HAL

Ethernet Switch

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

V6.3.x.x

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC090304HAL12EN | Revision 12 | English | 2021-09 | Released | Public

Table of Contents

1 Introduction

1.1 About this Document

This manual describes the interface of the Ethernet Switch 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	2011-09-15	AO		Added functions for setting/getting 2nd MAC address
5	2012-02-08	AO		Added netX51 support
6	2013-09-11	AO		Changes to V5.0.x.x, DLR support
7	2014-02-14	BI		Changes to V5.1.x.x, changed function parameters
8	2016-03-08	AO		Changes to V6.0.x.x; no change of HAL API
9	2016-12-20	AO		Changes to V6.1.x.x Added netX4000 and netX90 support
10, 11	2017-05-24	BI, AO	2.3	Changes to V6.2.x.x
12	2021-09-07	BI,MM		Changes to V6.3.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
DLR	Device Level Ring Protocol
DSCP	Differentiated Services Code Point
EIP	EtherNet/IP
PDU	Protocol Data Unit
BPDU	Bridge PDU

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
2	THE CIP NETWORKS LIBRARY - Volume 2, EtherNet/IP Adaptation of CIP, Edition 1.15, April 2013

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 Switch
- Getting of Status Information of the Ethernet Switch
- Sending of Ethernet frame transmission requests to the Ethernet Switch
- Getting of Confirmations about processed transmission processes
- Getting of Ethernet frame indications from the Ethernet Switch
- Configuration of link/activity LED behavior for application specific use
- Generating of cyclic synchronization signals and interrupts

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
- Knowledge of the EtherNet/IP specification

1.5.3 Technical Data

- 2 integrated MACs, each 10BASE-T / 100BASE-TX/FX operation in full/half duplex
- Integrated Dual-PHY with MDIX and Auto-Negotiation capability
- Direct Access to PHY status information link, duplex and speed
- Quality of Service capable: 2 Queues (Traffic Classes)
  - Prioritization via IEEE 802.1Q/D (based on VLAN-Tag Priority) and DSCP
  - For IP frames the DSCP value overrules IEEE 802.1Q/D priority
  - For non-IP frames the priority in the 802.1Q header is be used
- Multicast pre-filtering capable based on 12 Bit hash value
- Number of Ethernet frame buffers: netX50/100/500: 40, netX6/51/52/4000/90: 82
- Configurable Link and Activity LED behavior
- Optional confirmations of processed transmission requests
- Dynamic learning based on 12-Bit hashing, aging
  - 12 bit Hash algorithm: SA[47:40|35:32] xor SA[31:24|19:16]
- Forwarding using „Cut-Through“ mechanism when possible
- Possibility to filter for 2 local interface MAC addresses
- Remove device’s own frames from network when receiving
- IEEE 1588 support
  - Receive and transmit time-stamping for every frame to allow support of IEEE1588 ordinary/boundary/transparent clocks
  - Timestamp precision netX10/50/100/500: 10 ns, netX51/4000/90: 1.25 ns
  - Implement IEEE1588 V2 End-to-End transparent clock
  - Generation of two cyclic events (external trigger and optional interrupts) based on IEEE1588 clock
- Filtering/Forwarding of Bridge PDUs adjustable
- EtherNet/IP Device Level Ring (DLR) Protocol support
  - Support for Beacon-based ring node
  - Special DLR Preserve IEEE 802.Q VLAN ID and tag priority of ring protocol frames
  - Mechanism to flag the port through which such a frame was received from ring
  - Mechanism to forward such frames from host CPU on to ring only through the port it was intended to go out.
  - Mechanism to allow sending of frames on both ports
  - Flush Unicast MAC address tables on ring state transitions
  - Unicast MAC address of self is not purged when MAC address table is flushed
  - Implement of Interface Counters and Media Counters
  - Possibility to disable ring ports for link debounce purpose

1.5.4 Limitations

- No frame buffer management - each Ethernet frame occupies 1560 Bytes Buffer
- No Gigabit operation
- No MAC-pause mechanism in full-duplex, no back-pressure in half-duplex
- No Broadcast-storm control
- No Static learning
- No EtherNet/IP Device Level Ring (DLR) Ring Supervisor support

1.5.5 Ethernet Switch Port Block Diagram

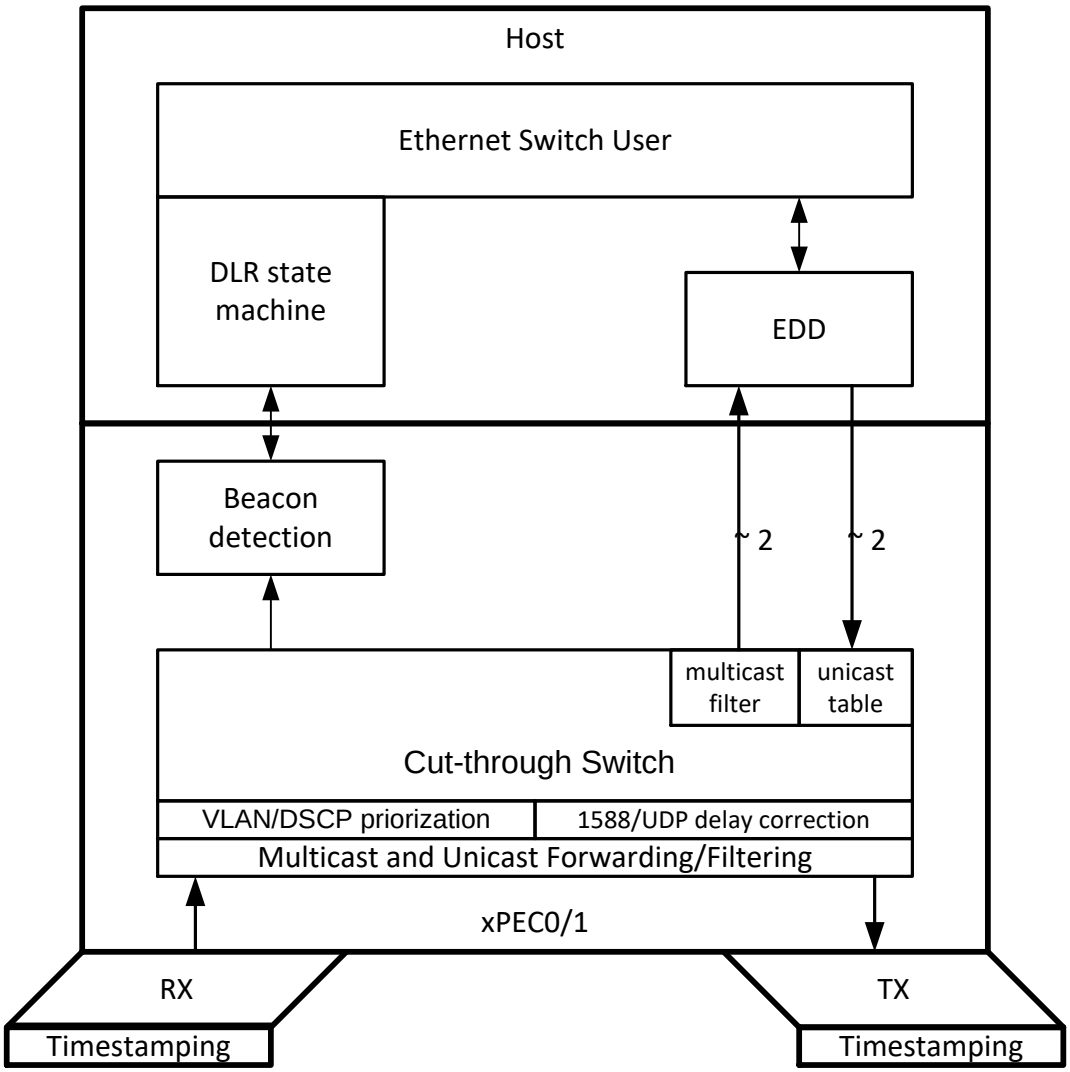

Figure 1: Ethernet Switch Block Diagram

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 Switch user with their associated service primitives and parameters.

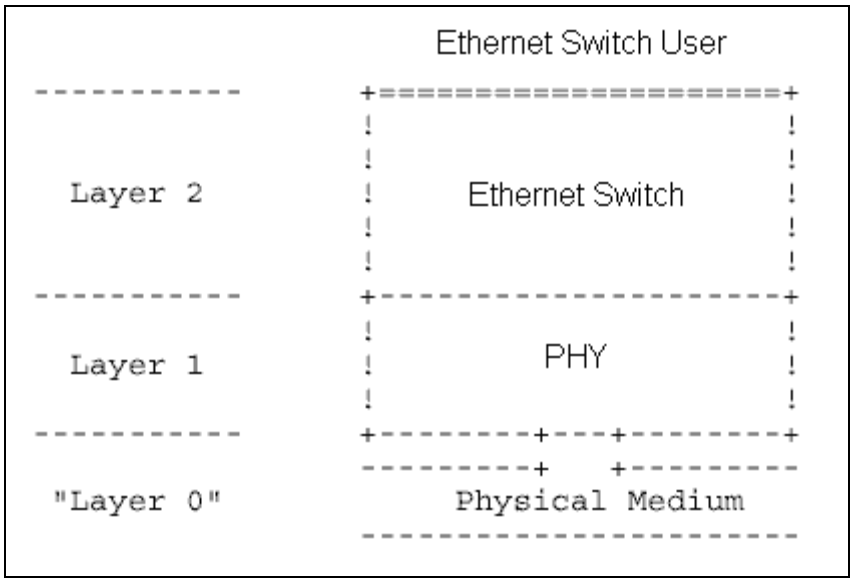

Figure 2: Interface between Interface User and Interface in Relation to Layer Model

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 Switch user to the Ethernet Switch.

Status

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

Communication

This service class defines the transmission of an Ethernet frames.

Precision Time Protocol

This service class defines the synchronization to an external master clock.

Cyclic Events

This service class defines the generation of cyclic interrupt requests and output signals.

EtherNet/IP DLR

This service class defines functions specific for the Device Level Ring protocol.

2.2 Overview of Functionality

2.2.1 FIFO usage

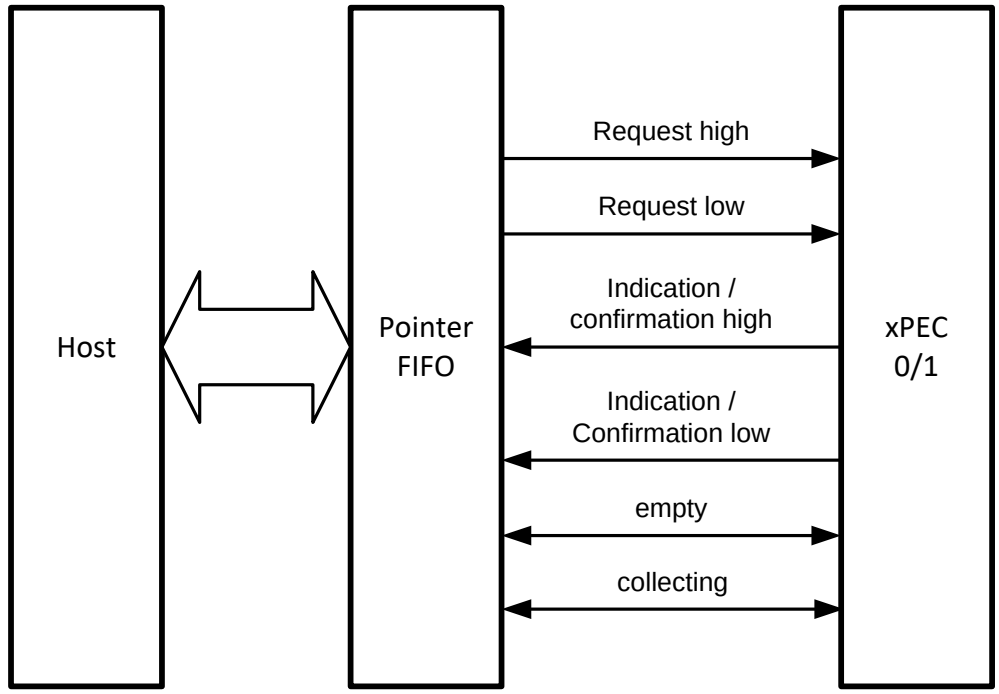

Figure 3: FIFO usage

FIFO numbers 0 to 15 are used for interfacing between ARM and the Ethernet Switch.

2.2.2 Signaling

Following signals are exchanged between xPEC and host (via xpec_irq register):

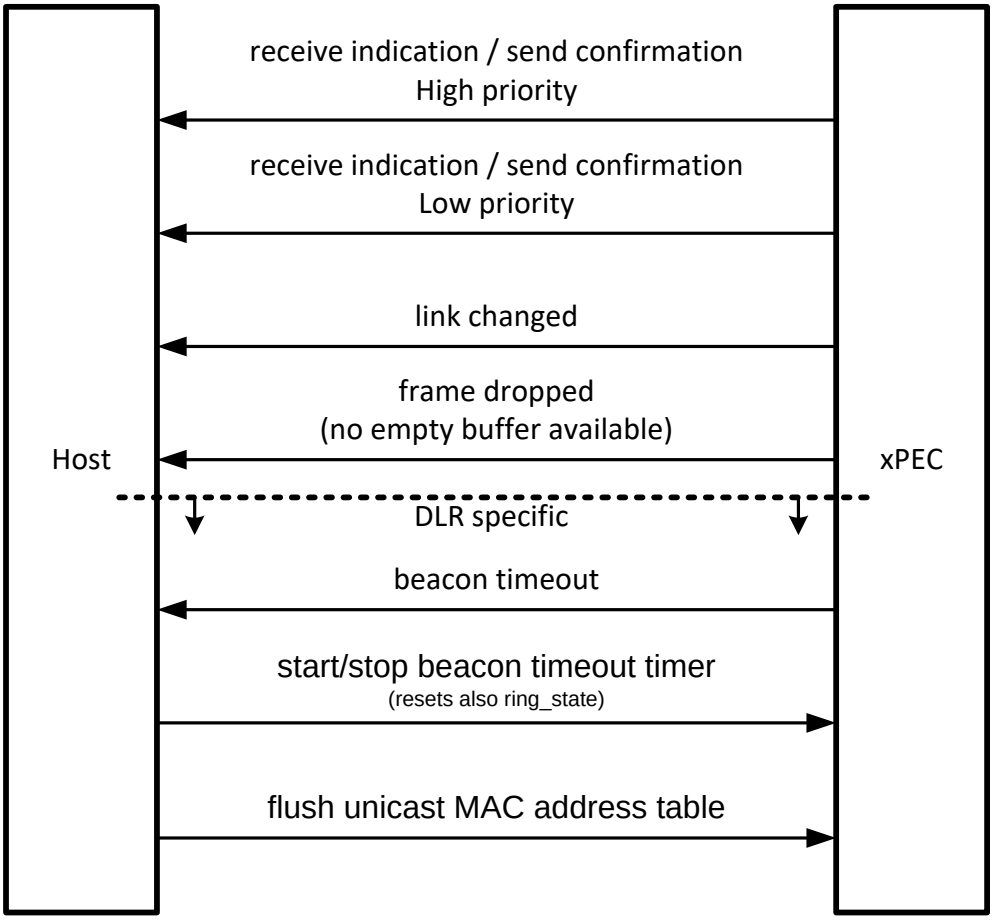

Figure 4: Signaling between xPEC and host

2.2.3 Filtering Database

The switch contains following Filtering Database (FDB). Please note that forwarding behavior depends on configured switch modes “DLR support” and “BPDU support”.

	DA				Traffic Class	Traffic Class
					w/o DLR support	with DLR-Support
FRAME_TYPE		VLAN-Tag.Prio	LT	LEARNING	Indication/Forwarding***	Indication/Forwarding***
DLR: Beacon	01-21-6C-00-00-01	7*	0x80E1*	w/o DLR support :YES with DLR support : NO	Hi** / Hi	Hi**** / Hi
DLR: Neighbor_Check_Req, Neighbor_Check_Rsp, Sign_On	01-21-6C-00-00-02	7*	0x80E1*	w/o DLR support :YES with DLR support : NO	Hi** / Hi	Hi / -
DLR: Announce, Locate_Fault, Flush_Tables	01-21-6C-00-00-03	7*	0x80E1*	w/o DLR support :YES with DLR support : NO	Hi** / Hi	Hi / Hi
DLR: Advertise	01-21-6C-00-00-04	7*	0x80E1*	w/o DLR support :YES with DLR support : NO	Hi** / Hi	Hi / Hi
DLR: Learning_Update	01-21-6C-00-00-05	7*	0x80E1*	w/o DLR support :YES with DLR support : NO	Hi** / Hi	Hi / Hi
DLR: Others	01-21-6C-00-xx-xx	7*	0x80E1*	w/o DLR support :YES with DLR support : NO	Hi** / Hi	Hi / Hi
BPDU (RSTP,LLDP,...)	01-80-C2-00-xx-xx	NON, 0..3		w/o BPDU support: YES with BPDU support: NO	w/o BPDU support: Lo** / Lo with BPDU support: Lo / -	
	01-80-C2-00-xx-xx	4..7		w/o BPDU support: YES with BPDU support: NO	w/o BPDU support: Hi** / Hi with BPDU support: Hi / -	
OTHERS	UC Match	NON, 0..3		Yes	Lo / -	
	UC Match	4..7		Yes	Hi / -	
	UC Mismatch	NON, 0..3		Yes	- / Lo	
	UC Mismatch	4..7		Yes	- / Hi	
	BC	NON, 0..3		Yes	Lo / Lo	
	BC	4..7		Yes	Lo / Lo	
	MC	NON, 0..3		Yes	Lo** / Lo	
	MC	4..7		Yes	Hi** / Hi	

NOTE:

UC_MATCH : (DA == 1st/2nd CHASSIS MAC)

UC_MISMATCH : (DA != 1st/2nd CHASSIS MAC)

* .. Fixed according EIP specification

** .. If reception of Multicasts is enabled, else "-"

*** .. Forwarding to other bridge ports must be enabled globally

**** .. Depends on Beacon-Based ring node FSM

Note: If IP.UDP.SrcPort==IP.UDP.DstPort==PTP_Event then update PTP correction field

Figure 5: Filtering Database

2.2.4 Receive Process

The receive xPEC will parse all incoming Ethernet telegrams for specific fields in order to detect Beacon and PTP_event frames and process these within the xPEC.

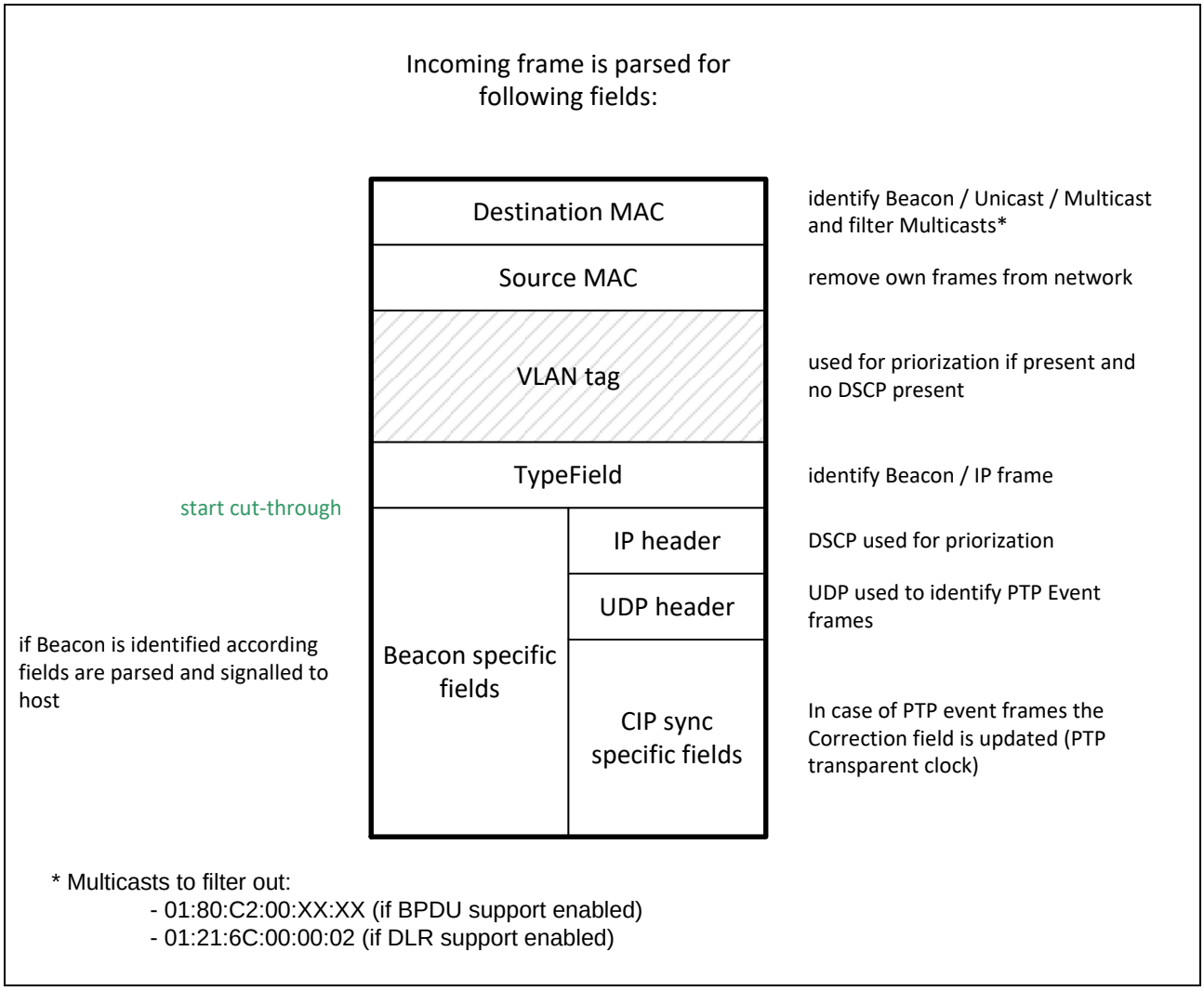

Figure 6: Information in received frame

2.2.5 Transmit Process

The transmit xPEC will send out all frames in send queues as they are, without any changes.

One exception to this rule exists:

In IEEE1588 V2 PTP_event frames the correction field is updated according to the bridge delay (the time the frame has reside in the switch during forwarding).

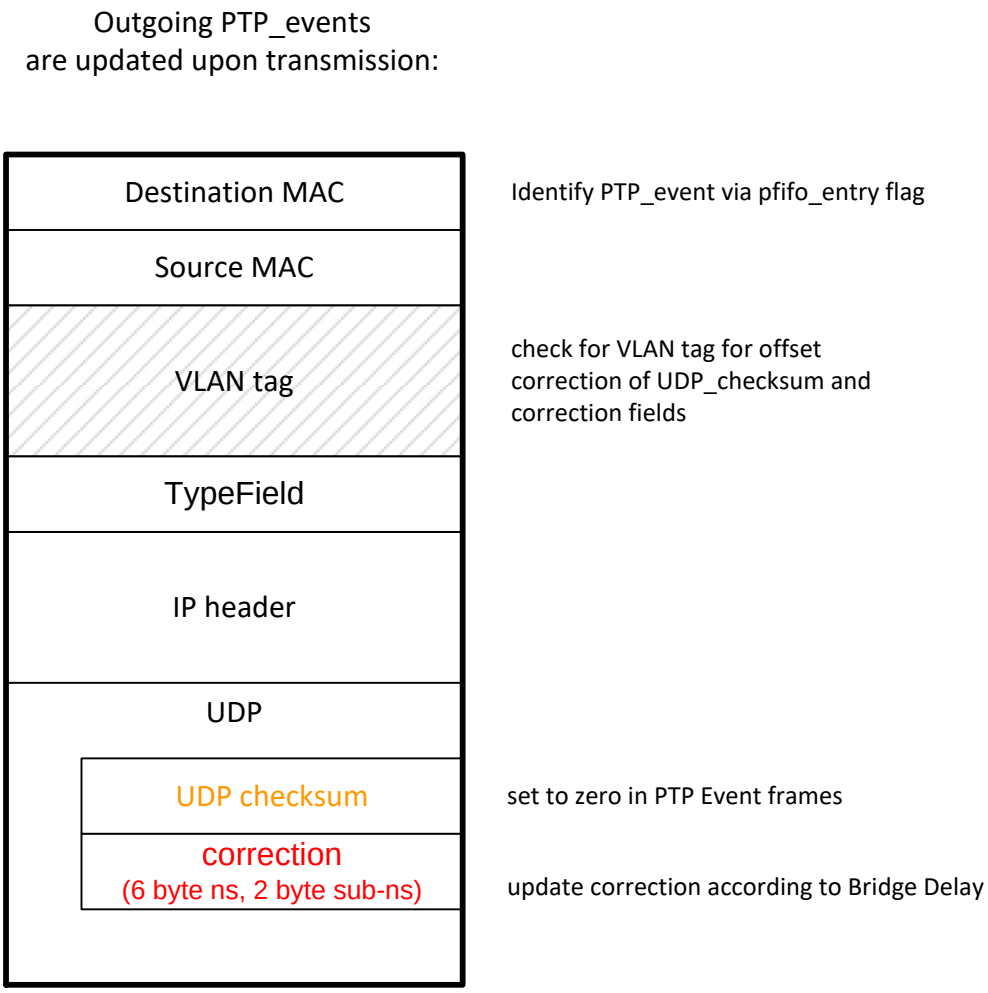

Figure 7: Information sent in PTP_event frame

2.2.6 DLR State-Machine and Beacon Detection

In order to reduce load on the host CPU for beacon based ring-nodes, Beacon frames are pre-processed within the xPEC and only passed to the host CPU when interaction is required.

For this purpose the state-machines, “State-Event-Action Table for Beacon Based Non-Supervisor Ring Node” and “State-Event-Action Table for Ring Supervisor Node” has been split in two parts, one called Beacon-detection, which is running in xPEC, and the other part, which implements the according state-machine on the host CPU. For Announce-based nodes there exists no xPEC support, the state-machine of such nodes must run only on the host CPU.

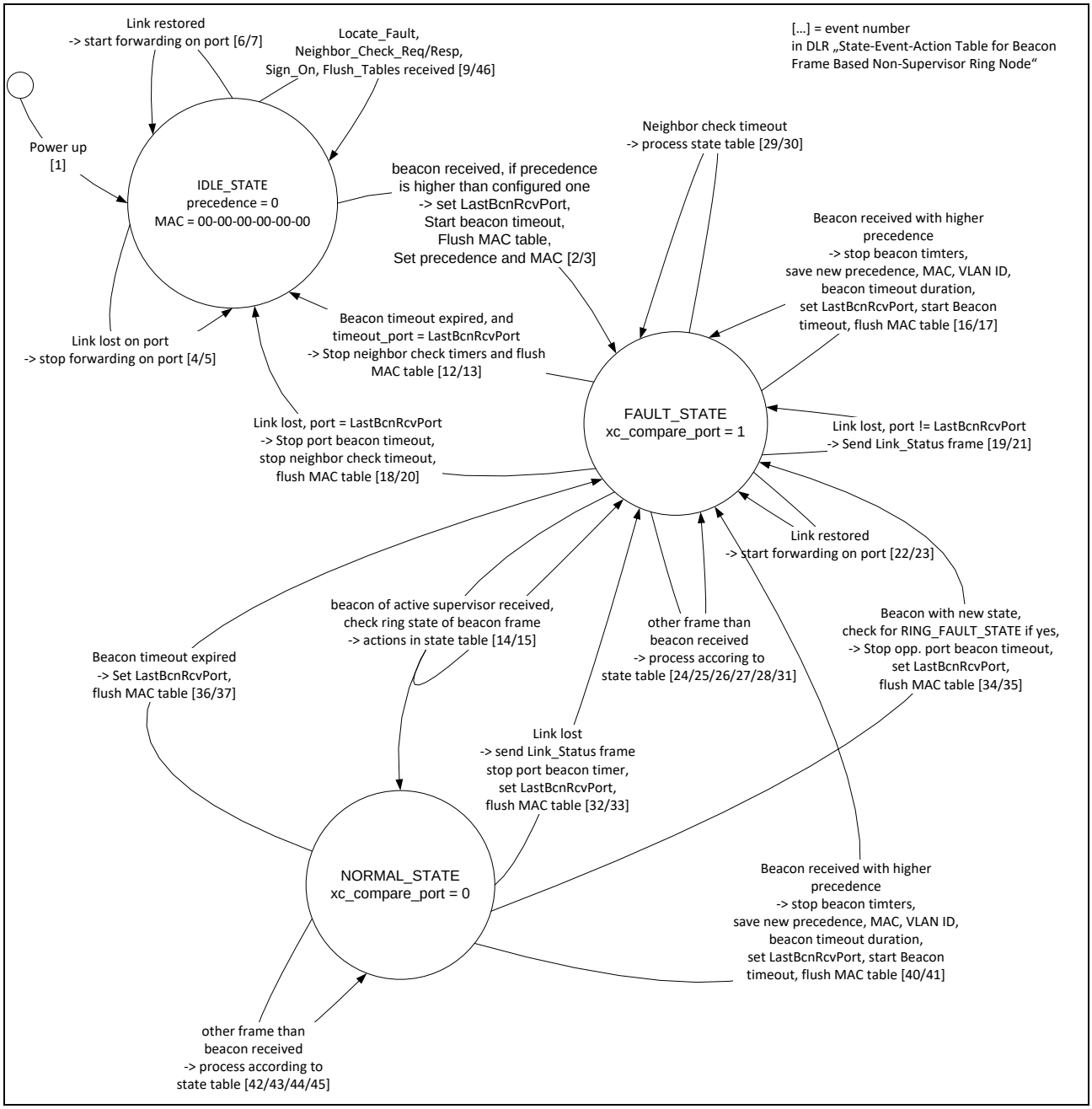

Figure 8: Beacon-based node state-machine (host part)

xPEC parses every beacon frame it receives and generates events to host CPU only when interaction is required. This reduces load on host. All actions taken on Beacon frames are shown in the following figure.

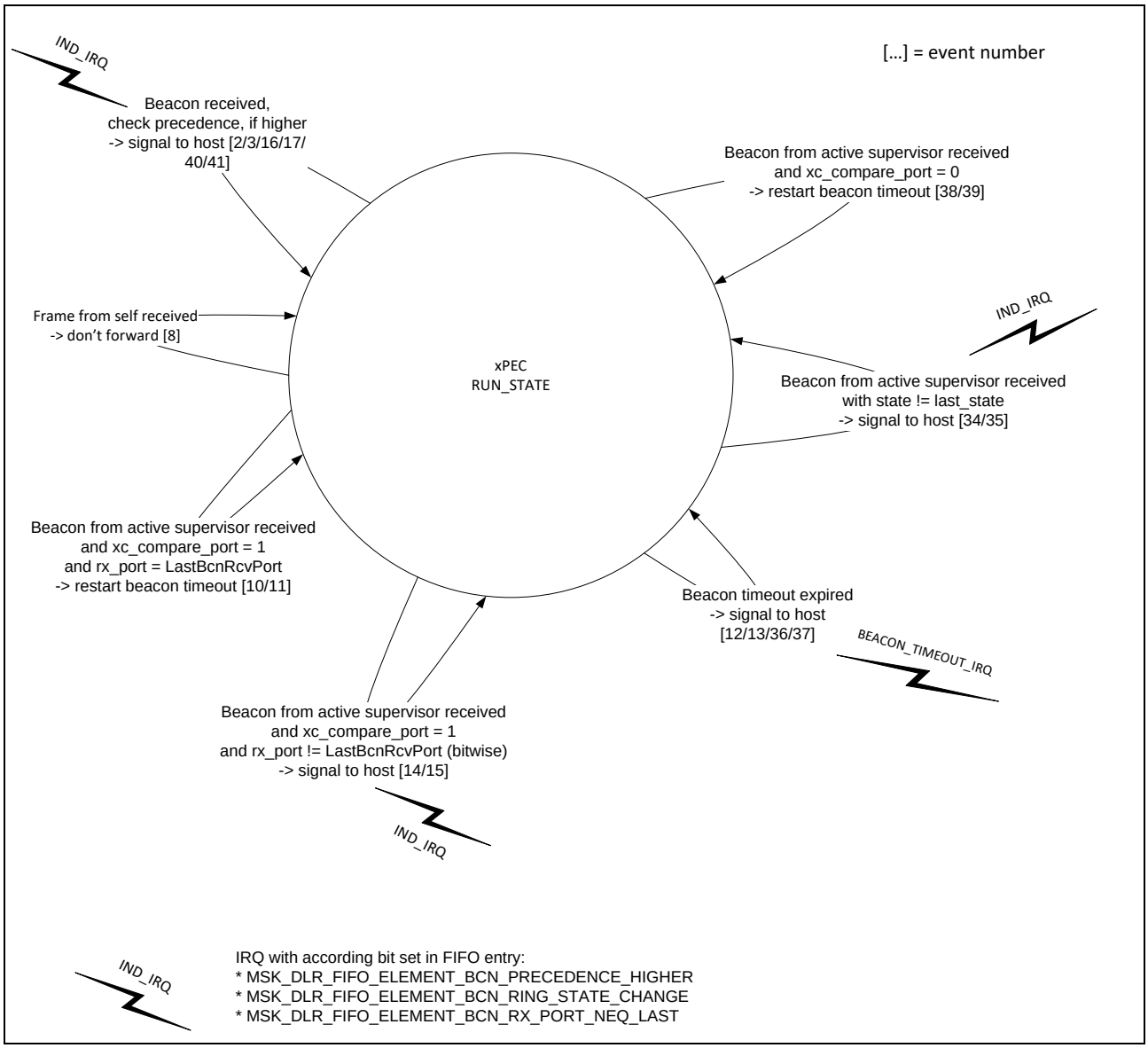

Figure 9: Beacon state-machine (xPEC part)

2.3 Communication Service Class

2.3.1 `Eth2PS_GetFrame()` - Get Empty Ethernet Frame Block

Gets an element from the empty pointer FIFO.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_GetFrame( ETH2PS_FRAME_HANDLE_T*  ptFrame )
```

Function Arguments

Argument	Direction	Description
`ptFrame`	[out]	Frame handle structure filled out by the function on success

Table 4: `Eth2PS_GetFrame()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_FIFO_EMPTY`	Indication FIFO is empty

Table 5: `Eth2PS_GetFrame()` - Function Return Values

2.3.2 `Eth2PS_GetIndCnf()` - Get Indication/Confirmation

Retrieves a received frame (indication) or a transmitted frame (confirmation) from the combined indication/confirmation FIFO.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_GetIndCnf( unsigned int            uPriority,
                  ETH2PS_FRAME_HANDLE_T*  ptFrame,
                  ETH2PS_FRAME_INFO_T*    ptFrameInfo )
```

Function Arguments

Argument	Direction	Description
`uPriority`	[in]	0: low priority, 1: high priority
`ptFrame`	[out]	Frame handle filled by the function
`ptFrameInfo`	[out]	Additional frame information

Table 6: `Eth2PS_GetIndCnf()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_FIFO_EMPTY`	Indication FIFO is empty

Table 7: `Eth2PS_GetIndCnf()` - Function Return Values

2.3.3 `Eth2PS_InitFrameHandleFromFifoEntry()` - Get fully qualified frame handle from FIFO entry

Initialize a fully qualified frame handle from FIFO entry.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_InitFrameHandleFromFifoEntry( uint32_t                ulFifoEntry,
                                     ETH2PS_FRAME_HANDLE_T*  ptFrame )
```

Function Arguments

Argument	Direction	Description
`ulFifoEntry`	[in]	FIFO entry
`ptFrame`	[out]	Frame handle structure filled out by the function on success

Table 8: `Eth2PS_InitFrameHandleFromFifoEntry()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_INVAL_PARAM`	Invalid parameter in function call

Table 9: `Eth2PS_InitFrameHandleFromFifoEntry()` - Function Return Values

2.3.4 `Eth2PS_ReleaseFrame()` - Release Ethernet Frame Buffer

Puts an Ethernet frame buffer back into the empty pointer FIFO.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_ReleaseFrame( ETH2PS_FRAME_HANDLE_T*  ptFrame )
```

Function Arguments

Argument	Direction	Description
ptFrame	[in]	Handle of Ethernet frame

Table 10: `Eth2PS_ReleaseFrame()` - Function Arguments

Function Return Values

Definition	Description
ETH2PS_OKAY	Successful
ETH2PS_ERR_INVAL_PARAM	Invalid parameter in function call

Table 11: `Eth2PS_ReleaseFrame()` - Function Return Values

2.3.5 `Eth2PS_Send()` - Send Ethernet Frame

Initiates a transmission request.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_Send( unsigned int             uiPortInfo,
             ETH2PS_FRAME_HANDLE_T*   ptFrame,
             unsigned int             uPriority,
             bool                     fConfirmationEn,
             unsigned int*            puCnfCnt )
```

Function Arguments

Argument	Direction	Description
`uiPortInfo`	[in]	0: Transmit on port 0 only
`ptFrame`	[in]	Frame handle of the frame to be transmitted
`uPriority`	[in]	0: low priority, 1: high priority
`fConfirmationEn`	[in]	false: Frame is released automatically after transmission
`puCnfCnt`	[out]	Number of confirmations produced by the transmission.

Table 12: `Eth2PS_Send()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful

Table 13: `Eth2PS_Send()` - Function Return Values

2.3.6 `Eth2PS_SetFrameLengthFromFifoEntry()` - Set frame length from FIFO entry

Set frame length within xPEC DRAM regarding frame buffer number got by FIFO entry.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_SetFrameLengthFromFifoEntry( uint32_t  ulFifoEntry,
                                    uint16_t  usLength )
```

Function Arguments

Argument	Direction	Description
`ulFifoEntry`	[in]	FIFO entry
`usLength`	[in]	Frame length

Table 14: `Eth2PS_SetFrameLengthFromFifoEntry()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_INVAL_PARAM`	Invalid parameter in function call

Table 15: `Eth2PS_SetFrameLengthFromFifoEntry()` - Function Return Values

2.4 Control Service Class

2.4.1 `Eth2PS_AddGroupAddr()` - Add Group Address

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

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_AddGroupAddr( const ETH2PS_MAC_ADDR_T  tMacAddr )
```

Function Arguments

Argument	Direction	Description
tMacAddr	[in]	Multicast MAC address value

Table 16: `Eth2PS_AddGroupAddr()` - Function Arguments

Function Return Values

Definition	Description
ETH2PS_OKAY	Successful
ETH2PS_ERR_OUT_OF_MEMORY	Not enough resources
ETH2PS_ERR_INVAL_PARAM	Invalid parameter in function call

Table 17: `Eth2PS_AddGroupAddr()` - Function Return Values

2.4.2 `Eth2PS_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

```
ETH2PS_RESULT_E
Eth2PS_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 18: `Eth2PS_CfgMii()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_INVAL_PARAM`	Invalid parameter in function call

Table 19: `Eth2PS_CfgMii()` - Function Return Values

2.4.3 `Eth2PS_DeleteGroupAddr()` - Delete Group Address

Delete the given Multicast group address from the address recognition.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_DeleteGroupAddr( const ETH2PS_MAC_ADDR_T  tMacAddr )
```

Function Arguments

Argument	Direction	Description
tMacAddr	[in]	Multicast MAC address value

Table 20: `Eth2PS_DeleteGroupAddr()` - Function Arguments

Function Return Values

Definition	Description
ETH2PS_OKAY	Successful
ETH2PS_ERR_INVAL_PARAM	Invalid parameter in function call
ETH2PS_ERR_INVAL_STATE	Invalid port state

Table 21: `Eth2PS_DeleteGroupAddr()` - Function Return Values

2.4.4 `Eth2PS_FlushLearningTable()` - Flush Learning Table

Deletes all entries in the MAC address learning table. The switch ”forgets” all received source MAC addresses.

Function Prototype

```
void
Eth2PS_FlushLearningTable( void )
```

2.4.5 `Eth2PS_FlushLearningTablePort()` - Flush Learning Table at port

Deletes all entries in the MAC address learning table of a port. The switch ”forgets” all received source MAC addresses.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_FlushLearningTablePort( unsigned int  uiPort )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	Switch port number

Table 22: `Eth2PS_FlushLearningTablePort()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_INVAL_PORT`	Invalid switch port number

Table 23: `Eth2PS_FlushLearningTablePort()` - Function Return Values

2.4.6 `Eth2PS_Initialize()` - Initialize 2-Port Switch

Initializes the switch and configures it with the default parameter settings.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_Initialize( ETH2PS_CFG_T*  ptCfg,
                   void*          pvUser )
```

Function Arguments

Argument	Direction	Description
`ptCfg`	[in]	HAL configuration
`pvUser`	[in]	User specific parameter

Table 24: `Eth2PS_Initialize()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_INIT_FAILED`	Error occurred during initialization

Table 25: `Eth2PS_Initialize()` - Function Return Values

2.4.7 `Eth2PS_SetLinkMode()` - Set Link Mode

This function sets the link mode of a switch port. 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

```
ETH2PS_RESULT_E
Eth2PS_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 26: `Eth2PS_SetLinkMode()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_INVAL_PARAM`	Invalid parameter in function call

Table 27: `Eth2PS_SetLinkMode()` - Function Return Values

2.4.8 `Eth2PS_SetMacAddr()` - Set MAC Address

Sets a MAC address. Note: The Chassis MAC addresses shall be set before the switch is started. Note: The DLR supervisor address is used for Beacon handling, but only if DLR support is enabled (see parameter ETH2PS_PARAM_DLR_SUPPORT_ENABLE).

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_SetMacAddr( ETH2PS_MAC_ADDRESS_TYPE_E  eType,
                   const ETH2PS_MAC_ADDR_T    tMacAddr )
```

Function Arguments

Argument	Direction	Description
eType	[in]	Defines which MAC address shall be configured
tMacAddr	[in]	MAC address value

Table 28: `Eth2PS_SetMacAddr()` - Function Arguments

Function Return Values

Definition	Description
ETH2PS_OKAY	Successful
ETH2PS_ERR_INVAL_PARAM	Invalid parameter in function call

Table 29: `Eth2PS_SetMacAddr()` - Function Return Values

2.4.9 `Eth2PS_SetParameter()` - Set Parameter

Sets a parameter in the 2-Port Switch.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_SetParameter( unsigned int    uiPort,
                     ETH2PS_PARAM_E  eParam,
                     uint32_t        ulValue )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	Switch port number
`eParam`	[in]	Parameter to be set
`ulValue`	[in]	Value to set parameter to

Table 30: `Eth2PS_SetParameter()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_INVAL_PARAM`	Invalid parameter in function call

Table 31: `Eth2PS_SetParameter()` - Function Return Values

2.4.10 `Eth2PS_Start()` - Start 2-Port Switch

Confirms all pending interrupts and starts the 2-port switch.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_Start( void*  pvUser )
```

Function Arguments

Argument	Direction	Description
pvUser	[in]	User specific parameter

Table 32: `Eth2PS_Start()` - Function Arguments

Function Return Values

Definition	Description
ETH2PS_OKAY	Successful
ETH2PS_ERR_INIT_FAILED	Error occurred during initialization

Table 33: `Eth2PS_Start()` - Function Return Values

2.5 Cyclic Service Class

2.5.1 `Eth2PS_CyclicConfig()` - Configure the generation of Cyclic Events

This function sets all parameters for the generation of cyclic events.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_CyclicConfig( ETH2PS_CYCLIC_CFG_T*  ptCfg )
```

Function Arguments

Argument	Direction	Description
`ptCfg`	[in]	Parameter structure

Table 34: `Eth2PS_CyclicConfig()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_INVAL_PARAM`	Invalid parameter in function call

Table 35: `Eth2PS_CyclicConfig()` - Function Return Values

2.5.2 `Eth2PS_CyclicGetCnfIrq()` - Get Cyclic Interrupts

Retrieves and confirms the current interrupt requests from the synchronization unit.

Function Prototype

```
uint32_t
Eth2PS_CyclicGetCnfIrq( void )
```

Function Return Values

Bit mask of pending interrupts

2.5.3 `Eth2PS_CyclicInitialize()` - Initialize the generation of Cyclic Events

This function initializes the cyclic event generator. All other functions of the Cyclic Service Class have to be called after this function.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_CyclicInitialize( void*  pvUser )
```

Function Arguments

Argument	Direction	Description
`pvUser`	[in]	User specific parameter

Table 36: `Eth2PS_CyclicInitialize()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_INIT_FAILED`	Error occurred during initialization

Table 37: `Eth2PS_CyclicInitialize()` - Function Return Values

2.5.4 `Eth2PS_CyclicStart()` - Start Generation of Cyclic Events

Starts the generation of cyclic events.

Function Prototype

```
void
Eth2PS_CyclicStart( uint32_t  ulStartTimeNs )
```

Function Arguments

Argument	Direction	Description
`ulStartTimeNs`	[in]	Start time of the first cycle [ns] */

Table 38: `Eth2PS_CyclicStart()` - Function Arguments

2.5.5 `Eth2PS_CyclicStop()` - Reset Generation of Cyclic Events

Resets the generation of cyclic trigger events.

Function Prototype

```
void
Eth2PS_CyclicStop( void )
```

2.6 DLR Service Class

2.6.1 `Eth2PS_GetBeaconState()` - Get Beacon State

Retrieves the Beacon state of a received Beacon frame frame.

Function Prototype

```
ETH2PS_BCNSTATE_E
Eth2PS_GetBeaconState( uint32_t  ulFifoEntry )
```

Function Arguments

Argument	Direction	Description
`ulFifoEntry`	[in]	FIFO entry from the frame handle to query the beacon state

Table 39: `Eth2PS_GetBeaconState()` - Function Arguments

Function Return Values

Beacon State

2.7 Precision Time Protocol Service Class

2.7.1 `Eth2PS_PtpConfigPll()` - Configure PLL

Control the PLL. This function should be called periodically to synchronize to the PTP clock master.

Function Prototype

```
void
Eth2PS_PtpConfigPll( unsigned int  uIAmp2Pow,
                     unsigned int  uPAmp2Pow,
                     uint32_t      ulClockSpeedVariancePpm )
```

Function Arguments

Argument	Direction	Description
`uIAmp2Pow`	[in]	The 2-base exponent of amplification factor of the integral term of the controller
`uPAmp2Pow`	[in]	The 2-base exponent of amplification factor of the proportional term of the controller
`ulClockSpeedVariancePpm`	[in]	The maximum/minimum variance of the clock speed in parts per million

Table 40: `Eth2PS_PtpConfigPll()` - Function Arguments

2.7.2 `Eth2PS_PtpControlPll()` - Control PLL

Control the PLL. This function should be called periodically to synchronize to the PTP clock master.

Function Prototype

```
void
Eth2PS_PtpControlPll( int32_t                          lDiff,
                      ETH2PS_PI_CONTROLLER_OUTPUT_T*   ptOutput,
                      void*                            pvUser )
```

Function Arguments

Argument	Direction	Description
`lDiff`	[in]	Control deviation
`ptOutput`	[out]	If not NULL, this structure is filled with the controller output
`pvUser`	[in]	User specific parameter */

Table 41: `Eth2PS_PtpControlPll()` - Function Arguments

2.7.3 `Eth2PS_PtpResetPll()` - Reset PLL

Resets the PLL speed to initial value.

Function Prototype

```
void
Eth2PS_PtpResetPll( void*  pvUser )
```

Function Arguments

Argument	Direction	Description
`pvUser`	[in]	User specific parameter */

Table 42: `Eth2PS_PtpResetPll()` - Function Arguments

2.8 Status Service Class

2.8.1 `Eth2PS_GetCnfIrq()` - Get and Confirm Communication Interrupts

Retrieves and confirms the current interrupt requests from the according communication port.

Function Prototype

```
uint32_t
Eth2PS_GetCnfIrq( unsigned int  uiPort,
                  bool          fHiPriority )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	Switch port number
`fHiPriority`	[in]	IRQ priority selector (0: COM IRQs, 1: MSYNC IRQs)

Table 43: `Eth2PS_GetCnfIrq()` - Function Arguments

Function Return Values

Bit mask of pending interrupts

2.8.2 `Eth2PS_GetCounters()` - Get Status Counters

Gets the status counters of the according switch port.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_GetCounters( unsigned int         uiPort,
                    ETH2PS_COUNTERS_T*   ptCounters )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	Switch port number
`ptCounters`	[out]	Counter structure filled out by the function

Table 44: `Eth2PS_GetCounters()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_INVAL_PORT`	Invalid switch port number

Table 45: `Eth2PS_GetCounters()` - Function Return Values

2.8.3 `Eth2PS_GetEmptyFillLevel()` - Get Empty Pointer FIFO Fill Level

Returns the current fill level of the empty pointer FIFO.

Function Prototype

```
uint32_t
Eth2PS_GetEmptyFillLevel( void )
```

Function Return Values

FIFO fill level

2.8.4 `Eth2PS_GetIndCnfFillLevel()` - Get Indication/Confirmation FIFO Fill Level

Gets the fill level of the indication/confirmation FIFO.

Function Prototype

```
uint32_t
Eth2PS_GetIndCnfFillLevel( unsigned int  uPriority )
```

Function Arguments

Argument	Direction	Description
`uPriority`	[in]	0: low priority, 1: high priority

Table 46: `Eth2PS_GetIndCnfFillLevel()` - Function Arguments

Function Return Values

FIFO fill level

2.8.5 `Eth2PS_GetReqFillLevel()` - Get Request FIFO Fill Level

Returns the current fill level of the transmit request FIFO.

Function Prototype

```
uint32_t
Eth2PS_GetReqFillLevel( unsigned int  uiPort,
                        unsigned int  uPriority )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	Switch port number
`uPriority`	[in]	0: low priority, 1: high priority

Table 47: `Eth2PS_GetReqFillLevel()` - Function Arguments

Function Return Values

FIFO fill level

2.9 Synchronization Service Class

2.9.1 `Eth2PS_GetPhyPhaseOffset()` - Get PHY Phase Offset

Gets the current phase offset of the PHY, which means additionally receive delay offset. This function should be called once after upon the first received packet after assertion of Ethernet link, in order to determine the exact delay. The value of phase offset is kept (constant) until link is down.

Function Prototype

```
ETH2PS_RESULT_E
Eth2PS_GetPhyPhaseOffset( unsigned int  uiPort,
                          uint8_t*      pbPhyPhaseOffsetNs,
                          void*         pvUser )
```

Function Arguments

Argument	Direction	Description
`uiPort`	[in]	Switch port number
`pbPhyPhaseOffsetNs`	[out]	Current PHY receive delay offset [nanoseconds]
`pvUser`	[in]	User specific parameter

Table 48: `Eth2PS_GetPhyPhaseOffset()` - Function Arguments

Function Return Values

Definition	Description
`ETH2PS_OKAY`	Successful
`ETH2PS_ERR_INVAL_PORT`	Invalid switch port number

Table 49: `Eth2PS_GetPhyPhaseOffset()` - Function Return Values

2.10 Structure Definitions

2.10.1 `ETH2PS_CFG_T` - HAL Configuration Structure

Configuration for HAL initialization.

ETH2PS_CFG_T

Name	Type	Description
`ePhyLedCfg`	ETH2PS_PHYLED_CFG_E	PHY LED behavior
`ulActLedFlashPeriod`	uint32_t	Flash frequency of activity LED [milliseconds], The blink frequency shall not be larger than 100, larger values may lead to malfunction of the LED

Table 50: `ETH2PS_CFG_T` - Structure

2.10.2 ETH2PS_CONNECTION_STATE_T - Link Status Structure

Describes the current state of the Ethernet PHY.

ETH2PS_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 51: ETH2PS_CONNECTION_STATE_T - Structure

2.10.3 ETH2PS_COUNTERS_T - Status Counters

The following structure contains the status counters of one port.

ETH2PS_COUNTERS_T

Name	Type	Description
ulTxOutOctets	uint32_t	The total number of octets transmitted out of the interface, including framing characters
ulTxSingleCollisions	uint32_t	The number of collisions during the first try of a frame transmission
ulTxMultipleCollisions	uint32_t	The number of collisions during a retry of a frame transmission
ulTxLateCollisions	uint32_t	The number of collisions after the first 64 octets of the packet have been transmitted
ulTxUnderrun	uint32_t	The number of TX MAC FIFO underruns that occur when the xPEC is too slow
ulTxAborted	uint32_t	The number of outbound packets which were aborted during a cut-through transmission because the incoming packed was erroneous
ulTxDiscarded	uint32_t	The number of outbound packets which were chosen to be discarded because the port is disabled
ulRxInOctets	uint32_t	The total number of octets received on the interface, including framing characters
ulRxFcsErrors	uint32_t	Count of received frames that are an integral number of octets in length and do not pass the FCS check
ulRxAlignmentErrors	uint32_t	Count of received frames that are not an integral number of octets in length and do not pass the FCS check
ulRxFrameLengthErrors	uint32_t	Count of received frames that exceed the maximum permitted frame size
ulRxRuntFrames	uint32_t	Count of received frames that have a length between 42 and 63 Bytes and pass the FCS check
ulRxCollisionFragments	uint32_t	Count of received frames that have a length of 63 Bytes or less and do not pass the FCS check
ulRxOverflow	uint32_t	The number of RX MAC FIFO overflows that occur when the xPEC is too slow
ulRxDiscarded	uint32_t	Count of inbound packets which were chosen to be discarded even though no error was detected. Possible reasons are low resources or the port is disabled.
ulRxCirculatingFrmBlocked	uint32_t	Count of inbound packets which were chosen to be discarded because the source MAC address matched one of our Chassis MAC addresses
ulRxUnknownErrors	uint32_t	Count of illegal error states reported from the RPU

Table 52: ETH2PS_COUNTERS_T - Structure

2.10.4 `ETH2PS_CYCLIC_CFG_T` - Cyclic Configuration

This structure hold all parameters for the cyclic event generation.

ETH2PS_CYCLIC_CFG_T

Name	Type	Description
`fTrgPinsControlledByHost`	bool	true: trigger 0 and 1 pins unchanged but interrupts are generated during cyclic operation
`ulTrgPulseLen`	uint32_t	Pulse length of trigger signals in 10ns units
`atTrgCfg[2]`	ETH2PS_CYCLIC_EVENT_T	Event configuration

Table 53: `ETH2PS_CYCLIC_CFG_T` - Structure

2.10.5 `ETH2PS_CYCLIC_EVENT_T` - Cyclic Event

This structure defines an event to be generated cyclically. One event can raise an IRQ and/or drive a pulse signal at the sync pin.

ETH2PS_CYCLIC_EVENT_T

Name	Type	Description
`ulStartOffset`	uint32_t	Time offset to ulStartTimeCyclicOp of first trigger event
`ulPeriod`	uint32_t	Cycle time of trigger event
`fIrqEn`	bool	true/false: IRQ generation enabled/disabled
`fTrgEn`	bool	true/false: trigger signal generation enabled/disabled
`fTrgPolarity`	bool	true/false: trigger signal is High/Low active
`fTrgOe`	bool	true/false: trigger signal output enable enabled/disabled

Table 54: `ETH2PS_CYCLIC_EVENT_T` - Structure

2.10.6 `ETH2PS_FRAME_BUF_HDR_T` - Ethernet Frame Buffer Header Structure

This structure defines the content at the front of each frame buffer.

ETH2PS_FRAME_BUF_HDR_T

Name	Type	Description
`ulUserData0`	uint32_t	Not used by HAL, for application usage
`ulUserData1`	uint32_t	Not used by HAL, for application usage

Table 55: `ETH2PS_FRAME_BUF_HDR_T` - Structure

2.10.7 `ETH2PS_FRAME_HANDLE_T` - Frame Handle

This structure is used to handle a single Ethernet frame.

ETH2PS_FRAME_HANDLE_T

Name	Type	Description
`ulFifoEntry`	uint32_t	Frame handle from Pointer FIFO
`usLength`	uint16_t	Total size of frame data in bytes
`ptHdr`	ETH2PS_FRAME_BUF_HDR_T*	Header of frame buffer
`pbData`	uint8_t*	Frame data pointer, pointing to first byte of destination MAC address

Table 56: `ETH2PS_FRAME_HANDLE_T` - Structure

2.10.8 ETH2PS_FRAME_INFO_T - Frame Information

This structure holds additional information after a frame was transmitted or received. The information is provided by the function Eth2PS_GetIndCnf().

ETH2PS_FRAME_INFO_T

Name	Type	Description
uPortNo	unsigned int	Switch port number (frame origin)
ulTimeNs	uint32_t	Value of systime_ns register, latched at transmission/reception of SFD (Start of Frame Delimiter)
ulTimeS	uint32_t	Value of systime_s register, latched at transmission/reception of SFD (Start of Frame Delimiter)
fCnf	bool	Flag indicating whether this frame was a confirmation (true) or indication (false)
eCnfResult	ETH2PS_CNF_ERROR_CODE_E	Result of the transmission process, only valid if fCnf==true

Table 57: ETH2PS_FRAME_INFO_T - Structure

2.10.9 ETH2PS_PI_CONTROLLER_OUTPUT_T - PI Controller Output

This structure holds information about the current PI controller output.

ETH2PS_PI_CONTROLLER_OUTPUT_T

Name	Type	Description
lPTerm	int32_t	Proportional term
ulITerm	uint32_t	Integral term
ulOutput	uint32_t	Output value

Table 58: ETH2PS_PI_CONTROLLER_OUTPUT_T - Structure

2.11 Enumeration Definitions

2.11.1 ETH2PS_BCNSTATE_E - Beacon Status

These values are used to determine why a received Beacon frame was indicated.

ETH2PS_BCNSTATE_E

Definition	Description
ETH2PS_BCNSTATE_NO_BEACON	This is no Beacon frame
ETH2PS_BCNSTATE_PRECEDENCE_HIGHER	Precedence of this Beacon is higher than active one
ETH2PS_BCNSTATE_RX_PORT_NEQ_LAST	Beacon receive port is not equal to LastBcnRcvPort
ETH2PS_BCNSTATE_RING_STATE_CHANGE	Beacon with other ring state than last Beacon was received

Table 59: ETH2PS_BCNSTATE_E - Enumeration

2.11.2 ETH2PS_CNF_ERROR_CODE_E - Transmit Confirmation Error Codes

The function Eth2PS_GetIndCnf() provides one of the following error codes for each transmit confirmation.

ETH2PS_CNF_ERROR_CODE_E

Definition	Description
ETH2PS_CNF_ERR_CODE_SUCCESSFUL_WITHOUT_RETRIES	Success on first try
ETH2PS_CNF_ERR_CODE_SUCCESSFUL_WITH_RETRIES	Success after retries
ETH2PS_CNF_ERR_CODE_FAILED_LATE_COLLISION	Error (late collision)
ETH2PS_CNF_ERR_CODE_FAILED_EXCESSIVE_COLLISION	Error (collision excess)
ETH2PS_CNF_ERR_CODE_FAILED_UTX_UNDERRUN	Error (FIFO under-run error)
ETH2PS_CNF_ERR_CODE_TX_FAILED_ABORTED	Error (Frame aborted)

Table 60: ETH2PS_CNF_ERROR_CODE_E - Enumeration

2.11.3 ETH2PS_MAC_ADDRESS_TYPE_E - MAC addresses

Describes the different types of MAC addresses.

ETH2PS_MAC_ADDRESS_TYPE_E

Definition	Description
ETH2PS_MAC_ADDRESS_CHASSIS	Primary Chassis MAC address valid all both ports
ETH2PS_MAC_ADDRESS_2ND_CHASSIS	Secondary Chassis MAC address valid all both ports
ETH2PS_MAC_ADDRESS_DLR_SUPERVISOR	MAC address of ring supervisor

Table 61: ETH2PS_MAC_ADDRESS_TYPE_E - Enumeration

2.11.4 ETH2PS_PARAM_E - Port Parameters

These parameters can be set for each port individually. Use the function Eth2PS_SetParameter() to set one of the parameters.

ETH2PS_PARAM_E

Definition	Description
ETH2PS_PARAM_IRQ_EN_MSK	Interrupt enable mask, the value shall be a sum of MSK_ETH2PS_IRQ_* bits
ETH2PS_PARAM_AGING_TIME	Aging time [milliseconds]
ETH2PS_PARAM_PORT_ENABLE	1/0: enable/disable port, a disabled port does not transmit, receive and forward any frames
ETH2PS_PARAM_LINK_INPUT_ENABLE	1/0: enable/disable usage of switch port's link input Note that if disabled user has to use PORT_ENABLE to discard transmission of frames in case of link down Not supported by netX100/500, netX50, netX5, netX51/52 and netX6.
ETH2PS_PARAM_BPDU_SUPPORT_ENABLE	1/0: enable/disable special handling of BPDU (Bridge Protocol Data Unit) frames (DA == 01:80:C2:00:xx:xx) if enabled, BPDU frames are always blocked and indicated regardless of other parameters
ETH2PS_PARAM_BPDU_ONLY	1: only BPDUs are received and transmitted at this port, all other frames are dropped
ETH2PS_PARAM_DISABLE_LEARNING	0/1: learning of non-BPDUs enabled/disabled To model state STP.Disabled: PORT_ENABLE = 0, BPDU_ONLY = don't care, DISABLE_LEARNING = don't care To model state STP.Blocking / RSTP.Discarding: PORT_ENABLE = 1, BPDU_ONLY = 1, DISABLE_LEARNING = 1 To model state STP.Listening / RSTP.Discarding: PORT_ENABLE = 1, BPDU_ONLY = 1, DISABLE_LEARNING = 1 To model state STP.Learning / RSTP.Learning: PORT_ENABLE = 1, BPDU_ONLY = 1, DISABLE_LEARNING = 0 To model state STP.Forwarding / RSTP.Forwarding: PORT_ENABLE = 1, BPDU_ONLY = 0, DISABLE_LEARNING = don't care (always 0)
ETH2PS_PARAM_DSCP_PRIORIZATION	IP based DSCP priority, disabled by default if enabled recommendation is 43; that means frames with DSCP < 43 / >=43 treated as low/high prior frames parameter must adapted to EthernetIP specific values, DSCP priorization overrules VLAN-Tag priority
ETH2PS_PARAM_DLR_SUPPORT_ENABLE	1/0: enable/disable special handling of DLR frames if enabled, Beacon frames are parsed (DA == 01:21:6c:00:00:01), NEIGHBOR_REQ/RSP and SIGN_ON frames are blocked (DA == 01:21:6c:00:00:02)
ETH2PS_PARAM_DLR_BCN_IND_ENABLE	1/0: enable/disable indication of Beacon frames, only used if ETH2PS_PARAM_DLR_SUPPORT_ENABLE is set
ETH2PS_PARAM_DLR_BCN_PORT_MATCH_ENABLE	1/0: enable/disable Beacon port match, if enabled match is always inclusive, This parameter has no effect unless ETH2PS_PARAM_DLR_SUPPORT_ENABLE is set
ETH2PS_PARAM_DLR_BCN_RCV_PORT	last Beacon receive port for comparison, This parameter has no effect unless ETH2PS_PARAM_DLR_SUPPORT_ENABLE is set
ETH2PS_PARAM_DLR_BCN_PRECEDENCE	last Beacon precedence for comparison, This parameter has no effect unless ETH2PS_PARAM_DLR_SUPPORT_ENABLE is set
ETH2PS_PARAM_DLR_BCN_TIMEOUT	0: Disable and stop Beacon timeout timer else: Start Beacon timeout timer with given value [10 ns] This parameter has no effect unless ETH2PS_PARAM_DLR_SUPPORT_ENABLE is set

Definition	Description
ETH2PS_PARAM_INGRESS_LATENCY	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 MAC sample delays + PHY receive delay + eventually PHY phase offsets
ETH2PS_PARAM_EGRESS_LATENCY	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 MAC sample delays + PHY transmit delay

Table 62: ETH2PS_PARAM_E - Enumeration

2.11.5 `ETH2PS_PHYLED_CFG_E` - PHY LED configuration

ETH2PS_PHYLED_CFG_E

Definition	Description
`ETH2PS_PHYLED_STATIC`	separate link and activity LEDs, activity statically on
`ETH2PS_PHYLED_BLINK`	separate link and activity LEDs, activity blinking when active
`ETH2PS_PHYLED_SINGLE`	single LED, combined link and blink on activity
`ETH2PS_PHYLED_OFF`	PHY LEDs are disabled

Table 63: `ETH2PS_PHYLED_CFG_E` - Enumeration

2.11.6 ETH2PS_RESULT_E - Functions Result Codes

All functions return one of the following values after returning from the function call. Function return values shall always be evaluated by the calling function.

ETH2PS_RESULT_E

Definition	Description
ETH2PS_OKAY	Successful
ETH2PS_ERR_INVAL_PORT	Invalid switch port number
ETH2PS_ERR_INVAL_PARAM	Invalid parameter in function call
ETH2PS_ERR_FIFO_EMPTY	Indication FIFO is empty
ETH2PS_ERR_INIT_FAILED	Error occurred during initialization
ETH2PS_ERR_INVAL_STATE	Invalid port state
ETH2PS_ERR_OUT_OF_MEMORY	Not enough resources

Table 64: ETH2PS_RESULT_E - Enumeration

3 Appendix

3.1 Differences in Features between V3/4 and V5

Feature	Version 3/4	Version 5
Forwarding Mechanism	Store and Forward	Cut-Through and Store and Forward in contention
DLR support	no	yes: Beacon-based non-supervisor ring node
IEEE1588 transparent Clock Support	no	yes
Quality of Service	Prioritization via IEEE 802.1Q/D (based on VLAN-Tag Priority)	Prioritization via IEEE 802.1Q/D (based on VLAN-Tag Priority) and DSCP (IP frames only)
Monitoring Mode	yes	no
Multicast DA Filtering	yes, based on 8 bit hash	yes, based on 12 bit hash
Possibility to disable switch port from transmission/reception	no	yes
Possibility to flush MAC address learning table	no	yes
Removing of device's own frames from network when receiving	no	yes
Error Counters for host		"LINK_DOWN_DURING_TRANSMISSION", "TX_FATAL_ERR", "RX_FATAL_ERR" not supported anymore, "FRAMES_DROPPED_DUE_LOW_RESOURCE" changed to "IN_FRAMES_DISCARDED"
Interrupts to host		Interrupts "Collision", "EarlyRcv", "RxErr", "TxErr" not supported anymore
FIFO interface to host	Indication FIFO separated by port and by priority Confirmation FIFO separated by port and by priority	Common Indication/Confirmation for all port separated by priority

Table 65: Differences in features between Version 3/4 and Version 5

3.2 PHY Latencies

100BaseTX/FX:

PHY	Reception (Ingress)	Transmission (Egress)
netX10/50/100/500 internal PHY	288 ns + PHY Phase Offset (0/8/16/24/32 ns) See function “Eth2PS_GetPhyPhaseOffset()” for details	72 ns
netX51/52/4000 internal PHY	215 ns	34 ns
netX90 internal PHY	220 ns + PHY Phase Offset (0/8/16/24/32 ns) See function “Eth2PS_GetPhyPhaseOffset()” for details	116 ns
Broadcom BCM5241 external PHY	170 ns	57 ns

Notes:

- 10BaseT: There are no delay values available for 10BaseT.
- MII sample delays (MAC samples MII) are taken into account within HAL interface. Do not consider them.

3.3 List of Tables

3.4 List of Figures

3.5 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