ENET_RMON_R_PACKETS_ADDR16 0x0284U`

3.8.1.1.35 Define ENET_RMON_R_UNDERSIZE_ADDR16

Received Packets < 64 bytes, good CRC.

Definition:`#define ENET_RMON_R_UNDERSIZE_ADDR16 0x0294U`

3.8.1.1.36 Define ENET_RMON_T_BC_PKT_ADDR16

Transmitted Broadcast Packets count.

Definition:`#define ENET_RMON_T_BC_PKT_ADDR16 0x0208U`

3.8.1.1.37 Define ENET_RMON_T_COL_ADDR16

Transmission collisions count.

Definition:`#define ENET_RMON_T_COL_ADDR16 0x0224U`

3.8.1.1.38 Define ENET_RMON_T_CRC_ALIGN_ADDR16

Transmitted Packets with CRC or Align error count.

Definition:`#define ENET_RMON_T_CRC_ALIGN_ADDR16 0x0210U`

3.8.1.1.39 Define ENET_RMON_T_DROP_ADDR16

Count of frames not counted correctly.

Definition:`#define ENET_RMON_T_DROP_ADDR16 0x0200U`

3.8.1.1.40 Define ENET_RMON_T_FRAG_ADDR16

Transmitted Packets < 64 bytes, bad CRC.

Definition:`#define ENET_RMON_T_FRAG_ADDR16 0x021CU`

3.8.1.1.41 Define ENET_RMON_T_JAB_ADDR16

Transmitted Packets > MAX_FL bytes, bad CRC.

Definition:`#define ENET_RMON_T_JAB_ADDR16 0x0220U`

3.8.1.1.42 Define ENET_RMON_T_MC_PKT_ADDR16

Transmitted Multicast Packets count.

Definition:`#define ENET_RMON_T_MC_PKT_ADDR16 0x020CU`

3.8.1.1.43 Define ENET_RMON_T_OCTETS_ADDR16

Transmitted octets count.

Definition:`#define ENET_RMON_T_OCTETS_ADDR16 0x0244U`

3.8.1.1.44 Define ENET_RMON_T_OVERSIZE_ADDR16

Transmitted Packets > MAX_FL bytes, good CRC.

Definition:`#define ENET_RMON_T_OVERSIZE_ADDR16 0x0218U`

3.8.1.1.45 Define ENET_RMON_T_P_GTE2048_ADDR16

Transmitted packets with length greater than 2047 byte count.

Definition:`#define ENET_RMON_T_P_GTE2048_ADDR16 0x0240U`

3.8.1.1.46 Define ENET_RMON_T_P1024TO2047_ADDR16

Transmitted packets with length 1024 to 2047 byte count.

Definition:`#define ENET_RMON_T_P1024TO2047_ADDR16 0x023CU`

3.8.1.1.47 Define ENET_RMON_T_P128TO255_ADDR16

Transmitted packets with length 128 to 255 byte count.

Definition:`#define ENET_RMON_T_P128TO255_ADDR16 0x0230U`

3.8.1.1.48 Define ENET_RMON_T_P256TO511_ADDR16

Transmitted packets with length 256 to 511 byte count.

Definition:`#define ENET_RMON_T_P256TO511_ADDR16 0x0234U`

3.8.1.1.49 Define ENET_RMON_T_P512TO1023_ADDR16

Transmitted packets with length 512 to 1023 byte count.

Definition:`#define ENET_RMON_T_P512TO1023_ADDR16 0x0238U`

3.8.1.1.50 Define ENET_RMON_T_P64_ADDR16

Transmitted packets with length equal to 64 byte count.

Definition:`#define ENET_RMON_T_P64_ADDR16 0x0228U`

3.8.1.1.51 Define ENET_RMON_T_P65TO127_ADDR16

Transmitted packets with length 65 to 127 byte count.

Definition:`#define ENET_RMON_T_P65TO127_ADDR16 0x022CU`

3.8.1.1.52 Define ENET_RMON_T_PACKETS_ADDR16

Transmitted Packets count.

Definition:`#define ENET_RMON_T_PACKETS_ADDR16 0x0204U`

3.8.1.1.53 Define ENET_RMON_T_UNDERSIZE_ADDR16

Transmitted Packets < 64 bytes, good CRC.

Definition:`#define ENET_RMON_T_UNDERSIZE_ADDR16 0x0214U`

3.8.2 Enum Reference

Enumeration of all constants supported by the driver are as per AUTOSAR ETH Driver software specification Version 4.2 Rev0002 .

3.8.3 Function Reference

Functions of all functions supported by the driver are as per AUTOSAR ETH Driver software specification Version 4.2 Rev0002 .

3.8.3.1 API Reference

The API description of all functions supported by the ETH Driver can be found in the AUTOSAR 4.2 Rev002 ETH Driver Software Specification Document[1]. There are 13 API functions and two interrupt handlers defined by the specification, all of them are implemented by the ETH Driver:

1. Eth_Init

2. Eth_SetControllerMode
3. Eth_GetControllerMode
4. Eth_GetPhysAddr
5. Eth_SetPhysAddr
6. Eth_UpdatePhysAddrFilter
7. Eth_WriteMii
8. Eth_ReadMii
9. Eth_GetDropCount
10. Eth_GetEtherStats
11. Eth_MainFunction
12. Eth_SetGlobalTime
13. Eth_SetCorrectionTime
14. Eth_GetIngressTimeStamp
15. Eth_GetEgressTimeStamp
16. Eth_EnableEgressTimeStamp
17. Eth_GetCurrentTime
18. Eth_ProvideTxBuffer
19. Eth_Transmit
20. Eth_Receive
21. Eth_TxConfirmation
22. Eth_GetVersionInfo

- Eth_RxIrqHdlr_0
- Eth_TxIrqHdlr_0

3.8.3.1.1 Function Eth_Init

Initializes the Ethernet Driver.

Prototype: `void Eth_Init(const Eth_ConfigType *CfgPtr);`

Table 3-50. Eth_Init Arguments

Type	Name	Direction	Description
const Eth_ConfigType *	CfgPtr	**input**	Points to the implementation specific structure containing the Eth driver configuration Compiler_Warning: this warning due to behavior of compiler depend on configs.

Passed configuration pointer is internally stored and the driver is initialized.

Note

Function should be called only once.

CAUTION

Second call can cause undefined behavior. Call the `Eth_SetControllerMode()` and pass ETH_MODE_DOWN to the CtrlMode argument before the second Eth_Init call to avoid problems.

3.8.3.1.2 Function Eth_SetControllerMode

Enables or disables the given controller.

Prototype: `Std_ReturnType Eth_SetControllerMode(uint8 CtrlIdx, Eth_ModeType CtrlMode);`

Table 3-51. Eth_SetControllerMode Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller to be enabled or disabled. The index is valid within the context of the Ethernet Driver only.
Eth_ModeType	CtrlMode	**input**	Mode which shall be entered • ETH_MODE_DOWN: disable the controller • ETH_MODE_ACTIVE: enable the controller

Return: Error status

Table 3-52. Eth_SetControllerMode Returns

Value	Description
E_OK	No error was detected during the function execution.
E_NOT_OK	Development error was detected and the function failed.

CAUTION

Disabling the controller clears all receive and transmit buffers. The application should ensure that no data is lost.

3.8.3.1.3 Function Eth_GetControllerMode

Obtains the mode of the given controller.

Prototype: `Std_ReturnType Eth_GetControllerMode(uint8 CtrlIdx, Eth_ModeType *CtrlModePtr);`

Table 3-53. Eth_GetControllerMode Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which state shall be read. The index is valid within the context of the Ethernet Driver only.
Eth_ModeType *	CtrlModePtr	**output**	Pointer where to store the current controller mode.

Return: Error status

Table 3-54. Eth_GetControllerMode Returns

Value	Description
E_OK	No error was detected during the function execution.
E_NOT_OK	Development error was detected and the function failed.

3.8.3.1.4 Function Eth_GetPhysAddr

Obtains the physical source address used by the indexed controller (the node MAC address).

Prototype: `void Eth_GetPhysAddr(uint8 CtrlIdx, uint8 *PhysAddrPtr);`

Table 3-55. Eth_GetPhysAddr Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which MAC address should be read. The index is valid within the context of the Ethernet Driver only.
uint8 *	PhysAddrPtr	**output**	Pointer where to store physical source address (MAC address). The address in network byte order is stored into 6 bytes at the given memory address.

3.8.3.1.5 Function Eth_SetPhysAddr

Set or change physical address to the defined controller.

Prototype: `void Eth_SetPhysAddr(uint8 CtrlIdx, const uint8 *PhysAddrPtr);`

Table 3-56. Eth_SetPhysAddr Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which PHY address should be changed. The index is valid within the context of the Ethernet Driver only.
const uint8 *	PhysAddrPtr	**input**	Pointer to PHY address which should be set to the controller. The address is stored in 6 bytes of memory in network byte order. This function may be called only when the controller is down. Call of function Eth_ControllerInit change MAC address to the default value!

3.8.3.1.6 Function Eth_UpdatePhysAddrFilter

Adds or removes the specific PhysAddrPtr address to or from a multicast address pool at controller specified by CtrlIdx index.

Prototype: `Std_ReturnType Eth_UpdatePhysAddrFilter(uint8 CtrlIdx, const uint8 *PhysAddrPtr, Eth_FilterActionType Action);`

Table 3-57. Eth_UpdatePhysAddrFilter Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller. The index is valid within the context of the Ethernet Driver only.
const uint8 *	PhysAddrPtr	**input**	Pointer to PHY address which shall be added or removed to or from multicast pool. The address in network byte order stored into 6 bytes of memory.
Eth_FilterActionType	Action	**input**	Determine whenever the defined address will be added to the pool ETH_ADD_TO_FILTER or removed from it ETH_REMOVE_FROM_FILTER.

Enables or disables reception for specified unicast physical address. Operations for special Physical addresses follow. If Physical Address ff:ff:ff:ff:ff:ff is added into a filter (Action=ETH_ADD_TO_FILTER) the filter is completely open and any address is accepted at reception. Later on when Physical Address ff:ff:ff:ff:ff:ff is removed from the filter (Action=ETH_REMOVE_FROM_FILTER) the filtering is recovered and the reception is allowed again only for addresses remaining in the filter. If Physical Address 00:00:00:00:00:00 is added into a filter, no matter whether action is ETH_ADD_TO_FILTER or ETH_REMOVE_FROM_FILTER, the filter is completely closed and all items from table are removed. Note that operations of full open or close are in exclusive disjunction. Operation of full open excludes full close and vice versa.

3.8.3.1.7 Function Eth_WriteMii

Writes to a transceiver (physical layer driver) register.

Prototype: `Std_ReturnType Eth_WriteMii(uint8 CtrlIdx, uint8 TrcvIdx, uint8 RegIdx, uint16 RegVal);`

Table 3-58. Eth_WriteMii Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which transceiver register shall be written. The index is valid within the context of the Ethernet Driver only.
uint8	TrcvIdx	**input**	Index of the transceiver connected the MII. The value shall be within the range 0..31.

Table continues on the next page...

Table 3-58. Eth_WriteMii Arguments (continued)

Type	Name	Direction	Description
uint8	RegIdx	**input**	Index of the transceiver register to be written. The value shall be withing the range 0..31.
uint16	RegVal	**input**	Value to be written into the indexed register.

Table 3-59. Eth_WriteMii Returns

Value	Description
E_OK	No error was detected during the function execution.
E_NOT_OK	Development error or the function failed.
ETH_E_NO_ACCESS	Inaccessible to tranceiver.

The management frame is assembled and the MII bus transaction is issued in order to transfer the data. Function waits until the bus transaction finishes.

CAUTION

This function is blocking the execution until the MII bus transaction is finished.

3.8.3.1.8 Function Eth_ReadMii

Reads a transceiver (physical layer driver) register.

Prototype: `Std_ReturnType Eth_ReadMii(uint8 CtrlIdx, uint8 TrcvIdx, uint8 RegIdx, uint16 *RegValPtr);`

Table 3-60. Eth_ReadMii Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which transceiver register shall be read. The index is valid within the context of the Ethernet Driver only.
uint8	TrcvIdx	**input**	Index of the transceiver connected on the MII. The value shall be within the range 0..31.
uint8	RegIdx	**input**	Index of the transceiver register to be read. The Value shall be within the range 0..31.
uint16 *	RegValPtr	**output**	Filled with the register content of the indexed register

Table 3-61. Eth_ReadMii Returns

Value	Description
E_OK	No error was detected during the function execution.

Table continues on the next page...

Table 3-61. Eth_ReadMii Returns (continued)

Value	Description
E_NOT_OK	Development error or the function failed.
ETH_E_NO_ACCESS	Inaccessible to tranceiver.

The management frame is assembled and the MII bus transaction is issued in order to transfer the data. Function waits until the bus transaction finishes and then returns the read data.

CAUTION

This function is blocking the execution until the MII bus transaction is finished.

3.8.3.1.9 Function Eth_GetDropCount

Reads a list with drop counter values of the corresponding controller.

Prototype: `Std_ReturnType Eth_GetDropCount(uint8 CtrlIdx, uint8 *CountValuesPtr, uint32 *DropCountPtr);`

Table 3-62. Eth_GetDropCount Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which shall be be read the drop package counts.
Commented parameter CountValues does not exist in function Eth_GetDropCount.	CountValues	**input, output**	The number of values which return. -In: Maximal number of values which can be written from DropCount. -Out: Number of values which are returned in the DropCount list.
Commented parameter DropCount does not exist in function Eth_GetDropCount.	DropCount	**output**	The interpretation of this list of values is hardware dependent

Return: Error status

Table 3-63. Eth_GetDropCount Returns

Value	Description
E_OK	No error was detected during the function execution.
E_NOT_OK	Development error was detected or inaccessible to counters register and the function.

Reads a list with drop counter values of the corresponding controller. The meaning of these values is hardware dependent. However, the list DropCount[] shall contain the following values in the given order, where the maximal possible value shall denote an invalid value, e.g. if this counter is not available: 1.) dropped packets due to buffer overrun 2.) dropped packets due to CRC errors 3.) number of undersize packets which were less than 64 octets long (excluding framing bits, but including FCS octets) and were otherwise will formed. (see IETF RFC 1757) 4.) number of oversize packets which are longer than 1518 octets (excluding framing bits, but including FCS octets) and were otherwise well formed. (see IETF RFC 1757) 5.) number of alignment errors, i.e. packets which are received and are not an integral number of octets in length and do not pass the CRC. 6.) SQE test error according to IETF RFC1643 dot3StatsSQETestErrors 7.) The number of inbound packets which were chosen to be discarded even though no errors had been detected to prevent their being deliverable to a higher layer protocol. One possible reason for discarding such a packet could be to free up buffer space. (see IETF RFC 2233 ifInDiscards) 8.) total number of erroneous inbound packets 9.) The number of outbound packets which were chosen to be discarded even though no errors had been detected to prevent their being transmitted. One possible reason for discarding such a packet could be to free up buffer space. (see IETF RFC 2233 ifOutDiscards) 10.) total number of erroneous outbound packets 11.) Single collision frames: A count of successfully transmitted frames on a particular interface for which transmission is inhibited by exactly one collision. (see IETF RFC1643 dot3StatsSingleCollisionFrames) 12.) Multiple collision frames: A count of successfully transmitted frames on a particular interface for which transmission is inhibited by more than one collision. (see IETF RFC1643 dot3StatsMultipleCollisionFrames) 13.) Number of deferred transmission: A count of frames for which the first transmission attempt on a particular interface is delayed because the medium is busy. (see IETF RFC1643 dot3StatsDeferredTransmissions) 14.) Number of late collisions: The number of times that a collision is detected on a particular interface later than 512 bit times into the transmission of a packet. (see IETF RFC1643 dot3StatsLateCollisions) 15.) the following positions in the list can contain hardware dependent counter values

3.8.3.1.10 Function Eth_GetEtherStats

Read the status of a controller Returns the following list according to IETF RFC2819, where the maximal possible value shall denote an invalid value, e.g. if this counter is not available:

Prototype: `Std_ReturnType Eth_GetEtherStats(uint8 CtrlIdx, uint32 *etherStats);`

Table 3-64. Eth_GetEtherStats Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	Input	Index of the controller which shall be read the status register.
uint32 *	etherStats	Output	Pointer to 32 bit long memory space to be filled with the list values according to IETF RFC 2819 (Remote Network Monitoring Management Information Base).

Return: Error status

Table 3-65. Eth_GetEtherStats Returns

Value	Description
E_OK	No error was detected during the function execution.
E_NOT_OK	Development error was detected or inaccessible to counters register and the function.

Following are fields with corresponding index for EtherStats in the pointers:

1. etherStatsDropEvents
2. etherStatsOctets
3. etherStatsPkts
4. etherStatsBroadcastPkts
5. etherStatsMulticastPkts
6. etherStatsCrcAlignErrors
7. etherStatsUndersizePkts
8. etherStatsOversizePkts
9. etherStatsFragments
10. etherStatsJabbers
11. etherStatsCollisions
12. etherStatsPkts64Octets
13. etherStatsPkts65to127Octets
14. etherStatsPkts128to255Octets
15. etherStatsPkts256to511Octets
16. etherStatsPkts512to1023Octets
17. etherStatsPkts1024to1518Octets

3.8.3.1.11 Function Eth_MainFunction

The function checks for controller errors and lost frames. Used for polling state changes. Calls EthIf_CtrlModeIndication when the controller mode changed.

Prototype: `void Eth_MainFunction(void);`

3.8.3.1.12 Function Eth_SetGlobalTime

Allows the Time Master to adjust the global ETH Reference clock in HW.

Prototype: `Std_ReturnType Eth_SetGlobalTime(uint8 CtrlIdx, Eth_TimeStampType *timeStampPtr);`

Table 3-66. Eth_SetGlobalTime Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the TM controller which time shall be adjusted.
const Eth_TimeStampType *	timeStampPtr	**input**	Pointer to new time stamp

Note

We can use this method to set a global time base on ETH in general or to synchronize the global ETH time base with another time base, e.g. FlexRay.

3.8.3.1.13 Function Eth_SetCorrectionTime

Allows the Time Slave to adjust the local ETH Reference clock in HW.

Prototype: `void Eth_SetCorrectionTime(uint8 CtrlIdx, Eth_TimeIntDiffType *timeOffsetPtr, Eth_RateRatioType *rateRatioPtr);`

Table 3-67. Eth_SetCorrectionTime Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which time shall be corrected
const Eth_TimeIntDiffType *	timeOffsetPtr	**input**	offset between time stamp grandmaster and time stamp by local clock.
const Eth_RateRatioType *	rateRatioPtr	**input**	time elements to calculate and to modify the ratio of the frequency of the grandmaster in relation to the frequency of the Local Clock

Note

Only use this function when this controller used as Time Slave.

3.8.3.1.14 Function Eth_GetIngressTimeStamp

Reads back the egress time stamp on a dedicated message object.

Prototype: `void Eth_GetIngressTimeStamp(uint8 CtrlIdx, Eth_DataType *DataPtr, Eth_TimeStampQualType *timeQualPtr, Eth_TimeStampType *timeStampPtr);`

Table 3-68. Eth_GetIngressTimeStamp Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which the egress timestamp shall be read.
Eth_DataType *	DataPtr	**input**	Pointer to the message buffer, where Application expects ingress time stamping
Eth_TimeStampQualType *	timeQualPtr	**output**	quality of HW time stamp, e.g. based on current drift
Eth_TimeStampType *	timeStampPtr	**output**	current time stamp

Note

It must be called within the TxConfirmation() function.

3.8.3.1.15 Function Eth_GetEgressTimeStamp

Reads back the egress time stamp on a dedicated message object.

Prototype: `void Eth_GetEgressTimeStamp(uint8 CtrlIdx, uint8 BufIdx, Eth_TimeStampQualType *timeQualPtr, Eth_TimeStampType *timeStampPtr);`

Table 3-69. Eth_GetEgressTimeStamp Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which the egress timestamp shall be read.
uint8	BufIdx	**input**	Index of the message buffer, where Application expects egress time stamping
Eth_TimeStampQualType *	timeQualPtr	**output**	quality of HW time stamp, e.g. based on current drift
Eth_TimeStampType *	timeStampPtr	**output**	current time stamp

Note

It must be called within the TxConfirmation() function.

3.8.3.1.16 Function Eth_EnableEgressTimeStamp

Activates egress time stamping on a dedicated message object.

Prototype: `void Eth_EnableEgressTimeStamp(uint8 CtrlIdx, uint8 BufIdx);`

Table 3-70. Eth_EnableEgressTimeStamp Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which counter state shall be enable the TimeStamp
uint8	BufIdx	**input**	Index of the message buffer, where Application expects egress time stamping

Note

Some HW does store once the egress time stamp marker and some HW needs it always before transmission. There will be no disable functionality, due to the fact, that the message type is always "time stamped" by network design.

3.8.3.1.17 Function Eth_GetCurrentTime

Returns a time value out of the HW registers according to the capability of the HW.

Prototype: `Std_ReturnType Eth_GetCurrentTime(uint8 CtrlIdx, Eth_TimeStampQualType *timeQualPtr, Eth_TimeStampType *timeStampPtr);`

Table 3-71. Eth_GetCurrentTime Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller shall be read the time value.
Eth_TimeStampQualType *	timeQualPtr	**output**	quality of HW time stamp, e.g. based on current drift
Eth_TimeStampType *	timeStampPtr	**output**	current time stamp

Return: Error status

Table 3-72. Eth_GetCurrentTime Returns

Value	Description
+E_OK	successfully read the timestamp
+E_NOT_OK	development error was detected or fail to read the TimeStamp.

Note

Is the HW resolution is lower than the Eth_TimeStampType resolution resp. range, than an the remaining bits will be filled with 0.

3.8.3.1.18 Function Eth_ProvideTxBuffer

Provides access to a transmit buffer of the specified controller.

Prototype: `BufReq_ReturnType Eth_ProvideTxBuffer(uint8 CtrlIdx, Eth_BufIdxType *BufIdxPtr, uint8 **BufPtr, uint16 *LenBytePtr);`

Table 3-73. Eth_ProvideTxBuffer Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which buffer shall be provided. The index is valid within the context of the Ethernet Driver only.
Eth_BufIdxType *	BufIdxPtr	**output**	Index to the granted transmit buffer resource. It uniquely identifies the buffer in all subsequent calls of functions `Eth_Transmit()` and `Eth_TxConfirmation()`.
uint8 **	BufPtr	**output**	Pointer to the granted buffer. This is the space where the data to be transmitted shall be stored.
uint16 *	LenBytePtr	**input,** **output**	Buffer payload length • In: desired length in bytes • Out: granted length in bytes

Return: Error and buffer status

Table 3-74. Eth_ProvideTxBuffer Returns

Value	Description
BUFREQ_OK	Buffer was successfully granted and no error has occurred.
BUFREQ_E_NOT_OK	A development error was detected and no buffer was granted.
BUFREQ_E_BUSY	All available buffers in use therefore no buffer was granted. No error has been detected.

CAUTION

The application should handle possible difference between the requested and granted buffer lengths. It is not necessary to use whole granted buffer i.e. some space at the end may not be written.

3.8.3.1.19 Function Eth_Transmit

Triggers transmission of a previously granted and then filled transmit buffer.

Prototype: `Std_ReturnType Eth_Transmit(uint8 CtrlIdx, Eth_BufIdxType BufIdx, Eth_FrameType FrameType, boolean TxConfirmation, uint16 LenByte, uint8 *PhysAddrPtr);`

Table 3-75. Eth_Transmit Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which buffer shall be transmitted. The index is valid within the context of the Ethernet Driver only.
Eth_BufIdxType	BufIdx	**input**	Index of the buffer resource to be transmitted.
Eth_FrameType	FrameType	**input**	Desired value of the Ethernet frame type in the frame header.
boolean	TxConfirmation	**input**	Activates transmission confirmation.
uint16	LenByte	**input**	Buffer data length in bytes (payload length).
const uint8 *	PhysAddrPtr	**input**	Physical target address (MAC address) in network byte order.

Return: Error status

Table 3-76. Eth_Transmit Returns

Value	Description
E_OK	No error was detected during the function execution.
E_NOT_OK	Development error was detected and the function failed.

3.8.3.1.20 Function Eth_Receive

Triggers frames reception notifications.

Prototype: `void Eth_Receive(uint8 CtrlIdx, Eth_RxStatusType *RxStatusPtr);`

Table 3-77. Eth_Receive Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which shall be checked whether any new frames were received. The index is valid within the context of the Ethernet Driver only.
Eth_RxStatusType *	RxStatusPtr	**output**	Informs the caller whether a frame was received (ETH_RECEIVED or ETH_NOT_RECEIVED) and whether more frames are available in the queue (ETH_RECEIVED or ETH_RECEIVED_MORE_DATA_AVAILABLE).

All receive buffers are checked and the first received frame is passed to the EthIf module. The caller is notified whether any frame was received and whether more frames are available in the receive queue.

3.8.3.1.21 Function Eth_TxConfirmation

Triggers frame transmission confirmations.

Prototype: `void Eth_TxConfirmation(uint8 CtrlIdx);`

Table 3-78. Eth_TxConfirmation Arguments

Type	Name	Direction	Description
uint8	CtrlIdx	**input**	Index of the controller which shall be checked whether any frame transmission has finished. The index is valid within the context of the Ethernet Driver only.

All transmit buffers are checked and upper layers are informed about successfully transmitted frames. Buffers containing transmitted frames are unlocked after the confirmation.

3.8.3.1.22 Function Eth_GetVersionInfo

Returns the version information of this module.

Prototype: `void Eth_GetVersionInfo(Std_VersionInfoType *VersionInfoPtr);`

Table 3-79. Eth_GetVersionInfo Arguments

Type	Name	Direction	Description
Std_VersionInfoType *	VersionInfoPtr	**output**	Pointer where to store the version information of this particular module instance.

3.8.3.1.23 Function Eth_RxIrqHdlr_0

Reception interrupt handler for the controller 0.

Prototype: `void Eth_RxIrqHdlr_0(void);`

All receive buffers are checked and upper layers are notified about received frames. Received data are passed to the upper layers.

3.8.3.1.24 Function Eth_TxIrqHdlr_0

Transmission interrupt handler for the controller 0.

Prototype: `void Eth_TxIrqHdlr_0(void);`

All transmit buffers are checked and upper layers are notified about successfully transmitted frames. Buffers containing transmitted frames are unlocked after the confirmation.

3.8.4 Structs Reference

Data structures supported by the driver are as per AUTOSAR ETH Driver software specification Version 4.2 Rev0002 .

3.8.5 Types Reference

Types supported by the driver are as per AUTOSAR ETH Driver software specification Version 4.2 Rev0002 .

3.9 Symbolic Names Disclaimer

All containers having the symbolic name tag set as true in the Autosar schema will generate defines like:

#define <Container_Short_Name> <Container_ID>

For this reason it is forbidden to duplicate the name of such containers across the MCAL configuration, or to use names that may trigger other compile issues (e.g. match existing #ifdefs arguments).

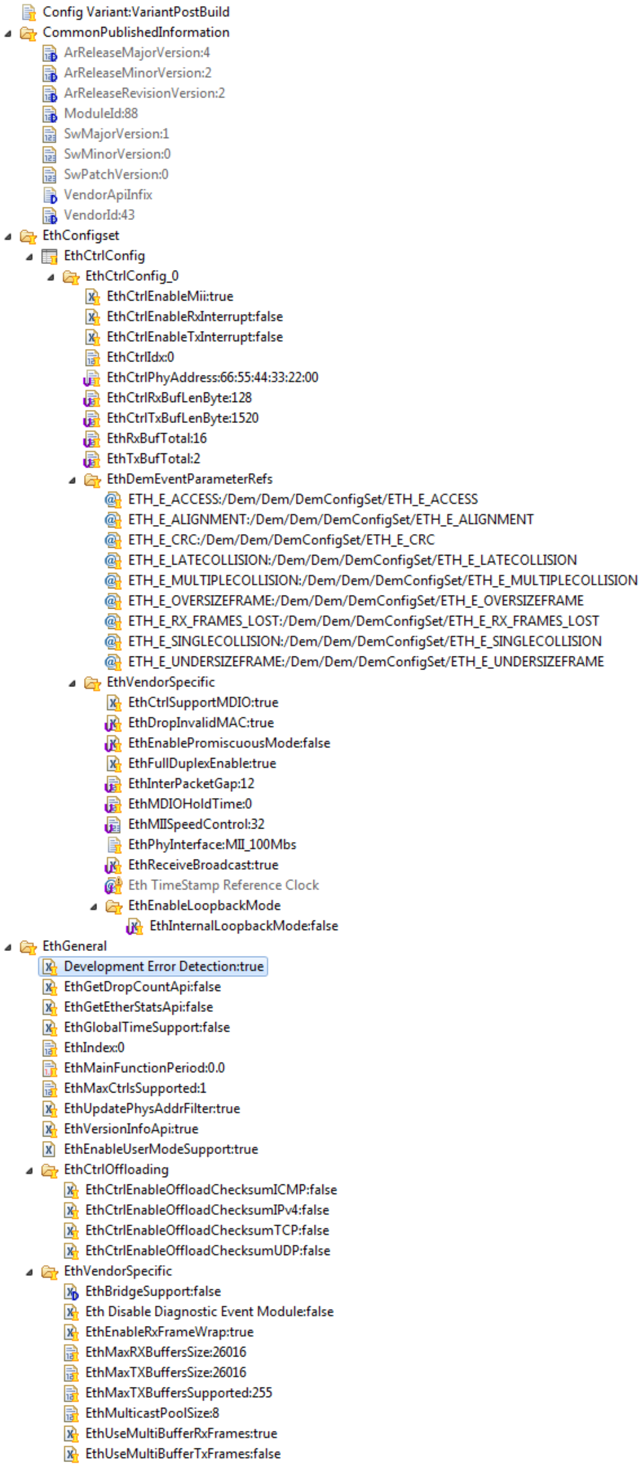
Chapter 4
Tresos Configuration Plug-in

The Tresos plug-in allows configuration of the ETH Driver by means of user friendly graphical user interface which allows an automatic configuration parameters checking. All computations are done by the configuration generator with the appropriate checks.

4.1 Configuration Elements of ETH

4.1.1 Configuration Parameters in Tresos

All ETH Driver configuration parameters are organized into tree-like structure according to the AUTOSAR 4.2 Rev002 ETH Driver Software Specification Document [Reference List] as shown in the Figure Figure 4-1. All of them are described in the Configuration Parameters section.



```
Config Variant:VariantPostBuild
CommonPublishedInformation
    ArReleaseMajorVersion:4
    ArReleaseMinorVersion:2
    ArReleaseRevisionVersion:2
    ModuleId:88
    SwMajorVersion:1
    SwMinorVersion:0
    SwPatchVersion:0
    VendorApiInfix
    VendorId:43
EthConfigset
    EthCtrlConfig
        EthCtrlConfig_0
            EthCtrlEnableMii:true
            EthCtrlEnableRxInterrupt:false
            EthCtrlEnableTxInterrupt:false
            EthCtrlIdx:0
            EthCtrlPhyAddress:66:55:44:33:22:00
            EthCtrlRxBufLenByte:128
            EthCtrlTxBufLenByte:1520
            EthRxBufTotal:16
            EthTxBufTotal:2
            EthDemEventParameterRefs
                ETH_E_ACCESS:/Dem/Dem/DemConfigSet/ETH_E_ACCESS
                ETH_E_ALIGNMENT:/Dem/Dem/DemConfigSet/ETH_E_ALIGNMENT
                ETH_E_CRC:/Dem/Dem/DemConfigSet/ETH_E_CRC
                ETH_E_LATECOLLISION:/Dem/Dem/DemConfigSet/ETH_E_LATECOLLISION
                ETH_E_MULTIPLECOLLISION:/Dem/Dem/DemConfigSet/ETH_E_MULTIPLECOLLISION
                ETH_E_OVERSIZEFRAME:/Dem/Dem/DemConfigSet/ETH_E_OVERSIZEFRAME
                ETH_E_RX_FRAMES_LOST:/Dem/Dem/DemConfigSet/ETH_E_RX_FRAMES_LOST
                ETH_E_SINGLECOLLISION:/Dem/Dem/DemConfigSet/ETH_E_SINGLECOLLISION
                ETH_E_UNDERSIZEFRAME:/Dem/Dem/DemConfigSet/ETH_E_UNDERSIZEFRAME
            EthVendorSpecific
                EthCtrlSupportMDIO:true
                EthDropInvalidMAC:true
                EthEnablePromiscuousMode:false
                EthFullDuplexEnable:true
                EthInterPacketGap:12
                EthMDIOHoldTime:0
                EthMIISpeedControl:32
                EthPhyInterface:MII_100Mbs
                EthReceiveBroadcast:true
                Eth TimeStamp Reference Clock
                EthEnableLoopbackMode
                    EthInternalLoopbackMode:false
EthGeneral
    Development Error Detection:true
    EthGetDropCountApi:false
    EthGetEtherStatsApi:false
    EthGlobalTimeSupport:false
    EthIndex:0
    EthMainFunctionPeriod:0.0
    EthMaxCtrlsSupported:1
    EthUpdatePhysAddrFilter:true
    EthVersionInfoApi:true
    EthEnableUserModeSupport:true
    EthCtrlOffloading
        EthCtrlEnableOffloadChecksumICMP:false
        EthCtrlEnableOffloadChecksumIPv4:false
        EthCtrlEnableOffloadChecksumTCP:false
        EthCtrlEnableOffloadChecksumUDP:false
    EthVendorSpecific
        EthBridgeSupport:false
        Eth Disable Diagnostic Event Module:false
        EthEnableRxFrameWrap:true
        EthMaxRXBuffersSize:26016
        EthMaxTXBuffersSize:26016
        EthMaxTXBuffersSupported:255
        EthMulticastPoolSize:8
        EthUseMultiBufferRxFrames:true
        EthUseMultiBufferTxFrames:false
```

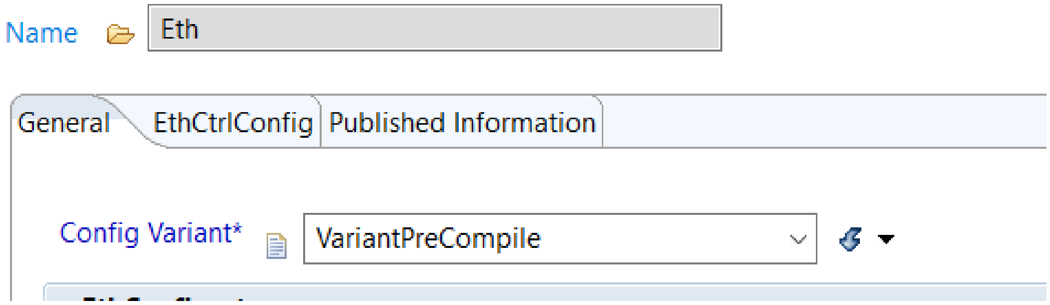
Figure 4-1. Configuration Parameters Tree

4.1.2 Configuration Generator

The configuration generator prepares the appropriate macros and/or structure instances from the values entered in the Tresos graphical user interface. It also performs the additional checks and provides detailed information about each configuration by means of the comments in configuration files. For example the sizes of the receive and the transmit buffers memory blocks are computed according to the description in the Buffers Memory section and appended to each generated of the configurations.

4.2 Form IMPLEMENTATION_CONFIG_VARIANT

VariantLinkTime: Linking time configuration parameters. VariantPreCompile: Precompile time configuration parameters. VariantPostBuild: Mix of precompile and postbuild time configuration parameters.


Name: Eth

Config Variant*: VariantPreCompile

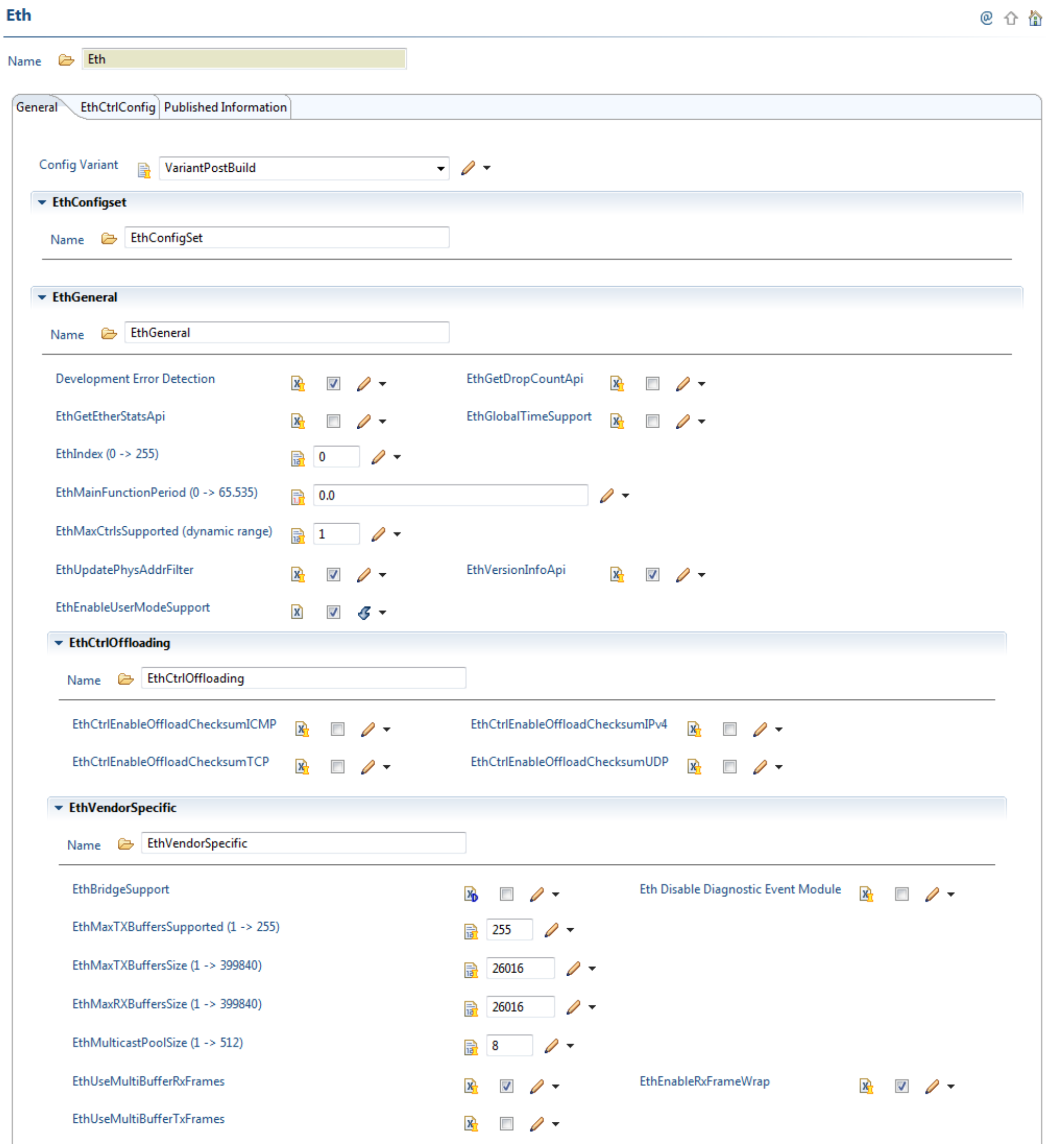
Figure 4-2. Tresos Plugin snapshot for IMPLEMENTATION_CONFIG_VARIANT

Table 4-1. Attribute IMPLEMENTATION_CONFIG_VARIANT detailed description

Property	Value
Label	Config Variant
Default	VariantPostBuild
Range	VariantLinkTime VariantPostBuild VariantPreCompile

4.3 EthGeneral Container

The *EthGeneral* container contains the ETH Driver settings common for all multiple configurations. The *EthGeneral* container is extended by the implementation specific configuration parameters grouped into the *EthVendorSpecific* sub-container.



Eth

Name: Eth

General | EthCtrlConfig | Published Information

Config Variant: VariantPostBuild

EthConfigset

Name: EthConfigSet

EthGeneral

Name: EthGeneral

Development Error Detection ☑
EthGetDropCountApi □
EthGetEtherStatsApi □
EthGlobalTimeSupport □
EthIndex (0 -> 255): 0
EthMainFunctionPeriod (0 -> 65.535): 0.0
EthMaxCtrlsSupported (dynamic range): 1
EthUpdatePhysAddrFilter ☑
EthVersionInfoApi ☑
EthEnableUserModeSupport ☑

EthCtrlOffloading

Name: EthCtrlOffloading

EthCtrlEnableOffloadChecksumICMP □
EthCtrlEnableOffloadChecksumIPv4 □
EthCtrlEnableOffloadChecksumTCP □
EthCtrlEnableOffloadChecksumUDP □

EthVendorSpecific

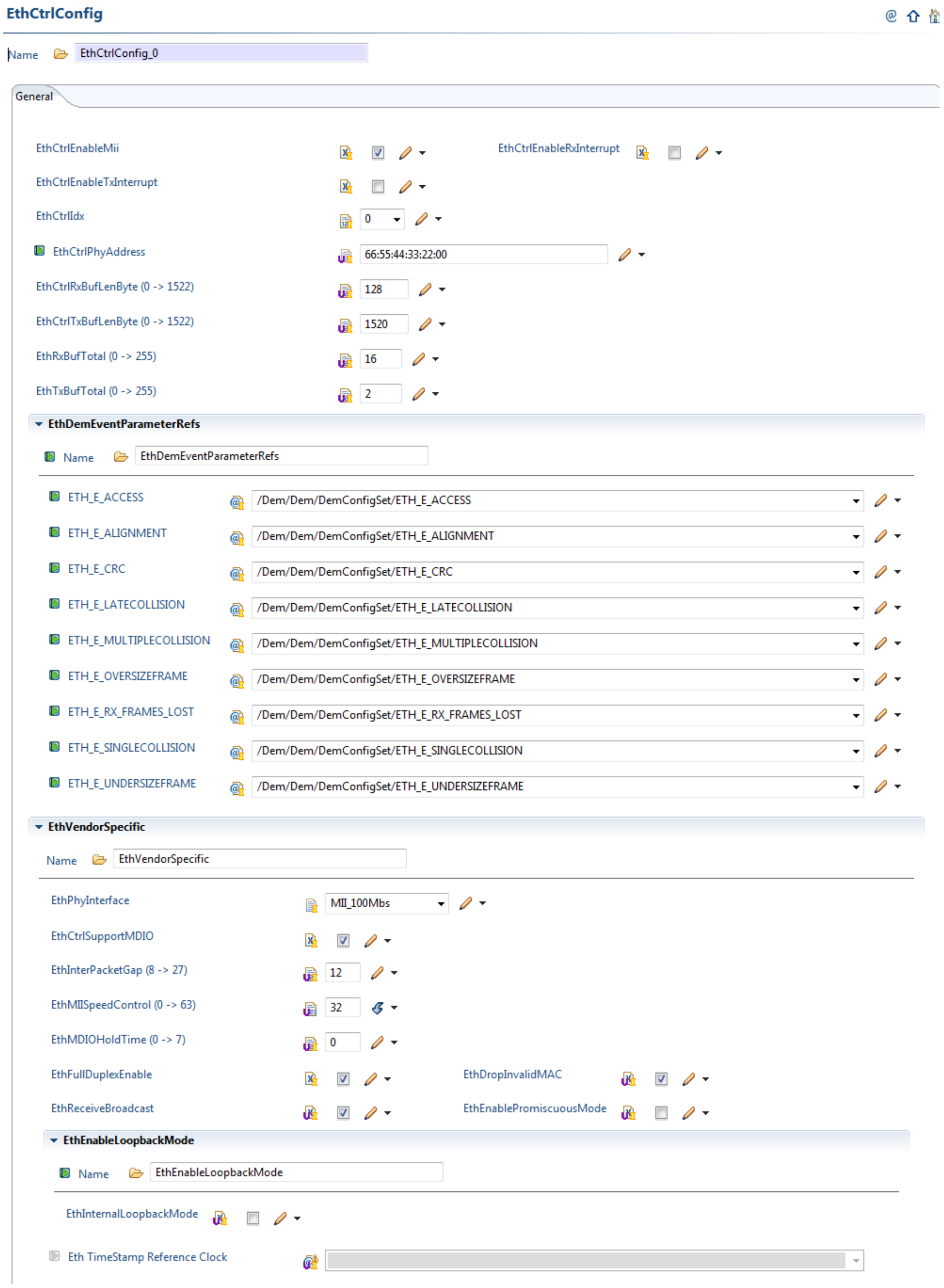
Name: EthVendorSpecific

EthBridgeSupport □
Eth Disable Diagnostic Event Module □
EthMaxTXBuffersSupported (1 -> 255): 255
EthMaxTXBuffersSize (1 -> 399840): 26016
EthMaxRXBuffersSize (1 -> 399840): 26016
EthMulticastPoolSize (1 -> 512): 8
EthUseMultiBufferRxFrames ☑
EthEnableRxFrameWrap ☑
EthUseMultiBufferTxFrames □

Figure 4-3. EthGeneral Container

4.4 EthConfigSet Container Children

Each EthConfigSet contains number of *EthCtrlConfig* (depend on *EthGeneral/EthMaxCtrlsSupported*). Each *EthCtrlConfig* contains the value specific for each controller. There are also some specific parameters which are grouped into the *EthVendorSpecific* sub-container.



EthCtrlConfig

Name EthCtrlConfig_0

General

EthCtrlEnableMii ☑

EthCtrlEnableRxInterrupt □

EthCtrlEnableTxInterrupt □

EthCtrlIdx 0

EthCtrlPhyAddress 66:55:44:33:22:00

EthCtrlRxBufLenByte (0 -> 1522) 128

EthCtrlTxBufLenByte (0 -> 1522) 1520

EthRxBufTotal (0 -> 255) 16

EthTxBufTotal (0 -> 255) 2

EthDemEventParameterRefs

Name EthDemEventParameterRefs

ETH_E_ACCESS /Dem/Dem/DemConfigSet/ETH_E_ACCESS

ETH_E_ALIGNMENT /Dem/Dem/DemConfigSet/ETH_E_ALIGNMENT

ETH_E_CRC /Dem/Dem/DemConfigSet/ETH_E_CRC

ETH_E_LATECOLLISION /Dem/Dem/DemConfigSet/ETH_E_LATECOLLISION

ETH_E_MULTIPLECOLLISION /Dem/Dem/DemConfigSet/ETH_E_MULTIPLECOLLISION

ETH_E_OVERSIZEFRAME /Dem/Dem/DemConfigSet/ETH_E_OVERSIZEFRAME

ETH_E_RX_FRAMES_LOST /Dem/Dem/DemConfigSet/ETH_E_RX_FRAMES_LOST

ETH_E_SINGLECOLLISION /Dem/Dem/DemConfigSet/ETH_E_SINGLECOLLISION

ETH_E_UNDERSIZEFRAME /Dem/Dem/DemConfigSet/ETH_E_UNDERSIZEFRAME

EthVendorSpecific

Name EthVendorSpecific

EthPhyInterface MII_100Mbs

EthCtrlSupportMDIO ☑

EthInterPacketGap (8 -> 27) 12

EthMIISpeedControl (0 -> 63) 32

EthMDIOHoldTime (0 -> 7) 0

EthFullDuplexEnable ☑

EthDropInvalidMAC ☑

EthReceiveBroadcast ☑

EthEnablePromiscuousMode □

EthEnableLoopbackMode

Name EthEnableLoopbackMode

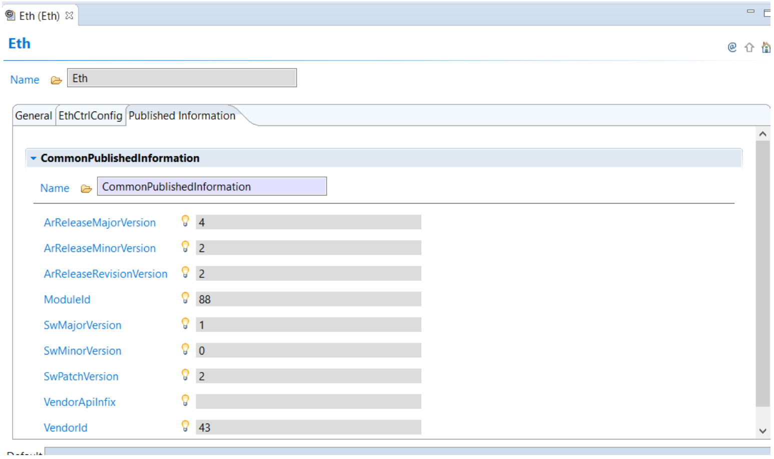
EthInternalLoopbackMode □

Eth TimeStamp Reference Clock

Figure 4-4. EthCtrlConfig Container Children

4.5 Form CommonPublishedInformation

Common container, aggregated by all modules. It contains published information about vendor and versions.



Name: Eth

CommonPublishedInformation

Name: CommonPublishedInformation

ArReleaseMajorVersion	4
ArReleaseMinorVersion	2
ArReleaseRevisionVersion	2
ModuleId	88
SwMajorVersion	1
SwMinorVersion	0
SwPatchVersion	2
VendorApiInfix	
VendorId	43

Figure 4-5. Tresos Plugin snapshot for CommonPublishedInformation form.

4.5.1 ArReleaseMajorVersion (CommonPublishedInformation)

Major version number of AUTOSAR specification on which the appropriate implementation is based on.

Table 4-2. Attribute ArReleaseMajorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Major Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	4
Invalid	Range >=4 <=4

4.5.2 ArReleaseMinorVersion (CommonPublishedInformation)

Minor version number of AUTOSAR specification on which the appropriate implementation is based on.

Table 4-3. Attribute ArReleaseMinorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Minor Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range >=2 <=2

4.5.3 ArReleaseRevisionVersion (CommonPublishedInformation)

Revision version number of AUTOSAR specification on which the appropriate implementation is based on.

Table 4-4. Attribute ArReleaseRevisionVersion (CommonPublishedInformation) detailed description

Property	Value
Label	AUTOSAR Release Revision Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range >=2 <=2

4.5.4 ModuleId (CommonPublishedInformation)

Module ID of this module from Module List.

Table 4-5. Attribute ModuleId (CommonPublishedInformation) detailed description

Property	Value
Label	Module Id
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false

Table continues on the next page...

Table 4-5. Attribute ModuleId (CommonPublishedInformation) detailed description (continued)

Property	Value
Default	
Invalid	Range >= <=

4.5.5 SwMajorVersion (CommonPublishedInformation)

Major version number of the vendor specific implementation of the module. The numbering is vendor specific.

Table 4-6. Attribute SwMajorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Major Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	1
Invalid	Range >=1 <=1

4.5.6 SwMinorVersion (CommonPublishedInformation)

Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.

Table 4-7. Attribute SwMinorVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Minor Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	0
Invalid	Range >=0 <=0

4.5.7 SwPatchVersion (CommonPublishedInformation)

Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.

Table 4-8. Attribute SwPatchVersion (CommonPublishedInformation) detailed description

Property	Value
Label	Software Patch Version
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	2
Invalid	Range >=2 <=2

4.5.8 VendorApiInfix (CommonPublishedInformation)

In driver modules which can be instantiated several times on a single ECU, BSW00347 requires that the name of APIs is extended by the VendorId and a vendor specific name. This parameter is used to specify the vendor specific name. In total, the implementation specific name is generated as follows: <ModuleName>_>VendorId>_<VendorApiInfix><Api name from SWS>. E.g. assuming that the VendorId of the implementor is 123 and the implementer chose a VendorApiInfix of "v11r456" a api name Can_Write defined in the SWS will translate to Can_123_v11r456Write. This parameter is mandatory for all modules with upper multiplicity > 1. It shall not be used for modules with upper multiplicity =1.

Table 4-9. Attribute VendorApiInfix (CommonPublishedInformation) detailed description

Property	Value
Label	Vendor Api Infix
Type	STRING_LABEL
Origin	Custom
Symbolic Name	false
Default	
Enable	false

4.5.9 VendorId (CommonPublishedInformation)

Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list.

Table 4-10. Attribute VendorId (CommonPublishedInformation) detailed description

Property	Value
Label	Vendor Id
Type	INTEGER_LABEL
Origin	Custom
Symbolic Name	false
Default	43
Invalid	Range >=43 <=43

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2019 NXP B.V.