Integration Manual

for S32K1 EEP Driver

Document Number: IM2EEPASR4.4 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 Eep Driver for S32K1 microcontrollers.

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
DET	Default Error Tracer
DEM	Diagnostic Event Manager
ECC	Error Correcting Code
VLE	Variable Length Encoding
N/A	Not Available
MCU	Microcontroller Unit
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
EA	EEPROM Abstraction
EEP	EEPROM driver
XML	Extensible Markup Language

2.5 Reference List

#	Title	Version
1	Specification of Fls Driver	AUTOSAR Release 4.4.0
2	Reference Manual	S32K1xx Series Reference Manual, Rev. 14, 09/2021
3	Datasheet	S32K1xx Data Sheet, Rev. 14, 08/2021
4	Errata	S32K116_0N96V Rev. 22/OCT/2021
		S32K118_0N97V Rev. 22/OCT/2021
		S32K142_0N33V Rev. 22/OCT/2021
		S32K144_0N57U Rev. 22/OCT/2021
		S32K144W_0P64A Rev. 22/OCT/2021
		S32K146_0N73V Rev. 22/OCT/2021
		S32K148_0N20V Rev. 22/OCT/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 (if assembler code) and link the Eep driver for S32 microcontrollers. To avoid integration of incompatible files, all the include files from other modules shall have the same AR_MAJOR_VERSION and AR_MINOR_VERSION, i.e. only files with the same AUTOSAR major and minor versions can be compiled.

#Eep files:

- Include file:
  - Eep_TS_T40D2M10I1R0\include\Eep.h
  - Eep_TS_T40D2M10I1R0\include\Eep_InternalTypes.h
  - Eep_TS_T40D2M10I1R0\include\Eep_IPW.h
  - Eep_TS_T40D2M10I1R0\include\Eep_Types.h
  - Eep_TS_T40D2M10I1R0\include\Ftfc_Ip.h
  - Eep_TS_T40D2M10I1R0\include\Ftfc_Ip_Types.h
- Source file:
  - Eep_TS_T40D2M10I1R0\src\Eep.c
  - Eep_TS_T40D2M10I1R0\src\Eep_IPW.c
  - Eep_TS_T40D2M10I1R0\src\Ftfc_Ip.c

  Note
  - Eep_PBcfg.c - this file should be generated by the user using a configuration/generation tool
  - Eep_PBcfg.h - this file should be generated by the user using a configuration/generation tool
  - Eep_Cfg.h - this file should be generated by the user using a configuration/generation tool
  - Eep_Cfg.c - this file should be generated by the user using a configuration/generation tool
  - Ftfc_Eep_Ip_PBcfg.c - this file should be generated by the user using a configuration/generation tool
  - Ftfc_Eep_Ip_Cfg.h - this file should be generated by the user using a configuration/generation tool

3.2.1 Other includes files:

3.2.1.1 Files from MemIf folder

- MemIf_TS_T40D2M10I1R0\include\MemIf_Types.h

3.2.1.2 Files from Base common folder

- Base_TS_T40D2M10I1R0\include\Compiler.h
- Base_TS_T40D2M10I1R0\include\Compiler_Cfg.h
- Base_TS_T40D2M10I1R0\include\ComStack_Types.h
- Base_TS_T40D2M10I1R0\include\Fls_MemMap.h
- Base_TS_T40D2M10I1R0\include\Mcal.h
- Base_TS_T40D2M10I1R0\include\Platform_Types.h
- Base_TS_T40D2M10I1R0\include\Std_Types.h
- Base_TS_T40D2M10I1R0\include\Reg_eSys.h
- Base_TS_T40D2M10I1R0\include\Soc_Ips.h
- Base_TS_T40D2M10I1R0\include\Reg_Macros.h
- Base_TS_T40D2M10I1R0\include\SilRegMacros.h
- Base_TS_T40D2M10I1R0\header\S32K148.h

3.2.1.3 Files from Det folder:

- Det_TS_T40D2M10I1R0\include\Det.h

3.2.1.4 Files from Dem folder:

- Dem_TS_T40D2M10I1R0\include\Dem.h

3.2.1.5 Files from Platform folder:

- Platform_TS_T40D2M10I1R0\include\Platform.h

3.2.1.6 Files from Rte folder:

- Rte_TS_T40D2M10I1R0\include\SchM_Eep.h

3.2.1.7 Files from Os folder:

- Os_TS_T40D2M10I1R0\include\Os.h

3.3 Setting up the plugins

The Eep Driver was designed to be configured by using the EB Tresos Studio (version 27.1.0 or later)

3.3.0.0.1 Steps to generate the configuration:

1. Copy the module folders:
   - Base_TS_T40D2M10I1R0
   - Det_TS_T40D2M10I1R0
   - Dem_TS_T40D2M10I1R0
   - EcuC_TS_T40D2M10I1R0
   - Eep_TS_T40D2M10I1R0
   - MemIf_TS_T40D2M10I1R0
   - Os_TS_T40D2M10I1R0
   - Platform_TS_T40D2M10I1R0
   - Resource_TS_T40D2M10I1R0
   - Rte_TS_T40D2M10I1R0 into the Tresos plugins folder.
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.

3.3.0.0.2 Location of various files inside the EEP module folder:

- VSMD (Vendor Specific Module Definition) file in EB Tresos Studio XDM format:
  - Eep_TS_T40D2M10I1R0\config\Eep.xdm
- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:
  - Eep_TS_T40D2M10I1R0\autosar\Eep_<subderivative_name>.epd
- Code Generation Templates for variant aware parameters:
  - Eep_TS_T40D2M10I1R0\generate_PB\include\Eep_PBcfg.h
  - Eep_TS_T40D2M10I1R0\generate_PB\src\Eep_PBcfg.c
  - Eep_TS_T40D2M10I1R0\generate_PB\src\Ftfc_Eep_Ip_PBcfg.c
- Code Generation Templates for parameters without variation points:
  - Eep_TS_T40D2M10I1R0\generate_PC\include\Eep_Cfg.h
  - Eep_TS_T40D2M10I1R0\generate_PC\include\Ftfc_Eep_Ip_Cfg.h
  - Eep_TS_T40D2M10I1R0\generate_PC\src\Eep_Cfg.c

Chapter 4

Function calls to module

4.1 Function Calls during Start-up

EEP shall be initialized during STARTUP phase of EcuM initialization. The API to be called for this is Eep_Init(). The MCU module should be initialized before the EEP is initialized.

4.2 Function Calls during Shutdown

None.

4.3 Function Calls during Wake-up

None.

Chapter 5

Module requirements

- Exclusive areas to be defined in BSW scheduler
- Exclusive areas not available on this platform
- Peripheral Hardware Requirements
- ISR to configure within AutosarOS - dependencies
- ISR Macro
- Other AUTOSAR modules - dependencies
- Data Cache Restrictions
- User Mode support
- Multicore support

5.1 Exclusive areas to be defined in BSW scheduler

- In the current implementation, EEP is using the services of Run-Time Environment (RTE) for entering and exiting the critical regions.
- RTE implementation is done by the integrators of the RTD using OS or non-OS services.
- For testing the EEP, stubs are used for RTE.
- EEP driver has nine exclusive areas (EA)
  - EEP_EXCLUSIVE_AREA_01
  - EEP_EXCLUSIVE_AREA_02
  - EEP_EXCLUSIVE_AREA_03
  - EEP_EXCLUSIVE_AREA_04
  - EEP_EXCLUSIVE_AREA_05
  - EEP_EXCLUSIVE_AREA_07
  - EEP_EXCLUSIVE_AREA_08
  - EEP_EXCLUSIVE_AREA_10

  - EEP_EXCLUSIVE_AREA_11

- The purpose of these exclusive areas is to make the functions Eep_Erase, Eep_Write, Eep_Read Eep_↩ Compare, Eep_QuickWrite and Eep_MainFunction thread safe and thus protect EEP internal job variables.

EEP_EXCLUSIVE_AREA_01 is used in function Eep_Erase. It protects the variables:

- Eep_u32EepromAddrIt
- Eep_u32RemainingLength
- Eep_eJob
- Eep_eJobResult

EEP_EXCLUSIVE_AREA_02 is used in function Eep_Write. It protects the variables:

- Eep_u32EepromAddrIt
- Eep_pu8JobSrcAddrPtr
- Eep_u32RemainingLength
- Eep_eJob
- Eep_eJobResult

EEP_EXCLUSIVE_AREA_03 is used in function Eep_Read. It protects the variables:

- Eep_u32EepromAddrIt
- Eep_pu8JobDataDestPtr
- Eep_u32RemainingLength
- Eep_eJob
- Eep_eJobResult

EEP_EXCLUSIVE_AREA_04 is used in function Eep_Compare. It protects the variables:

- Eep_u32EepromAddrIt
- Eep_pu8JobSrcAddrPtr
- Eep_u32RemainingLength
- Eep_eJob
- Eep_eJobResult

EEP_EXCLUSIVE_AREA_05 is used in function Eep_QuickWrite. It protects the variables:

- Eep_u32EepromAddrIt

- Eep_pu8JobSrcAddrPtr
- Eep_u32RemainingLength
- Eep_eJob
- Eep_eJobResult

EEP_EXCLUSIVE_AREA_07 is used in function Ftfc_Eep_Ip_Write. It protects write from FLS and CSEC.

EEP_EXCLUSIVE_AREA_08 is used in function Ftfc_Eep_Ip_CmdSetFlexramFunction. It protects read from FLS and CSEC.

EEP_EXCLUSIVE_AREA_10 is used in function Eep_MainFunction. It protects the following variables:

- Ftfc_Eep_Ip_xAsyncJob.TicksStarted
- Ftfc_Eep_Ip_xAsyncJob.TicksElapsed
- Ftfc_Eep_Ip_xAsyncJob.Result

EEP_EXCLUSIVE_AREA_11 is used in function Eep_Cancel and Eep_MainFunction. It protects the following variables:

- Ftfc_Eep_Ip_xAsyncJob.TicksStarted
- Ftfc_Eep_Ip_xAsyncJob.TicksElapsed
- Ftfc_Eep_Ip_xAsyncJob.Result

5.1.1 Critical Region Exclusive Matrix

- Below is the table depicting the exclusivity between different critical region IDs from the EEP driver.
- If there is an "X" in a table, it means that those 2 critical regions cannot interrupt each other.

Exclusive Area ID	EEP_EXCLUSIVE_AREA_01 (Eep_Erase)	EEP_EXCLUSIVE_AREA_02 (Eep_Write)	EEP_EXCLUSIVE_AREA_03 (Eep_Read)	EEP_EXCLUSIVE_AREA_04 (Eep_Compare)	EEP_EXCLUSIVE_AREA_05 (Eep_QuickWrite)	EEP_EXCLUSIVE_AREA_07 protect write from FLS and CSEC (Ftfc_Eep_Ip_Write)	EEP_EXCLUSIVE_AREA_08 protect write from FLS and CSEC (Ftfc_Eep_Ip_CmdSetFlexramFunction)	EEP_EXCLUSIVE_AREA_10 (Ftfc_Eep_Ip_Write)	EEP_EXCLUSIVE_AREA_11 (Ftfc_Eep_Ip_GetJobResult)
EEP_EXCLUSIVE_AREA_01 (Eep_Erase)		x	x	x	x				
EEP_EXCLUSIVE_AREA_02 (Eep_Write)	x		x	x	x				
EEP_EXCLUSIVE_AREA_03 (Eep_Read)	x	x		x	x				
EEP_EXCLUSIVE_AREA_04 (Eep_Compare)	x	x	x		x				
EEP_EXCLUSIVE_AREA_05 (Eep_QuickWrite)	x	x	x	x					
EEP_EXCLUSIVE_AREA_07 protect write from FLS and CSEC (Ftfc_Eep_Ip_Write)									
EEP_EXCLUSIVE_AREA_08 protect read from FLS and CSEC (Ftfc_Eep_Ip_CmdSetFlexramFunction)									
EEP_EXCLUSIVE_AREA_10 (Ftfc_Eep_Ip_Write)									x
EEP_EXCLUSIVE_AREA_11 (Ftfc_Eep_Ip_GetJobResult)								x	

5.2 Exclusive areas not available on this platform

List of exclusive areas which are not available on this platform (or blank if they're all available).

- **EEP_EXCLUSIVE_AREA_00**
- **EEP_EXCLUSIVE_AREA_06**
- **EEP_EXCLUSIVE_AREA_12**
- **EEP_EXCLUSIVE_AREA_13**
- **EEP_EXCLUSIVE_AREA_14**

5.3 Peripheral Hardware Requirements

The EEP driver uses/controls the "FTFC" MCU peripheral. For more details about peripheral and its structure refers to MCU reference manual.

5.4 ISR to configure within AutosarOS - dependencies

None.

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

- **Base**: This module provides basic data types and auxiliary macros or functions commonly used by other AUTOSAR modules.
- **Dem**: This module is necessary for enabling reporting of production relevant error status. The API function used is Dem_ReportErrorStatus().
- **Det** This module is necessary for enabling Development error detection. The API function used is Det_↩ ReportError(). The activation/deactivation of Development error detection is configurable using 'EepDev↩ ErrorDetect' configuration parameter.
- **EcuC**: The ECUC module is used for ECU configuration. RTD modules need ECUC to retrieve the variant information.
- **Rte**: Exclusive areas implementations.
- **MemIf**: Memory Interface.

5.7 Data Cache Restrictions

None.

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

No special measures need to be taken to run Eep module from user mode. The Eep driver code can be executed at any time from both supervisor and user mode.

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.

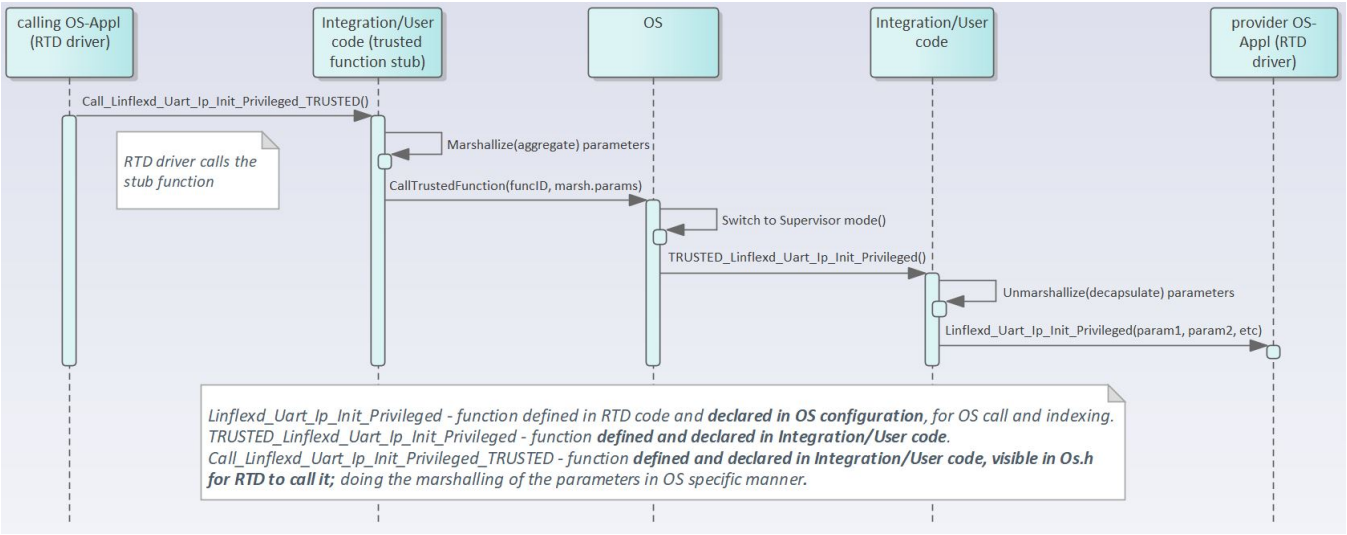

Figure 5.1 Example sequence chart for calling `Linflexd_Uart_Ip_Init_Privileged` as trusted function

5.9 Multicore support

None.

Chapter 6

Main API Requirements

6.1 Main function calls within BSW scheduler

Eep_MainFunction (call rate depends on target application, i.e. how fast the data needs to be read/written/compare in FTFC memory)

6.2 API Requirements

None.

6.3 Calls to Notification Functions, Callbacks, Callouts

The EEP driver provides notifications that are user configurable:

- EepJobEndNotification (usually routed to EA module)
- EepJobErrorNotification (usually routed to EA module)

Chapter 7

Memory allocation

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

7.1 Sections to be defined in Eep_MemMap.h

Section name	Type of section	Description
EEP_START_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	Start of Memory Section for Config Data. Used for variables, constants, structure, array and unions when SIZE (alignment) does not fit the criteria of 8,16 or 32 bit. For instance used for variables of unknown size.
EEP_STOP_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	End of above section
EEP_START_SEC_CODE	Code	Start of memory Section for Code.
EEP_STOP_SEC_CODE	Code	End of above section
EEP_START_SEC_RAMCODE	Code	Start of memory section for code placed and executed from RAM.
EEP_STOP_SEC_RAMCODE	Code	End of above section
EEP_START_SEC_VAR_CLEARED_UNSPECIFIED	Variables	Start of memory Section for Variables. Used for variables, constants, structure, array and unions when SIZE (alignment) does not fit the criteria of 8, 16 or 32 bit. For instance used for variables of unknown size. These variables are cleared to zero by start-up code.
EEP_STOP_SEC_VAR_CLEARED_UNSPECIFIED	Variables	End of above section
EEP_START_SEC_VAR_CLEARED_8	Variables	Start of Memory Section for Variable 8 bits. These variables are cleared to zero by start-up code.
EEP_STOP_SEC_VAR_CLEARED_8	Variables	End of above section
EEP_START_SEC_VAR_CLEARED_16	Variables	Start of Memory Section for Variable 16 bits. These variables are cleared to zero by start-up code.

Section name	Type of section	Description
EEP_STOP_SEC_VAR_CLEARED_16	Variables	End of above section
EEP_START_SEC_VAR_CLEARED↩ _32	Variables	Start of Memory Section for Variable 32 bits. These variables are cleared to zero by start-up code.
EEP_STOP_SEC_VAR_CLEARED_32	Variables	End of 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 EEP driver into the application.

External Assumption Req ID	External Assumption Text
EA_RTD_00071	If interrupts are locked, a centralized function pair to lock and unlock interrupts shall be used.
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
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.