HAL

Ethernet MAC

netX 5/10/50/51/90/100/500/4000

V6.2.x.x

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC090303HAL09EN | Revision 9 | English | 2018-11 | Released | Public

Table of Contents

1 Introduction

1.1 About this Document

This manual describes the interface of the Ethernet MAC with the aim to support and lead you during the integration process of the given unit into your application running under your own operating system.

It is a description of how to configure and to exchange data with it in general.

1.2 List of Revisions

Rev	Date	Name	Chapter	Revision
1	2009-03-23	AO		Created
2	2009-12-17	AO		Documents merged for all netX50/100/500 Added user specific pointer for function Init() and Start()
3	2010-02-22	AO		Some corrections in function descriptions
4	2012-02-08	AO		Added netX51
5	2016-03-08	AO		Changes to V5.0.x.x; no change of HAL API
6	2016-12-13	AO		Changes to V5.1.x.x Unified all HAL APIs Renewed HAL
7	2017-02-02	AO		Changes to V6.0.x.x Merged Ethernet Standard MAC and Ethernet Standard MAC for external PHY
8	2017-05-15	BI	2	Updated HAL API
9	2018-11-22	AO		Changes to V6.2.x.x

Table 1: List of Revisions

1.3 Terms, Abbreviations and Definitions

Term	Description
MAC	Media Access Controller
QoS	Quality of Service
VLAN	Virtual Local Area Network
MII	Media Independent Interface

Table 2: Terms, Abbreviations and Definitions

All variables, parameters, and data used in this manual have the LSB/MSB (“Intel”) data format. This corresponds to the convention of the Microsoft C Compiler.

All IP addresses in this document have host byte order.

1.4 References

This document based on the following specification:

Number	Document
1	IEEE802.3 - 2002

Table 3: References

1.5 Functional Overview

You as a user are getting a capable and a general-purpose Software interface package with following features:

- Initialization of the integrated transceivers (Dual-PHY)
- Configuration of the Ethernet MAC
- Getting of Status Information of the Ethernet MAC
- Sending of Ethernet frame transmission requests to the Ethernet MAC
- Getting of Confirmations about processed transmission processes
- Getting of Ethernet frame indications from the Ethernet MAC
- Configuration of link/activity LED behavior for application specific use

1.5.1 System Requirements

The software package has the following system requirements to its environment:

- netX-Chip as CPU hardware platform
- operating system independency

1.5.2 Intended Audience

This manual is suitable for software developers with the following background:

- Knowledge of the programming language C
- Knowledge of the IEEE802.3 specification
- Knowledge of the IEEE1588 specification

1.5.3 Technical Data

- 10BASE-T/100BASE-TX/FX operation in full/half duplex
- Integrated Dual-PHY with MDIX and Auto-Negotiation capability
- Quality of Service capable: 2 Traffic Classes (adjustable)
- Promiscuous mode (for monitoring purposes)
- Multicast pre-filtering capable
- Direct Access to PHY status information link, duplex and speed
- Number of Ethernet frame buffers
  - netX5/10/50/100/500: 20
  - netX51/4000/90: 41
- Configurable LED behavior
- possibility to suppress confirmation of processed transmission requests
- Time stamping of incoming and outgoing Ethernet Frames at MII in IEEE1588 format
  - Precision netX10/50/100/500: 10 ns
  - Precision netX51/4000/90: 1.25 ns

1.5.4 Limitations

- no frame buffer management - each Ethernet frame occupies 1560 Bytes Buffer
- no Gigabit operation
- SYSTIME resolution must be 1 ns (clock_count_val = 0xa0000000) to get high-precise timestamps
- netX100/500 external PHY at XC Port 2 and 3: Ethernet LEDs like Link, Speed, Duplex or Activity must be controlled via PHY itself

1.5.5 External PHY selection requirements

- PHY shall provide MII interface and shall be accessible via MDIO interface
- PHY shall provide information about Link status, Duplex mode and Speed via MDIO access
- Generally, a Static Link signal from PHY connected to netX required
- netX5/100/500: Static Link signal must be low-active

1.5.6 Speed and Duplex setting

For proper network operation, both the MAC and PHY must be properly configured.

When using a copper PHY, the PHY performs the auto-negotiation function. Auto-negotiation provides a method for two link partners to exchange information in a systematic manner in order to establish a link configuration providing the highest common level of functionality supported by both partners. Once configured, the link partners exchange configuration information to resolve link settings such as

- speed 10 or 100 Mb/s
- duplex: full or half

Duplex mismatch

On an Ethernet connection, a duplex mismatch is a condition where two connected devices operate in different duplex modes, that is, one operates in half duplex while the other one operates in full duplex. The effect of a duplex mismatch is a link that operates inefficiently. Duplex mismatch may be caused by manually setting two connected network interfaces at different duplex modes or by connecting a device that performs auto-negotiation to one that is manually set to a full duplex mode.

A duplex mismatch can be recognized by checking the MAC error counters. On the full duplex side there are receive errors (collision fragments, FCS errors). On the half-duplex side there are many late collisions (collision after first 64 bytes of an Ethernet frame). From the MAC user’s point of view, the performance of link with duplex mismatch is very poor, especially on the half-duplex side due to many retries and frame transmission aborts because of late collisions.

The speed and duplex status of the link must be determined by reading of PHY registers directly by the MAC user. This functionality is not part of the Ethernet MAC HAL.

Because of the speed and duplex mode changes after link change it is recommended to use the “Link Changed” interrupt of the Ethernet MAC to update speed and duplex setting of the Ethernet MAC quickly. Alternatively, it is possible update the speed and duplex cyclically (via polling of the PHY registers).

1.6 Legal Notes

1.6.1 Copyright

1.6.2 Important Notes

The user manual, accompanying texts and the documentation were created for the use of the products by qualified experts, however, errors cannot be ruled out. For this reason, no guarantee can be made and neither juristic responsibility for erroneous information nor any liability can be assumed. Descriptions, accompanying texts and documentation included in the user manual do not present a guarantee nor any information about proper use as stipulated in the contract or a warranted feature. It cannot be ruled out that the user manual, the accompanying texts and the documentation do not correspond exactly to the described features, standards or other data of the delivered product. No warranty or guarantee regarding the correctness or accuracy of the information is assumed.

We reserve the right to change our products and their specification as well as related user manuals, accompanying texts and documentation at all times and without advance notice, without obligation to report the change. Changes will be included in future manuals and do not constitute any obligations. There is no entitlement to revisions of delivered documents. The manual delivered with the product applies.

Hilscher Gesellschaft für Systemautomation mbH is not liable under any circumstances for direct, indirect, incidental or follow-on damage or loss of earnings resulting from the use of the information contained in this publication.

1.6.3 Exclusion of Liability

The software was produced and tested with utmost care by Hilscher Gesellschaft für Systemautomation mbH and is made available as is. No warranty can be assumed for the performance and flawlessness of the software for all usage conditions and cases and for the results produced when utilized by the user. Liability for any damages that may result from the use of the hardware or software or related documents, is limited to cases of intent or grossly negligent violation of significant contractual obligations. Indemnity claims for the violation of significant contractual obligations are limited to damages that are foreseeable and typical for this type of contract.

It is strictly prohibited to use the software in the following areas:

- for military purposes or in weapon systems;
- for the design, construction, maintenance or operation of nuclear facilities;
- in air traffic control systems, air traffic or air traffic communication systems;
- in life support systems;
- in systems in which failures in the software could lead to personal injury or injuries leading to death.

We inform you that the software was not developed for use in dangerous environments requiring fail-proof control mechanisms. Use of the software in such an environment occurs at your own risk. No liability is assumed for damages or losses due to unauthorized use.

1.6.4 Warranty

Although the hardware and software was developed with utmost care and tested intensively, Hilscher Gesellschaft für Systemautomation mbH does not guarantee its suitability for any purpose not confirmed in writing. It cannot be guaranteed that the hardware and software will meet your requirements, that the use of the software operates without interruption and that the software is free of errors. No guarantee is made regarding infringements, violations of patents, rights of ownership or the freedom from interference by third parties. No additional guarantees or assurances are made regarding marketability, freedom of defect of title, integration or usability for certain purposes unless they are required in accordance with the law and cannot be limited. Warranty claims are limited to the right to claim rectification.

1.6.5 Export Regulations

The delivered product (including the technical data) is subject to export or import laws as well as the associated regulations of different counters, in particular those of Germany and the USA. The software may not be exported to countries where this is prohibited by the United States Export Administration Act and its additional provisions. You are obligated to comply with the regulations at your personal responsibility. We wish to inform you that you may require permission from state authorities to export, re-export or import the product.

2 The Interface

This section describes the data transfer services available to the Ethernet MAC user with their associated service primitives and parameters.

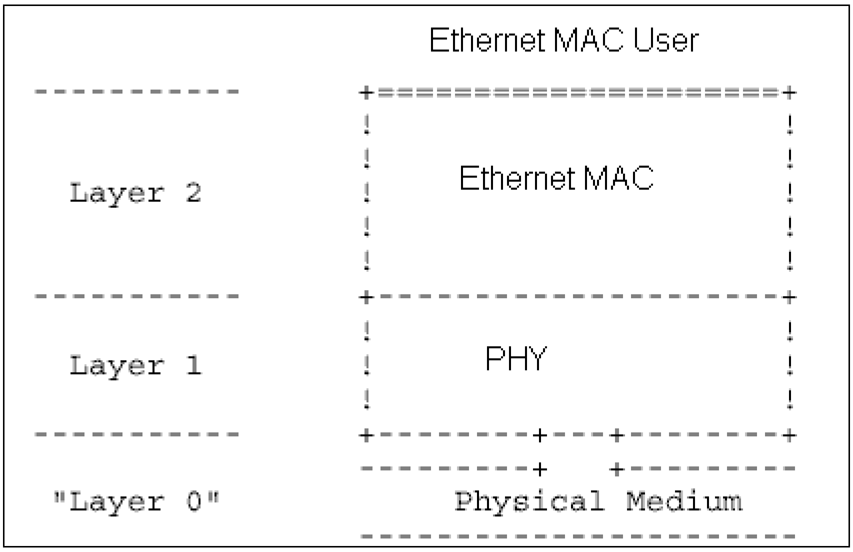

Figure 1: Interface between Interface User and Interface in Relation to Layer Model

2.1 Overview of Service Classes

The user of Layer 2 is provided with the following service classification:

Control

This service class defines the transfer of control commands from an Ethernet MAC user to the Ethernet MAC.

Status

This service class defines the transfer of status information from the Ethernet MAC to an Ethernet MAC user.

Transmission

This service class defines the transfer of an Ethernet frame from the Ethernet MAC user to the Ethernet MAC.

Reception

This service class defines the transfer of an Ethernet frame from the Ethernet MAC to an Ethernet MAC user.

2.2 Control Service Class

2.2.1 `EthMac_AddGroupAddr()` - Add Group Address

Add the given Multicast group address to the address recognition filter.

Function Prototype

```
ETHERNET_RESULT
EthMac_AddGroupAddr( unsigned int               uiPort,
                     const ETHERNET_MAC_ADDR_T  tMacAddr )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	XC port number
`tMacAddr`	[in]	Multicast MAC address value

Table 4: `EthMac_AddGroupAddr()` - Function Arguments

Function Return Values

Definition	Description
`ETH_OKAY`	Successful
`ETH_ERR_OUT_OF_MEMORY`	Not enough resources
`ETH_ERR_INVALID_PARAMETER`	Invalid parameter

Table 5: `EthMac_AddGroupAddr()` - Function Return Values

2.2.2 `EthMac_CfgMii()` - Configure MII

This function configures the MII that is used. Note: Only call this function before XC started. Note: Use this function only when connecting an external PHY to compensate delays between external PHY and internal MAC logic. Note: Default value fits to internal PHYs if available else to external MII.

Function Prototype

```
ETHERNET_RESULT
EthMac_CfgMii( unsigned int  uiPort,
               unsigned int  uiCfg )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	XC port number
`uiCfg`	[in]	MII configuration; 0: internal PHY, 1: external MII

Table 6: `EthMac_CfgMii()` - Function Arguments

Function Return Values

Definition	Description
`ETH_OKAY`	Successful
`ETH_ERR_INVALID_PARAMETER`	Invalid parameter

Table 7: `EthMac_CfgMii()` - Function Return Values

2.2.3 `EthMac_ConfirmIrq()` - Confirm Interrupts

This function confirms a set of interrupts that were requested by the Ethernet MAC.

Function Prototype

```
ETHERNET_RESULT
EthMac_ConfirmIrq( unsigned int  uiPort,
                   uint32_t      ulConfirmIrqMask )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	XC port number
`ulConfirmIrqMask`	[in]	Mask to confirm interrupt events

Table 8: `EthMac_ConfirmIrq()` - Function Arguments

Function Return Values

Definition	Description
`ETH_OKAY`	Successful
`ETH_ERR_INVALID_PARAMETER`	Invalid parameter

Table 9: `EthMac_ConfirmIrq()` - Function Return Values

2.2.4 `EthMac_DeleteGroupAddr()` - Delete Group Address

Delete the given Multicast group address from the address recognition.

Function Prototype

```
ETHERNET_RESULT
EthMac_DeleteGroupAddr( unsigned int              uiPort,
                        const ETHERNET_MAC_ADDR_T  tMacAddr )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	XC port number
`tMacAddr`	[in]	Multicast MAC address value

Table 10: `EthMac_DeleteGroupAddr()` - Function Arguments

Function Return Values

Definition	Description
`ETH_OKAY`	Successful
`ETH_ERR_INVALID_PARAMETER`	Invalid parameter

Table 11: `EthMac_DeleteGroupAddr()` - Function Return Values

2.2.5 `EthMac_Initialize()` - Initialize Ethernet MAC

This function initializes the according XC port as Ethernet MAC and configures it with the default parameter settings.

Note that LEDs must be disabled when external PHYs are used.

Function Prototype

```
ETHERNET_RESULT
EthMac_Initialize( unsigned int            uiPort,
                   ETHERNET_PHY_LED_CFG_E  ePhyLedCfg,
                   uint32_t                ulActLedFlashPeriod,
                   void*                   pvUser )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	XC port number
`ePhyLedCfg`	[in]	PHY LED behavior (0: two separate LEDs for link and activity not blinking; 1: as 0, but activity is blinking; 2: one single combined link/activity LED; 3: LEDs are disabled)
`ulActLedFlashPeriod`	[in]	Flash frequency of activity LED [10ns], default: 5000000 = 50 milliseconds; The blink frequency shall not be smaller than 10ms and larger than 80 milliseconds, other values may lead to malfunction of the LED
`pvUser`	[in]	User specific parameter

Table 12: `EthMac_Initialize()` - Function Arguments

Function Return Values

Definition	Description
`ETH_OKAY`	Successful
`ETH_ERR_INIT_FAILED`	Initialization has failed
`ETH_ERR_INVALID_PARAMETER`	Invalid parameter

Table 13: `EthMac_Initialize()` - Function Return Values

2.2.6 EthMac_ModePromisc() - Enable/Disable Promiscuous Mode

This function enables/disables promiscuous mode at the Ethernet MAC. When promiscuous mode is enabled all error-free received Ethernet frames are transferred into the according indication FIFO. Otherwise only all error-free received broadcast Ethernet frames, not filtered MultiCast Ethernet frames and UniCast Ethernet frames that match the local MAC address are transferred into the according indication FIFO.

Function Prototype

```
ETHERNET_RESULT
EthMac_ModePromisc( unsigned int  uiPort,
                    unsigned int  uEnable )
```

Function Arguments

Argument	Direction	Description
uiPort	[in]	XC port number
uEnable	[in]	1/0: enables/disables promiscuous mode

Table 14: EthMac_ModePromisc() - Function Arguments

Function Return Values

Definition	Description
ETH_OKAY	Successful
ETH_ERR_INVALID_PARAMETER	Invalid parameter

Table 15: EthMac_ModePromisc() - Function Return Values

2.2.7 `EthMac_SetIrqMask()` - Enable Interrupts

This function sets a set of interrupts to be enabled or disabled to the Ethernet MAC.

Function Prototype

```
ETHERNET_RESULT
EthMac_SetIrqMask( unsigned int  uiPort,
                   uint32_t      ulIrqMask )
```

Function Arguments

Argument	Direction	Description
uiPort	[in]	XC port number
ulIrqMask	[in]	Inclusively-ORed mask of interrupts to be enabled, otherwise they will be disabled

Table 16: `EthMac_SetIrqMask()` - Function Arguments

Function Return Values

Definition	Description
ETH_OKAY	Successful
ETH_ERR_INVALID_PARAMETER	Invalid parameter

Table 17: `EthMac_SetIrqMask()` - Function Return Values

2.2.8 EthMac_SetLinkMode() - Set Link Mode

This function sets the link mode of the MAC. Note: These values must match the mode the connected PHY is set to. Also in case of link down this function has to be called.

Function Prototype

```
ETHERNET_RESULT
EthMac_SetLinkMode( unsigned int  uiPort,
                    bool          fValid,
                    unsigned int  uiSpeed,
                    bool          fFdx )
```

Function Arguments

Argument	Direction	Description
uiPort	[in]	XC port number
fValid	[in]	true: link up
uiSpeed	[in]	10/100
fFdx	[in]	true: FDX

Table 18: EthMac_SetLinkMode() - Function Arguments

Function Return Values

Definition	Description
ETH_OKAY	Successful
ETH_ERR_INVALID_PARAMETER	Invalid parameter

Table 19: EthMac_SetLinkMode() - Function Return Values

2.2.9 `EthMac_SetMacAddr()` - Set MAC Address

Sets a MAC address for the according XC port. Note: The Chassis MAC addresses shall be set before the switch is started.

Function Prototype

```
ETHERNET_RESULT
EthMac_SetMacAddr( unsigned int             uiPort,
                   ETH_MAC_ADDRESS_TYPE_E   eType,
                   const ETHERNET_MAC_ADDR_T  tMacAddr )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	XC port number
`eType`	[in]	Defines which MAC address shall be configured
`tMacAddr`	[in]	MAC address value

Table 20: `EthMac_SetMacAddr()` - Function Arguments

Function Return Values

Definition	Description
`ETH_OKAY`	Successful
`ETH_ERR_INVALID_PARAMETER`	Invalid parameter

Table 21: `EthMac_SetMacAddr()` - Function Return Values

2.2.10 EthMac_SetTrafficClassArrangement() - Set Traffic Classes

This function sets the VLAN Tag priority classification between the two traffic classes.

TC=0: High: VLAN tag priority 7..0 Low: Non-tagged frame

TC=1: High: VLAN tag priority 7..1 Low: VLAN tag priority 0, non-tagged

TC=2: High: VLAN tag priority 7..2 Low: VLAN tag priority 1..0, non-tagged

TC=3: High: VLAN tag priority 7..3 Low: VLAN tag priority 2..0, non-tagged

TC=4: High: VLAN tag priority 7..4 Low: VLAN tag priority 3..0, non-tagged

TC=5: High: VLAN tag priority 7..5 Low: VLAN tag priority 4..0, non-tagged

TC=6: High: VLAN tag priority 7..6 Low: VLAN tag priority 5..0, non-tagged

TC=7: High: VLAN tag priority 7 Low: VLAN tag priority 6..0, non-tagged

TC=8: High: - Low: VLAN tag priority 7..0, non-tagged

Function Prototype

```
ETHERNET_RESULT
EthMac_SetTrafficClassArrangement( unsigned int  uiPort,
                                   unsigned int  uClass )
```

Function Arguments

Argument	Direction	Description
uiPort	[in]	XC port number
uClass	[in]	Traffic Class Arrangement

Table 22: EthMac_SetTrafficClassArrangement() - Function Arguments

Function Return Values

Definition	Description
ETH_OKAY	Successful
ETH_ERR_INVALID_PARAMETER	Invalid parameter

Table 23: EthMac_SetTrafficClassArrangement() - Function Return Values

2.2.11 `EthMac_Start()` - Start Ethernet MAC

This function starts the previously initialized Ethernet MAC.

Function Prototype

```
ETHERNET_RESULT
EthMac_Start( unsigned int  uiPort,
              void*         pvUser )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	XC port number
`pvUser`	[in]	User specific parameter

Table 24: `EthMac_Start()` - Function Arguments

Function Return Values

Definition	Description
`ETH_OKAY`	Successful
`ETH_ERR_INVALID_PARAMETER`	Invalid parameter
`ETH_ERR_INIT_FAILED`	Initialization has failed

Table 25: `EthMac_Start()` - Function Return Values

2.3 Reception Service Class

2.3.1 `EthMac_GetRecvFillLevel()` - Get Fill Level Indication FIFO

This function retrieves the fill level of the according indication FIFO.

Function Prototype

```
ETHERNET_RESULT
EthMac_GetRecvFillLevel( unsigned int  uiPort,
                         unsigned int  uHighPriority,
                         uint32_t *    pulFillLevel )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	XC port number
`uHighPriority`	[in]	Indication priority selector (1/0: high/low priority)
`pulFillLevel`	[out]	Indication FIFO fill level

Table 26: `EthMac_GetRecvFillLevel()` - Function Arguments

Function Return Values

Definition	Description
`ETH_OKAY`	Successful
`ETH_ERR_INVALID_PARAMETER`	Invalid parameter

Table 27: `EthMac_GetRecvFillLevel()` - Function Return Values

2.3.2 `EthMac_Recv()` - Get Ethernet Frame Indication

This function retrieves an indication from the according indication element at the Ethernet MAC.

Note: The IEEE 1588 time stamp is to be found at offset 1536 within each Ethernet-Frame Buffer.

Function Prototype

```
ETHERNET_RESULT
EthMac_Recv( unsigned int        uiPort,
             ETHERNET_FRAME_T**  pptFrame,
             void**              phFrame,
             uint32_t*           pulLength,
             unsigned int        uHighPriority )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	XC port number
`pptFrame`	[out]	Pointer to pointer to Ethernet frame
`phFrame`	[out]	Pointer to handle to Ethernet frame
`pulLength`	[out]	Pointer to Ethernet frame length of indication
`uHighPriority`	[in]	Indication priority selector (1/0: high/low priority)

Table 28: `EthMac_Recv()` - Function Arguments

Function Return Values

Definition	Description
ETH_OKAY	Successful
ETH_ERR_FIFO_EMPTY	The FIFO is empty
ETH_ERR_INVALID_PARAMETER	Invalid parameter

Table 29: `EthMac_Recv()` - Function Return Values

2.3.3 `EthMac_ReleaseFrame()` - Release Ethernet Frame Block

This function puts an Ethernet frame block back into empty pointer FIFO of the Ethernet MAC.

Function Prototype

```
ETHERNET_RESULT
EthMac_ReleaseFrame( unsigned int  uiPort,
                     void*         hFrame )
```

Function Arguments

Argument	Direction	Description
uiPort	[in]	XC port number
hFrame	[in]	Handle to Ethernet frame

Table 30: `EthMac_ReleaseFrame()` - Function Arguments

Function Return Values

Definition	Description
ETH_OKAY	Successful
ETH_ERR_INVALID_PARAMETER	Invalid parameter

Table 31: `EthMac_ReleaseFrame()` - Function Return Values

2.4 Status Service Class

2.4.1 `EthMac_GetCounters()` - Get Diagnostic Counters

This function gets the diagnostic counters of the according Ethernet MAC.

Function Prototype

```
ETHERNET_RESULT
EthMac_GetCounters( unsigned int        uiPort,
                    ETHMAC_COUNTERS_T*  ptCounters )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	XC port number
`ptCounters`	[out]	Pointer to returned counter values

Table 32: `EthMac_GetCounters()` - Function Arguments

Function Return Values

Definition	Description
`ETH_OKAY`	Successful
`ETH_ERR_INVALID_PARAMETER`	Invalid parameter

Table 33: `EthMac_GetCounters()` - Function Return Values

2.4.2 `EthMac_GetIrq()` - Get Interrupt(s)

This function retrieves the current interrupt requests from the according Ethernet MAC.

Function Prototype

```
ETHERNET_RESULT
EthMac_GetIrq( unsigned int  uiPort,
               uint32_t *    pulIrq )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	XC port number
`pulIrq`	[out]	Pointer to interrupt events

Table 34: `EthMac_GetIrq()` - Function Arguments

Function Return Values

Definition	Description
`ETH_OKAY`	Successful
`ETH_ERR_INVALID_PARAMETER`	Invalid parameter

Table 35: `EthMac_GetIrq()` - Function Return Values

2.4.3 `EthMac_GetIrqMask()` - Get Interrupt Mask

This function gets a set of currently enabled interrupts at the according Ethernet MAC.

Function Prototype

```
ETHERNET_RESULT
EthMac_GetIrqMask( unsigned int  uiPort,
                   uint32_t *    pulIrqMask )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	XC port number
`pulIrqMask`	[out]	Pointer to unsigned long to receive the mask of enabled interrupts on the Ethernet MAC

Table 36: `EthMac_GetIrqMask()` - Function Arguments

Function Return Values

Definition	Description
`ETH_OKAY`	Successful
`ETH_ERR_INVALID_PARAMETER`	Invalid parameter

Table 37: `EthMac_GetIrqMask()` - Function Return Values

2.4.4 `EthMac_GetMacAddr()` - Get MAC Address

This function gets the MAC address of the according Ethernet MAC.

Function Prototype

```
ETHERNET_RESULT
EthMac_GetMacAddr( unsigned int           uiPort,
                   ETH_MAC_ADDRESS_TYPE_E  eType,
                   ETHERNET_MAC_ADDR_T*    ptMacAddr )
```

Function Arguments

Argument	Direction	Description
uiPort	[in]	XC port number
eType	[in]	Defines which MAC address shall be read
ptMacAddr	[out]	Pointer to MAC address buffer

Table 38: `EthMac_GetMacAddr()` - Function Arguments

Function Return Values

Definition	Description
ETH_OKAY	Successful
ETH_ERR_INVALID_PARAMETER	Invalid parameter

Table 39: `EthMac_GetMacAddr()` - Function Return Values

2.5 Transmission Service Class

2.5.1 `EthMac_GetFrame()` - Get empty Ethernet Frame Block

This function gets an element from the empty FIFO.

Function Prototype

```
ETHERNET_RESULT
EthMac_GetFrame( unsigned int        uiPort,
                 ETHERNET_FRAME_T**  pptFrame,
                 void**              phFrame )
```

Function Arguments

Argument	Direction	Description
uiPort	[in]	XC port number
pptFrame	[out]	Pointer to pointer to Ethernet frame
phFrame	[out]	Pointer to handle to Ethernet frame

Table 40: `EthMac_GetFrame()` - Function Arguments

Function Return Values

Definition	Description
ETH_OKAY	Successful
ETH_ERR_FIFO_EMPTY	The FIFO is empty
ETH_ERR_INVALID_PARAMETER	Invalid parameter

Table 41: `EthMac_GetFrame()` - Function Return Values

2.5.2 EthMac_GetSendCnf() - Get Confirmation of Transmission Request

This function retrieves a confirmation of the according confirmation FIFO of the Ethernet MAC.

Note: The IEEE 1588 time stamp is to be found at offset 1536 within each Ethernet-Frame Buffer.

Function Prototype

```
ETHERNET_RESULT
EthMac_GetSendCnf( unsigned int        uiPort,
                   ETHERNET_FRAME_T**  pptFrame,
                   void**              phFrame,
                   uint32_t*           pulLength,
                   unsigned int        uHighPriority,
                   ETHERNET_RESULT*    peResult )
```

Function Arguments

Argument	Direction	Description
uiPort	[in]	XC port number
pptFrame	[out]	Pointer to Ethernet frame
phFrame	[out]	Pointer to handle to Ethernet frame
pulLength	[out]	Pointer to Ethernet frame length of processed request
uHighPriority	[in]	Confirmation priority selector
peResult	[out]	Pointer to result code

Table 42: EthMac_GetSendCnf() - Function Arguments

Function Return Values

Definition	Description
ETH_OKAY	Successful
ETH_ERR_TX_SUCCESSFUL_WITH_RETRIES	Transmission successful with retries
ETH_ERR_TX_FAILED_LATE_COLLISION	Transmission failed due late collision
ETH_ERR_TX_FAILED_LINK_DOWN_DURING_TX	Transmission failed due link down
ETH_ERR_TX_FAILED_EXCESSIVE_COLLISION	Transmission failed due excessive collisions
ETH_ERR_TX_FAILED_UTX_UFL_DURING_TX	Transmission failed due UTX FIFO underflow
ETH_ERR_TX_FAILED_FATAL_ERROR	Transmission failed due fatal error
ETH_ERR_FIFO_EMPTY	The FIFO is empty
ETH_ERR_INVALID_PARAMETER	Invalid parameter

Table 43: EthMac_GetSendCnf() - Function Return Values

2.5.3 EthMac_GetSendCnfFillLevel() - Get Fill Level Confirmation FIFO

This function gets the fill level of the according confirmation FIFO.

Function Prototype

```
ETHERNET_RESULT
EthMac_GetSendCnfFillLevel( unsigned int  uiPort,
                            unsigned int  uHighPriority,
                            uint32_t *    pulCnfFillLevel )
```

Function Arguments

Argument	Direction	Description
uiPort	[in]	XC port number
uHighPriority	[in]	Confirmation priority selector (1/0: high/low priority)
pulCnfFillLevel	[out]	Pointer to confirmation FIFO fill level

Table 44: `EthMac_GetSendCnfFillLevel()` - Function Arguments

Function Return Values

Definition	Description
ETH_OKAY	Successful
ETH_ERR_INVALID_PARAMETER	Invalid parameter

Table 45: `EthMac_GetSendCnfFillLevel()` - Function Return Values

2.5.4 EthMac_Send() - Send Ethernet Frame with Confirmation

This function initiates a transmission request. After the processed transmission request the Host will get a confirmation.

Function Prototype

```
ETHERNET_RESULT
EthMac_Send( unsigned int  uiPort,
             void*         hFrame,
             uint32_t      ulLength,
             unsigned int  uHighPriority )
```

Function Arguments

Argument	Direction	Description
uiPort	[in]	XC port number
hFrame	[in]	Handle to Ethernet frame
ulLength	[in]	Ethernet frame length
uHighPriority	[in]	Request priority selector (1/0: high/low priority)

Table 46: EthMac_Send() - Function Arguments

Function Return Values

Definition	Description
ETH_OKAY	Successful
ETH_ERR_INVALID_PARAMETER	Invalid parameter

Table 47: EthMac_Send() - Function Return Values

2.5.5 EthMac_SendWithoutCnf() - Send Ethernet Frame without Confirmation

This function initiates a transmission request and suppresses confirmation. After processing the frame buffer will be released automatically by xPEC.

Function Prototype

```
ETHERNET_RESULT
EthMac_SendWithoutCnf( unsigned int  uiPort,
                       void*         hFrame,
                       uint32_t      ulLength,
                       unsigned int  uHighPriority )
```

Function Arguments

Argument	Direction	Description
uiPort	[in]	XC port number
hFrame	[in]	Handle to Ethernet frame
ulLength	[in]	Ethernet frame length
uHighPriority	[in]	Request priority selector (1/0: high/low priority)

Table 48: EthMac_SendWithoutCnf() - Function Arguments

Function Return Values

Definition	Description
ETH_OKAY	Successful
ETH_ERR_INVALID_PARAMETER	Invalid parameter

Table 49: EthMac_SendWithoutCnf() - Function Return Values

2.6 Structure Definitions

2.6.1 ETHERNET_CONNECTION_STATE_T - Link Status Structure

ETHERNET_CONNECTION_STATE_T

Name	Type	Description
uSpeed	unsigned int	SPEED (100/10)
uIsLinkUp	unsigned int	LINK state (!=0 -> Link UP)
uIsFullDuplex	unsigned int	DUPLEX state (!=0 -> FDX)

Table 50: ETHERNET_CONNECTION_STATE_T - Structure

2.6.2 ETHERNET_FRAME_T - Ethernet Frame Structure

ETHERNET_FRAME_T

Name	Type	Description
tDstAddr	ETHERNET_MAC_ADDR_T	Destination MAC address (DA)
tSrcAddr	ETHERNET_MAC_ADDR_T	Source MAC address (SA)
usType	uint16_t	Frame length/type (LT)
abData[1504]	uint8_t	Frame data excluding DA,SA,LT,FCS
abRes[18]	uint8_t	reserved, shall be zero
ulTimestampNs	uint32_t	receive time stamp [nanoseconds]
ulTimestampS	uint32_t	receive time stamp [s]

Table 51: ETHERNET_FRAME_T - Structure

2.6.3 ETHMAC_COUNTERS_T - Ethernet Counter Structure

ETHMAC_COUNTERS_T

Name	Type	Description
ulETHMAC_OUT_FRAMES_OKAY	uint32_t	count of frames that are transmitted successfully
ulETHMAC_OUT_OCTETS	uint32_t	count of bytes transmitted (without Preamble, SFD and FCS)
ulETHMAC_SINGLE_COLLISION_FRAMES	uint32_t	count of frames that are involved into a single collision
ulETHMAC_MULTIPLE_COLLISION_FRAMES	uint32_t	count of frames that are involved into more than one collisions
ulETHMAC_LATE_COLLISIONS	uint32_t	count of the times that a collision has been detected later than 512 bit times into the transmitted packet
ulETHMAC_LINK_DOWN_DURING_TRANSMISSION	uint32_t	count of the times that a frame was discarded during link down
ulETHMAC_UTX_UNDERFLOW_DURING_TRANSMISSION	uint32_t	UTX FIFO underflow at transmission time
ulETHMAC_IN_FRAMES_OKAY	uint32_t	count of frames that are received without any error
ulETHMAC_IN_OCTETS	uint32_t	count of bytes in valid MAC frames received excluding Preamble, SFD and FCS
ulETHMAC_FRAME_CHECK_SEQUENCE_ERRORS	uint32_t	count of frames that are an integral number of octets in length and do not pass the FCS check
ulETHMAC_ALIGNMENT_ERRORS	uint32_t	count of frames that are not an integral number of octets in length and do not pass the FCS check
ulETHMAC_FRAME_TOO_LONG_ERRORS	uint32_t	count of frames that are received and exceed the maximum permitted frame size
ulETHMAC_RUNT_FRAMES_RECEIVED	uint32_t	count of frames that have a length between 42..63 bytes and a valid CRC
ulETHMAC_COLLISION_FRAGMENTS_RECEIVED	uint32_t	count of frames that are smaller than 64 bytes and have an invalid CRC
ulETHMAC_FRAMES_DROPPED_DUE_LOW_RESOURCE	uint32_t	no empty pointer available at indication time
ulETHMAC_FRAMES_DROPPED_DUE_URX_OVERFLOW	uint32_t	URX FIFO overflow at indication time
ulETHMAC_TX_FATAL_ERROR	uint32_t	counts unknown error numbers from TX xMAC, should never occur
ulETHMAC_RX_FATAL_ERROR	uint32_t	counts unknown error numbers from RX xMAC, should never occur

Table 52: ETHMAC_COUNTERS_T - Structure

2.7 Enumeration Definitions

2.7.1 `ETH_MAC_ADDRESS_TYPE_E` - MAC addresses

Describes the different types of MAC addresses.

ETH_MAC_ADDRESS_TYPE_E

Definition	Description
`ETH_MAC_ADDRESS_CHASSIS`	Primary Chassis MAC address
`ETH_MAC_ADDRESS_2ND_CHASSIS`	Secondary Chassis MAC address

Table 53: `ETH_MAC_ADDRESS_TYPE_E` - Enumeration

2.7.2 ETHERNET_PHY_LED_CFG_E - PHY LED Configuration

ETHERNET_PHY_LED_CFG_E

Definition	Description
ETH_PHY_LED_STATIC	separate link and activity LEDs
ETH_PHY_LED_BLINK	separate link and activity LEDs, activity blinking when active
ETH_PHY_LED_SINGLE	single LED, combined link and blink on activity
ETH_PHY_LED_OFF	PHY LEDs are disabled

Table 54: ETHERNET_PHY_LED_CFG_E - Enumeration

2.7.3 ETHERNET_RESULT - Result Codes for Ethernet Functions

ETHERNET_RESULT

Definition	Description
ETH_OKAY	Successful
ETH_ERR_FIFO_EMPTY	The FIFO is empty
ETH_ERR_INIT_FAILED	Initialization has failed
ETH_ERR_INVALID_PARAMETER	Invalid parameter
ETH_ERR_TX_SUCCESSFUL_WITH_RETRIES	Transmission successful with retries
ETH_ERR_TX_FAILED_LATE_COLLISION	Transmission failed due late collision
ETH_ERR_TX_FAILED_LINK_DOWN_DURING_TX	Transmission failed due link down
ETH_ERR_TX_FAILED_EXCESSIVE_COLLISION	Transmission failed due excessive collisions
ETH_ERR_TX_FAILED_UTX_UFL_DURING_TX	Transmission failed due UTX FIFO underflow
ETH_ERR_TX_FAILED_FATAL_ERROR	Transmission failed due fatal error
ETH_ERR_INVAL_STATE	Invalid port state
ETH_ERR_OUT_OF_MEMORY	Not enough resources

Table 55: ETHERNET_RESULT - Enumeration

3 Appendix

3.1 netX100/500 connection to external PHY

Std Func	Mux Func	MUX Select	Notes
pio26	mii2_rxd0	sel_mii2	MII
pio29	mii2_rxd1	sel_mii2	MII
pio27	mii2_rxd2	sel_mii2	MII
pio28	mii2_rxd3	sel_mii2	MII
pio30	mii2_rxdv	sel_mii2	MII
pio8	mii2_rxclk	sel_mii2	MII
pio9	mii2_rxer	sel_mii2	MII
pio10	mii2_crs	sel_mii2	MII
pio11	mii2_col	sel_mii2	MII
pio12	mii2_txd0	sel_mii2	MII
pio13	mii2_txd1	sel_mii2	MII
pio14	mii2_txd2	sel_mii2	MII
xm2_rx	mii2_txd3	sel_mii2	MII
xm2_tx	mii2_txen	sel_mii2	MII
xm2_io0	mii2_txclk	sel_mii2	MII
xm2_io1	mii2_txer	sel_mii2	MII
pio16	mii_mdio	sel_mii23	MDIO
pio17	mii_mdc	sel_mii23	MDIO
pio0	mii2_led0	sel_led_mii2	Static link signal
pio1	mii2_led1	sel_led_mii2	Not used
pio2	mii2_led2	sel_led_mii2	Not used
pio3	mii2_led3	sel_led_mii2	Not used
rst_out_n			Optional when external PHY shall be reset via netX100/500
clk_out			Optional when netX100/500 shall be clock source for external PHY

Table 56: netX100/500 pinning and multiplex options for external PHY connected to XC port 2

3.2 Usable PHYs

Micrel KSZ8041NL

- Connect mii2_led0 to LED0/NWAYEN pin
- use LED mode [01] to get static link signal (low-active) at LED0 (required)
- Set PHY address via strapping unequal internal Dual-PHY address
- Set CONFIG0/1/2 to MII mode via strapping
- Disable ISO via strapping
- Set SPEED/DUPLEX/NWAYEN to requested values via strapping
- get FullDuplex status via MDIO read access to Register 31 (1fh) Bit 4
- get Speed status via MDIO read access to Register 31 (1fh) Bit 3
- get Link status via MDIO read access to Register 1 (01h) Bit 2

National DP83848I

- Connect mii2_led0 to LED_LINK/AN0 pin
- configure LED mode 1 (LED_CFG[0]=1) to get static link signal (low-active) via strapping at pin 40 or via MDIO after PHY power on (required)
- Set PHY address via strapping unequal internal Dual-PHY address
- Disable ISO via strapping
- get FullDuplex status via MDIO read access to Register 16 (10h) Bit 2
- get Speed status via MDIO read access to Register 16 (10h) Bit 1 (invert!)
- get Link status via MDIO read access to Register 16 (10h) Bit 0

Broadcom BCM5241

- Connect mii2_led0 to LED1
- configure LED1=Link, LED2=Activity to get static link signal (low-active) via strapping at LED1/2 or via MDIO after PHY power on
- Set PHY address via strapping unequal internal Dual-PHY address
- Disable ISO via strapping
- Set F100/AN<EN/STANDBY to requested values via strapping
- get FullDuplex status via MDIO read access to Register 24 (18h) Bit 0
- get Speed status via MDIO read access to Register 24 (18h) Bit 1
- get Link status via MDIO read access to Register 1 (01h) Bit 2

3.3 PHY Latencies

100BaseTX:

PHY	Reception (Ingress)	Transmission (Egress)
netX10/50/100/500 internal PHY	288 ns + PHY Phase Offset (0/8/16/24/32 ns) MII sample delay = 20 ns	72 ns MII sample delay = 20 ns
netX51/52/4000 internal PHY	215 ns	34 ns
netX90 internal PHY	220 ns + PHY Phase Offset (0/8/16/24/32 ns)	116 ns
Broadcom BCM5241 external PHY	170 ns	57 ns

The MAC takes the timestamps in ingress and egress direction at reception/transmission of SFD at MII. For some netX types MII sample delays must be taken into account.

- Ingress: Timestamp_corrected = Timestamp – MII sample delay
- Egress: Timestamp_corrected = Timestamp + MII sample delay

10BaseT

There are no delay values available for 10BaseT.

PHY Phase Offset

- “Phase Indicator”, is for only “100BASE-TX” and “100BASE-FX” modes
- The value of “Phase Indicator” can be detected upon the first received packet after asserting “LINKLED” and kept until link is down.
- netX10/50/100/500: The value of “Phase Indicator” can be read via SMI (System Management Interface) at register 27 Bits 10:8 (000/001/010/011/100 = 0/8/16/24/32 ns)
- netX90: The value of “Phase Indicator” can be read from register *Int_phy_cfg_status* (000/001/010/011/100 = 0/32/24/16/8 ns)

IEEE1588 V2: Timestamp reference plane

Ingress:

All PTP event messages are time stamped on ingress. The timestamp shall be the time at which the event message timestamp point passes the reference plane (boundary between PTP node and network). This implementation generates event message timestamps at detection of SFD at MII. Use this parameter to correct appropriately.

ingressTimestamp = ingressMeasuredTimestamp – ingressLatency

Normally ingressLatency contains + PHY receive delay + eventually PHY phase offsets

Egress:

All PTP event messages are time stamped on egress. The timestamp shall be the time at which the event message timestamp point passes the reference plane (boundary between PTP node and network). This implementation generates event message timestamps at detection of SFD at MII. Use this parameter to correct appropriately.

egressTimestamp = egressMeasuredTimestamp + egressLatency

Normally engressLatency contains + PHY transmit delay

3.4 Ethernet MAC availability regarding XC Ports

netX	XC Port	Internal PHY	External PHY	Note
netX100/500	0	X	-	
	1	X	-	
	2	-	X	Static Link signal from PHY shall be low-active
	3	-	-	
netX50	0	X	-	
	1	X	-	
netX5	0	-	X	Static Link signal from PHY shall be low-active
	1	-	X	Static Link signal from PHY shall be low-active
netX10	0	X	-	
netX51/52	0	X	-	
	1	X	-	
netX4000	0	X	-	
	1	X	-	
	2	-	X	If Static Link signal from PHY is low-active then invert it via MMIO Configuration Register
	3	-	X	If Static Link signal from PHY is low-active then invert it via MMIO Configuration Register
netX90	0	X	X	If Static Link signal from external PHY is low-active then invert it via ASIC_CTRL.PhyCtrl0 Register
	1	X	X	If Static Link signal from external PHY is low-active then invert it via ASIC_CTRL.PhyCtrl0 Register

Table 57: Ethernet MAC availability regarding XC ports

3.5 List of Tables

3.6 List of Figures

3.7 Contacts

Headquarters

Germany
Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
E-Mail: info@hilscher.in

Italy
Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it

Support
Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp

Support
Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea
Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us

Support
Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com