MICROSAR Ethernet Switch Driver

Technical Reference

NXP SJA1105P/Q/R/S 5-Port Automotive Ethernet Switch

Version 5.0.1

Authors	Ingo Schröck, David Fessler, Mark Harsch, Benjamin Groebner
Status	Released

Document Information

History

Author	Date	Version	Remarks
Ingo Schröck	2017-07-26	1.00.00	Initial version of the Technical Reference for the MICROSAR Ethernet Switch driver for the NXP SJA1105PEL, SJA1105QEL, SJA1105REL, SJA1105SEL
Ingo Schröck	2017-11-24	2.00.00	Added component version 2.00.00
David Fessler	2017-12-18	3.00.00	Added chapter for switch cascading
Ingo Schröck	2018-02-14	3.00.01	Updated reference documents, minor fixes, review integration
Mark Harsch	2018-11-20	4.00.00	▶ Updated Version of Refence Document [7] ▶ Minor editorial corrections ▶ STORYC-5868: Provide possibility to correct the switch clock by an offset ▶ STORYC-5869: Provide possibility to correct the switch clock by a rate ▶ Mirroring description ▶ STORYC-5872: Provide possibility to start/stop the Qbv scheduling
Benjamin Groebner	2019-02-06	5.00.00	▶ STORY-9435: Adapt configuration blocks to download L2 policing table ▶ STORY-9449: Add validators for L2 policing configuration ▶ STORY-9439: Add L2 policing section in TechRef
Ingo Schröck	2019-03-01	5.00.01	▶ ESCAN00101470: Typo in technical reference

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_EthernetSwitchDriver.pdf	4.2.1
[2]	AUTOSAR	AUTOSAR_SWS_DET.pdf	4.2.1
[3]	AUTOSAR	AUTOSAR_SWS_DEM.pdf	4.2.1
[4]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[5]	NXP	um420810 - UM11040 Software user manual for SJA1105P_Q_R_S (1.0).pdf	Rev. 1 - 24 November 2017
[6]	NXP	ds420710 - SJA1105P_Q_R_S Objective Data Sheet (1.0).pdf	Rev. 1 - 24 November 2017
[7]	NXP	an428912 - AH1704 (1.2).pdf SJA1105PQRS Application Hints	Rev. 1.2 - 08. Mar 2018

Scope of the Document

This technical reference describes the general use of the Ethernet Switch driver for the NXP SJA1105P/Q/R/S 5-Port Automotive Ethernet Switch.

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

Contents

Illustrations

Tables

1 Component History

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

Component Version	New Features
1.00.xx	Initial version of the Ethernet Switch Driver for SJA1105P/Q/R/S
2.00.xx	Release of DrvEthSwitch_Sja1105PQRS with ASR 4.2.1 feature set
3.00.xx	Added cascading and RGMII support
4.00.xx	▶ Possibility to correct the switch clock by an offset and/or rate ▶ Support of port mirroring functionality ▶ Possibility to start/stop the Qbv scheduling
5.00.xx	▶ Support of L2 policing

Table 1-1 Component history

2 Introduction

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

Supported AUTOSAR Release*:	4.2.1	
Supported Configuration Variants:	pre-compile	
Vendor ID:	ETHSWT_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	ETHSWT_MODULE_ID	89 decimal (according to ref. [4])

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

The Ethernet Switch driver for the NXP SJA1105P/Q/R/S Automotive Ethernet Switch provides a hardware independent access to the functionality of the Switch according to the AUTOSAR standard noted above.

2.1 Architecture Overview

The following figure shows where the EthSwt is located in the AUTOSAR architecture.

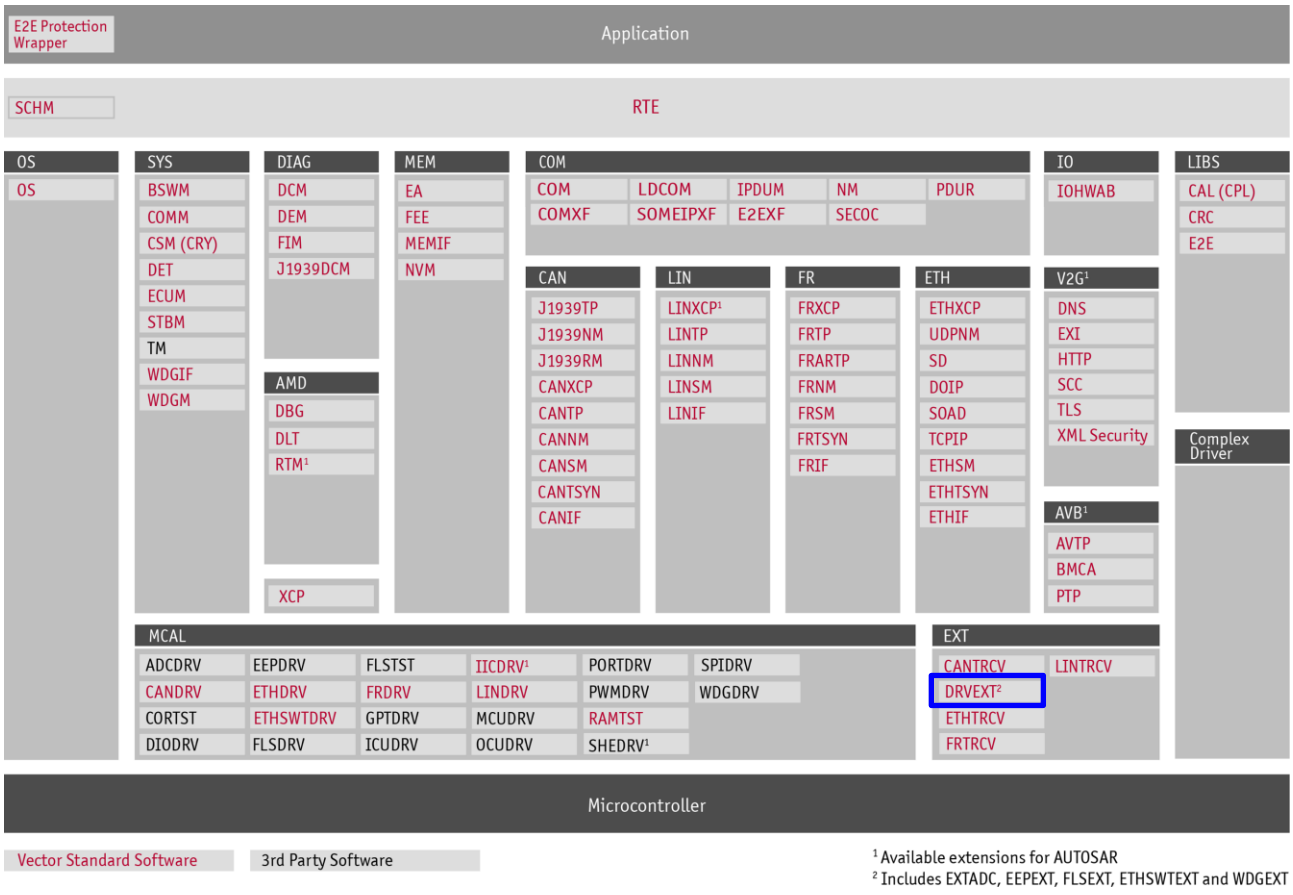

Figure 2-1 AUTOSAR 4.2 Architecture Overview

The next figure shows the interfaces to adjacent modules of the EthSwt. These interfaces are described in chapter 5.

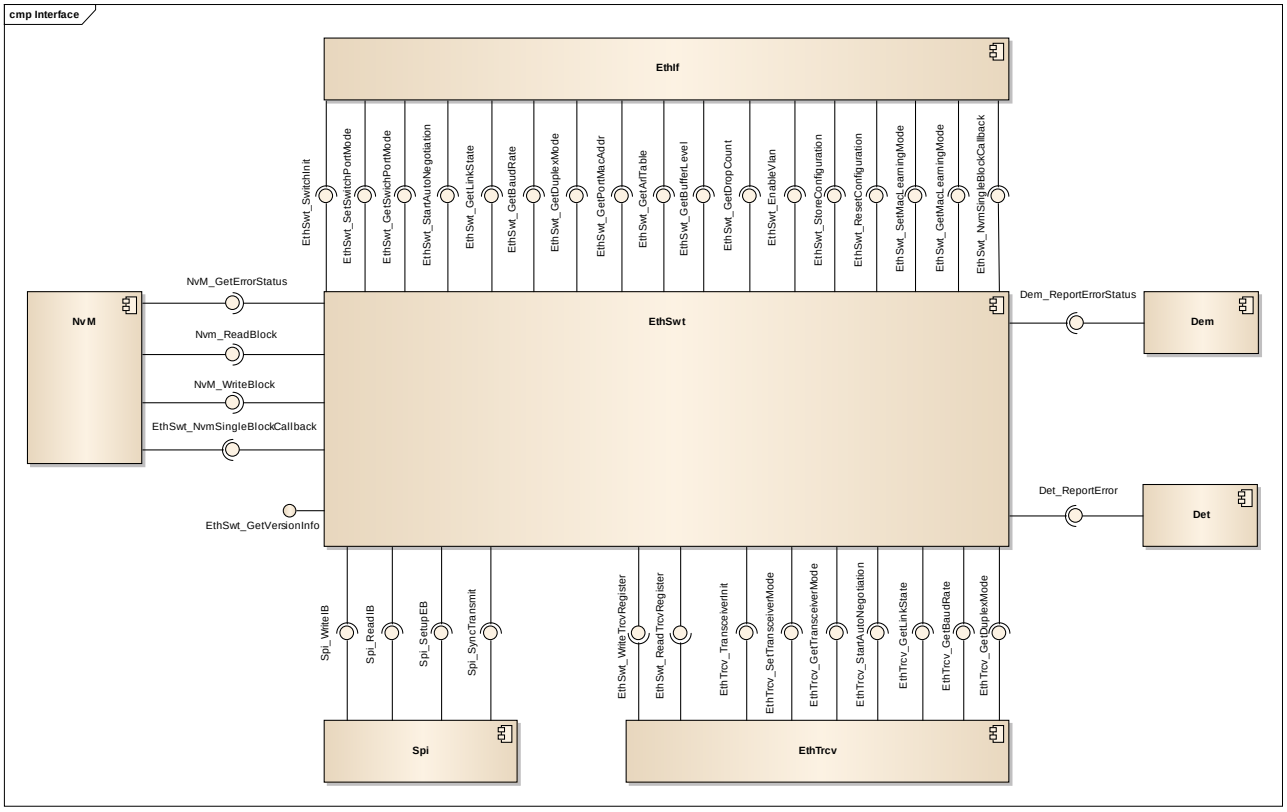

Figure 2-2 Interfaces to adjacent modules of the EthSwt

3 Functional Description

3.1 Features

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

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

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further EthSwt functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-4 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Ethernet Switch Driver initialization
Influencing/retrieving the mode of a port
Triggering the auto-negotiation process of a port
Retrieving the link state of a port
Retrieving the baud rate of a port
Retrieving the duplex mode of a port
Retrieving the port a MAC address is linked to in L2 lookup table
Retrieving the L2 lookup table
Retrieving the drop counts
Retrieving the Ethernet statistic counts
Influencing the VLAN forwarding behavior of a port
Storing/resetting of learned entries retrieved from address resolution table in NV RAM
Loading of learned entries previously stored in NV RAM during initialization
Optional user callout to indicate the finish of a NvM job
Influencing/retrieving the MAC learning mode
Configuration of VLAN Modification of untagged frames on ingress side
Configuration of VLAN forwarding rules on egress side (don’t forward, forward without modification, forward untagged)
Configuration of egress elements and their structure (composition of FIFOs, Shapers, Schedulers)
Hardware access through SPI
Development error reporting by DET
Production error reporting by DEM

Supported AUTOSAR Standard Conform Features
Optional link up/down user callback

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

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

Not Supported AUTOSAR Standard Conform Features
Traffic Class handling
Ingress tagging of VLAN-tagged frames

Table 3-2 Not supported AUTOSAR standard conform features

The sub-sections of this chapter describe the deviations that apply to the listed features.

3.1.1.1 Traffic Class Handling

Due to the hardware architecture of SJA1105P/Q/R/S, which doesn't define an internal frame priority denoted as 'Traffic Class' by AUTOSAR, the related configuration parameters either aren't used or have a different meaning. The following table lists the affected configuration elements and describes their behavior (if applicable).

Configuration Element	Description
EthSwtPortTrafficClassAssignment	Applies a default PCP for an untagged Ethernet frame (please see 3.5.2 for a detailed description).
EthSwtPriorityTrafficClassAssignment ▶ EthSwtPriorityTrafficClassAssignmentPriority ▶ EthSwtPriorityTrafficClassAssignmentTrafficClass	Configuration container and its parameters aren't used.

Table 3-3 Traffic Class Handling Deviations

3.1.1.2 Ingress Tagging of VLAN tagged frames

Due to hardware architecture of SJA1105P/Q/R/S, which doesn't allow changing the VLAN-ID of already VLAN-tagged Ethernet frames (at least not for all possible VLAN-IDs, it is limited to 32 VLAN-IDs) the configuration parameter `EthSwtPortIngressVlanModification` only defines the VLAN-ID that un-tagged traffic shall be tagged with on ingress side of the related port. For a description of this functionality please see 3.5.2.

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Asynchronous calls to AUTOSAR API
Frame Management
Global Time support

Features Provided Beyond The AUTOSAR Standard
Possibility to correct the switch clock used as time base for the time stamps by an offset and/or a rate
Port mirroring (preliminary ASR4.4.x implementation)
L2 Policing (according to ASR 4.3.x with limitation of VLAN ID membership based policing and extension of Maximum Frame Burst Size specification)

Table 3-4 Features provided beyond the AUTOSAR standard

3.1.2.1 Asynchronous Calls to AUTOSAR API

AUTOSAR defines the most EthSwt driver APIs to be called in a synchronous way. However, some APIs must do very time-consuming operations with a lot of interaction with the switch through the SPI interface. This most probably will result in harming timing restriction for task activation of the OS.

This extension allows changing the behavior of the APIs to an asynchronous way. The API call will just trigger the start for processing and the caller will be notified by a callout if the processing is finished. The following table shows the APIs, which are affected by this extension.

AUTOSAR API
EthSwt_30_Sja1105PQRS_GetArlTable
EthSwt_30_Sja1105PQRS_GetDropCount

Table 3-5 Affected APIs of Asynchronous Calls Extension

Caution
The in/out function parameters passed during the call to the API must not be changed and are not valid until the user was notified about the end of processing through the callout function.

Note
The feature is enabled by setting `EthSwtApiCallType` to `ETHSWT_ASYNC_CALL`. In addition to this the User for the finish notification callout must be given by the configuration parameter `EthSwtAsyncNotificationUser`. For a description of the notification please refer to 5.5.1.1.

3.1.3 Limitations

3.1.3.1 Availability of Drop Counters

The AUTOSAR API `EthSwt_GetDropCount()` allows to retrieve counters to packet drop events on ingress and egress for the switch. Which counters to be retrieved is specified by AUTOSAR.

However, the availability of these counters is limited by hardware. Which counters are supported by the SJA1105P/Q/R/S can be seen with the help of the following table.

#	AUTOSAR Diagnostic Counter Description	Availability
1	Dropped packets due to buffer overrun	Supported
2	Dropped packets due to CRC error	Supported
3	Number of undersize packets which were less than 64 octets long (excluding framing bits, but including FCS octets) and were otherwise will formed. (see IETF RFC 1757)	Supported
4	Number of oversize packets which are longer than 1518 octets (excluding framing bits, but including FCS octets) and were otherwise well formed. (see IETF RFC 1757)	Not supported (hardware lacks this counter)
5	Number of alignment errors, i.e. packets which are received and are not an integral number of octets in length and do not pass the CRC.	Supported
6	SQE test error according to IETF RFC1643 dot3StatsSQETestErrors	Not supported (hardware lacks this counter)
7	The number of inbound packets which were chosen to be discarded even though no errors had been detected to prevent their being deliverable to a higher-layer protocol. One possible reason for discarding such a packet could be to free up buffer space. (see IETF RFC 2233 ifInDiscards)	Not supported (hardware lacks this counter)
8	Total number of erroneous inbound packets	Supported (sum of the supported counters 1-7)
9	The number of outbound packets which were chosen to be discarded even though no errors had been detected to prevent their being transmitted. One possible reason for discarding such a packet could be to free up buffer space. (see IETF RFC 2233 ifOutDiscards)	Not supported (hardware lacks this counter)
10	Total number of erroneous outbound packets	Not supported (would be the sum of the supported counters 9, 11-14)
11	Single collision frames: A count of successfully transmitted frames on a particular interface for which transmission is inhibited by exactly one collision. (see IETF RFC1643 dot3StatsSingleCollisionFrames)	Not supported (hardware lacks this counter)

12	Multiple collision frames: A count of successfully transmitted frames on a particular interface for which transmission is inhibited by more than one collision. (see IETF RFC1643 dot3StatsMultipleCollisionFrames)	Not supported (hardware lacks this counter)
13	Number of deferred transmission: A count of frames for which the first transmission attempt on a particular interface is delayed because the medium is busy. (see IETF RFC1643 dot3StatsDeferredTransmissions)	Not supported (hardware lacks this counter)
14	Number of late collisions: The number of times that a collision is detected on a particular interface later than 512 bit-times into the transmission of a packet. (see IETF RFC1643 dot3StatsLateCollisions)	Not supported (hardware lacks this counter)

Table 3-6 AUTOSAR Diagnostic Counter Limitations

Note
For not supported counters the driver will pass the maximum value of the data type of the variable the counter shall be written to.

Note
Due to the lack of some drop counters the provided DEM events available to report to the DEM is also limited. Please refer to 3.14.2 for more information about available DEM events.

3.1.3.2 Availability of Ethernet Statistic Counters

The AUTOSAR API `EthSwt_GetEtherStats()` allows to retrieve the Ethernet statistic counters for a switch port. Which counters to be retrieved is specified by AUTOSAR.

However, the availability of these counters is limited by hardware. Which counters are supported by the SJA1105P/Q/R/S can be seen with the help of the following table.

#	AUTOSAR Ethernet Statistic Counter Description	Availability
1	etherStatsDropEvents	Not supported (hardware lacks this counter)
2	etherStatsOctets	Supported
3	etherStatsPkts	Supported
4	etherStatsBroadcastPkts	Supported
5	etherStatsMulticastPkts	Supported

6	etherStatsCrcAlignErrors	Supported
7	etherStatsUndersizePkts	Supported
8	etherStatsOversizePkts	Not supported (hardware lacks this counter)
9	etherStatsFragments	Not supported (hardware lacks this counter)
10	etherStatsJabbers	Not supported (hardware lacks this counter)
11	etherStatsCollisions	Not supported (hardware lacks this counter)
12	etherStatsPkts64Octets	Supported
13	etherStatsPkts65to127Octets	Supported
14	etherStatsPkts128to255Octets	Supported
15	etherStatsPkts256to511Octets	Supported
16	etherStatsPkts512to1023Octets	Supported
17	etherStatsPkts1024to1518Octets	Supported

Table 3-7 AUTOSAR Ethernet Statistics Limitations

Note
For not supported counters the driver will pass the maximum value of the data type of the variable the counter shall be written to.

3.2 Initialization

3.2.1 Initialization of Switch Driver

The Ethernet Switch Driver is initialized by calling the service `EthSwt_30_Sja1105PQRS_Init()`.

In conjunction to this API it must be ensured that the `*_INIT_*` variables are either be initialized by the startup code of the MCU or by the call to `EthSwt_30_Sja1105PQRS_InitMemory()`.

Note
It is implicitly assumed that the switch connected to the host CPU is initialized before the consecutive switches of the cascade are initialized. Time synchronization works only in this case.

Note
The correct initialization of the `*_INIT_*` variables is checked if `EthSwtDevErrorDetect` is enabled. If the variables weren't enabled before `EthSwt_30_Sja1105PQRS_Init()` is called `ETHSWT_30_SJA1105PQRS_E_MEMORY_NOT_INITIALIZED` is issued to the DET module.

3.2.2 Initialization of Dependent Modules

Ethernet Transceiver Driver

The Ethernet Switch driver utilizes the Ethernet Transceiver driver to configure the PHYs connected to its ports. Therefore, the Ethernet Transceiver driver has to be initialized according to its reference manual.

SPI Driver

For operation the switch driver utilizes the AUTOSAR SPI driver to exchange data with the SJA1105P/Q/R/S. Therefore, the SPI driver must be initialized according to its manual.

Port Driver

For the SPI driver being able to drive the signal lines the SJA1105P/Q/R/S's SPI interface is connected to, the pins of the MCU must be configured with the help of the AUTOSAR Port driver.

3.3 States

Initially the driver is in state `ETHSWT_STATE_UNINIT`. By calling the services like specified in 3.2.1 the driver will transition to `ETHSWT_STATE_INIT`. This state allows loading the configuration to the switch. The loading procedure is triggered by the Ethernet Interface during the communication request. After successfully loading the configuration the switch is in state `ETHSWT_STATE_ACTIVE` and operational.

3.4 Main Functions

The driver provides two main functions. They are described in the following table.

Main Function	Purpose
EthSwt_30_Sja1105PQRS_MainFunction()	Monitors the drop counts if production error detection is

	enabled and reports the corresponding DEM events to the Diagnostic Event Manager.
EthSwt_30_Sja1105PQRS_AsyncProcessingMainFunction()	Processes asynchronous API calls. For further information see 3.1.2.1.
EthSwt_30_Sja1105PQRS_MainFunctionTime()	Queue Handling for asynchronous timestamp retrieval.

Table 3-8 Main Functions

3.5 Untagged Traffic Handling

Internally the SJA1105P/Q/R/S only can handle Ethernet frames extended by a VLAN-Tag according to IEEE802.1Q. Therefore, every untagged Ethernet frame is extended by a default VLAN-Tag.

3.5.1 Global default VLAN-Tag

If no port-specific default VLAN-tag configuration is given the global settings are used to generate the VLAN-tag for untagged Ethernet frames.

Note
The two configuration parameters for the global default VLAN-Tag are located under the configuration container `EthSwtConfig`. `EthSwtDefaultVlanId` allows to configure the VLAN-ID and `EthSwtDefaultPcp` allows to configure the PCP used to generate the VLAN-Tag.

Caution
Untagged frames extended with the global default VLAN-Tag will only put out on ports where a forwarding rule for this VLAN-ID is configured or the VLAN is unlocked during runtime (see 3.6)

3.5.2 Port-based default VLAN-Tag

In addition to the global default VLAN-Tag it is possible to use port specific settings for the generation of the default VLAN-Tag. In this case it is possible to only define the VLAN-ID or the PCP or both. These settings override the global default VLAN-Tag settings so only these are used to generate the VLAN-tag on the related port.

Note
The configuration parameters for the port-based default VLAN-Tag are located under the configuration container `EthSwtPort`. `EthSwtIngressVlanModification` allows to configure the VLAN-ID and `EthSwtTrafficClassAssignment` allows to configure the PCP used to generate the VLAN-Tag.

Caution
Untagged frames extended with the port-based default VLAN-Tag will only put out on ports where a forwarding rule for this VLAN-ID is configured or the VLAN is unlocked during runtime (see 3.6).

3.6 Tagged Traffic Handling

The SJA1105P/Q/R/S can apply special treatment for Ethernet Traffic tagged with a VLAN-Tag according to IEEE802.1Q. This handling is done with the help of the VLAN-ID contained in the VLAN-Tag and can be applied port-wise. There are two possibilities to define these rules, which are explained in this section.

Note
SJA1105P/Q/R/S only handles Ethernet traffic for known VLANs. I. e. if a frame tagged with a VLAN-ID that is neither setup and allowed during configuration nor unlocked during runtime is received on a port, this frame will be dropped.

3.6.1 Manipulation via Configuration

Ethernet traffic for a specific VLAN is unlocked by configuring a forwarding rule on the egress side of a port. In addition to just allow Ethernet traffic for this respective VLAN the forwarding rule also allows to configure how the frame shall be transmitted on the port. It either keeps its VLAN tag or the VLAN-tag will be removed and the frame is put out as an untagged Ethernet frame.

Note
The configuration parameters for the port-based VLAN unlocking and forwarding are located in the configuration container `EthSwtPortVlanForwarding`. `EthSwtPortVlanForwardingId` defines the VLAN-ID the rule shall be applied to and `EthSwtPortVlanForwardingType` defines if the frame shall be transmitted as it is (with VLAN-tag) or if the VLAN-tag shall be removed.

3.6.2 Manipulation during Runtime

In addition to the configuration the switch driver allows to dynamically unlock/lock VLANs for a specific port. This can be done by utilizing the API `EthSwt_30_Sja1105PQRS_EnableVlan()`.

Note
The `EthSwt_30_Sja1105PQRS_EnableVlan()` API is provided if the configuration parameter `EthSwtEnableVlanApi` is turned on.

If the API is available the lock/unlock of a VLAN already provided during configuration can be done any time for a specific port. The type of forwarding (VLAN-tag is preserved, frame is untagged) remains unchanged. This behavior will remain as provided during configuration.

Additionally, to the VLANs provided during configuration, the driver can handle a specific count of additional VLANs. The amount of VLANs that can be handled in parallel must be provided during configuration. In case of such a VLAN the forwarding type is always to preserve the VLAN-tag.

Note
The configuration parameter `EthSwtDynamicVlanNum` allows to define the number of VLANs that the driver can handle in addition to the VLANs provided during configuration. It defines the maximum amount of additional VLANs that can be allowed at the same time.

3.7 Priority Handling

In contrast to the AUTOSAR SWS (see [1]) the priority management of the SJA1105P/Q/R/S isn’t based on an internal priority (entitled as Traffic-Class) in AUTOSAR but on the VLAN PCP specified by IEEE802.1Q.

Note
Due to this limitation some configuration elements provided by AUTOSAR are not used by the driver.

3.7.1 VLAN-Priority

As denoted in the introduction of this section the priority handling of the SJA1105P/Q/R/S is based on the PCP of the VLAN-Tag of an Ethernet frame. The hardware utilizes two mechanisms to apply a priority handling for frames within the switching engine.

Ingress PCP Re-Mapping

The switch hardware allows re-mapping the ingress PCP to an egress PCP of an Ethernet frame on a per port basis. This re-mapping results in changing the PCP of the VLAN-Tag without adapting any other fields of the frame header (except of the FCS, which must be recalculated).

Note
The re-mapping is configured by creating an `EthSwtPriorityRegeneration` container in the ingress configuration of a switch port. This container represents a tuple of an ingress PCP and egress PCP.
If no rule for a specific ingress PCP is given the ingress PCP will be used as egress PCP (no re-mapping applied).

Egress PCP to Queue assignment

Priority handling for Ethernet frames on egress side of a switch port is done by assignment of egress PCP to a switch port queue. Each queue has an explicitly assigned priority where queue 0 has the lowest priority and queue 7 the highest.

Note
The egress PCP to queue assignment is configured by creating `EthSwtPortFifoTrafficClassAssignment` parameters (representing the egress PCP) for an `EthSwtPortFifo` container (representing the queue).
If no rule for a specific egress PCP is given it is mapped to the queue having the same priority level. E.g. if egress PCP is 2 and no rule is given the frame will be forwarded to queue 2.

3.7.2 Egress Queue Scheduling

All queues of a port are served by one scheduler. The scheduling algorithm used is “strict priority”. So, the queue with the highest priority having frames ready to transmit is served until no more frames are queued.

Caution
If any of the queues has assigned a traffic shaper the scheduling isn’t done according to “strict priority scheduling” anymore. Even if a higher priority queue with no shaper assigned has frames for transmission the shaper assigned to a lower priority queue is able to override the transmission request.

3.8 Traffic Shaping

Traffic shaping allows reserving but also limiting bandwidth for a queue of a port.

The SJA1105P/Q/R/S provides 10 shapers that can be assigned to any queue on any port during configuration time. However, the shaper can only handle one queue at a time.

Example
The most common use-case is to use 2 shapers on each port that allows handling 3 classes of traffic on the port.

Best effort traffic that is handled by the queues not assigned to a shaper.

More prior traffic that is handled by the queue with a shaper assigned that has the lower priority of the two queues.

Highest prior traffic that is handled by the queue with a shaper assigned that has the higher priority of the two queues.

Note
The bandwidth that shall be assigned to a shaper can be defined by configuring the parameter `EthSwtPortShaperIdleSlope` that is located in the container `EthSwtPortShaper`.

In addition to the bandwidth limitation and reservation aspects of the shaper it is also possible to reduce traffic burstiness caused by a sender before the traffic arrives at the node designated to receive the traffic. Instead of letting the frames pass in the burst they were received the shaper can put out the frames with a defined distance in time.

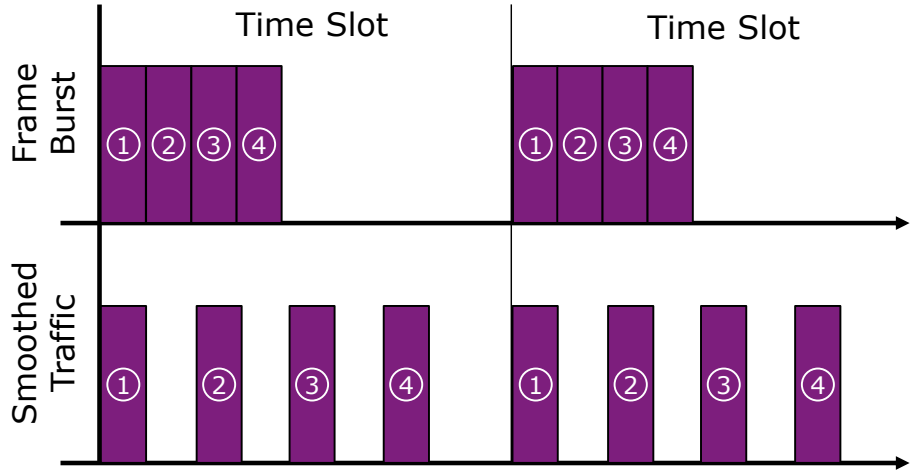

Figure 3-1 Traffic Shaping

3.9 L2 Policing

L2 Policing allows to limit the input bandwidth for each port. The bandwidth limitation can be individually applied for each VLAN PCP priority (0-7) per port by an ingress policer configuration. Additionally, the maximum allowed ingress frame burst size can be configured for each policer. This maximum frame burst size is the maximum amount of bandwidth credit that can be reached if there is no related ingress traffic for a longer time and serves additionally as the initial starting bandwidth credit of a policer. For any received frame the value decreased from the related bandwidth credit by a configured policer is the total number of bytes of the ingress frame including also the header and checksum bytes.

Note
It is recommended to set the value for “Maximum Burst Size of Ingress Frames” at least to 1522 bytes for a typical Ethernet communication.

If no policer configuration is existing for a port and priority combination then no policing is applied for the related ingress frames. If no priority value is existing for a policer of a port then this policer configuration is considered as a priority wildcard configuration and the policer is applied for any non-explicitly existing priority of the related port.

Caution
The mapping of MAC broadcast frames is always statically to the policer configuration of the related port which has the VLAN PCP priority value of 0 even if the received broadcast frame has a different VLAN PCP priority value (1-7).

3.10 Frame Management

The frame management feature allows the SJA1105P/Q/R/S to handle special Ethernet frames like PTP frames. These special frames are identified by having a reserved multicast address as destination address.

If the switch recognizes such an Ethernet frame on reception it traps this frame instead of flooding it to all other ports (like it is done for normal multicast traffic). The frame is redirected to the Host-CPU and enriched by management information including such information like the port the frame was received on. This information is provided to the protocol stack running on the Host-CPU by the driver in a follow up frame (META frame). The destination multicast address of that frame is configurable in Da Vinci Configurator Pro.

Note
The destination multicast MAC address for the META frames can be configured by setting the configuration parameter `EthSwtManagementMetaFrameDestMac`.

Caution
The destination multicast MAC address for the META frame must be unique and not used by other frames in that system.

Additionally, to treating the special Ethernet frames on reception the driver also has the capability to apply management information on transmission of a frame. This allows the protocol stack to exactly define on which port a frame shall be transmitted.

Note
The frame management capabilities of the driver and therefore of the SJA1105P/Q/R/S can be enabled by setting the configuration parameter `EthSwtManagementSupport`

3.11 Time Synchronization

The time synchronization feature allows the SJA1105P/Q/R/S to time stamp PTP frames when they are entering the switch.

The driver indicates this time stamps to the respective protocol stack and therefore enables the stack to provide or synchronize to a time base.

Note
The time synchronization capabilities of the driver and therefore of the SJA1105P/Q/R/S can be enabled by setting the configuration parameter `EthSwtGlobalTimeSupport`.

3.12 IEEE802.1 Qbv

The IEEE802.1 Qbv feature allows the derivatives SJA1105Q/S to apply a switch clock based time-based schedule to the egress queues of the Ethernet switch port.

This time-based schedule enables the switch to send Ethernet frames based on their traffic class in a deterministic manner.

Caution
The SJA1105Q/S is - with respect to IEEE802.1Qbv - a pre-standard implementation and has some differences to the final IEEE802.1Qbv-2015 specification.

Please refer to section 6.4.1.6 of [7] for a detailed description.

3.12.1 Correction of the Ethernet Switch clock

As a precondition for the time-based schedule of Ethernet traffic the switch must be synchronized to the global time of the Ethernet network. Therefore, the switch clock, which is used as base for the switch time must be corrected to align to the networks global time.

The driver supports the correction by an offset and/or rate. However, as soon as the time-base schedule is started no offset correction is allowed anymore.

Since AUTOSAR has – to current point in time - some limitations with respect to taking timestamps on uplink ports (AUTOSAR only allows one ingress and one egress timestamp for an Ethernet frame, which will be the timestamps related to the ports the frame enters and leaves the cascade), no timestamps on uplink ports can be taken. This limitation has an impact on the hardware architecture of a switch cascade.

Caution
If a switch cascade is used and IEEE802.1Qbv shall be enabled, the switch clocks must be driven by the same oscillator source.

In case of the SJA1105Q/S two options are possible:

- All switches of the cascade are connected to the same external oscillator
- The master switch of the cascade is connected to an external oscillator and all other switches are daisy-chained to the `CLK_OUT` pin of the previous switch in the cascade

For recommendations on how to design the switch cascade with respect to the oscillator source please refer to section 2.3.3 of [7].

3.12.2 QBV scheduling

IEEE802.1Qbv allows the scheduling of traffic in a switch. It is possible to define a schedule table for each port. Each “row” of this table defines which traffic classes can send data. There are 8 different traffic classes. So, it is possible to reserve timeslots for distinct traffic to guaranty a defined latency or throughput for this traffic class. Egress traffic is scheduled by this mechanism only.

3.12.2.1 QBV scheduling example

The scheduling can look like this:

EthSwtQbvSchedulers	Absolute time when the entry shall be scheduled [ms]	Gate Duration [ms]	Gate (FIFO)
EthSwtQbvScheduler_000	0 *	10	
EthSwtQbvScheduler_001	10	0.0004	EthSwtPortFifo_P1_0
EthSwtQbvScheduler_002	10.0004	39.9996	
EthSwtQbvScheduler_003	50	50	
EthSwtQbvScheduler_004	100	50	
EthSwtQbvScheduler_005	150	50	
EthSwtQbvScheduler_006	200	0.0002	EthSwtPortFifo_P1_4
EthSwtQbvScheduler_007	200.0002	49.9998	
EthSwtQbvScheduler_008	250	50	
EthSwtQbvScheduler_009	300	50	
EthSwtQbvScheduler_010	350	0.0002	

The scheduling is defined by the absolute time of the different schedulers. There are two short slots which are open, EthSwtQbvScheduler_001 and EthSwtQbvScheduler_006. The Gate Duration column shows the length of each scheduler. So EthSwtQbvScheduler_001 is open for 400 nanoseconds, EthSwtQbvScheduler_006 is open for 200 nanoseconds. The last row does only define the end of the schedule. The Gate Duration and the Gate parameters are not used. When the end of the schedule is reached, scheduling starts automatically from the beginning. Therefore, the first entry always starts with an absolute time value of 0.

3.12.2.2 Limitations of QBV scheduling

Due to hardware limitations there is no full IEEE802.1Qbv scheduling available. These are the limitations:

> SJA1105Q and SJA1105S are supported only.

- Minimum scheduling time 200 nanoseconds
  So every absolute scheduling time and all gate duration times must be a multiple of 200 nanoseconds.
- Maximum scheduling length of 52428800 nanoseconds (52,4 milliseconds)
  The longest gate duration time can be 52428800 nanoseconds, if longer periods are needed identical schedules can be concatenated.
- Only global schedule available
  The hardware supports only one schedule table for all ports. Therefore, the individual schedule tables of each port must be merged into one table. This is only possible if two entries with the same absolute time use the same Gate (FIFO) settings. If this is not possible, one of these entries must have its absolute time increased or decreased. The smallest value is 200 nanoseconds. All schedules must have the same absolute length, too. There are some hints regarding preparing schedule tables and merging in [7]. The merging itself is done by the generator.
- 1023 scheduler entries possible
  Because of the merging into the global schedule table it depends on the individual configuration of the schedule table of the different ports how many schedule entries are possible per port.

3.13 Port mirroring

The port mirroring feature allows to capture traffic on specific ports and copy them to a dedicated capture port for debug or diagnostic purposes. To have a more dedicated selection of traffic that one is interested in the following filter options can be used:

- Direction: Ingress packets, egress packets or both directions
- Address: Source MAC address, destination MAC address or both
- Used VLAN ID

The output of the mirrored packets can be influenced in following ways:

- Output dividing of packets: Only every second, third, … packet is mirrored
- The packet VLAN can be changed to a new VLAN or a new double VLAN

Optionally the port mirroring configuration can be saved in NV-Memory. In this case it is restored after a power cycle automatically.

Caution
Due to hardware limitations it is possible that not all packets are mirrored, depending on the load of the switch or of the mirroring switch port.

3.13.1 Example: Enable Port Mirroring

The following example describes how enabling the port mirroring could look like:

Example
Enable port mirroring

```
#include "EthSwt.h"

Std_ReturnType EnablePortMirroring(uint8 Switch, uint8 MirroringPort)
{
  Std_ReturnType retVal;
  EthSwt_PortMirrorCfgType MirrorConfig;

  MirrorConfig.CapturePortIdx = MirroringPort;
  /* Mirror egress and ingress traffic of switch port 1 and 2 */
  MirrorConfig.MirroredSwitchEgressPortIdxBitMask = 0x03;
  MirrorConfig.MirroredSwitchIngressPortIdxBitMask = 0x03;

  /* No output manipulation */
  MirrorConfig.MirroringMode = ETHSWT_MIRROR_MODE_NO_VLAN_RETAGGING;
  MirrorConfig.MirroringPacketDivider = 1;

  /* No input filtering */
  MirrorConfig.VlanIdFilter = 0xFFFF;
  MirrorConfig.dstMacAddrFilter[0] = 0;
  MirrorConfig.dstMacAddrFilter[1] = 0;
  MirrorConfig.dstMacAddrFilter[2] = 0;
  MirrorConfig.dstMacAddrFilter[3] = 0;
  MirrorConfig.dstMacAddrFilter[4] = 0;
  MirrorConfig.dstMacAddrFilter[5] = 0;
  MirrorConfig.srcMacAddrFilter[0] = 0;
  MirrorConfig.srcMacAddrFilter[1] = 0;
  MirrorConfig.srcMacAddrFilter[2] = 0;
  MirrorConfig.srcMacAddrFilter[3] = 0;
  MirrorConfig.srcMacAddrFilter[4] = 0;
  MirrorConfig.srcMacAddrFilter[5] = 0;

  /* Set and activate mirroring configuration */
  retVal = EthSwt_WritePortMirrorConfiguration(Switch, &MirrorConfig);
  if( retVal == E_OK )
  {
    retVal = EthSwt_SetPortMirrorState(Switch, ETHSWT_PORT_MIRROR_ENABLED);
  }

  return retVal;
}
```

3.13.2 Example: Disable Port Mirroring

The following example describes how disabling the port mirroring could look like:

Example
Disable port mirroring

```
#include "EthSwt.h"

Std_ReturnType DisablePortMirroring(uint8 Switch)
{
  Std_ReturnType retVal;
  EthSwt_PortMirrorStateType MirrorState;

  /* Stop possible active mirroring configuration */
  retVal = EthSwt_GetPortMirrorState(Switch, &MirrorState);
  if( retVal == E_OK && MirrorState == ETHSWT_PORT_MIRROR_ENABLED )
  {
    retVal = EthSwt_SetPortMirrorState(Switch, ETHSWT_PORT_MIRROR_DISABLED);
  }

  /* Invalidate mirroring configuration and delete it in NV-Memory */
  if( retVal == E_OK )
  {
    retVal = EthSwt_DeletePortMirrorConfiguration(Switch);
  }

  return retVal;
}
```

3.13.3 Example: Modify Port Mirroring Configuration

The following example describes how modifying the port mirroring configuration could look like:

Example
Modify port mirroring configuration

```
#include "EthSwt.h"

Std_ReturnType ChangePortMirroringPort(uint8 Switch, uint8 MirroringPort)
{
  Std_ReturnType retVal;
  EthSwt_PortMirrorStateType MirrorState;
  EthSwt_PortMirrorCfgType MirrorConfig;

  /* Stop possible active mirroring configuration */
  retVal = EthSwt_GetPortMirrorState(Switch, &MirrorState);
  if( retVal == E_OK && MirrorState == ETHSWT_PORT_MIRROR_ENABLED )
  {
    retVal = EthSwt_SetPortMirrorState(Switch, ETHSWT_PORT_MIRROR_DISABLED);
  }

  /* Get current configuration */
  if( retVal == E_OK )
  {
    retVal = EthSwt_ReadPortMirrorConfiguration(Switch, &MirrorConfig);
  }

  /* Modify configuration, set and activate it */
  if( retVal == E_OK )
  {
    MirrorConfig.CapturePortIdx = MirroringPort;
    retVal = EthSwt_WritePortMirrorConfiguration(Switch, &MirrorConfig);
    if( retVal == E_OK )
    {
      retVal = EthSwt_SetPortMirrorState(Switch, ETHSWT_PORT_MIRROR_ENABLED);
    }
  }

  return retVal;
}
```

3.13.4 Limitations of port mirroring

The driver is currently not able to provide the following functionality:

- Filtering of packets with a specified Source MAC address, destination MAC address or both
- Filtering of packets with a specified VLAN ID
- Output dividing
- Retagging the packets with a new VLAN ID

3.14 Error Handling

3.14.1 Development Error Reporting

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

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

The reported EthSwt ID is 89.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	`EthSwt_30_Sja1105PQRS_Init()`
0x01	`EthSwt_30_Sja1105PQRS_SwitchInit()`
0x02	`EthSwt_30_Sja1105PQRS_SetSwitchPortMode()`
0x03	`EthSwt_30_Sja1105PQRS_GetSwitchPortMode()`
0x04	`EthSwt_30_Sja1105PQRS_StartSwitchPortAutoNegotiation()`
0x05	`EthSwt_30_Sja1105PQRS_GetLinkState()`
0x06	`EthSwt_30_Sja1105PQRS_GetBaudRate()`
0x07	`EthSwt_30_Sja1105PQRS_GetDuplexMode()`
0x08	`EthSwt_30_Sja1105PQRS_GetPortMacAddr()`
0x09	`EthSwt_30_Sja1105PQRS_GetArlTable()`
0x0A	`EthSwt_30_Sja1105PQRS_GetBufferLevel()`
0x0B	`EthSwt_30_Sja1105PQRS_GetDropCount()`
0x0C	`EthSwt_30_Sja1105PQRS_EnableVlan()`
0x0D	`EthSwt_30_Sja1105PQRS_StoreConfiguration()`
0x0E	`EthSwt_30_Sja1105PQRS_ResetConfiguration()`
0x0F	`EthSwt_30_Sja1105PQRS_NvmSingleBlockCallback()`
0x10	`EthSwt_30_Sja1105PQRS_MainFunction()`
0x11	`EthSwt_30_Sja1105PQRS_GetVersionInfo()`
0x12	`EthSwt_30_Sja1105PQRS_SetMacLearningMode()`
0x13	`EthSwt_30_Sja1105PQRS_GetMacLearningMode()`
0x14	`EthSwt_30_Sja1105PQRS_GetEtherStats()`
0x23	`EthSwt_30_Sja1105PQRS_UpdateMCastPortAssignment()`
0x30	`EthSwt_30_Sja1105PQRS_GetSwitchReg()`
0x31	`EthSwt_30_Sja1105PQRS_SetSwitchReg()`
0x32	`EthSwt_30_Sja1105PQRS_ReadTrcvRegister()`
0x33	`EthSwt_30_Sja1105PQRS_WriteTrcvRegister()`
0x36	`EthSwt_30_Sja1105PQRS_WritePortMirrorConfiguration()`

Service ID	Service
0x37	EthSwt_30_Sja1105PQRS_ReadPortMirrorConfiguration()
0x38	EthSwt_30_Sja1105PQRS_GetPortMirrorState()
0x39	EthSwt_30_Sja1105PQRS_SetPortMirrorState()
0x40	EthSwt_30_Sja1105PQRS_VSwitchInit()
0x55	EthSwt_30_Sja1105PQRS_DeletePortMirrorConfiguration()
0xF0	Internal Frame Management API
0xF1	EthSwt_30_Sja1105PQRS_EnableEgressTimestamp()
0xF2	EthSwt_30_Sja1105PQRS_SetCorrectionTime()
0xF3	EthSwt_30_Sja1105PQRS_StartQbvSchedule()
0xF4	EthSwt_30_Sja1105PQRS_StopQbvSchedule()

Table 3-9 Service IDs

The errors reported to DET are described in the following table:

Error Code	Description
0x01	ETHSWT_30_SJA1105PQRS_E_INV_SWITCH_IDX API called with invalid switch index.
0x02	ETHSWT_30_SJA1105PQRS_E_NOT_INITIALIZED API called although module isn't initialized.
0x03	ETHSWT_30_SJA1105PQRS_E_INV_POINTER API called with invalid pointer (null pointer).
0x04	ETHSWT_30_SJA1105PQRS_E_ALREADY_INITIALIZED API called although module is already initialized.
0x05	ETHSWT_30_SJA1105PQRS_E_INV_API Invalid API called. E.g. EthTrcv wrapper API like EthSwt_30_Sja1105PQRS_GetLinkState() is called and corresponding EthTrcv doesn't provide the API.
0x06	ETHSWT_30_SJA1105PQRS_E_INV_SWITCHPORT_IDX API called with invalid port index.
0x07	ETHSWT_30_SJA1105PQRS_E_MEMORY_NOT_INITIALIZED *_INIT_* variables weren't neither be initialized by startup code nor by usage of EthSwt_30_Sja1105PQRS_InitMemory().

Table 3-10 Errors reported to DET

3.14.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service Dem_ReportErrorStatus() as specified in [3], if production error reporting is enabled (i.e. pre-compile parameter ETHSWT_PROD_ERROR_DETECT==STD_ON).

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

The errors reported to DEM are described in the following table:

Error Code	Description
E_ALIGNMENT	Ethernet frame with alignment error detected.
E_BUFFEROVERRUN	Buffer overrun detected.
E_CRC	Ethernet frame with CRC error detected.
E_UNDERSIZEPCKT	Undersized Ethernet frame detected.
E_INERROR	Erroneous Ethernet frame on ingress side detected.

Table 3-11 Errors reported to DEM

4 Integration

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

4.1 Embedded Implementation

The delivery of the EthSwt consists out of these files:

File Name	Description	Integration Tasks
EthSwt_GeneralTypes.h	Header-file containing the types and defines specified by AUTOSAR.	-
EthSwt_30_Sja1105PQRS.h	Header-file containing the declaration of the public API, types and defines. It also provides access to the symbolic name values generated by the configuration Tool.	-
EthSwt_30_Sja1105PQRS_Types.h	Header-file containing the derivative specific types and defines.	-
EthSwt_30_Sja1105PQRS_Int.h	Header-file containing private module defines and data not to be used by foreign modules.	-
EthSwt_30_Sja1105PQRS_Regs.h	Header-file containing the defines used for register addressing and manipulation.	-
EthSwt_30_Sja1105PQRS_LL.h	Header-file containing the declaration of the lower-layer API, types and defines used by the higher-layer.	-
EthSwt_30_Sja1105PQRS.c	Source-file containing the higher-layer implementation of Ethernet Switch driver. Contains the implementation of the AUTOSAR APIs and corresponding helper functions.	-
EthSwt_30_Sja1105PQRS_LL.c	Source-file containing the lower-layer implementation of Ethernet Switch driver. Contains the implementation of the hardware access abstraction (SPI, MII).	-
EthSwt_30_Sja1105PQRS_Mgmt.c	Source file containing the implementation for Frame Management.	-
EthSwt_30_Sja1105PQRS_Mgmt.h	Header-file containing the declaration of the public API,	-

File Name	Description	Integration Tasks
	types and defines for Frame Management.	
EthSwt_30_Sja1105PQRS_Time.c	Source file containing the implementation for Global Time Support / Time Synchronization.	-
EthSwt_30_Sja1105PQRS_Time.h	Header-file containing the declaration of the public API, types and defines for Global Time Support.	-
EthSwt_30_Sja1105PQRS_Time_Int.h	Header-file containing private module defines and data for Global Time Support not to be used by foreign modules.	-
EthSwt_30_Sja1105PQRS_Casc.c	Source file containing he implementation for switch cascading.	-
EthSwt_30_Sja1105PQRS_Casc.h	Header-file containing the declaration of the public API for switch cascading.	-
EthSwt_30_Sja1105PQRS _Mirror_Int.h	Static header file containing private types, data and functions of the port mirroring sub-module only to be used by the Ethernet switch driver itself.	-
EthSwt_30_Sja1105PQRS _Mirror.h	Static header file providing access to the public API of the port mirroring sub-module of the Ethernet switch driver.	-
EthSwt_30_Sja1105PQRS _Mirror.c	Static source file containing the implementation of the port mirroring sub-module of the Ethernet switch driver.	-
EthSwt_30_Sja1105PQRS _LL_Mirror.h	Static header file containing private types, data and function related to the port mirroring sub-module provided by the hardware specific driver implementation of the Ethernet switch driver only to be used by the driver itself.	-
EthSwt_30_Sja1105PQRS_Cfg.h	Header-file containing the pre-compile defines and the symbolic name values.	-
EthSwt_30_Sja1105PQRS_Lcfg.h	Header-file containing the declaration for the variables and constants of generated configuration data.	-

File Name	Description	Integration Tasks
EthSwt_30_Sja1105PQRS_Lcfg.c	Source-file containing the generated variables and constants.	-

Table 4-1 Implementation files

Do not edit manually
The static files must not be edited manually!

Do not edit manually
The generated files must not be edited manually but be generated by the DaVinci Configurator PRO where configuration changes are done.

4.2 Compiler Abstraction and Memory Mapping

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

The following table contains the memory section names and the compiler abstraction definitions, which are defined for the Ethernet Switch Driver, and illustrates their assignment among each other.

For better readability the table contains shortened compiler abstraction definitions. Each **`ETHSWT_30_SJA1105PQRS`** is substituted with **`ETHSWT`**.

Compiler Abstraction Definitions / Memory Mapping Sections	ETHSWT_CONST	ETHSWT_VAR_INIT	ETHSWT_VAR_NOINIT	ETHSWT_CODE	ETHSWT_APPL_CONST	ETHSWT_APPL_VAR
ETHSWT_START_SEC_CONST_8BIT ETHSWT_STOP_SEC_CONST_8BIT	■					
ETHSWT_START_SEC_CONST_16BIT ETHSWT_STOP_SEC_CONST_16BIT	■					
ETHSWT_START_SEC_CONST_32BIT ETHSWT_STOP_SEC_CONST_32BIT	■					

Compiler Abstraction Definitions / Memory Mapping Sections	ETHSWT_CONST	ETHSWT_VAR_INIT	ETHSWT_VAR_NOINIT	ETHSWT_CODE	ETHSWT_APPL_CONST	ETHSWT_APPL_VAR
ETHSWT_START_SEC_CONST_UNSPECIFIED ETHSWT_STOP_SEC_CONST_UNSPECIFIED	■					
ETHSWT_START_SEC_VAR_INIT_8BIT ETHSWT_STOP_SEC_VAR_INIT_8BIT		■				
ETHSWT_START_SEC_VAR_NOINIT_8BIT ETHSWT_STOP_SEC_VAR_NOINIT_8BIT			■			
ETHSWT_START_SEC_VAR_NOINIT_UNSPECIFIED ETHSWT_STOP_SEC_VAR_NOINIT_UNSPECIFIED			■			
ETHSWT_START_SEC_CODE ETHSWT_STOP_SEC_CODE				■		

Table 4-2 Compiler Abstraction and Memory Mapping

4.3 Critical Sections

This section describes the critical sections utilized by the Ethernet Switch Driver to ensure data consistency. All critical sections are listed in Table 4-3.

Note
For better readability the prefix **ETHSWT_30_SJA1105PQRS_EXCLUSIVE_AREA** was removed and must be considered during integration.
Example: **ETHSWT_30_SJA1105PQRS_EXCLUSIVE_AREA_SPI_SEQUENCE**

Critical Section	Description
SPI_SEQUENCE	The exclusive area ensures that setup of buffers (Spi_WriteIB, Spi_SetupEB) and transmission of the data on SPI (Spi_SyncTransmit, Spy_AsyncTransmit, Spi_GetSequenceResult) is consistent. The Interrupt for a finished SPI transaction must not be blocked by this exclusive area!
RX_MNGMT_ELEMENT_PROCESSING	The exclusive area ensures consistent handling of the management element pool.

Critical Section	Description
TX_MNGMT_ELEMENT_PROCESSING	The exclusive area ensures consistent handling of the management element pool.
TS_QUEUE_PROCESSING	The exclusive area ensures consistent handling of the timestamp element pool.
SET_CORRECTION_TIME	Ensure that the sequence of setting the correction time cannot be interrupted.
ARL_TABLE_ACCESS	Ensure that the sequence of searching the ARL table and writing or reading values cannot be interrupted.
QBV_START_STOP_SCHEDULE	Ensure that the sequence of starting or stopping the Qbv scheduling cannot be interrupted.

Table 4-3 Critical Sections

It is recommended to use OS Resources for the critical sections instead of using **`SuspendAllInterrupts()`** and **`ResumeAllInterrupts()`** for better performance.

4.4 Initialization

AUTOSAR specifies the **`EthSwt_SwitchInit()`** API to work in a synchronous way. However the initialization of the switch needs too much time to be done within the context of a task triggered in a periodic way. Doing so could lead to multiple task activation errors issued by the operating system.

Until AUTOSAR changes the behavior from synchronous to asynchronous, which also would lead in introducing changes to the Ethernet Interface, the following workaround is provided.

Workaround
Following section describes a workaround for the synchronous behavior of **`EthSwt_SwitchInit()`**

Additionally to the AUTOSAR API **`EthSwt_30_Sja1105PQRS_SwitchInit()`** the Vector specific API **`EthSwt_30_Sja1105PQRS_VSwitchInit()`** was introduced. It allows to shift the processing for the switch initialization into a background/idle task not triggered by a periodic event/timer. This allows the switch to be initialized without harming the timing restrictions.

The AUTOSAR **`EthSwt_30_Sja1105PQRS_SwitchInit()`** API will still be called by the Ethernet Interface but will only return **`E_OK`** if **`EthSwt_30_Sja1105PQRS_VSwitchInit()`** has finished successfully.

Example
Example implementation of the background/idle task

```
#include "EthSwt_30_Sja1105PQRS.h"

TASK(BackgroundTask)
{
  /* The BackgroundTask task is configured to be the idle task.
   * The task is preemptable and has the lowest priority.
   */

  uint8_least LoopIdx;
  boolean     InitDone[ETHSWT_30_SJA1105PQRS_SWITCH_NUM];

  /* initialize local state variable */
  for( LoopIdx = 0; LoopIdx < ETHSWT_30_SJA1105PQRS_SWITCH_NUM; LoopIdx++ )
  {
    InitDone[LoopIdx] = FALSE;
  }

  /* Tasks endless loop */
  while( 1 )
  {
    for( LoopIdx = 0; LoopIdx < ETHSWT_30_SJA1105PQRS_SWITCH_NUM; LoopIdx++ )
    {
      /* check if initialization was already performed */
      if( FALSE == InitDone[LoopIdx] )
      {
        /* trigger initialization */
        if( E_OK == EthSwt_30_Sja1105PQRS_VSwitchInit(LoopIdx) )
        {
          /* initialization successfully finished */
          InitDone[LoopIdx] = TRUE;
        }
        else
        {
          /* error handling */
        }
      }
    }
  }
}
```

Caution
It must be ensured that **`EthSwt_30_Sja1105PQRS_VSwitchInit()`** is only called once until it returns. Concurrent calls are not allowed.

Caution
It must be ensured that EthSM doesn't abort the try to process the **FULL_COMMUNICATION** request for an Ethernet Interface Controller, which provides an abstracted access to the underlying switch hardware.

4.5 NV-Memory

The Ethernet Switch driver can store parameters during runtime so they are persisted and can be restored after a system restart.

This mechanism utilizes the AUTOSAR NV-Manager (NvM), which processes the store and restore of the parameters. This is done by a memory handle called "NvM Block Descriptor", which is a representation of an NvM Block.

The following sections describe how the functionality can be enabled in the driver and how the related NvM Block Descriptor must be configured for each parameter the driver is able to store.

Note
The proper configuration of the "NvM Block Descriptor" is ensured by DaVinci Configurator PRO.

4.5.1 Learned Switch Table Entries

The Ethernet Switch driver can store the learned address lookup table entries (MAC-Address to port resolution).

The storage of the learned entries of the table is triggered by a call to the API `EthSwt_30_Sja1105PQRS_StoreConfiguration()`. If storage was triggered once the driver will download the entries stored to the switch with each call to `EthSwt_30_Sja1105PQRS_VSwitchInit()`. Commonly the initialization is only triggered after a reset of the Host-CPU.

To stop the download again, the stored information can be invalidated by a call to `EthSwt_30_Sja1105PQRS_ResetConfiguration()`.

Both APIs `EthSwt_30_Sja1105PQRS_StoreConfiguration()` and `EthSwt_30_Sja1105PQRS_ResetConfiguration()` trigger the change of the NV-Data only. If it is necessary to get informed about the finish of the change a user notification can be enabled. Please see 5.5.1.1 for information about this notification API.

Note
The storage and restore of the learned switch table entries is enabled by setting the parameters `EthSwtStoreConfigurationApi` and `EthSwtResetConfigurationApi`.

The following table shows the configuration of the “NvM Block Descriptor”

NvM Block Descriptor Parameter	Value
RAM Block Data) (`NvMRamBlockDataAddress`)	`NULL_PTR`
ROM Block Data (`NvMRomBlockDataAddress`)	`NULL_PTR`
Block Length (`NvMNvBlockLength`)	The NV-Data consists of a header storing meta information and the count of entries able to store. Header: 2 Byte Entries: 8 Byte * `EthSwtMaxArlTableEntries`
Select Block For ReadAll (`NvMSelectBlockForReadAll`)	`FALSE`
Select Block For WriteAll (`NvMSelectBlockForWriteAll`)	`FALSE`
Single Block Callback (`NvMSingleBlockCallback`)	`EthSwt_30_Sja1105PQRS_NvmSingleBlockCallback`

Table 4-4 Learned Switch Table Entries NvM Block Descriptor configuration

The storage and restore of the NV-Data is triggered by the driver itself and must not be handled during `NvM_ReadAll()` and `NvM_WriteAll()`.

4.5.2 Port Mirroring Configuration

The Ethernet Switch driver can store the port mirroring configuration and the state of the port mirroring.

The storage of the port mirroring configuration and its state is triggered by a call to the API `EthSwt_30_Sja1105PQRS_WritePortMirrorConfiguration()` or by a call to the API `EthSwt_30_Sja1105PQRS_SetPortMirrorState()`. The saved data is automatically restored with each call to `EthSwt_30_Sja1105PQRS_VSwitchInit()`. Commonly the initialization is only triggered after a reset of the Host-CPU.

To disable the automatic port mirroring re-configuration during initialization again, the stored information can be invalidated by a call to `EthSwt_30_Sja1105PQRS_DeletePortMirrorConfiguration()`.

Note
The storage and restore of the port mirroring configuration and state is enabled by referencing an NVM block descriptor by parameter `EthSwtConfigurationNvmBlockDescriptorRef`.

The following table shows the configuration of the “NvM Block Descriptor”

NvM Block Descriptor Parameter	Value
RAM Block Data) (`NvMRamBlockDataAddress`)	`NULL_PTR`
ROM Block Data (`NvMRomBlockDataAddress`)	`NULL_PTR`
Block Length (`NvMNvBlockLength`)	The NV-Data consists of the status and the configuration of the port mirroring: > EthSwt_PortMirrorStateType: 1 Byte > EthSwt_PortMirrorCfgType: 19 Byte
Select Block for ReadAll (`NvMSelectBlockForReadAll`)	`FALSE`
Select Block for WriteAll (`NvMSelectBlockForWriteAll`)	`FALSE`
Single Block Callback (`NvMSingleBlockCallback`)	`EthSwt_NvmSingleBlockCallback`

Table 4-5 Port mirroring configuration NvM Block Descriptor configuration

The storage and restore of the NV-Data is triggered by the driver itself and must not be handled during `NvM_ReadAll()` and `NvM_WriteAll()`.

4.6 SPI configuration

For configuration of the SJA1105P/Q/R/S specific settings in the SPI driver please refer to the documentation of the SJA1105P/Q/R/S [5], [6].

Note
Please note that the timing and especially “t2” mentioned in the “SPI read/write timing” chapter of the manual [5], [6] is important for correct operation of the switch and this software.

5 API Description

5.1 Type Definitions

The types defined by the EthSwt are described in this chapter.

5.1.1 General Types

This section describes the types defined by the AUTOSAR SWS and shared between all Ethernet Switch driver implementations. They are located in the `EthSwt_GeneralTypes.h` header-file.

Type Name	C-Type	Description	Value Range
EthSwt_StateType	uint8	AUTOSAR type for module state	`ETHSWT_STATE_UNINIT` Driver not initialized
			`ETHSWT_STATE_INIT` Driver initialized
			`ETHSWT_STATE_ACTIVE` Switch instance initialized (Configuration was loaded and switch is operational)
EthSwt_SwitchIdxType	uint8	AUTOSAR type for switch index	`0 .. 255`
EthSwt_PortIdxType	uint8	AUTOSAR type for port index	`0 .. 255`
EthSwt_BufferLevelType	uint32	AUTOSAR type for buffer level	`0 .. 4.294.967.295`
EthSwt_MacLearningType	uint8	AUTOSAR type for MAC learning type	`ETHSWT_MACLEARNING_HWDISABLED` MAC learning disabled
			`ETHSWT_MACLEARNING_HWENABLED` MAC learning by hardware enabled
			`ETHSWT_MACLEARNING_SWENABLED` MAC learning by software enabled
EthSwt_PortModeType	uint8	AUTOSAR type for port mode	`ETHSWT_MODE_DOWN_SWITCHPORT` Port disabled
			`ETHSWT_MODE_ACTIVE_SWITCHPORT` Port enabled
EthSwt_MCastPortAssignActionType	uint8	AUTOSAR type for multicast port assignment action	`ETHSWT_MCAST_PORT_ASSIGN_ACTION_ADD` Add multicast assignment to port
			`ETHSWT_MCAST_PORT_ASSIGN_ACTION_REMOVE`

Type Name	C-Type	Description	Value Range
			Remove multicast assignment from port
EthSwt_PortMirrorStateType	uint8	AUTOSAR type for port mirroring state	ETHSWT_PORT_MIRROR_DISABLED Port mirroring disabled
			ETHSWT_PORT_MIRROR_ENABLED Port mirroring enabled
			ETHSWT_PORT_MIRROR_INVALID Port mirroring invalid

Table 5-1 General type definitions

EthSwt_ConfigType

This structure contains the information needed in multiple configuration use-case. However AUTOSAR specifies the Ethernet Switch driver to pre-compile only and therefore the use-case isn't supported.

Struct Element Name	C-Type	Description	Value Range
DummyData	uint8	Dummy member to be able to provide structure	0 .. 255

Table 5-2 EthSwt_ConfigType

EthSwt_MacVlanType

This structure contains the information about an address resolution table entry.

Struct Element Name	C-Type	Description	Value Range
MacAddr	uint8[6]	MAC address	0 .. 255
VlanId	uint16	VLAN ID	0 .. 4094
SwitchPort	uint8	Index of the port	0 .. 255

Table 5-3 EthSwt_MacVlanType

EthSwt_MgmtInfoType

This structure contains the frame management information.

Struct Element Name	C-Type	Description	Value Range
SwitchIdx	EthSwt_SwitchIdxType	EthSwt switch index	Refere to the value range defined for EthSwt_SwitchIdxType.
PortIdx	uint8	Index of the port	0 .. 255
Eth_FrameIdType	FrameId	Id to identify frames in queues and pools	0..65535

Table 5-4 EthSwt_MacVlanType

EthSwt_PortMirrorCfgType

This structure contains the mirroring configuration.

Struct Element Name	C-Type	Description	Value Range
CapturePortIdx	uint8	Switch port which captures mirrored traffic - symbolic name value from configuration must be used	0 .. 255
srcMacAddrFilter	uint8[6]	Source MAC address of packets which shall be mirrored. This element is ignored if it is set to {0, 0, 0, 0, 0, 0}.	0 .. 255
dstMacAddrFilter	uint8[6]	Destination MAC address of packets which shall be mirrored. This element is ignored if it is set to {0, 0, 0, 0, 0, 0}.	0 .. 255
VlanIdFilter	uint16	VLAN ID of frames which shall be mirrored. This element is ignored if it is set to 65535.	0 .. 4094; 65535
MirroringPacketDivider	uint8	Limits mirrored frames: 1 -> all frames, 2 -> every second frame, ... This element is ignored if it is set to 1.	1 .. 255
MirroringMode	uint8	Retagging of mirrored frames: 0x00 == no VLAN retagging, 0x01 == VLAN retagging, 0x02 == VLAN double retagging	0 .. 2
VlanOuterTag	uint16	VLAN ID used for VLAN retagging or as outer tag for VLAN double retagging	0 .. 4094
VlanInnerTag	uint16	VLAN ID used as inner tag for VLAN double retagging	0 .. 4094
MirroredSwitchEgressPortIdxBitMask	uint32	Bit mask defining ports where egress frames shall be mirrored - symbolic name values from configuration must be used	0 .. 4.294.967.295
MirroredSwitchIngressPortIdxBitMask	uint32	Bit mask defining ports where ingress frames shall be mirrored - symbolic name values from configuration must be used	0 .. 4.294.967.295

Table 5-5 EthSwt_PortMirrorCfgType

5.1.2 Driver Specific Types

This section describes the derivative specific. They are located in the `EthSwt_30_Sja1105PQRS_Types.h` header-file.

5.2 Services provided by EthSwt

5.2.1 AUTOSAR API

This section contains the API description for the APIs specified by AUTOSAR.

5.2.1.1 EthSwt_30_Sja1105PQRS_InitMemory

Prototype	
`void` **`EthSwt_30_Sja1105PQRS_InitMemory`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Function for *_INIT_*-variable initialization.	
Particularities and Limitations	
> Module must not be initialized > Function shall be called from task level Service to initialize module global variables at power up. This function can be used to initialize the variables in *_INIT_* sections in case they are not initialized by the startup code.	
Call context	
> Task level	

Table 5-6 EthSwt_30_Sja1105PQRS_InitMemory

5.2.1.2 EthSwt_30_Sja1105PQRS_Init

Prototype	
`void` **`EthSwt_30_Sja1105PQRS_Init`** `(const EthSwt_ConfigType* ConfigPtr)`	
Parameter	
ConfigPtr [in]	Configuration structure for initializing the module
Return code	
void	none
Functional Description	
Initialization function of the EthSwt.	
Particularities and Limitations	
Specification of module initialization > Interrupts have to be disabled.The module has to be uninitialized.	

This function initializes the module EthSwt. It initializes all variables and sets the module state to initialized.
Call context
> Task level

Table 5-7 EthSwt_30_Sja1105PQRS_Init

5.2.1.3 EthSwt_30_Sja1105PQRS_SwitchInit

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_SwitchInit`** `(EthSwt_SwitchIdxType SwitchIdx)`	
Parameter	
SwitchIdx [in]	Index of the switch instance to be initialized
Return code	
Std_ReturnType	> E_NOT_OK EthSwt_30_Sja1105PQRS_VSwitchInit() is still pending or hasn't finished successful. > E_OK EthSwt_30_Sja1105PQRS_VSwitchInit() has finished successful.
Functional Description	
Configuration function of an EthSwt instance.	
Particularities and Limitations	
AUTOSAR specifies this function to be called synchronously. However the initialization process does take as much time to hurt the timings of task activation. Therefore this function only checks if EthSwt_30_Sja1105PQRS_VSwitchInit(), which has to run in a preemptable idle task, has finished successful. The module has to be initialized. This function resets and configures the switch instance. Configuration of ingress-, egress-behavior, writing of static address resolution table entries and internal ports setup.	
Call context	
> Task level	

Table 5-8 EthSwt_30_Sja1105PQRS_SwitchInit

5.2.1.4 EthSwt_30_Sja1105PQRS_VSwitchInit

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_VSwitchInit`** `(EthSwt_SwitchIdxType SwitchIdx)`	
Parameter	
SwitchIdx [in]	Index of the switch instance to be initialized
Return code	
Std_ReturnType	> E_NOT_OK Switch instance initialization not successful.

	> E_OK Switch instance initialization successful.
Functional Description	
Configuration function of an EthSwt instance.	
Particularities and Limitations	
AUTOSAR specifies the function EthSwt_SwitchInit() to be called synchronously. However the initialization process does take as much time to hurt the timings of task activation. Therefore this function shifts the processing of the initialization into a preemptable idle task. The module has to be initialized. This function resets and configures the switch instance. Configuration of ingress-, egress-behavior, writing of static address resolution table entries and internal ports setup.	
Call context	
> Must be called in preemptable idle task.	

Table 5-9 EthSwt_30_Sja1105PQRS_VSwitchInit

5.2.1.5 EthSwt_30_Sja1105PQRS_SetSwitchPortMode

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_SetSwitchPortMode`** `(EthSwt_SwitchIdxType SwitchIdx, EthSwt_PortIdxType PortIdx, EthTrcv_ModeType PortMode)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
PortIdx [in]	Index of the port
PortMode [in]	Mode that shall be applied ETHTRCV_MODE_DOWN: Disable communication ability on port ETHTRCV_MODE_ACTIVE: Enable communication ability on port
Return code	
Std_ReturnType	> E_NOT_OK Mode change not successful > E_OK Mode change successful
Functional Description	
Mode change function for a switch port.	
Particularities and Limitations	
Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows to change the mode of a switch port. It either redirects the call to the corresponding Transceiver module (for external configured ports) or applies the change by itself.	
Call context	
> Task context	

Table 5-10 EthSwt_30_Sja1105PQRS_SetSwitchPortMode

5.2.1.6 EthSwt_30_Sja1105PQRS_GetSwitchPortMode

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_GetSwitchPortMode`** `(EthSwt_SwitchIdxType SwitchIdx, EthSwt_PortIdxType PortIdx, EthTrcv_ModeType* ModePtr)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
PortIdx [in]	Index of the port
ModePtr [out]	Retrieved Mode ETHTRCV_MODE_DOWN: Communication ability on port is disabled ETHTRCV_MODE_ACTIVE: Communication ability on port is enabled
Return code	
Std_ReturnType	> E_NOT_OK Mode retrieval not successful > E_OK Mode retrieval successful
Functional Description	
Mode retrieval function for a switch port.	
Particularities and Limitations	
Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows retrieving the mode of a switch port. It either redirects the call to the corresponding Transceiver module (for external configured ports) or retrieves the mode by itself.	
Call context	
> Task context	

Table 5-11 EthSwt_30_Sja1105PQRS_GetSwitchPortMode

5.2.1.7 EthSwt_30_Sja1105PQRS_StartSwitchPortAutoNegotiation

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_StartSwitchPortAutoNegotiation`** `(EthSwt_SwitchIdxType SwitchIdx, EthSwt_PortIdxType PortIdx)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
PortIdx [in]	Index of the port
Return code	
Std_ReturnType	> E_NOT_OK Auto Negotiation triggering not successful > E_OK Auto Negotiation triggering successful
Functional Description	
Trigger Auto Negotiation function for a switch port.	
Particularities and Limitations	
Switch instance has to be initialized by EthSwt_SwitchInit().	

This function allows triggering the Auto Negotiation on a switch port. It either redirects the call to the corresponding Transceiver module (for external configured ports) or triggers the Auto Negotiation by itself.
Call context
> Task context

Table 5-12 EthSwt_30_Sja1105PQRS_StartSwitchPortAutoNegotiation

5.2.1.8 EthSwt_30_Sja1105PQRS_GetLinkState

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_GetLinkState`** `(EthSwt_SwitchIdxType SwitchIdx, EthSwt_PortIdxType PortIdx, EthTrcv_LinkStateType* LinkStatePtr)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
PortIdx [in]	Index of the port
LinkStatePtr [out]	Retrieved Link State ETHTRCV_LINK_STATE_DOWN: No link established on port ETHTRCV_LINK_STATE_ACTIVE: Link established on port
Return code	
Std_ReturnType	> E_NOT_OK Link State retrieval not successful > E_OK Link State retrieval successful
Functional Description	
Link State retrieval function for a switch port.	
Particularities and Limitations	
Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows retrieving the Link State of a switch port. It either redirects the call to the corresponding Transceiver module (for external configured ports) or retrieves the Link State by itself.	
Call context	
> Task context	

Table 5-13 EthSwt_30_Sja1105PQRS_GetLinkState

5.2.1.9 EthSwt_30_Sja1105PQRS_GetBaudRate

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_GetBaudRate`** `(EthSwt_SwitchIdxType SwitchIdx, EthSwt_PortIdxType PortIdx, EthTrcv_BaudRateType* BaudRatePtr)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
PortIdx [in]	Index of the port
BaudRatePtr [out]	Retrieved Baud Rate

	ETHTRCV_BAUD_RATE_10MBIT: 10 MBit Baud Rate ETHTRCV_BAUD_RATE_100MBIT: 100 MBit Baud Rate ETHTRCV_BAUD_RATE_1000MBIT: 1000 MBit Baud Rate
Return code	
Std_ReturnType	> E_NOT_OK Baud Rate retrieval not successful > E_OK Baud Rate retrieval successful
Functional Description	
Baud Rate retrieval function for a switch port.	
Particularities and Limitations	
Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows to retrieve the Baud Rate of a switch port. It either redirects the call to the corresponding Transceiver module (for external configured ports) or retrieves the Baud Rate by itself.	
Call context	
> Task context	

Table 5-14 EthSwt_30_Sja1105PQRS_GetBaudRate

5.2.1.10 EthSwt_30_Sja1105PQRS_GetDuplexMode

Prototype	
`Std_ReturnType EthSwt_30_Sja1105PQRS_GetDuplexMode (EthSwt_SwitchIdxType SwitchIdx, EthSwt_PortIdxType PortIdx, EthTrcv_DuplexModeType* DuplexModePtr)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
PortIdx [in]	Index of the port
DuplexModePtr [out]	Retrieved Duplex Mode ETHTRCV_DUPLEX_MODE_HALF: Half Duplex Mode ETHTRCV_DUPLEX_MODE_FULL: Full Duplex Mode
Return code	
Std_ReturnType	> E_NOT_OK Duplex Mode retrieval not successful > E_OK Duplex Mode retrieval successful
Functional Description	
Duplex Mode retrieval function for a switch port.	
Particularities and Limitations	
Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows retrieving the Duplex Mode of a switch port. It either redirects the call to the corresponding Transceiver module (for external configured ports) or retrieves the Duplex Mode by itself.	
Call context	
> Task context	

Table 5-15 EthSwt_30_Sja1105PQRS_GetDuplexMode

5.2.1.11 EthSwt_30_Sja1105PQRS_GetPortMacAddr

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_GetPortMacAddr`** `(const uint8* MacAddrPtr, EthSwt_SwitchIdxType* SwitchIdxPtr, EthSwt_PortIdxType* PortIdxPtr)`	
Parameter	
MacAddrPtr [in]	MAC address to be queried
SwitchIdxPtr [out]	Index of the switch instance the corresponding frame was received on
PortIdxPtr [out]	Index of the port the corresponding frame was received on
Return code	
Std_ReturnType	> E_NOT_OK Retrieval not successful > E_OK Retrieval successful
Functional Description	
Function retrieves the switch instance and port a MAC address is assigned to.	
Particularities and Limitations	
Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows retrieving the switch instance and port an Ethernet frame with a source MAC address matching the given MAC address was received on.	
Call context	
> Task context	

Table 5-16 EthSwt_30_Sja1105PQRS_GetPortMacAddr

5.2.1.12 EthSwt_30_Sja1105PQRS_GetArlTable

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_GetArlTable`** `(EthSwt_SwitchIdxType SwitchIdx, uint32* LenPtr, EthSwt_MacVlanType* ArlTablePtr)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
LenPtr [in/out]	in: Size of the passed buffer the entries shall be written to out: Number of entries written into buffer
ArlTablePtr [out]	Pointer to the buffer the data shall be written to
Return code	
Std_ReturnType	> E_NOT_OK Retrieval not successful > E_OK Retrieval successful
Functional Description	
Function retrieves the complete address resolution table.	

Particularities and Limitations
Switch instance has to be initialized by EthSwt_SwitchInit().
This function allows retrieving the valid entries of the address resolution table of a switch instance.
Call context
> Task context

Table 5-17 EthSwt_30_Sja1105PQRS_GetArlTable

5.2.1.13 EthSwt_30_Sja1105PQRS_GetBufferLevel

Prototype	
`Std_ReturnType EthSwt_30_Sja1105PQRS_GetBufferLevel (EthSwt_SwitchIdxType SwitchIdx, EthSwt_BufferLevelType* BufferLevelPtr)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
BufferLevelPtr [out]	Level of buffer occupation in bytes
Return code	
Std_ReturnType	> E_NOT_OK Retrieval not successful > E_OK Retrieval successful
Functional Description	
Function retrieves current buffer occupation.	
Particularities and Limitations	
Not supported by hardware. Will always return E_NOT_OK. Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows retrieving the buffer occupation of a switch instance.	
Call context	
> Task context	

Table 5-18 EthSwt_30_Sja1105PQRS_GetBufferLevel

5.2.1.14 EthSwt_30_Sja1105PQRS_GetDropCount

Prototype	
`Std_ReturnType EthSwt_30_Sja1105PQRS_GetDropCount (EthSwt_SwitchIdxType SwitchIdx, uint16* LenPtr, uint32* DropCountPtr)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
LenPtr [in/out]	in: Size of the passed buffer the drop counts shall be written to out: Number of drop counts written into buffer
DropCountPtr [out]	Pointer to the buffer the data shall be written to

Return code	
Std_ReturnType	> E_NOT_OK Retrieval not successful > E_OK Retrieval successful
Functional Description	
Function retrieves the drop counts according to the AUTOSAR SWS.	
Particularities and Limitations	
No hardware specific drop counts are written. Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows retrieving the drop counts specified by the AUTOSAR SWS. Each count is the sum of the drop count of all ports.	
Call context	
> Task context	

Table 5-19 EthSwt_30_Sja1105PQRS_GetDropCount

5.2.1.15 EthSwt_30_Sja1105PQRS_GetEtherStats

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_GetEtherStats`** `(EthSwt_SwitchIdxType SwitchIdx, EthSwt_PortIdxType SwitchPortIdx, uint32* etherStats)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
SwitchPortIdx [in]	Index of the Port
etherStats [out]	Pointer to the buffer the data shall be written to
Return code	
ETHSWT_Std_ReturnType	> E_NOT_OK Retrieval not successful > E_OK Retrieval successful
Functional Description	
Function retrieves the Ethernet statistic counters according to the AUTOSAR SWS.	
Particularities and Limitations	
Statistic counters that are not available contain the max value for the uint32 data type. Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows retrieving the Ethernet statistic counters specified by the AUTOSAR SWS for one Switch Port.	
Call context	
> Task context	

Table 5-20 EthSwt_30_Sja1105PQRS_GetEtherStats

5.2.1.16 EthSwt_30_Sja1105PQRS_GetSwitchReg

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_GetSwitchReg`** `(EthSwt_SwitchIdxType SwitchIdx, uint32 page, uint32 registerAddr, uint32* registerContent)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
page [in]	Addressed memory page (not applicable for SJA1105P/Q/R/S)
registerAddr [in]	Addressed register
registerContent [out]	Content of the register read
Return code	
Std_ReturnType	> E_NOT_OK Retrieval not successful > E_OK Retrieval successful
Functional Description	
Function retrieves the value of a switch register.	
Particularities and Limitations	
SJA1105P/Q/R/S 's memory space isn't divided into pages. Therefore the page parameter is unused. Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows retrieving the value of a switch register.	
Call context	
> Task context	

Table 5-21 EthSwt_30_Sja1105PQRS_GetSwitchReg

5.2.1.17 EthSwt_30_Sja1105PQRS_SetSwitchReg

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_SetSwitchReg`** `(EthSwt_SwitchIdxType SwitchIdx, uint32 page, uint32 registerAddr, uint32 registerContent)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
page [in]	Addressed memory page (not applicable for SJA1105P/Q/R/S)
registerAddr [in]	Addressed register
registerContent [in]	Content that shall be written to the register
Return code	
Std_ReturnType	> E_NOT_OK Retrieval not successful > E_OK Retrieval successful
Functional Description	
Function sets the value of a switch register.	

Particularities and Limitations
SJA1105P/Q/R/S 's memory space isn't divided into pages. Therefore the page parameter is unused. Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows setting the value of a switch register.
Call context
> Task context

Table 5-22 EthSwt_30_Sja1105PQRS_SetSwitchReg

5.2.1.18 EthSwt_30_Sja1105PQRS_EnableVlan

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_EnableVlan`** `(EthSwt_SwitchIdxType SwitchIdx, EthSwt_PortIdxType PortIdx, uint16 VlanId, boolean Enable)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
PortIdx [in]	Index of the port
VlanId [in]	ID of VLAN the rule shall be applied for
Enable [in]	Behavior that shall be applied FALS: VLAN shall not be forwarded TRUE: VLAN shall be forwarded
Return code	
Std_ReturnType	> E_NOT_OK Rule appliance not successful > E_OK Rule appliance successful
Functional Description	
Function allows changing the VLAN forwarding behavior of a port.	
Particularities and Limitations	
Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows changing the VLAN forwarding behavior of a port. The VLAN forwarding for the corresponding VLAN can either be disable or enabled.	
Call context	
> Task context	

Table 5-23 EthSwt_30_Sja1105PQRS_EnableVlan

5.2.1.19 EthSwt_30_Sja1105PQRS_StoreConfiguration

Prototype
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_StoreConfiguration`** `(EthSwt_SwitchIdxType SwitchIdx)`

Parameter	
SwitchIdx [in]	Index of the switch instance
Return code	
Std_ReturnType	> E_NOT_OK Storage not successful > E_OK Storage successful
Functional Description	
Function allows storing the current MAC/Port table in NV RAM.	
Particularities and Limitations	
Concurrent calls will lead to returning E_NOT_OK until the async storage process of NvM has finished. Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows storing the current MAC/Port table retrieved out of the address resolution table of the switch in the NV Ram.	
Call context	
> Task context	

Table 5-24 EthSwt_30_Sja1105PQRS_StoreConfiguration

5.2.1.20 EthSwt_30_Sja1105PQRS_ResetConfiguration

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_ResetConfiguration`** `(EthSwt_SwitchIdxType SwitchIdx)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
Return code	
Std_ReturnType	> E_NOT_OK Reset not successful > E_OK Reset successful
Functional Description	
Function resets learned MAC port mapping table in NV RAM.	
Particularities and Limitations	
Concurrent calls will lead to returning E_NOT_OK until the async storage process of NvM has finished. Switch instance has to be initialized by EthSwt_SwitchInit(). The function resets the NV RAM used to store the learned MAC to port mapping table.	
Call context	
> Task context	

Table 5-25 EthSwt_30_Sja1105PQRS_ResetConfiguration

5.2.1.21 EthSwt_30_Sja1105PQRS_SetMacLearningMode

Prototype	
`Std_ReturnType EthSwt_30_Sja1105PQRS_SetMacLearningMode (EthSwt_SwitchIdxType SwitchIdx, EthSwt_PortIdxType PortIdx, EthSwt_MacLearningType MacLearningMode)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
PortIdx [in]	Index of the port
MacLearningMode [in]	Learning Mode to be applied ETHSWT_MACLEARNING_HWDISABLED: Disables Hardware MAC learning ETHSWT_MACLEARNING_HWENABLED: Enables Hardware MAC learning ETHSWT_MACLEARNING_SWENABLED: Enables Software MAC learning (disables Hardware MAC learning)
Return code	
Std_ReturnType	> E_NOT_OK MAC learning mode change not successful > E_OK MAC learning mode change successful
Functional Description	
Function allows influencing the MAC learning behavior of the switch.	
Particularities and Limitations	
Enabling Software MAC learning by passing ETHSWT_MACLEARNING_SWENABLED isn't supported! Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows changing the MAC learning mode of the switch.	
Call context	
> Task context	

Table 5-26 EthSwt_30_Sja1105PQRS_SetMacLearningMode

5.2.1.22 EthSwt_30_Sja1105PQRS_GetMacLearningMode

Prototype	
`Std_ReturnType EthSwt_30_Sja1105PQRS_GetMacLearningMode (EthSwt_SwitchIdxType SwitchIdx, EthSwt_PortIdxType PortIdx, ETHSWT_P2EthSwt_MacLearningType MacLearningModePtr)`	
Parameter	
SwitchIdx [in]	Index of the switch instance
PortIdx [in]	Index of the port
MacLearningModePtr [out]	Learning Mode retrieved ETHSWT_MACLEARNING_HWDISABLED: Hardware MAC learning disabled ETHSWT_MACLEARNING_HWENABLED: Hardware MAC learning enabled ETHSWT_MACLEARNING_SWENABLED: Software MAC learning enabled

Return code	
Std_ReturnType	> E_NOT_OK MAC learning mode retrieval not successful > E_OK MAC learning mode retrieval successful
Functional Description	
Function allows retrieving the MAC learning behavior of the switch.	
Particularities and Limitations	
Because setting of ETHSWT_MACLEARNING_SWENABLED isn't supported API will never return this learning mode. Switch instance has to be initialized by EthSwt_SwitchInit(). This function allows retrieve the MAC learning mode of the switch.	
Call context	
> Task context	

Table 5-27 EthSwt_30_Sja1105PQRS_GetMacLearningMode

5.2.1.23 EthSwt_30_Sja1105PQRS_MainFunction

Prototype	
`void EthSwt_30_Sja1105PQRS_MainFunction (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Main function of the EthSwt.	
Particularities and Limitations	
Switch instance has to be initialized by EthSwt_SwitchInit(). This function is called periodically and recognizes the events that shall be directed to DEM.	
Call context	
> Task context	

Table 5-28 EthSwt_30_Sja1105PQRS_MainFunction

5.2.1.24 EthSwt_30_Sja1105PQRS_AsyncProcessingMainFunction

Prototype	
`void EthSwt_30_Sja1105PQRS_AsyncProcessingMainFunction (void)`	
Parameter	
void	none
Return code	
void	none

Functional Description
Main function processing asynchronous tasks.
Particularities and Limitations
Switch instance has to be initialized by EthSwt_SwitchInit(). This function is called periodically and processes the asynchronous tasks of the driver.
Call context
> Backgounrd-/Idle-Task able to preempted by other tasks.

Table 5-29 EthSwt_30_Sja1105PQRS_MainFunction

5.2.1.25 EthSwt_30_Sja1105PQRS_GetVersionInfo

Prototype	
`void EthSwt_30_Sja1105PQRS_GetVersionInfo (Std_VersionInfoType* VersionInfoPtr)`	
Parameter	
VersionInfoPtr [out]	Pointer to buffer where to store the version information
Return code	
void	none
Functional Description	
Returns the version information.	
Particularities and Limitations	
> Input parameter must not be NULL.Function shall be called from task level EthSwt_30_Sja1105PQRS_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.	
Call context	
FunctionCallcontext	

Table 5-30 EthSwt_30_Sja1105PQRS_GetVersionInfo

5.2.1.26 EthSwt_30_Sja1105PQRS_UpdateMCastPortAssignment

Prototype	
`Std_ReturnType EthSwt_30_Sja1105PQRS_UpdateMCastPortAssignment (EthSwt_SwitchIdxType SwitchIdx, EthSwt_PortIdxType PortIdx, const uint8* MCastAddr, EthSwt_MCastPortAssignActionType Action)`	
Parameter	
SwitchIdx [in]	Index of the switch
PortIdx [in]	Index of the switch port
MCastAddr [in]	Pointer to the MAC Multicast Address
Action [in]	Action that shall be applied

	• ETHSWT_MCAST_PORT_ASSIGN_ACTION_ADD: Request passing of multicast on the port • ETHSWT_MCAST_PORT_ASSIGN_ACTION_REMOVE: Request removal of multicast on the port
Return code	
Std_ReturnType	E_OK Assignment adapted successfully
Std_ReturnType	E_NOT_OK Assignment could not be adapted
Functional Description	
This function allows to set the mirror filter mode for ingress/egress traffic on a per switch basis.	
Particularities and Limitations	
Call context	
> Task	

Table 5-31 EthSwt_30_Sja1105PQRS_UpdateMCastPortAssignment

5.2.1.27 EthSwt_30_Sja1105PQRS_WritePortMirrorConfiguration

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_WritePortMirrorConfiguration`** `(uint8 MirroredSwitchIdx, EthSwt_PortMirrorCfgType PortMirrorConfigurationPtr)`	
Parameter	
MirroredSwitchIdx [in]	Identifier of the switch which shall get the new configuration
PortMirrorConfigurationPtr [in]	Pointer to the new configuration
Return code	
Std_ReturnType	E_NOT_OK - Error writing configuration
Std_ReturnType	E_OK - Port mirroring configuration is valid and has been saved to buffer
Std_ReturnType	ETHSWT_PORT_MIRRORING_CONFIGURATION_NOT_SUPPORTED - Port mirroring configuration is invalid
Functional Description	
Write port mirroring configuration into internal buffer.	
Particularities and Limitations	
> Module is initialized. Ethernet switch is initialized. This function validates the given port mirroring configuration and writes it into an internal buffer in the microcontroller. The stored configuration will be written into the switch later when the function EthSwt_30_Sja1105PQRS_SetPortMirrorState() is called to set the mirroring state to active.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-32 EthSwt_30_Sja1105PQRS_WritePortMirrorConfiguration

5.2.1.28 EthSwt_30_Sja1105PQRS_ReadPortMirrorConfiguration

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_ReadPortMirrorConfiguration`** `(uint8 MirroredSwitchIdx, EthSwt_PortMirrorCfgType PortMirrorConfigurationPtr)`	
Parameter	
MirroredSwitchIdx [in]	Identifier of switch which configuration shall be read
PortMirrorConfigurationPtr [out]	Pointer to buffer where configuration shall be stored
Return code	
Std_ReturnType	E_NOT_OK - No port mirroring configuration has been found
Std_ReturnType	E_OK - Retrieval of port mirroring configuration has been successful
Functional Description	
Read port mirroring configuration from internal buffer.	
Particularities and Limitations	
> Module is initialized. Ethernet switch is initialized. This function reads the current mirroring configuration of the specified switch from the internal buffer in the microcontroller.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-33 EthSwt_30_Sja1105PQRS_ReadPortMirrorConfiguration

5.2.1.29 EthSwt_30_Sja1105PQRS_GetPortMirrorState

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_GetPortMirrorState`** `(uint8 MirroredSwitchIdx, EthSwt_PortMirrorStateType PortMirrorStatePtr)`	
Parameter	
MirroredSwitchIdx [in]	Identifier of switch which state shall be read
PortMirrorStatePtr [out]	Pointer to variable where state shall be stored
Return code	
Std_ReturnType	E_NOT_OK - Error getting port mirroring state
Std_ReturnType	E_OK - Retrieval of port mirroring state has been successful
Functional Description	
Get current port mirroring state.	
Particularities and Limitations	
> Module is initialized. Ethernet switch is initialized.	

This function returns the current port mirroring state of the specified switch.
Call context
> TASK > This function is Synchronous > This function is Non-Reentrant

Table 5-34 EthSwt_30_Sja1105PQRS_GetPortMirrorState

5.2.1.30 EthSwt_30_Sja1105PQRS_SetPortMirrorState

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_SetPortMirrorState`** `(uint8 MirroredSwitchIdx, EthSwt_PortMirrorStateType PortMirrorState)`	
Parameter	
MirroredSwitchIdx [in]	Identifier of switch which state shall be written
PortMirrorState [in]	New port mirroring state
Return code	
Std_ReturnType	E_NOT_OK - Error setting port mirroring state
Std_ReturnType	E_OK - Setting of port mirroring state has been successful
Functional Description	
Set new port mirroring state.	
Particularities and Limitations	
> Module is initialized.Ethernet switch is initialized. This function set the new port mirroring state of the specified switch.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-35 EthSwt_30_Sja1105PQRS_SetPortMirrorState

5.2.1.31 EthSwt_30_Sja1105PQRS_DeletePortMirrorConfiguration

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_DeletePortMirrorConfiguration`** `(uint8 MirroredSwitchIdx)`	
Parameter	
MirroredSwitchIdx [in]	Identifier of switch which configuration shall be deleted
Return code	
Std_ReturnType	E_NOT_OK - Error deleting port mirroring configuration
Std_ReturnType	E_OK - Deleting of port mirroring configuration has been successful
Functional Description	
Delete stored mirroring configuration.	

Particularities and Limitations
> Module is initialized.Ethernet switch is initialized.
This function deletes the port mirroring configuration of the specified switch. This is only possible if the port mirroring state is PORT_MIRRORING_DISABLED.
Call context
> TASK > This function is Synchronous > This function is Non-Reentrant

Table 5-36 EthSwt_30_Sja1105PQRS_DeletePortMirrorConfiguration

5.2.2 Frame Management API

5.2.2.1 EthSwt_30_Sja1105PQRS_SetMgmtInfo

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_SetMgmtInfo`** `(uint8 CtrlIdx, uint8 BufIdx, const EthSwt_MgmtInfoType* MgmtInfoPtr)`	
Parameter	
CtrlIdx [in]	Ethernet Controller index
BufIdx [in]	Ethernet Tx Buffer index
MgmtInfoPtr [in]	Pointer to the management information
Return code	
Std_ReturnType	void
Functional Description	
Sets the management information for the provided frame to achieve transmission only on specific ports.	
Particularities and Limitations	
Switch module must have been initialized.	
Call context	
> Task or interrupt	

Table 5-37 EthSwt_30_Sja1105PQRS_SetMgmtInfo

5.2.2.2 EthSwt_30_Sja1105PQRS_EthTxAdaptBufferLength

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_EthTxAdaptBufferLength`** `(uint16 LengthPtr)`	
Parameter	
LengthPtr [in/out]	[in]: Pointer to the length of the buffer without management information. [out]: Pointer to the modified length needed for buffer and management information.

Return code	
Std_ReturnType	void
Functional Description	
Modifies the buffer length to be able to insert management information (No modification done for Sja1105PQRS).	
Particularities and Limitations	
Switch module must have been initialized.	
Call context	
> Task or interrupt	

Table 5-38 EthSwt_30_Sja1105PQRS_EthTxAdaptBufferLength

5.2.2.3 EthSwt_30_Sja1105PQRS_EthTxPrepareFrame

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_EthTxPrepareFrame`** `(uint8 CtrlIdx, uint8 BufIdx, Eth_FrameIdType frameId, uint8 ** DataPtr, uint16* LengthPtr)`	
Parameter	
CtrlIdx [in]	Ethernet Controller index
BufIdx [in]	Ethernet Tx Buffer index
DataPtr [in/out]	[in]: Pointer to the position of the EtherType of a common Ethernet frame [out]: Pointer to the position of the EtherType in the management frame
LengthPtr [in/out]	[in]: Pointer to the length of the buffer without management information. [out]: Pointer to the modified length needed for buffer and management information.
Return code	
Std_ReturnType	void
Functional Description	
Prepares a the Ethernet frame for common Ethernet communication (frame shall be handled by switch according to the common address resolution behavior) and stores the information for processing of EthSwt_30_Sja1105PQRS_EthTxFinishedIndication().	
Particularities and Limitations	
Switch module must have been initialized.	
Call context	
> Task or interrupt	

Table 5-39 EthSwt_30_Sja1105PQRS_EthTxPrepareFrame

5.2.2.4 EthSwt_30_Sja1105PQRS_EthTxProcessFrame

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_EthTxProcessFrame`** `(uint8 CtrlIdx, uint8 BufIdx, Eth_FrameIdType frameId, uint8** DataPtr, uint16* LengthPtr)`	
Parameter	
CtrlIdx [in]	Ethernet Controller index
BufIdx [in]	Ethernet Rx Buffer index
DataPtr [in/out]	[in]: Pointer to the position of the EtherType of a common Ethernet frame [out]: Pointer to the position of the EtherType in the management frame
LengthPtr [in/out]	[in]: Pointer to the length of the frame received [out]: Pointer to the length decreased by the management information length
Return code	
Std_ReturnType	E_OK Frame successfully processed.
Std_ReturnType	E_NOT_OK Frame processing failed.
Functional Description	
Function sets management information for the Ethernet frame.	
Particularities and Limitations	
Switch module must have been initialized.	
Call context	
> Task or interrupt	

Table 5-40 EthSwt_30_Sja1105PQRS_EthTxProcessFrame

5.2.2.5 EthSwt_30_Sja1105PQRS_EthTxFinishedIndication

Prototype	
`void` **`EthSwt_30_Sja1105PQRS_EthTxFinishedIndication`** `(uint8 CtrlIdx, uint8 BufIdx, Eth_FrameIdType FrameId)`	
Parameter	
CtrlIdx [in]	Ethernet Controller index
BufIdx [in]	Ethernet Tx Buffer index
Return code	
void	void
Functional Description	
Indication for a finished transmit process for a specific Ethernet frame.	
Particularities and Limitations	
Switch module must have been initialized.	
Call context	
> Task or interrupt	

Table 5-41 EthSwt_30_Sja1105PQRS_EthTxFinishedIndication

For an interfaces overview please see Figure 2-2.

5.2.2.6 EthSwt_30_Sja1105PQRS_EthRxProcessFrame

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_EthRxProcessFrame`** `(uint8 CtrlIdx, uint8** DataPtr, uint16* LengthPtr, boolean* IsMgmtFrameOnlyPtr)`	
Parameter	
CtrlIdx [in]	Ethernet Controller index
DataPtr [in/out]	[in]: Pointer to the position of the EtherType of a common Ethernet frame [out]: Pointer to the position of the EtherType in the management frame
LengthPtr [in/out]	[in]: Pointer to the length of the frame received [out]: Pointer to the length decreased by the management information length
IsMgmtFrameOnlyPtr [out]	Information about the kind of frame FALSE: Frame is not only for management purpose but also for normal communication. TRUE: Frame is only for management purpose and MUST not be processed in common receive process.
Return code	
Std_ReturnType	E_OK Frame successfully processed.
Std_ReturnType	E_NOT_OK Frame processing failed.
Functional Description	
Function inspects the Ethernet frame passed by the data pointer for management information and stores it for later use in EthSwt_30_Sja1105PQRS_EthRxFinishedIndication().	
Particularities and Limitations	
Switch module must have been initialized.	
Call context	
> Task or interrupt	

Table 5-42 EthSwt_30_Sja1105PQRS_EthRxProcessFrame

5.2.2.7 EthSwt_30_Sja1105PQRS_EthRxFinishedIndication

Prototype	
`void` **`EthSwt_30_Sja1105PQRS_EthRxFinishedIndication`** `(uint8 CtrlIdx, uint8** DataPtr)`	
Parameter	
CtrlIdx [in]	Ethernet Controller index
Return code	
void	void

Functional Description
Indication for a finished receive process for a specific Ethernet frame, which results in providing the management information retrieved during EthSwt_30_Sja1105PQRS_EthProcessRxFrame() to the EthIf if frame received needs special treatment.
Particularities and Limitations
Switch module must have been initialized.
Call context
> Task or interrupt

Table 5-43 EthSwt_30_Sja1105PQRS_EthRxFinishedIndication

5.2.3 Global Time API

5.2.3.1 EthSwt_30_Sja1105PQRS_TimeInit

Prototype	
`void EthSwt_30_Sja1105PQRS_TimeInit (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Initialization of data structures necessary for global time support.	
Particularities and Limitations	
Call context	
FunctionCallcontext	

Table 5-44 EthSwt_30_Sja1105PQRS_TimeInit

5.2.3.2 EthSwt_30_Sja1105PQRS_MainFunctionTime

Prototype	
`void EthSwt_30_Sja1105PQRS_MainFunctionTime (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Mainfunction for handling cyclic events for global time support.	

Particularities and Limitations
Call context
FunctionCallcontext

Table 5-45 EthSwt_30_Sja1105PQRS_MainFunctionTime

5.2.3.3 EthSwt_30_Sja1105PQRS_EnableEgressTimeStamp

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_EnableEgressTimeStamp`** `(uint8 EthCtrlIdx, uint8 BufIdx, const EthSwt_MgmtInfoType *MgmtInfo)`	
Parameter	
EthCtrlIdx [in]	Switch index
BufIdx [in]	Spi Transaction data
MgmtInfo [in]	The data was successfully read
Return code	
Std_ReturnType	E_OK Enabling the egress timestamp was successful
Std_ReturnType	E_NOT_OK Enabling the egress timestamp failed
Functional Description	
Enables time stamping in Ethernet switch for the frame identified by the passed data.	
Particularities and Limitations	
Call context	
> Task or interrupt	

Table 5-46 EthSwt_30_Sja1105PQRS_EnableEgressTimeStamp

5.2.3.4 EthSwt_30_Sja1105PQRS_SetCorrectionTime

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_SetCorrectionTime`** `(uint8 SwitchIdx, const Eth_TimeIntDiffType* OffsetTimePtr, const Eth_RateRatioType* RateRatioPtr)`	
Parameter	
SwitchIndex [in]	Index of the switch instance
OffsetTimePtr [in]	Offset the switch clock shall be corrected with
RateRatioPtr [in]	Rate the switch clock shall be corrected with
Return code	
Std_ReturnType	E_OK – Switch clock was corrected
Std_ReturnType	E_NOT_OK – SPI access failed or requested correction can’t be applied

Functional Description
Performs a correction of the switch clock
Particularities and Limitations
This function performs offset and rate correction of the switch clock.
Call context
> Task

Table 5-47 EthSwt_30_Sja1105PQRS_SetCorrectionTime

5.2.3.5 EthSwt_30_Sja1105PQRS_StartQbvSchedule

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_StartQbvSchedule`** `(uint8 SwitchIdx)`	
Parameter	
SwitchIndex [in]	Index of the switch instance
Return code	
Std_ReturnType	E_OK – QBV scheduling was started
Std_ReturnType	E_NOT_OK – SPI access failed or QBV already running
Functional Description	
This function starts QBV scheduling on the specified switch. There are two cases: > No switch has QBV traffic scheduling activated: The QBV traffic scheduling will be started 1 millisecond later. This offset gives enough time for the necessary SPI operations. > One or more switches have QBV traffic scheduling activated: The QBV traffic scheduling will be started the next time the running QBV scheduling will itself restart its cycle. If two or more switches are used the length of the QBV traffic scheduling cycle should be the same.	
Particularities and Limitations	
Time must be synchronized on all connected switches.	
Call context	
> Task	

Table 5-48 EthSwt_30_Sja1105PQRS_StartQbvSchedule

5.2.3.6 EthSwt_30_Sja1105PQRS_StopQbvSchedule

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_StopQbvSchedule`** `(uint8 SwitchIdx)`	
Parameter	
SwitchIndex [in]	Index of the switch instance

Return code	
Std_ReturnType	E_OK – QBV scheduling was stopped
Std_ReturnType	E_NOT_OK – SPI access failed or QBV scheduling already stopped
Functional Description	
This function stops QBV scheduling on the specified switch.	
Particularities and Limitations	
Call context	
> Task	

Table 5-49 EthSwt_30_Sja1105PQRS_StopQbvSchedule

5.3 Services used by EthSwt

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

Component	API
Det	> Det_ReportError()
Dem	> Dem_ReportErrorStatus()
Spi	> Spi_SetupEB() > Spi_SyncTransmit() > Spi_AsyncTransmit() > Spi_GetSequenceResult()
NvM	> NvM_GetErrorStatus() > NvM_ReadBlock() > NvM_WriteBlock() > NvM_InvalidateNvBlock()
EthTrcv	> EthTrcv_TransceiverInit() > EthTrcv_SetTransceiverMode() > EthTrcv_GetTransceiverMode() > EthTrcv_StartAutoNegotiation() > EthTrcv_GetLinkState() > EthTrcv_GetBaudRate() > EthTrcv_GetDuplexMode()
IpBase	> IpBase_Copy()
EthIf	> EthIf_SwitchMgmtInfoIndication() > EthIf_SwitchIngressTimeStampIndication()

Component	API
	> EthIf_SwitchEgressTimeStampIndication()

Table 5-50 Services used by the EthSwt

5.4 Callback Functions

This chapter describes the callback functions that are implemented by the EthSwt and can be invoked by other modules. The prototypes of the callback functions are provided in the header file `EthSwt_30_Sja1105PQRS.h` by the EthSwt.

5.4.1 EthSwt_30_Sja1105PQRS_NvmSingleBlockCallback

Prototype	
`Std_ReturnType` **`EthSwt_30_Sja1105PQRS_NvmSingleBlockCallback`** `(NvM_ServiceIdType ServiceId, NvM_RequestResultType JobResult)`	
Parameter	
BlockId	ID of the NvM block
JobResult	Result of the NvM job > NVM_REQ_NOT_OK: Job processing finished not successful > NVM_REQ_OK: Job processing finished successful
Return Code	
Std_ReturnType	E_OK Always returns E_OK according to SWS_NvM_00368
Functional Description	
Callback function to indicate the finish of a NvM job.	
Particularities and Limitations	
none	
Pre-Conditions	
Switch instance has to be initialized by EthSwt_SwitchInit().	
Call Context	
Task context	

Table 5-51 EthSwt_30_Sja1105PQRS_NvmSingleBlockCallback

5.5 Configurable Interfaces

5.5.1 Notifications

At its configurable interfaces the EthSwt defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the EthSwt but can be performed at configuration time. The function prototypes that can be used for the configuration must match the appropriate function prototype signatures, which are described in the following sub-chapters.

5.5.1.1 Persistent Configuration Result Notification

This notification informs the user about the finish and the result of the store and reset of the learned switch table entries (please see 4.5.1).

Note
The notification is enabled by setting `EthSwtPersistenConfigurationResult` and the `<User>` prefix is provided by the parameter `EthSwtPersistentConfigurationResultUser`.

Prototype	
`void` **`<User>_PersistentConfigurationResult`** `( NvM_RequestResultType Result )`	
Parameter	
`Result`	Result of the asynchronous NvM job triggered by `EthSwt_30_Sja1105PQRS_StoreConfiguration()` or `EthSwt_30_Sja1105PQRS_ResetConfiguration()`
Return code	
`Void`	none
Functional Description	
The notification is called after the asynchronous NvM job triggered by `EthSwt_30_Sja1105PQRS_StoreConfiguration()` or `EthSwt_30_Sja1105PQRS_ResetConfigration()` has finished. The user function is able to analyze the result of the NvM job.	
Particularities and Limitations	
Call context	
> Context of `EthSwt_30_Sja1105PQRS_NvmSignleBlockCallback()`	

Table 5-52 Persistent Configuration Result Notification

5.5.1.2 Link Down Notification

This notification informs the user about a link state change from `LINK_ACTIVE` to `LINK_DOWN` for a specific switch port.

Note
The notification is enabled by setting `EthSwtLinkDownUser`, which also provides the `<User>` prefix for the notification.

Prototype	
`void <User>_LinkDown ( uint8 *SwitchIdxPtr, uint8 *PortIdxPtr )`	
Parameter	
`SwitchIdxPtr`	Pointer to the switch index the link down is issued for
`PortIdxPtr`	Pointer to the port index the link down is issued for
Return code	
`Void`	none
Functional Description	
The notification is called if a link change for a port from `LINK_ACTIVE` to `LINK_DOWN` is detected.	
Particularities and Limitations	
Call context	
> Context of `EthSwt_30_Sja1105PQRS_GetLinkState()`	

Table 5-53 Link Down Notification

5.5.1.3 Link Up Notification

This notification informs the user about a link state change from `LINK_DOWN` to `LINK_ACTIVE` for a specific switch port.

Note
The notification is enabled by setting `EthSwtLinkUpUser`, which also provides the `<User>` prefix for the notification.

Prototype	
`void <User>_LinkUp ( uint8 *SwitchIdxPtr, uint8 *PortIdxPtr )`	
Parameter	
`SwitchIdxPtr`	Pointer to the switch index the link up is issued for
`PortIdxPtr`	Pointer to the port index the link up is issued for
Return code	
`Void`	none
Functional Description	
The notification is called if a link change for a port from `LINK_DOWN` to `LINK_ACTIVE` is detected.	
Particularities and Limitations	

Call context
> Context of `EthSwt_30_Sja1105PQRS_GetLinkState()`

Table 5-54 Link Up Notification

5.5.1.4 Asynchronous Processing Finished Notification

This notification informs about the finish of a request to the switch driver processed asynchronously. Please see 3.1.2.1 for additional information.

Prototype	
`void` **`<User>_AsyncCallFinishedNotification`** `( uint8 SwitchIdx, uint8 ApiId, boolean Result )`	
Parameter	
`SwitchIdx`	Index of the switch the asynchronous processing was triggered for.
`ApiId`	API ID of the API the asynchronous processing was triggered for.
`Result`	Information about the processing result: ▶ FALSE: Processing wasn't finished successfully ▶ TRUE: Processing was finished successfully
Return code	
`void`	
Functional Description	
This notification is called if the asynchronous processing triggered for an AUTOSAR API listed in Table 3-9 Service IDs has finished. The context of the notification can be retrieved by inspecting the SwitchIdx and ApiId parameters, which is set to the API ID of the triggering function. The processing result passed by Result must be checked to find out if the returned data is valid or not.	
Particularities and Limitations	
Call context	
> interrupt or task context > notification is called if processing of the triggered task is finished	

Table 5-55 Asynchronous Processing Finished Notification

6 Configuration

The EthSwt is configured with the help of the following Vector tools:

- DaVinci Configurator PRO

6.1 Configuration Variants

The EthSwt supports the configuration variants

- `VARIANT-PRE-COMPILE`

The configuration classes of the EthSwt parameters depend on the supported configuration variants. For their definitions please see the EthSwt_30_Sja1105PQRS_bswmd.arxml file.

6.2 Configuration in DaVinci Configurator PRO

The Ethernet Switch driver is configured with the help of the Vector DaVinci Configurator PRO configuration tool.

Note
The configuration instructions only cover configurations for a stack containing Vector BSW components.

This section describes the configuration for proper operation. For configuration information please refer to the description texts provided within the DaVinci Configurator PRO configuration tool shown in the `Properties` view. Additionally the tool provides messages with different severity levels in its `Validation` view if a miss- or non-optimal-configuration is detected. Solving actions or help texts are provided to solve the issues detected.

6.3 Cascading configuration

The cascading configuration defines the connections between the switches (called uplinks) and the connection to the Host-CPU so the driver is able to control the switches and retrieve information according to the setup.

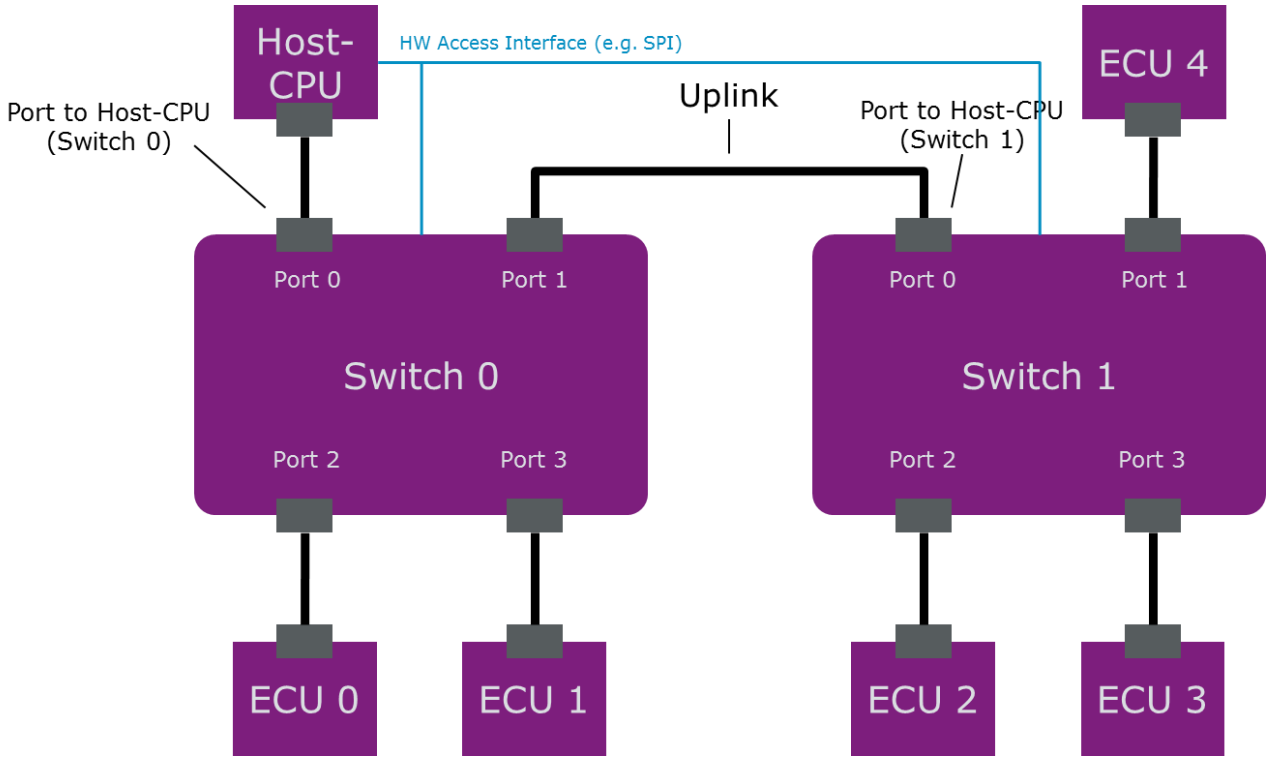

Figure 6-1 Exemplary hardware layout of a switch cascade setup

Figure 6-2 provides an example of a hardware layout used later on in this section to describe the needed configuration modifications to reflect this layout in the configuration.

Each switch that shall be controlled by the driver must be represented as an `EthSwtCascadingMap` within the configuration. To define which switch shall be represented by the map use the reference parameter `EthSwtCascadingSwitchRef` of the respective `EthSwtCascadingMap` and select the `EthSwtConfig` that shall be represented.

Example
For the provided cascading layout there have to exist two `EthSwtConfig`, each representing one of the switches, and two `EthSwtCascadingMap`, each selecting one of the `EthSwtConfig` and therefore providing the possibility to configure the connections to the other switch and the Host-CPU.

For defining the communication path to the Host-CPU the `EthSwtCascadingMap` provides the reference parameter `EthSwtPortToHostRef`. This reference allows selecting the `EthSwtPort` representing the port “closest” to the Host-CPU.

Example

For the provided cascading layout the ports that must be selected would be `EthSwtPort` representing Port 0 of Switch 0 for the `EthSwtCascadingMap` representing Switch 0 and `EthSwtPort` representing Port 0 of Switch 1 for the `EthSwtCascadingMap` representing Switch 1.

The following figure shows the final configuration including the `EthSwtConfigs`, `EthSwtCascadingMaps` and the `EthSwtPorts` selected to define the communication path to the Host-CPU:

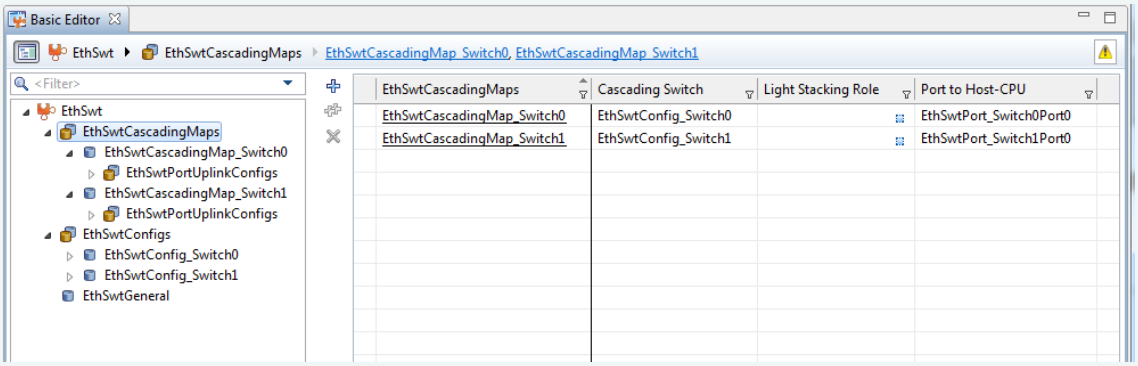

Figure 6-2 Cascading configuration example: EthSwtCascadingMaps

The hierarchy and data path between the switches is defined by the configuration of the uplinks. Each uplink must be configured on both switches involved in the uplink. Therefore, for each uplink two `EthSwtPortUplinkConfig` must exist one for the `EthSwtCascadingMap` representing the first switch involved in the uplink and one for the `EthSwtCascadingMap` representing the second switch involved in the uplink. Within the respective `EthSwtPortUplinkConfig` the local port and the remote port defining the uplink must be selected by `EthSwtLocalUplinkPortRef` and `EthSwtRemoteUplinkPortRef`.

Example

For the provided cascading layout each `EthSwtCascadingMap` has to configure one `EthSwtPortUplinkConfig`. In the `EthSwtPortUplinkConfig` located under the `EthSwtCascadingMap` representing Switch 0 `EthSwtLocalUplinkPortRef` must select the `EthSwtPort` representing Port 1 of Switch 0 and `EthSwtRemoteUplinkPortRef` must select the `EthSwtPort` representing Port 0 of Switch 1.
This applies for the `EthSwtPortUplinkConfig` of `EthSwtCascadingMap` representing Switch 1 too. However, the configuration for `EthSwtLocalUplinkPortRef` and `EthSwtRemoteUplinkPortRef` must be done the other way around. So `EthSwtLocalUplinkPortRef` selects the `EthSwtPort` representing Port 0 of Switch 1 and `EthSwtRemoteUplinkPortRef` selects the `EthSwtPort` representing Port 1 of Switch 0.

The following figures show the final configuration for each `EthSwtPortUplinkConfig` defining the uplink between the switches:

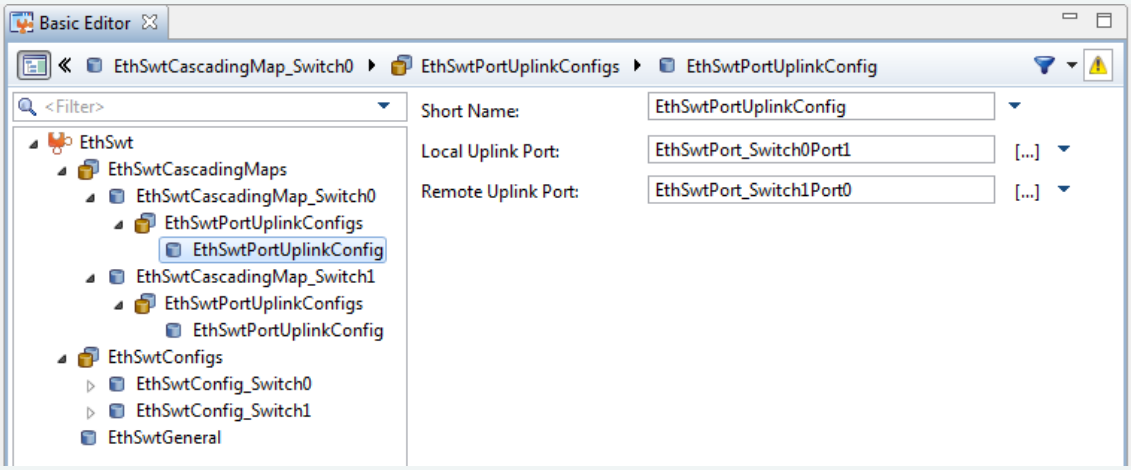

Figure 6-3 Cascading configuration example: EthSwtPortUplinkConfig of Switch 0

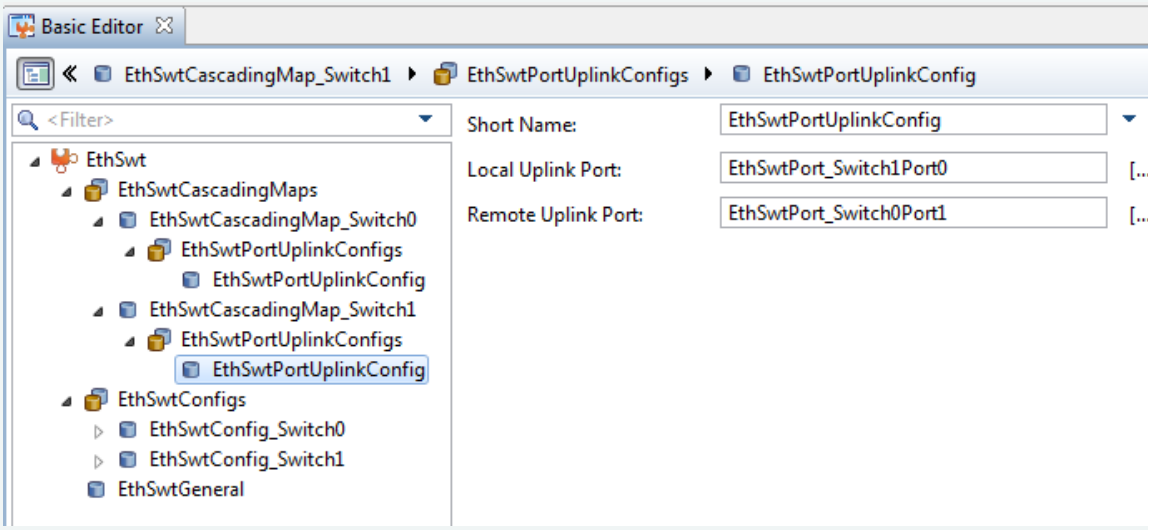

Figure 6-4 Cascading configuration example: EthSwtPortUplinkConfig of Switch 1

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
DaVinci Configurator PRO	Generation tool for the MICROSAR 4 BSW components

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
MAC	Media Access Control
MCU	Microcontroller Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PCP	Priority Code Point (field within VLAN tag)
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
VID	VLAN Identifier (field within the VLAN tag)
VLAN	Virtual Local Area Network

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com