Integration Manual

for S32K1 ETH Driver

Document Number: IM2ETHASR4.4 Rev0000R1.0.1 Rev. 1.0

Chapter 1

Revision History

Revision	Date	Author	Description
1.0	24.02.2022	NXP RTD Team	Prepared for release RTD S32K1 Version 1.0.1

Chapter 2

Introduction

This Integration Manual describes the integration requirements for NXP Semiconductors' AUTOSAR Ethernet Driver for S32K1.

2.1 Supported Derivatives

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

- s32k116_qfn32
- s32k116_lqfp48
- s32k118_lqfp48
- s32k118_lqfp64
- s32k142_lqfp48
- s32k142_lqfp64
- s32k142_lqfp100
- s32k142w_lqfp48
- s32k142w_lqfp64
- s32k144_lqfp48
- s32k144_lqfp64

- s32k144_lqfp100
- s32k144_mapbga100
- s32k144w_lqfp48
- s32k144w_lqfp64
- s32k146_lqfp64
- s32k146_lqfp100
- s32k146_mapbga100
- s32k146_lqfp144
- s32k148_lqfp100
- s32k148_mapbga100
- s32k148_lqfp144
- s32k148_lqfp176

All of the above microcontroller devices are collectively named as S32K1.

2.2 Overview

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

AUTOSAR:

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

2.3 About This Manual

This Technical Reference employs the following typographical conventions:

- **Boldface** style: Used for important terms, notes and warnings.
- *Italic* style: Used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

Note

This is a note.

Warning

This is a warning

2.4 Acronyms and Definitions

Term	Definition
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
DEM	Diagnostic Event Manager
DET	Default Error Tracer
ETH	Ethernet
ETHIF	Ethernet Interface
ETHTRCV	Ethernet Transceiver
ETHSWT	Ethernet Switch
MCU	Microcontroller Unit
MII	Media Independent Interface
N/A	Not Available
RMII	Reduced Media Independent Interface
RAM	Random Access Memory

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

2.5 Reference List

#	Title	Version
1	Specification of Ethernet Driver	AUTOSAR Release 4.4.0
2	S32K1xx Series Reference Manual	S32K1xx Series Reference Manual, Rev. 14, 09/2021
3	S32K148_0N20V Errata	Rev. 22/OCT/2021
4	S32K1xx Data Sheet	S32K1xx Data Sheet, Rev. 14, 08/2021

Chapter 3

Building the driver

- Build Options
- Files required for compilation
- Setting up the plugins

This section describes the source files and various compilers, linker options used for building the driver.
It also explains the EB Tresos Studio plugin setup procedure.

3.1 Build Options

- GCC Compiler/Assembler/Linker Options
- GHS Compiler/Assembler/Linker Options
- IAR Compiler/Assembler/Linker Options

The RTD driver files are compiled using:

- NXP GCC 9.2.0 20190812 (Build 1649 Revision gaf57174)
- IAR ANSI C/C++ Compiler V8.40.3.228/W32 for ARM Functional Safety
- Green Hills Multi 7.1.6d / Compiler 2020.1.4

The compiler, assembler, and linker flags used for building the driver are explained below.
The TS_T40D2M10I1R0 part of the plugin name is composed as follows:

- T = Target_Id (e.g. T40 identifies Cortex-M architecture)
- D = Derivative_Id (e.g. D2 identifies S32K1 platform)
- M = SW_Version_Major and SW_Version_Minor
- I = SW_Version_Patch
- R = Reserved

3.1.1 GCC Compiler/Assembler/Linker Options

3.1.1.1 GCC Compiler Options

Compiler Option	Description
-mcpu=cortex-m4	Targeted ARM processor for which GCC should tune the performance of the code (for S32K14x devices)
-mcpu=cortex-m0plus	Targeted ARM processor for which GCC should tune the performance of the code (for S32K11x devices)
-mthumb	Generates code that executes in Thumb state
-mlittle-endian	Generate code for a processor running in little-endian mode
-mfpu=fpv4-sp-d16	Specifies the floating-point hardware available on the target (for S32K14x devices)
-mfloat-abi=hard	Specifies the floating-point ABI to use. "hard" allows generation of floating-point instructions and uses FPU-specific calling conventions (for S32K14x devices)
-mfpu=auto	Specifies the floating-point hardware available on the target (for S32K11x devices)
-mfloat-abi=soft	Specifies the floating-point ABI to use. Specifying "soft" causes GCC to generate output containing library calls for floating-point operations (for S32K11x devices)
-std=c99	Specifies the ISO C99 base standard
-Os	Optimize for size. Enables all -O2 optimizations except those that often increase code size
-ggdb3	Produce debugging information for use by GDB using the most expressive format available, including GDB extensions if at all possible. Level 3 includes extra information, such as all the macro definitions present in the program
-Wall	Enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros
-Wextra	This enables some extra warning flags that are not enabled by -Wall
-pedantic	Issue all the warnings demanded by strict ISO C. Reject all programs that use forbidden extensions. Follows the version of the ISO C standard specified by the aforementioend -std option
-Wstrict-prototypes	Warn if a function is declared or defined without specifying the argument types
-Wundef	Warn if an undefined identifier is evaluated in an #if directive. Such identifiers are replaced with zero
-Wunused	Warn whenever a function, variable, label, value, macro is unused
-Werror=implicit-function-declaration	Make the specified warning into an error. This option throws an error when a function is used before being declared
-Wsign-compare	Warn when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned.
-Wdouble-promotion	Give a warning when a value of type float is implicitly promoted to double
-fno-short-enums	Specifies that the size of an enumeration type is at least 32 bits regardless of the size of the enumerator values.

Compiler Option	Description
-funsigned-char	Let the type char be unsigned by default, when the declaration does not use either signed or unsigned
-funsigned-bitfields	Let a bit-field be unsigned by default, when the declaration does not use either signed or unsigned
-fomit-frame-pointer	Omit the frame pointer in functions that dont need one. This avoids the instructions to save, set up and restore the frame pointer; on many targets it also makes an extra register available.
-fno-common	Makes the compiler place uninitialized global variables in the BSS section of the object file. This inhibits the merging of tentative definitions by the linker so you get a multiple-definition error if the same variable is accidentally defined in more than one compilation unit
-fstack-usage	Makes the compiler output stack usage information for the program, on a per-function basis
-fdump-ipa-all	Enables all inter-procedural analysis dumps
-c	Stop after assembly and produce an object file for each source file
-DS32K1XX	Predefine S32K1XX as a macro, with definition 1
-DS32K148	Predefine S32K148 as a macro, with definition 1
-DGCC	Predefine GCC as a macro, with definition 1
-DUSE_SW_VECTOR_MODE	Predefine USE_SW_VECTOR_MODE as a macro, with definition 1. By default, the drivers are compiled to handle interrupts in Software Vector Mode
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initalization in source file system.c under the Platform driver (for S32K14x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver (for S32K14x devices)
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPO↩ RT as a macro, with definition 1. Allows drivers to be configured in user mode.

3.1.1.2 GCC Assembler Options

Assembler Option	Description
-Xassembler-with-cpp	Specifies the language for the following input files (rather than letting the compiler choose a default based on the file name suffix)
-mcpu=cortex-m4	Targeted ARM processor for which GCC should tune the performance of the code (for S32K14x devices)
-mcpu=cortex-m0plus	Targeted ARM processor for which GCC should tune the performance of the code (for S32K11x devices)
-mthumb	Generates code that executes in Thumb state
-c	Stop after assembly and produce an object file for each source file

3.1.1.3 GCC Linker Options

Linker Option	Description
-Wl,-Map,filename	Produces a map file
-T linkerfile	Use linkerfile as the linker script. This script replaces the default linker script (rather than adding to it)
–entry=Reset_Handler	Specifies that the program entry point is Reset_Handler
-nostartfiles	Do not use the standard system startup files when linking
-mcpu=cortex-m4	Targeted ARM processor for which GCC should tune the performance of the code (for S32K14x devices)
-mcpu=cortex-m0plus	Targeted ARM processor for which GCC should tune the performance of the code (for S32K11x devices)
-mthumb	Generates code that executes in Thumb state
-mfpu=fpv4-sp-d16	Specifies the floating-point hardware available on the target (for S32K14x devices)
-mfloat-abi=hard	Specifies the floating-point ABI to use. "hard" allows generation of floating-point instructions and uses FPU-specific calling conventions (for S32K14x devices)
-mfpu=auto	Specifies the floating-point hardware available on the target (for S32K11x devices)
-mfloat-abi=soft	Specifies the floating-point ABI to use. Specifying "soft" causes GCC to generate output containing library calls for floating-point operations (for S32K11x devices)
-mlittle-endian	Generate code for a processor running in little-endian mode
-ggdb3	Produce debugging information for use by GDB using the most expressive format available, including GDB extensions if at all possible. Level 3 includes extra information, such as all the macro definitions present in the program
-lc	Link with the C library
-lm	Link with the Math library
-lgcc	Link with the GCC library
-n	Turn off page alignment of sections, and disable linking against shared libraries

3.1.2 GHS Compiler/Assembler/Linker Options

3.1.2.1 GHS Compiler Options

Compiler Option	Description
-cpu=cortexm4	Selects target processor: Arm Cortex M4 (for S32K14x devices)
-cpu=cortexm0plus	Selects target processor: Arm Cortex M0+ (for S32K11x devices)
-thumb	Selects generating code that executes in Thumb state
-fpu=vfpv4_d16	Specifies hardware floating-point using the v4 version of the VFP instruction set, with 16 double-precision floating-point registers (for S32K14x devices)
-fsingle	Use hardware single-precision, software double-precision FP instructions (for S32K14x devices)

Compiler Option	Description
-fsoft	Specifies software floating-point (SFP) mode. This setting causes your target to use integer registers to hold floating-point data and use library subroutine calls to emulate floating-point operations (for S32K11x devices)
-C99	Use (strict ISO) C99 standard (without extensions)
–ghstd=last	Use the most recent version of Green Hills Standard mode (which enables warnings and errors that enforce a stricter coding standard than regular C and C++)
-Osize	Optimize for size
–gnu_asm	Enables GNU extended asm syntax support
-dual_debug	Generate DWARF 2.0 debug information
-G	Generate debug information
-keeptempfiles	Prevents the deletion of temporary files after they are used. If an assembly language file is created by the compiler, this option will place it in the current directory instead of the temporary directory
-Wimplicit-int	Produce warnings if functions are assumed to return int
-Wshadow	Produce warnings if variables are shadowed
-Wtrigraphs	Produce warnings if trigraphs are detected
-Wundef	Produce a warning if undefined identifiers are used in #if preprocessor statements
–unsigned_chars	Let the type char be unsigned, like unsigned char
–unsigned_fields	Bitfelds declared with an integer type are unsigned
–no_commons	Allocates uninitialized global variables to a section and initializes them to zero at program startup
–no_exceptions	Disables C++ support for exception handling
–no_slash_comment	C++ style // comments are not accepted andgenerate errors
–prototype_errors	Controls the treatment of functions referenced or called when no prototype has been provided
–incorrect_pragma_warnings	Controls the treatment of valid #pragma directives that use the wrong syntax
-c	Stop after assembly and produce an object file for each source file
-DS32K1XX	Predefine S32K1XX as a macro, with definition 1
-DS32K148	Predefine S32K148 as a macro, with definition 1
-DGHS	Predefine GHS as a macro, with definition 1
-DUSE_SW_VECTOR_MODE	Predefine USE_SW_VECTOR_MODE as a macro, with definition 1. By default, the drivers are compiled to handle interrupts in Software Vector Mode
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initalization in source file system.c under the Platform driver (for S32K14x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver (for S32K14x devices)

Compiler Option	Description
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPO↩ RT as a macro, with definition 1. Allows drivers to be configured in user mode

3.1.2.2 GHS Assembler Options

Assembler Option	Description
-cpu=cortexm4	Selects target processor: Arm Cortex M4 (for S32K14x devices)
-cpu=cortexm0plus	Selects target processor: Arm Cortex M0+ (for S32K11x devices)
-preprocess_assembly_files	Controls whether assembly files with standard extensions such as .s and .asm are preprocessed
-list	Creates a listing by using the name and directory of the object file with the .lst extension
-c	Stop after assembly and produce an object file for each source file

3.1.2.3 GHS Linker Options

Linker Option	Description
-e Reset_Handler	Make the symbol Reset_Handler be treated as a root symbol and the start label of the application
-T linker_script_file.ld	Use linker_script_file.ld as the linker script. This script replaces the default linker script (rather than adding to it)
-map	Produce a map file
-keepmap	Controls the retention of the map file in the event of a link error
-Mn	Generates a listing of symbols sorted alphabetically/numerically by address
-delete	Instructs the linker to remove functions that are not referenced in the final executable. The linker iterates to find functions that do not have relocations pointing to them and eliminates them
-ignore_debug_references	Ignores relocations from DWARF debug sections when using -delete. DWA↩ RF debug information will contain references to deleted functions that may break some third-party debuggers
-Llibrary_path	Points to library_path (the libraries location) for thumb2 to be used for linking
-larch	Link architecture specific library
-lstartup	Link run-time environment startup routines. The source code for themodules in this library is provided in the src/libstartup directory
-lind_sd	Link language-independent library, containing support routines for features such as software floating point, run-time error checking, C99 complex numbers, and some general purpose routines of the ANSI C library (for S32K14x devices)
-lind_sf	Link language-independent library, containing support routines for features such as software floating point, run-time error checking, C99 complex numbers, and some general purpose routines of the ANSI C library (for S32K11x devices)
-v	Prints verbose information about the activities of the linker, including the libraries it searches to resolve undefined symbols
-keep=C40_Ip_AccessCode	Avoid linker remove function C40_Ip_AccessCode from Fls module because it is not referenced explicitly

Linker Option	Description
-nostartfiles	Controls the start files to be linked into the executable

3.1.3 IAR Compiler/Assembler/Linker Options

3.1.3.1 IAR Compiler Options

Compiler Option	Description
–cpu=Cortex-M4	Targeted ARM processor for which IAR should tune the performance of the code (for S32K14x devices)
–cpu=Cortex-M0+	Targeted ARM processor for which IAR should tune the performance of the code (for S32K11x devices)
–cpu_mode=thumb	Generates code that executes in Thumb state
–endian=little	Generate code for a processor running in little-endian mode
–fpu=FPv4-SP	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). Single-precision variant. (for S32K14x devices)
–fpu=none	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). No FPU. (for S32K11x devices)
-e	Enables all IAR C language extensions
-Ohz	Optimize for size. the compiler will emit AEABI attributes indicating the requested optimization goal. This information can be used by the linker to select smaller or faster variants of DLIB library functions
–debug	Makes the compiler include debugging information in the object modules. Including debug information will make the object files larger
–no_clustering	Disables static clustering optimizations. Static and global variables defined within the same module will not be arranged so that variables that are accessed in the same function are close to each other
–no_mem_idioms	Makes the compiler not optimize certain memory access patterns
–no_explicit_zero_opt	Do not treat explicit initializations to zero of static variables as zero initializations
–require_prototypes	Force the compiler to verify that all functions have proper prototypes. Generates an error otherwise
–no_wrap_diagnostics	Does not wrap long lines in diagnostic messages
–diag_suppress=Pa050	Suppresses diagnostic message Pa050
-DS32K1XX	Predefine S32K1XX as a macro, with definition 1
-DS32K148	Predefine S32K148 as a macro, with definition 1
-DIAR	Predefine IAR as a macro, with definition 1
-DUSE_SW_VECTOR_MODE	Predefine USE_SW_VECTOR_MODE as a macro, with definition 1. By default, the drivers are compiled to handle interrupts in Software Vector Mode.

Compiler Option	Description
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initalization in source file system.c under the Platform driver (for S32K14x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver (for S32K14x devices)
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPO↩ RT as a macro, with definition 1. Allows drivers to be configured in user mode.

3.1.3.2 IAR Assembler Options

Assembler Option	Description
–cpu=Cortex-M4	Targeted ARM processor for which IAR should tune the performance of the code (for S32K14x devices)
–cpu=Cortex-M0+	Targeted ARM processor for which IAR should tune the performance of the code (for S32K11x devices)
–cpu_mode thumb	Selects the thumb mode for the assembler directive CODE
-g	Disables the automatic search for system include files
-r	Generates debug information

3.1.3.3 IAR Linker Options

Linker Option	Description
–map filename	Produces a map file
–config linkerfile	Use linkerfile as the linker script. This script replaces the default linker script (rather than adding to it)
–cpu=Cortex-M4	Targeted ARM processor for which IAR should tune the performance of the code (for S32K14x devices)
–cpu=Cortex-M0+	Targeted ARM processor for which IAR should tune the performance of the code (for S32K11x devices)
–fpu=FPv4-SP	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). Single-precision variant. (for S32K14x devices)
–fpu=none	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). No FPU. (for S32K11x devices)
–entry _start	Treats _start as a root symbol and start label
–enable_stack_usage	Enables stack usage analysis. If a linker map file is produced, a stack usage chapter is included in the map file
–skip_dynamic_initialization	Dynamic initialization (typically initialization of C++ objects with static storage duration) will not be performed automatically during application startup
–no_wrap_diagnostics	Does not wrap long lines in diagnostic messages

3.2 Files required for compilation

This section describes the include files required to compile, assemble and link the AUTOSAR Ethernet Driver for S32K1 microcontrollers.

To avoid integration of incompatible files, all the include files from other modules shall have the same AR_MAJO↩ R_VERSION and AR_MINOR_VERSION, i.e. only files with the same AUTOSAR major and minor versions can be compiled.

3.2.0.0.1 Ethernet Driver Files:

- Eth_TS_T40D2M10I1R0\src\Eth.c
- Eth_TS_T40D2M10I1R0\src\Eth_Irq.c
- Eth_TS_T40D2M10I1R0\src\Eth_Ipw.c
- Eth_TS_T40D2M10I1R0\src\Eth_Ipw_Irq.c
- Eth_TS_T40D2M10I1R0\src\Enet_Ip.c
- Eth_TS_T40D2M10I1R0\src\Enet_Ip_Irq.c
- Eth_TS_T40D2M10I1R0\src\Enet_Ip_Hw_Access.c
- Eth_TS_T40D2M10I1R0\include\Eth.h
- Eth_TS_T40D2M10I1R0\include\Eth_Internal.h
- Eth_TS_T40D2M10I1R0\include\Eth_Ipw.h
- Eth_TS_T40D2M10I1R0\include\Enet_Ip.h
- Eth_TS_T40D2M10I1R0\include\Enet_Ip_Types.h
- Eth_TS_T40D2M10I1R0\include\Enet_Ip_Irq.h
- Eth_TS_T40D2M10I1R0\include\Enet_Ip_Hw_Access.h

3.2.0.0.2 Ethernet Driver Generated Files (must be generated by the user using a configuration tool):

- Enet_Ip_Cfg.c
- Eth_Cfg.h
- Eth_Ipw_Cfg.h
- Enet_Ip_Cfg.h
- Eth_[VariantName]_PBcfg.c
- Eth_Ipw_[VariantName]_PBcfg.c
- Enet_Ip_[VariantName]_PBcfg.c
- Eth_[VariantName]_PBcfg.h

- Eth_Ipw_[VariantName]_PBcfg.h
- Enet_Ip_[VariantName]_PBcfg.h
- Enet_Ip_Features.h
- Enet_Ip_CfgDefines.h

Note

As a deviation from the standard:

- Eth_[VariantName]_PBcfg.c, Eth_Ipw_[VariantName]_PBcfg.c, Enet_Ip_[VariantName]_PBcfg.c - These files will contain the definition for all parameters that are variant aware, independent of the configuration class that will be selected (PC, LT, PB)
- Enet_Ip_Cfg.c - This file will contain the definition for all configuration structures containing only variables that are not variant aware, configured and generated only once. This file alone does not contain the whole structure needed by Eth_Init function to configure the driver. Based on the number of variants configured in the EcuC, there can be more than one configuration structure for one module even for VariantPreCompile.

3.2.0.0.3 Base Files:

- Base_TS_T40D2M10I1R0\include\Eth_GeneralTypes.h
- Base_TS_T40D2M10I1R0\include\Mcal.h
- Base_TS_T40D2M10I1R0\include\Eth_MemMap.h
- Base_TS_T40D2M10I1R0\include\Platform_Types.h
- Base_TS_T40D2M10I1R0\include\Soc_Ips.h
- Base_TS_T40D2M10I1R0\include\Std_Types.h
- Base_TS_T40D2M10I1R0\include\OsIf.h
- Base_TS_T40D2M10I1R0\generate_PC\include\modules.h

3.2.0.0.4 DEM Files:

- Dem_TS_T40D2M10I1R0\include\Dem.h
- Dem_TS_T40D2M10I1R0\include\Dem_Types.h
- Dem_TS_T40D2M10I1R0\generate_PC\include\Dem_IntErrId.h
- Dem_TS_T40D2M10I1R0\src\Dem.c

3.2.0.0.5 DET Files:

- Det_TS_T40D2M10I1R0\include\Det.h
- Det_TS_T40D2M10I1R0\src\Det.c

3.2.0.0.6 RTE Files:

- Rte_TS_T40D2M10I1R0\include\SchM_Eth.h
- Rte_TS_T40D2M10I1R0\src\SchM_Eth.c

3.2.0.0.7 EthIf Files:

- EthIf_TS_T40D2M10I1R0\include\EthIf_Cbk.h
- EthIf_TS_T40D2M10I1R0\src\EthIf_Cbk.c

3.2.0.0.8 EthTrcv Files:

- EthTrcv_TS_T40D2M10I1R0\include\EthTrcv.h
- EthTrcv_TS_T40D2M10I1R0\src\EthTrcv.c

3.2.0.0.9 EthSwt Files:

- EthSwt_TS_T40D2M10I1R0\include\EthSwt.h
- EthSwt_TS_T40D2M10I1R0\src\EthSwt.c

3.3 Setting up the plugins

The Ethernet Driver was designed to be configured by using the EB Tresos Studio (version 27.1.0 b200625-0900 or later)

3.3.0.0.1 Location of various files inside the ETH module folder:

- VSMD (Vendor Specific Module Definition) file in EB Tresos Studio XDM format:
  - Eth_TS_T40D2M10I1R0\config\Eth.xdm
- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:
  - Eth_TS_T40D2M10I1R0\autosar\Eth_<subderivative_name>.epd
- Code Generation Templates for variant aware parameters:
  - Eth_TS_T40D2M10I1R0\generate_PB\src\Eth_PBcfg.c
  - Eth_TS_T40D2M10I1R0\generate_PB\src\Eth_Ipw_PBcfg.c
  - Eth_TS_T40D2M10I1R0\generate_PB\src\Enet_Ip_PBcfg.c
  - Eth_TS_T40D2M10I1R0\generate_PB\include\Eth_PBcfg.h
  - Eth_TS_T40D2M10I1R0\generate_PB\include\Eth_Ipw_PBcfg.h
  - Eth_TS_T40D2M10I1R0\generate_PB\include\Enet_Ip_PBcfg.h
- Code Generation Templates for parameters without variation points:
  - Eth_TS_T40D2M10I1R0\generate_PC\src\Enet_Ip_Cfg.c
  - Eth_TS_T40D2M10I1R0\generate_PC\include\Eth_Cfg.h
  - Eth_TS_T40D2M10I1R0\generate_PC\include\Eth_Ipw_Cfg.h
  - Eth_TS_T40D2M10I1R0\generate_PC\include\Enet_Ip_Cfg.h
  - Eth_TS_T40D2M10I1R0\generate_PC\include\Enet_Ip_CfgDefines.h
  - Eth_TS_T40D2M10I1R0\generate_PC\include\Enet_Ip_Features.h

3.3.0.0.2 Steps to generate the configuration:

1. Copy the following module folders into the Tresos plugins folder:
   - Eth_TS_T40D2M10I1R0
   - Base_TS_T40D2M10I1R0
   - Dem_TS_T40D2M10I1R0
   - Det_TS_T40D2M10I1R0
   - EcuC_TS_T40D2M10I1R0
   - Rte_TS_T40D2M10I1R0
   - Resource_TS_T40D2M10I1R0
   - Mcu_TS_T40D2M10I1R0
   - EthIf_TS_T40D2M10I1R0
   - EthTrcv_TS_T40D2M10I1R0
   - EthSwt_TS_T40D2M10I1R0
2. Set the desired Tresos Output location folder for the generated sources and header files.
3. Use the EB Tresos Studio GUI to modify ECU configuration parameters values.
4. Generate the configuration files

Chapter 4

Function calls to module

4.1 Function Calls during Start-up

The Ethernet Driver shall be initialized through the *Eth_Init* API function during the UP phase of EcuM (which means this module will be initialized by the BswM to allow for a short STARTUP phase). The Mcu and Port drivers shall be initialized beforehand (Port must be initialized before Mcu). The Ethernet Transceiver connected to the MAC has to be configured to the appropriate interface **prior** to the Ethernet Driver initialization. Configure fast slew-rate (and/or drive strength) for all Ethernet pins (otherwise packet loss may occur).

4.2 Function Calls during Shutdown

None.

4.3 Function Calls during Wake-up

None.

Chapter 5

Module requirements

5.1 Exclusive areas to be defined in BSW scheduler

No exclusive areas have been identified.

5.2 Unavailable exclusive areas

None

5.3 Peripheral Hardware Requirements

None.

5.4 ISR to configure within AutosarOS - dependencies

The following ISRs are used by the Ethernet Driver when interrupts are switched on (the driver can also be run in polling mode):

ISR Name	NVIC Interrupt ID
ENET0_TIMESTAMP_IRQHandler	72
ENET0_RING0_TX_IRQHandler	73
ENET0_RING0_RX_IRQHandler	74
ENET0_ERR_IRQHandler	75
ENET0_WAKEUP_IRQHandler	77

5.5 ISR Macro

RTD drivers use the ISR macro to define the functions that will process hardware interrupts. Depending on whether the OS is used or not, this macro can have different definitions.

5.5.1 Without an Operating System

The macro *USING_OS_AUTOSAROS* must not be defined.

5.5.1.1 Using Software Vector Mode

The macro *USE_SW_VECTOR_MODE* must be defined and the ISR macro is defined as:

```
#define ISR(IsrName) void IsrName(void)
```

In this case, the drivers' interrupt handlers are normal C functions and their prologue/epilogue will handle the context save and restore.

5.5.1.2 Using Hardware Vector Mode

The macro *USE_SW_VECTOR_MODE* must not defined and the ISR macro is defined as:

```
#define ISR(IsrName) INTERRUPT_FUNC void IsrName(void)
```

In this case, the drivers' interrupt handlers must also handle the context save and restore.

5.5.2 With an Operating System

Please refer to your OS documentation for description of the ISR macro.

5.6 Other AUTOSAR modules - dependencies

- **Port**: Needed to configure the pins to be used by the Ethernet driver.
- **DET**: Needed for detecting and reporting development errors. The API function used is *Det_ReportError*. The detection can be configured using the *EthDevErrorDetect* configuration parameter
- **DEM**: Needed for detecting and reporting extended production errors.
- **RTE**: Needed for implementing data consistency through exclusive areas.
- **MCU**: Needed to configure the clocks to be used by the Ethernet driver.
- **EcuC**: Needed to retrieve information about post-build variants.
- **Base**: Needed for common files/definitions which are used by all RTD modules.
- **OS**: Needed to define a mapping between EcuC partitions and EcuC core ids when multicore support is enabled.
- **EthIf**: The callbacks *EthIf_RxIndication* and *EthIf_TxConfirmation* are used to notify the upper layer about an ethernet frame transmission and reception. The callback *EthIf_CtrlModeIndication* is used to notify the upper layer about mode transitions in the controllers (i.e. ACTIVE -> DOWN or DOWN -> ACTIVE)
- **EthTrcv**: The callbacks _EthTrcv_ReadMiiIndication and _EthTrcv_WriteMiiIndication (or their vendor API infix variation) are used to notify the ethernet transceiver about a successful MDIO transfer.
- **EthSwt**: The callbacks *EthSwt_TxAdaptBufferLengthFunction* and *EthSwt_TxPrepareFrame↩ Function* are used to request the ethernet switch to do the preparations for a Tx buffer. The callbacks *Eth↩ Swt_TxProcessFrameFunction* and *EthSwt_TxFinishedIndicationFunction* are used to request the ethernet switch to process the ethernet frame before transmission. The callbacks *EthSwt_RxProcess↩ FrameFunction* and *EthSwt_RxFinishedIndicationFunction* are used to notify the ethernet switch of a received ethernet frame.

5.7 Data Cache Restrictions

The descriptors and data buffers are placed in a non-cacheable memory section delimited by *ETH_START_SEC↩ _VAR_NO_INIT_UNSPECIFIED_NO_CACHEABLE* and *ETH_STOP_SEC_VAR_NO_INIT_UNSPECIFIED_NO_CA↩ CHEABLE*. As long as the MPU is properly configured to define the memory region containing the aforementioned section as non-cacheable, there should be no data cache coherency problems.

5.8 User Mode support

- User Mode configuration in the module
- User Mode configuration in AutosarOS

5.8.1 User Mode configuration in the module

The Eth can be run in user mode if the following steps are performed:

- Enable **EthEnableUserModeSupport** from the configuration
- Call the following functions as trusted functions:

Function syntax	Description	Available via
void Enet_Ip_SetUserAccess↩ Allowed(const ENET_Type *Base)	Set the UAA bit in REG_PROT to make the Instance accessible in user mode.	Enet_Ip_Trusted↩ Functions.h
void Enet_Ip_ClrUserAccess↩ Allowed(const ENET_Type *Base)	Clears the UAA bit in REG_PROT to make the Instance unaccessible in user mode.	Enet_Ip_Trusted↩ Functions.h

5.8.2 User Mode configuration in AutosarOS

When User mode is enabled, the driver may has the functions that need to be called as trusted functions in AutosarOS context. Those functions are already defined in driver and declared in the header `<IpName>_Ip↩ _TrustedFunctions.h`. This header also included all headers files that contains all types definition used by parameters or return types of those functions. Refer the chapter User Mode configuration in the module for more detail about those functions and the name of header files they are declared inside. Those functions will be called indirectly with the naming convention below in order to AutosarOS can call them as trusted functions.

```
Call_<Function_Name>_TRUSTED(parameter1,parameter2,...)
```

That is the result of macro expansion `OsIf_Trusted_Call` in driver code:

```
#define OsIf_Trusted_Call[1-6params](name,param1,...,param6) Call_##name##_TRUSTED(param1,...,param6)
```

So, the following steps need to be done in AutosarOS:

- Ensure `MCAL_ENABLE_USER_MODE_SUPPORT` macro is defined in the build system or somewhere global.
- Define and declare all functions that need to call as trusted functions follow the naming convention above in Integration/User code. They need to visible in `Os.h` for the driver to call them. They will do the marshalling of the parameters and call `CallTrustedFunction()` in OS specific manner.
- `CallTrustedFunction()` will switch to privileged mode and call `TRUSTED_<Function_Name>()`.
- `TRUSTED_<Function_Name>()` function is also defined and declared in Integration/User code. It will unmarshalling of the parameters to call `<Function_Name>`() of driver. The `<Function_Name>`() functions are already defined in driver and declared in `<IpName>_Ip_TrustedFunctions.h`. This header should be included in OS for OS call and indexing these functions.

See the sequence chart below for an example calling `Linflexd_Uart_Ip_Init_Privileged()` as a trusted function.

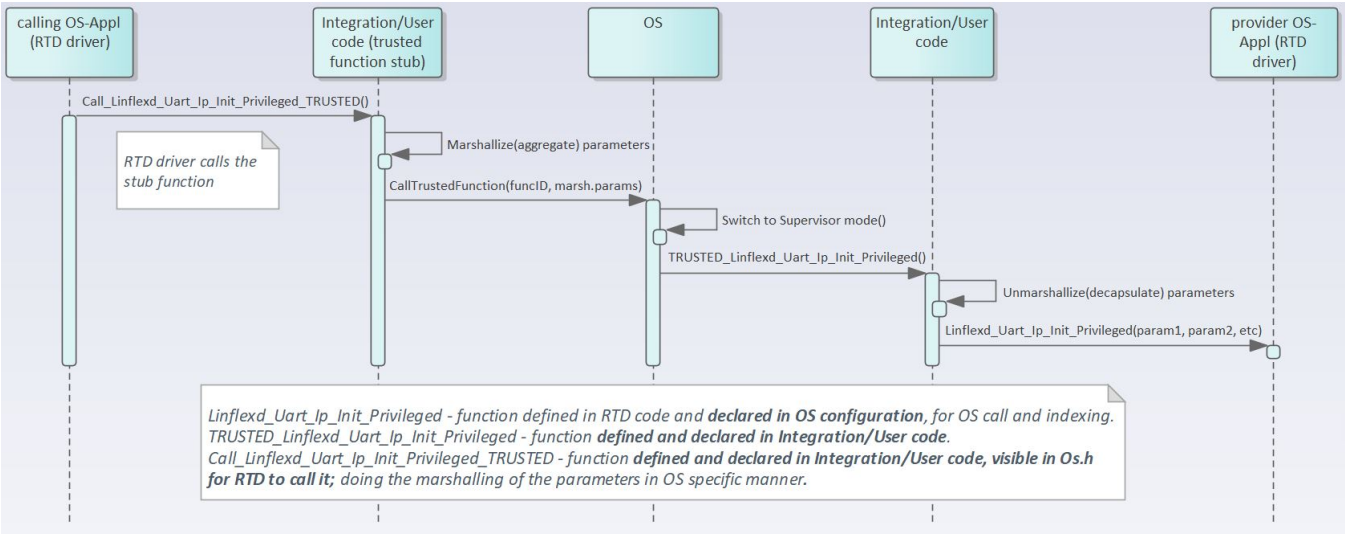

Figure 5.1 Example sequence chart for calling `Linflexd_Uart_Ip_Init_Privileged` as trusted function

5.9 Multicore Support

The Ethernet Driver implements the "Autosar 4.4 MCAL Multicore Distribution" according to type II, in which the mappable element is set to the Hardware Controller. For additional details, please refer to the "AUTOSAR_EXP↩ _BSWDistributionGuide" document.

The Ethernet Driver and its mappable elements can be allocated to zero, one or several EcuC partitions, by means of "EthEcucPartionRef". If the Ethernet Driver is mapped to zero EcuC partitions, the behavior reverts to single-core implementation, similar to previous AUTOSAR versions. Otherwise, if it is mapped to one or more EcuC partitions, then the following multi-core assumptions are enforced:

1. The module assumes there is only a single EcucPartition allocated per core. Internally, the module will use the Core ID returned by GetCoreID API to reference the appropriate global data and configuration elements.
2. The module assumes the EcucCoreIDs are defined in a compact/consecutive order, starting from zero. The rationale is that the number of EcucPartitions is used for dimensioning the internal variables and the Ecuc↩ CoreIDs are used for indexing those variables.
3. The module assumes that the initialization is performed on each core. "Eth_Init" is called individually for each core using the configuration structure intended for that core.
4. The module initialization expects the upper layer to pass the correct initialization pointer, specific to the partition in which the driver is to be used. For example, if EcucPartition_1 is assigned to CoreID 1, then "Eth_Init" shall be called with the "Eth_Config_EcucPartition_1" configuration structure on Core 1.
5. If DET error reporting is enabled, the module will check upon each API call if the requested resource is configured to be available on the current executing core.
6. The module assumes that the RTE module implements exclusive areas to be core-aware only. For single-core scope, the exclusive areas keep the same purpose as on previous AUTOSAR implementations.

7. The module assumes that each interrupt is routed by the system only to the core on which it is supposed to be serviced.

To enable multicore support:

1. Make sure the configuration parameter `_EthMulticoreSupport` is enabled.
2. Add at least one EcuC partition to the list *`EthEcucPartitionRef`*.
3. For each controller, make sure that the configuration parameter *`EthCtrlEcucPartitionRef`* is uniquely referencing one of the EcuC partitions previously defined in *`EthEcucPartitionRef`*.

Chapter 6

Main API Requirements

6.1 Main function calls within BSW scheduler

The Ethernet Driver supports a main function that can be configured to be scheduled by the BSW Scheduler:

void Eth_MainFunction(void);

This function checks for controller errors and lost frames and it is also used for polling state changes. Calls *Eth_CtrlModeIndication* when the controller mode is changed.

6.2 API Requirements

None

6.3 Calls to Notification Functions, Callbacks, Callouts

The Ethernet Driver provides no configurable notification functions, callbacks, or callouts.

Chapter 7

Memory allocation

- Sections to be defined in Eth_MemMap.h
- Linker command file

7.1 Sections to be defined in Eth_MemMap.h

Section Name	Section Type	Description
ETH_START_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	Start of Memory Section for Config Data
ETH_STOP_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	End of Memory Section for Config Data
ETH_START_SEC_CODE	Code	Start of Memory Section for Code
ETH_STOP_SEC_CODE	Code	End of Memory Section for Code
ETH_START_SEC_VAR_CLEARED_UNSPECIFIED	Variables	Used for variables, structures, arrays when the SIZE (alignment) does not fit the criteria of 8, 16 or 32 bit. These variables are never cleared and never initialized by start-up code
ETH_STOP_SEC_VAR_CLEARED_UNSPECIFIED	Variables	End of the above section
ETH_START_SEC_VAR_CLEARED_UNSPECIFIED_NO_CACHEABLE	Variables	Used for variables, structures, arrays when the SIZE (alignment) does not fit the criteria of 8, 16 or 32 bit. Normally, this section is used to store descriptors and data buffers. This section must also be cache inhibited
ETH_STOP_SEC_VAR_CLEARED_UNSPECIFIED_NO_CACHEABLE	Variables	End of the above section
ETH_START_SEC_VAR_CLEARED_32	Variables	Used for variables which have to be aligned to 32 bits. For instance used for 32-bits variables or used for composite data types (arrays, structs) containing elements of maximum 32 bits. These variables are cleared to zero by start-up code.
ETH_STOP_SEC_VAR_CLEARED_32	Variables	End of the above section
ETH_START_SEC_CONST_32	Constants	Used for constants that have to be aligned to 32 bits
ETH_STOP_SEC_CONST_32	Constants	End of the above section

7.2 Linker command file

Memory shall be allocated for every section defined in the driver's "<Module>"_MemMap.h.

Chapter 8

Integration Steps

This section gives a brief overview of the steps needed for integrating this module:

1. Generate the required module configuration(s). For more details refer to section Files Required for Compilation
2. Allocate the proper memory sections in the driver's memory map header file ("<Module>"_MemMap.h) and linker command file. For more details refer to section Sections to be defined in <Module>_MemMap.h
3. Compile & build the module with all the dependent modules. For more details refer to section Building the Driver

Chapter 9

External assumptions for driver

The section presents requirements that must be complied with when integrating the ETH driver into the application.

External Assumption Req ID	External Assumption Text
SWS_Eth_00026	Module - Header File - Imported Type - ComStack_Types - ComStack↩ Types.h - BufReq_ReturnType - Dem - Rte_Dem_Type.h - Dem_Event↩ IdType - Rte_Dem_Type.h - Dem_EventStatusType - Eth_GeneralTypes - Eth_GeneralTypes.h - Eth_BufIdxType - Eth_GeneralTypes.h - Eth↩ _CounterType - Eth_GeneralTypes.h - Eth_DataType - Eth_General↩ Types.h - Eth_FilterActionType - Eth_GeneralTypes.h - Eth_Frame↩ Type - Eth_GeneralTypes.h - Eth_ModeType - Eth_GeneralTypes.h - Eth_RxStatsType - Eth_GeneralTypes.h - Eth_RxStatusType - Eth_↩ GeneralTypes.h - Eth_TimeStampQualType - Eth_GeneralTypes.h - Eth↩ _TimeStampType - Eth_GeneralTypes.h - Eth_TxErrorCounterValues↩ Type - Eth_GeneralTypes.h - Eth_TxStatsType - Std_Types - Standard↩ Types.h - Std_ReturnType - StandardTypes.h - Std_VersionInfoType - Note: Under control of BASE, DEM, RTE module
SWS_Eth_00158	Name: - Eth_ModeType - Type: - Enumeration - Range: - ETH_MOD↩ E_DOWN - 0x00 - Controller disabled - ETH_MODE_ACTIVE - 0x01 - Controller enabled - Description: - This type defines the controller modes - Available via: - Eth_GeneralTypes.h - Note: Under control of BASE module
SWS_Eth_00159	Name: - Eth_StateType - Type: - Enumeration - Range: - ETH_STA↩ TE_UNINIT - 0x00 - Driver is not yet configured - ETH_STATE_INIT - 0x01 - Driver is configured - Description: - Status supervision used for Development Error Detection. The state shall be available for debugging. - Available via: - Eth_GeneralTypes.h - Note: Under control of BASE module
SWS_Eth_00160	Name: - Eth_FrameType - Type: - uint16 - Description: - This type defines the Ethernet frame type used in the Ethernet frame header - Available via: - Eth_GeneralTypes.h - Note: Under control of BASE module
SWS_Eth_00161	Name: - Eth_DataType - Type: - uint8, uint16, uint32 - Description: - This type defines the Ethernet data type used for data transmission. Its definition depends on the used CPU. - Available via: - Eth_GeneralTypes.h - Note: Under control of BASE module
SWS_Eth_00175	Name: - Eth_BufIdxType - Type: - uint32 - Description: - Ethernet buffer identifier type. - Available via: - Eth_GeneralTypes.h - Note: Under control of BASE module

External Assumption Req ID	External Assumption Text
SWS_Eth_00162	Name: - Eth_RxStatusType - Type: - Enumeration - Range: - ETH_↩ RECEIVED - 0x00 - Ethernet frame has been received, no further frames available - ETH_NOT_RECEIVED - 0x01 - Ethernet frame has not been received, no further frames available - ETH_RECEIVED_MORE_DAT↩ A_AVAILABLE - 0x02 - Ethernet frame has been received, more frames are available - Description: - Used as out parameter in Eth_Receive() indicates whether a frame has been received and if so, whether more frames are available or frames got lost. - Available via: - Eth_GeneralTypes.h - Note: Under control of BASE module
SWS_Eth_00163	Name: - Eth_FilterActionType - Type: - Enumeration - Range: - ETH↩ _ADD_TO_FILTER - 0x00 - add the MAC address to the filter, meaning allow reception - ETH_REMOVE_FROM_FILTER - 0x01 - remove the MAC address from the filter, meaning reception is blocked in the lower layer - Description: - The Enumeration Type Eth_FilterActionType describes the action to be taklen for the MAC address given in ∗PhysAddrPtr. - Available via: - Eth_GeneralTypes.h - Note: Under control of BASE module
SWS_Eth_00177	Name: - Eth_TimeStampQualType - Type: - – - Range: - ETH_VALID - 0 - – - ETH_INVALID - 1 - – - ETH_UNCERTAIN - 2 - – - Description: - Depending on the HW, quality information regarding the evaluated time stamp might be supported. If not supported, the value shall be always Valid. For Uncertain and Invalid values, the upper layer shall discard the time stamp. - Available via: - Eth_GeneralTypes.h - Note: Under control of BASE module
SWS_Eth_00178	Name: - Eth_TimeStampType - Type: - Structure - Element: - uint32 - nanoseconds - Nanoseconds part of the time - uint32 - seconds - 32 bit LSB of the 48 bits Seconds part of the time - uint16 - secondsHi - 16 bit MSB of the 48 bits Seconds part of the time - Description: - Variables of this type are used for expressing time stamps including relative time and absolute calendar time. The absolute time starts at 1970-01-01. 0 to 281474976710655s == 3257812230d [0xFFFF FFFF FFFF] 0 to 999999999ns [0x3B9A C9FF] invalid value in nanoseconds: [0x3B9A CA00] to [0x3FFF FFFF] Bit 30 and 31 reserved, default: 0 - Available via: - Eth_GeneralTypes.h - Note: Under control of BASE module
SWS_Eth_00179	Name: - Eth_TimeIntDiffType - Type: - Structure - Element: - Eth_↩ TimeStampType - diff - time difference - boolean - sign - Positive (True) / negative (False) time - Description: - Variables of this type are used to express time differences. - Available via: - Eth_GeneralTypes.h - Note: Under control of BASE module
SWS_Eth_00180	Name: - Eth_RateRatioType - Type: - Structure - Element: - Eth↩ _TimeIntDiffType - IngressTimeStampDelta - IngressTimeStampSync2 - IngressTimeStampSync1 - Eth_TimeIntDiffType - OriginTimeStamp↩ Delta - OriginTimeStampSync2[FUP2] - OriginTimeStampSync1[FUP1] - Description: - Variables of this type are used to express frequency ratios. - Available via: - Eth_GeneralTypes.h - Note: Under control of BASE module

External assumptions for driver

External Assumption Req ID	External Assumption Text
SWS_Eth_00120	API function - Header File - Description - Det_ReportError - Det.h - Service to report development errors. - EthSwt_EthRxFinishedIndication - EthSwt_Eth.h - Indication for a finished receive process for a specific Ethernet frame, which results in providing the management information retrieved during EthSwt_EthRxProcessFrame(). - EthSwt_EthRxProcessFrame - EthSwt_Eth.h - Function inspects the Ethernet frame passed by the data pointer for management information and stores it for later use in EthSwt_EthRxFinishedIndication(). - EthSwt_EthTxAdaptBufferLength - EthSwt_Eth.h - Modifies the buffer length to be able to insert management information. - EthSwt_EthTxFinishedIndication - EthSwt_Eth.h - Indication for a finished transmit process for a specific Ethernet frame. - EthSwt_EthTxPrepareFrame - EthSwt_Eth.h - Prepares 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_EthTxFinishedIndication(). - EthSwt_EthTxProcessFrame - EthSwt_Eth.h - Function inserts management information into the Ethernet frame. - Note: Under control EthSwt module
SWS_Eth_91001	Name: - Eth_MacVlanType - Type: - Structure - Element: - uint8[6] - MacAddr - Specifies the MAC address [0..255,0..255,0..255,0..255,0..255,0..255] - uint16 - VlanId - Specifies the VLAN address 0..65535 - uint32 - SwitchPort - Specifies the ports of the switch as bit mask (0x00000001->Port0, 0x80000001->Port31+Port0) - Description: - This type is used to read out addresses from the address resolution logic (ARL) table of the switch. typedef struct { uint8 MacAddr[6U]; uint16 VlanId; uint32 SwitchPort; } Eth_MacVlanType; In case of Macaddr contains a Multicast Address MacVlanType.SwitchPort shall be handled as Bitmask, each bit represents a Switch Port, Bit 0 represents EthSwichtPortIdx = 0 , Bit 1 represents EthSwichtPortIdx = 1 and so on. In case of Macaddr contains not a Multicast Address MacVlanType.SwitchPort shall be handled as a value representing the EthSwitchPortIdx. - Available via: - Eth_GeneralTypes.h - Note: Under BASE module, some HW don't support VLAN

External Assumption Req ID	External Assumption Text
SWS_Eth_91004	Name: - Eth_TxErrorCounterValuesType - Type: - Structure - Element: - uint32 - TxDroppedNoErrorPkts - 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. Also described in IETF RFC1213 MIB ifOutDiscards - uint32 - TxDroppedErrorPkts - transmitted because of errors. Also described in IETF RFC1213 MIB ifOutErrors - uint32 - TxDeferredTrans - A count of frames for which the first transmission attempt on a particular interface is delayed because the medium is busy. The count represented by an instance of this object does not include frames involved in collisions. Also described in IETF RFC1643 MIB dot3Stats↩ DeferredTransmissions - uint32 - TxSingleCollision - A count of successfully transmitted frames on a particular interface for which transmission is inhibited by exactly one collision. A frame that is counted by an instance of this object is also counted by the corresponding instance of either the Tx↩ UniCastPkts and TxNUcastPkts and is not counted by the corresponding instance of the TxMultipleCollision object. Also described in IETF RF↩ C1643 MIB dot3StatsSingleCollisionFrames - uint32 - TxMultipleCollision - A count of successfully transmitted frames on a particular interface for which transmission is inhibited by more than one collision. A frame that is counted by an instance of this object is also counted by the corresponding instance of either the TxUniCastPkts and TxNUcastPkts and is not counted by the corresponding instance of the TxSingleCollision object. Also described in IETF RFC1643 MIB dot3StatsMultipleCollisionFrames. - uint32 - TxLateCollision - The number of times that a collision is detected on a particular interface later than 512 bit-times into the transmission of a packet. Five hundred and twelve bit-times corresponds to 51.2 microseconds on a 10 Mbit/s system. A (late) collision included in a count represented by an instance of this object is also considered as a (generic) collision for purposes of other collision-related statistics. Also described in IETF RFC1643 MIB dot3StatsLateCollisions - uint32 - TxExcessiveCollison - A count of frames for which transmission on a particular interface fails due to excessive collisions. Also described in IETF RFC1643 MIB dot3StatsExcessiveCollisions - Description: - Statistic counters for diagnostics. - Available via: - Eth_↩ GeneralTypes.h - Note: Under BASE module
SWS_Eth_91003	Name: - Eth_TxStatsType - Type: - Structure - Element: - uint32 - Tx↩ NumberOfOctets - The total number of octets transmitted out of the interface, including framing characters. Also described in IETF RFC1213 MIB ifOutOctets. - uint32 - TxNUcastPkts - The total number of packets that higher-level protocols requested be transmitted to a non-unicast (i.e., a subnetwork-broadcast or subnetwork-multicast) address, including those that were discarded or not sent. Also described in IETF RFC1213 MIB ifOutNUcastPkts - uint32 - TxUniCastPkts - The total number of packets that higher-level protocols requested be transmitted to a subnetwork-unicast address, including those that were discarded or not sent. Also described in IETF RFC1213 MIB ifOutUcastPkts. - Description: - Statistic counter for diagnostics. - Available via: - Eth_GeneralTypes.h - Note: Under BASE module

External assumptions for driver

External Assumption Req ID	External Assumption Text
SWS_Eth_91002	Name: - Eth_RxStatsType - Type: - Structure - Element: - uint32 - RxStatsDropEvents - The total number of events in which packets were dropped by the probe due to lack of resources. Also described in IETF RFC 2819 MIB etherStatsDropEvents. - uint32 - RxStatsOctets - The total number of octets of data (including those in bad packets) received on the network (excluding framing bits but including FCS octets). Also described in IETF RFC 2819 MIB etherStatsOctets. - uint32 - RxStatsPkts - The total number of packets (including bad packets, broadcast packets, and multicast packets) received. Also described in IETF RFC 2819 MIB etherStatsPkts - uint32 - RxStatsBroadcastPkts - The total number of good packets received that were directed to the broadcast address. Also described in IETF RFC 2819 MIB etherStatsBroadcastPkts - uint32 - RxStatsMulticastPkts - The total number of good packets received that were directed to a multicast address. Also described in IETF RFC 2819 MIB etherStatsMulticastPkts. - uint32 - RxStatsCrcAlignErrors - The total number of packets received that had a length of bertween 64 and 1518 octets that had either a bad Frame Check Sequence (FCS) with an integral number of octets (FCS Error) or a bad FCS with a non-integral number of octets (Alignment Error). Also described in IETF RFC 2819 MIB etherStatsCRCAlignErrors - uint32 - RxStatsUndersizePkts - The total number of packets received that were less than 64 octets long (excluding framing bits, but including FCS octets) and were otherwise well formed. Also described in IETF RFC 2819 MIB etherStatsUndersizePkts. - uint32 - RxStatsOversizePkts - The total number of packets received that were longer than 1518 octets (excluding framing bits, but including FCS octets) and were otherwise well formed. Also described in IETF RFC 2819 MIB etherStatsOversizePkts - uint32 - RxStatsFragments - The total number of packets received that were less than 64 octets in length (excluding framing bits but including FCS octets) and had either a bad Frame Check Sequence (FCS) with an integral number of octets (FCS Error) or a bad FCS with a non-integral number of octets (Alignment Error). Also described in IETF RFC 2819 MIB etherStatsFragments. - uint32 - RxStatsJabbers - The total number of packets received that were longer than 1518 octets, and had either a bad Frame Check Sequence (FCS) with an integral number of octets (FCS Error) or a bad FCS with a non-integral number of octets (Alignment Error). Also described in IETF RFC 2819 MIB etherStatsJabbers. - uint32 - RxStatsCollisions - The best estimate of the total number of collisions on this Ethernet segment. Also described in IETF RFC 2819 MIB etherStatsCollisions - uint32 - RxStatsPkts64Octets - The total number of packets (including bad packets) received that were 64 octets in length. Also described in IETF RFC 2819 MIB etherStatsPkts64Octets - uint32 - RxStatsPkts65to127Octets - The total number of packets (including bad packets) received that were between 65 and 127 octets in length. Also described in IETF RFC 2819 MIB etherStatsPkts65to127Octets - uint32 - RxStatsPkts128to255Octets - The total number of packets (including bad packets) received that were between 128 and 255 octets in length. Also described in IETF RFC 2819 MIB etherStatsPkts128to255Octets - uint32 - RxStatsPkts256to511Octets - The total number of packets (including bad packets) received that were between 256 and 511 octets in length. Also described in IETF RFC 2819 MIB etherStatsPkts256to511Octets - uint32 - RxStatsPkts512to1023Octets - The total number of packets (including bad packets) received that were between 512 and 1023 octets in length. Also described in IETF RFC 2819 MIB etherStatsPkts512to1023Octets - uint32 - RxStatsPkts1024to1518Octets - The total number of packets (including bad packets) received that were between 1024 and 1518 octets in length. Also described in IETF RFC 2819 MIB etherStatsPkts1024to1518Octets - uint32 - RxUnicastFrames - The number of subnetwork-unicast packets delivered to a higher-layer protocol. Also described in IETF RFC 1213 MIB ifInUcastPkts - Description: - Statistic counter for diagnostics - Available

External Assumption Req ID	External Assumption Text
SWS_Eth_00260	DRAFT: The ECUC partitions referenced by EthCtrlEcucPartitionRef shall be a subset of the ECUC partitions referenced by EthEcucPartitionRef. Note: S32K1XX/SJA11XX don't support multiple core
SWS_Eth_00261	DRAFT: EthCtrlConfig, EthTrcvConfig and EthSwtConfig (if existent in configuration) of one communication channel shall all reference the same ECUC partition Note: S32K1XX/SJA11XX don't support multiple core
SWS_Eth_91007	Name: - Eth_CounterType - Type: - Structure - Element: - uint32 - DropPktBufOverrun - dropped packets due to buffer overrun - uint32 - DropPktCrc - dropped packets due to CRC errors - uint32 - UndersizePkt - number of undersize packets which were less than 64 octets long (excluding framing bits, but including FCS octets) and were otherwise well formed. (see IETF RFC 1757) - uint32 - OversizePkt - 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) - uint32 - AlgnmtErr - 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. - uint32 - SqeTestErr - SQE test error according to IETF RFC1643 dot3StatsSQETestErrors - uint32 - DiscInbdPkt - 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) - uint32 - ErrInbdPkt - total number of erroneous inbound packets - uint32 - DiscOtbdPkt - 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) - uint32 - ErrOtbdPkt - total number of erroneous outbound packets - uint32 - SnglCollPkt - 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) - uint32 - MultCollPkt - 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) - uint32 - DfrdPkt - 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) - uint32 - LatCollPkt - 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) - uint32 - HwDepCtr0 - hardware dependent counter value - uint32 - HwDepCtr1 - hardware dependent counter value - uint32 - HwDepCtr2 - hardware dependent counter value - uint32 - HwDepCtr3 - hardware dependent counter value - Description: - Statistic counter for diagnostics. - Available via: - Eth_GeneralTypes.h - Note: Under Base module
EA_RTD_00082	When caches are enabled and data buffers are allocated in cacheable memory regions the buffers involved in DMA transfer shall be aligned with both start and end to cache line size. Note: **Rationale**: This ensures that no other buffers/variables compete for the same cache lines.
EA_RTD_00106	Standalone IP configuration and HL configuration of the same driver shall be done in the same project

External assumptions for driver

External Assumption Req ID	External Assumption Text
EA_RTD_00107	The integrator shall use the IP interface only for hardware resources that were configured for standalone IP usage. Note: The integrator shall not directly use the IP interface for hardware resources that were allocated to be used in HL context.
EA_RTD_00108	The integrator shall use the IP interface to a build a CDD, therefore the BSWMD will not contain reference to the IP interface

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

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

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