Integration Manual

for S32K14X EEP Driver

Document Number: IM2EEPASR4.3 Rev0001R1.0.1
Rev. 1.0

Contents

Section number **Title** **Page**

Chapter 1
Revision History

Table 1-1. Revision History

Revision	Date	Author	Description
1.0	21/06/2019	NXP MCAL Team	Updated version for ASR 4.3.1S32K14XR1.0.1

Chapter 2
Introduction

This integration manual describes the integration requirements for EEP Driver for S32K14X microcontrollers.

2.1 Supported Derivatives

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

Table 2-1. S32K14X Derivatives

NXP Semiconductors	s32k148_lqfp144, s32k148_lqfp176, s32k148_mapbga100, s32k146_lqfp144, s32k146_lqfp100, s32k146_lqfp64, s32k146_mapbga100, s32k144_lqfp100, s32k144_lqfp64, s32k144_mapbga100, s32k142_lqfp100, s32k142_lqfp64, s32k118_lqfp48, s32k118_lqfp64, s32k142_lqfp48, s32k144_lqfp48, s32k148_lqfp100

All of the above microcontroller devices are collectively named as S32K14X .

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 type: Bold is used for important terms, notes and warnings.

Italic font: Italic typeface is 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.

2.4 Acronyms and Definitions

Table 2-2. Acronyms and Definitions

Term	Definition
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
CSEC	Cryptographic Services Engine driver
DEM	Diagnostic Event Manager
DET	Default Error Tracer
EA	EEPROM Abstraction
ECC	Error Correcting Code
ECU	Electronic Control Unit
EEP	Autosar EEPROM driver
EEPROM	Electrically Erasable Programmable Read-Only Memory
FLS	Autosar Flash driver
MCU	Micro Controller Unit
N/A	Not Applicable

Table continues on the next page...

Table 2-2. Acronyms and Definitions (continued)

Term	Definition
VLE	Variable Length Encoding
XML	Extensible Markup Language

2.5 Reference List

Table 2-3. Reference List

#	Title	Version
1	Specification of EEP Driver	AUTOSAR Release 4.3.1
2	S32K14X Reference Manual	Reference Manual, Rev. 9, 9/2018
3	S32K142 Mask Set Errata for Mask 0N33V (0N33V)	30/11/2017
4	S32K144 Mask Set Errata for Mask 0N57U (0N57U)	30/11/2017
5	S32K146 Mask Set Errata for Mask 0N73V (0N73V)	30/11/2017
6	S32K148 Mask Set Errata for Mask 0N20V (0N20V)	25/10/2018
7	S32K118 Mask Set Errata for Mask 0N97V (0N97V)	07/01/2019

Chapter 3
Building the Driver

This section describes the source files and various compilers, linker options used for building the Autosar EEP driver for NXP SemiconductorsS32K14X . It also explains the EB Tresos Studio plugin setup procedure.

3.1 Build Options

The EEP driver files are compiled using

- Green Hills Multi 7.1.4 / Compiler 2017.1.4
- (Linaro GCC 6.3-2017.06~dev) 6.3.1 20170509 (Wed Jan 24 16:21:45 CST 2018 build.sh rev=g27a1317 s=L631 Earmv7 -V release_g27a1317_build_Fed_Earmv7)
- IAR: V8.11.2

The compiler, linker flags used for building the driver are explained below:

Note

The TS_T40D2M10I1R0 plugin name is composed as follow:

TS_T = Target_Id

D = Derivative_Id

M = SW_Version_Major

I = SW_Version_Minor

R = Revision

(i.e. Target_Id = 40 identifies CORTEXM architecture and Derivative_Id = 2 identifies the S32K14X)

3.1.1 GHS Compiler/Linker/Assembler Options

Table 3-1. Compiler Options

Option	Description
-cpu=cortexm4	Selects target processor: Arm Cortex M4
-cpu=cortexm0plus	Selects target processor: Arm Cortex M0+
-ansi	Specifies ANSI C with extensions. This mode extends the ANSI X3.159-1989 standard with certain useful and compatible constructs.
-Osize	Optimize for size.
-dual_debug	Enables the generation of DWARF, COFF, or BSD debugging information in the object file
-G	Generates source level debugging information and allows procedure call from debugger's command line.
--no_exceptions	Disables support for exception handling
-Wundef	Generates warnings for undefined symbols in preprocessor expressions
-Wimplicit-int	Issues a warning if the return type of a function is not declared before it is called
-Wshadow	Issues a warning if the declaration of a local variable shadows the declaration of a variable of the same name declared at the global scope, or at an outer scope
-Wtrigraphs	Issues a warning for any use of trigraphs
-Wall	Enables all the warnings about constructions that some users consider questionable, and that are easy to avoid even in conjunction with macros.
--prototype_errors	Generates errors when functions referenced or called have no prototype
--incorrect_pragma_warnings	Valid #pragma directives with wrong syntax are treated as warnings
-noslashcomment	C++ like comments will generate a compilation error
-preprocess_assembly_files	Preprocesses assembly files
-nostartfile	Do not use Start files
--short_enum	Store enumerations in the smallest possible type
-c	Produces an object file (called input-file.o) for each source file.
--no_commons	Allocates uninitialized global variables to a section and initializes them to zero at program startup.
-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. Produces an object file (called input-file.o) for each source file.
-list	Creates a listing by using the name of the object file with the .lst extension. Assembler option
-DAUTOSAR_OS_NOT_USED	-D defines a preprocessor symbol and optionally can set it to a value. AUTOSAR_OS_NOT_USED: By default in the package, the drivers are compiled to be used without Autosar OS. If the drivers are used with Autosar OS, the compiler option '-DAUTOSAR_OS_NOT_USED' must be removed from project options
-DDISABLE_MCAL_INTERMODULE_ASR_CHECK	-D defines a preprocessor symbol to disable the inter-module version check for AR_RELEASE versions. DISABLE_MCAL_INTERMODULE_ASR_CHECK: By default in the package, drivers are compiled to perform the inter-module version check as per Autosar BSW004. When the inter-module version check needs to be disabled then the DISABLE_MCAL_INTERMODULE_ASR_CHECK global define must be added to the list of compiler options.
-DGHS	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the GHS preprocessor symbol.

Table 3-2. Assembler Options

Option	Description
-cpu=cortexm4	Selects target processor: Arm Cortex M4
-cpu=cortexm0plus	Selects target processor: Arm Cortex M0+
-c	Produces an object file (called input-file.o) for each source file.
-preprocess_assembly_files	Preprocesses assembly files
-asm=list	Creates a listing by using the name of the object file with the .lst extension. Assembler option

Table 3-3. Linker Options

Option	Description
-Mn	Map file numeric ordering
-delete	Removal from the executable of functions that are unused and unreferenced
-v	Display removed unused functions
-ignore_debug_references	Ignores relocations from DWARF debug sections when using -delete.
-map	Creates a detailed map file
-keepmap	Keep the map file in the event of a link error
-lstartup	Link libstartup library -Run-time environment startup routines
-lsys	Link libsys library -Run-time environment system routines
-larch	Link libarch library -Target-specific run-time support. Any file produced by the Green Hills Compiler may depend on symbols in this library.
-lansi	Link libansi library -the standard C library
-L(/lib/thumb2)	Link thumb2 library
-lutf8_s32	Include utf8_s32.a to use the Wide Character Functions

3.1.2 IAR Compiler/Linker/Assembler Options

Table 3-4. Compiler Options

Option	Description
--cpu=Cortex-M4	Selects target processor: Arm Cortex M4
--cpu=Cortex-M0+	Selects target processor: Arm Cortex M0+
--cpu_mode=thumb	Selects generating code that executes in Thumb state.
--endian=little	Specifies the endianess of core: little endian.
-Ohz	Sets the optimization level to High, favoring size.
-c	Produces an object file (called input-file.o) for each source file.
--no_clustering	Disables static clustering optimizations.
--no_mem_idioms	Makes the compiler to not optimize code sequences that clear, set, or copy a memory region.
--no_explicit_zero_opt	Places the zero initialized variables in data section instead of bss.
--debug	Makes the compiler include information in the object modules.

Table continues on the next page...

Table 3-4. Compiler Options (continued)

Option	Description
--diag_suppress=Pa050	Suppresses diagnostic messages (warnings) about non-standard line endings.
-DAUTOSAR_OS_NOT_USED	-D defines a preprocessor symbol and optionally can set it to a value. AUTOSAR_OS_NOT_USED: By default in the package, the drivers are compiled to be used without Autosar OS. If the drivers are used with Autosar OS, the compiler option '-DAUTOSAR_OS_NOT_USED' must be removed from project options
-DIAR	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the IAR preprocessor symbol.
--require_prototypes	Forces the compiler to verify that all functions have proper prototypes.
--no_wrap_diagnostics	Disables line wrapping of diagnostic messages issued by compiler.
--no_system_include	Disables the automatic search for system include files.
-e	Enables language extensions. This option is needed by FLS driver which uses _packed structures.

Table 3-5. Assembler Options

Option	Description
--cpu=Cortex-M4	Selects target processor: Arm Cortex M4
--cpu=Cortex-M0+	Selects target processor: Arm Cortex M0+
--cpu_mode=thumb	Selects generating code that executes in Thumb state.
-g	Use this option to disable the automatic search for system include files.

Table 3-6. Linker Options

Option	Description
--cpu=Cortex-M4	Selects target processor: Arm Cortex M4
--cpu=Cortex-M0+	Selects target processor: Arm Cortex M0+
--map filename	Produces a map file.
--no_library_search	Disables automatic runtime library search.
--entry _start	Treats the symbol _start as a root symbol and as the start of the application.
--enable_stack_usage	Enables stack usage analysis.
--skip_dynamic_initialization	Suppress dynamic initialization during system startup.
--no_wrap_diagnostics	Disables line wrapping of diagnostic messages issued by linker.
--config	Specifies the configuration file to be used by the linker.

3.1.3 GCC Compiler/Linker/Assembler Options

Table 3-7. Compiler Options

Option	Description
-c	Produces an object file (called input-file.o) for each source file.
-Os	Use optimization for size.
-ggdb3	Produce debugging information for use by GDB. Level 3 includes extra information, such as all the macro definitions present in the program.
-mcpu=cortex-m4	Selects target processor: Arm Cortex M4
-mcpu=cortex-m0plus	Selects target processor: Arm Cortex M0+
-mthumb	Selects generating code that executes in Thumb state.
-ansi	Specifies ANSI C with extensions.
-mlittle-endian	Generate code for a processor running in little-endian mode.
-fomit-frame-pointer	Removes the frame pointer for all functions, which might make debugging harder.
-msoft-float	Use software floating-point instructions.
-fno-common	Specifies that the compiler should place uninitialized global variables in the data section of the object file, rather than generating them as common blocks.
-Wall	Enables all the warnings about constructions that some users consider questionable, and that are easy to avoid even in conjunction with macros.
-Wextra	Enables some extra warning flags that are not enabled by '-Wall'.
-Wstrict-prototypes	Warn if a function is declared or defined without specifying the argument types.
-Wno-sign-compare	Do not warn when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned.
-fstack-usage	Geneates an extra file that specifies the maximum amount of stack used, on a per-function basis.
-fdump-ipa-all	Enables all inter-procedural analysis dumps.
-Werror=implicit-function-declaration	Generates an error when the prototype of the function is not defined..
-DAUTOSAR_OS_NOT_USED	-D defines a preprocessor symbol and optionally can set it to a value. AUTOSAR_OS_NOT_USED: By default in the package, the drivers are compiled to be used without Autosar OS. If the drivers are used with Autosar OS, the compiler option '-DAUTOSAR_OS_NOT_USED' must be removed from project options
-DGCC	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the GCC preprocessor symbol.
-std=c99	C programming language standard version c99

Table 3-8. Assembler Options

Option	Description
-mcpu=cortex-m4	Selects target processor: Arm Cortex M4
-mcpu=cortex-m0plus	Selects target processor: Arm Cortex M0+
-c	Produces an object file (called input-file.o) for each source file.
-mthumb	This option specifies that the assembler should start assembling Thumb instructions.
-x assembler-with-cpp	Indicates that the assembly code contains C directives and the C preprocessor must be run.

Table 3-9. Linker Options

Option	Description
-Map=filename	Print a link map to the file mapfile.
-T scriptfile	Use scriptfile as the linker script. This script replaces ld's default linker script(rather than adding to it), so commandfile must specify everything necessary to describe the output file.
--disable-newlib-supplied-syscalls -specs=nosys.specs	These options support for using newlib on core M0+
-u _printf_float -u _scanf_float	These options support generating profile report.
-nostartfiles	Do not use the standard system startup files when linking
-e _start	Specify that the program entry point is _start
-static	The --static flag tells the linker to link a static, not a dynamically linked
-lc	The -lc flag tells the linker to link this binary against the C library, which is newlib in our case.
-lnosys	The -lnosys flag tells the linker to link this binary against the "nosys" library
$(TOOLCHAIN_DIR)/arm-none-eabi/newlib/lib/thumb/v6-m $(TOOLCHAIN_DIR)/lib/gcc/arm-none-eabi/6.3.1/thumb/v6-m	Library for core M0+, added with -L and -B option
$(TOOLCHAIN_DIR)/arm-none-eabi/newlib/lib/thumb $(TOOLCHAIN_DIR)/arm-none-eabi/newlib/lib)	Library for core M4, added with -L and -B option

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 S32K14X 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

- ..\Eep_TS_T40D2M10I1R0\src\Eep.c
- ..\Eep_TS_T40D2M10I1R0\src\Eep_Ac.c
- ..\Eep_TS_T40D2M10I1R0\include\Eep.h
- ..\Eep_TS_T40D2M10I1R0\include\Eep_Api.h
- ..\Eep_TS_T40D2M10I1R0\include\Eep_InternalTypes.h
- ..\Eep_TS_T40D2M10I1R0\include\Eep_Types.h
- ..\Eep_TS_T40D2M10I1R0\include\Reg_eSys_Ftfc.h

- Eep_PBcfg.c - 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

Other includes files:

Files from MemIf folder:
- ..\MemIf_TS_T40D2M10I1R0\include\MemIf_Types.h

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\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

Files from Dem folder:
- ..\Dem_TS_T40D2M10I1R0\include\Dem.h

Files from Det folder:
- ..\Det_TS_T40D2M10I1R0\include\Det.h

Files from RTE folder:
- ..\Rte_TS_T40D2M10I1R0\include\SchM_Eep.h

3.3 Setting up the Plug-ins

The EEP driver was designed to be configured by using the EB Tresos Studio (version EB tresos Studio 24.0.1 b180321-0610 or later.)

- VSMD (Vendor Specific Module Definition) file in EB tresos Studio XDM format: .. \Eep_TS_T40D2M10I1R0\config\Eep.xdm
- VSMD (Vendor Specific Module Definition) file in AUTOSAR compliant EPD format: ..\Eep_TS_T40D2M10I1R0\autosar\(one EPD file for each supported sub-derivative)

- Code Generation Templates for Post-Build time configuration parameters:
  - ..\Eep_TS_T40D2M10I1R0\generate\include\Eep_Cfg.h
  - ..\Eep_TS_T40D2M10I1R0\generate\src\Eep_PBcfg.c
- Code Generation Templates for Pre-Compile time configuration parameters:
  - ..\Eep_TS_T40D2M10I1R0\generate\include\Eep_Cfg.h
  - ..\Eep_TS_T40D2M10I1R0\generate\src\Eep_Cfg.c

Steps to generate the configuration:

1. Copy the module folders Eep_TS_T40D2M10I1R0 , Base_TS_T40D2M10I1R0, Resource_TS_T40D2M10I1R0, Dem_TS_T40D2M10I1R0, MemIf_TS_T40D2M10I1R0, Det_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.

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.

If the EEP driver is used in User Mode, be sure that the Eeprom memory controller registers are accessible and that accessed Eeprom memory partition is not protected. For more information please refer to the "Memory Protection Unit" and "Register Protection" chapters in the device reference manual.

The Eeprom memory physical sectors that are going to be modified by Eep driver (i.e. erase and write operations) have to be unprotected for a successful operation.

The FlexNVM memory has to be partitioned for EEPROM emulation.

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

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 MCAL 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_06,EEP_EXCLUSIVE_AREA_07,EEP_EXCLUSIVE_AREA_08,EEP_EXCLUSIVE_AREA_09. The purpose of EEP_EXCLUSIVE_AREA_01, EEP_EXCLUSIVE_AREA_02, EEP_EXCLUSIVE_AREA_03, EEP_EXCLUSIVE_AREA_04,EEP_EXCLUSIVE_AREA_05,EEP_EXCLUSIVE_AREA_06 exclusive areas is to protect EEP internal job variables. The purpose of EEP_EXCLUSIVE_AREA_07, EEP_EXCLUSIVE_AREA_08, EEP_EXCLUSIVE_AREA_09 is to protect EEP from CSEC and FLS operations. If EEP, FLS and CSEC are used together, then the integrator must call their APIs from the same task or map the dedicated exclusive area from each driver on the same OS resource. For EEP, exclusive areas EEP_EXCLUSIVE_AREA_07, EEP_EXCLUSIVE_AREA_08, EEP_EXCLUSIVE_AREA_09 must be mapped on a OS resource which is shared with FLS and CSEC.

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.

EEP_EXCLUSIVE_AREA_01 Used in Eep_Erase.

EEP_EXCLUSIVE_AREA_02 Used in Eep_Write.

EEP_EXCLUSIVE_AREA_03 Used in Eep_Read.

EEP_EXCLUSIVE_AREA_04 Used in Eep_Compare.

EEP_EXCLUSIVE_AREA_05 Used in Eep_QuickWrite.

EEP_EXCLUSIVE_AREA_06 Used in Eep_Cancel.

EEP_EXCLUSIVE_AREA_07 Used in Eep_MainFunction for a write job to protect from FLS and CSEC.

EEP_EXCLUSIVE_AREA_08 Used in Eep_MainFunction for a read job to protect from FLS and CSEC.

EEP_EXCLUSIVE_AREA_09 Used in Eep_MainFunction for a compare job to protect from FLS and CSEC.

Table 5-1. Exclusive Areas

	EEP_EXCLUSIVE_AREA_01	EEP_EXCLUSIVE_AREA_02	EEP_EXCLUSIVE_AREA_03	EEP_EXCLUSIVE_AREA_04	EEP_EXCLUSIVE_AREA_05	EEP_EXCLUSIVE_AREA_06	EEP_EXCLUSIVE_AREA_07	EEP_EXCLUSIVE_AREA_08	EEP_EXCLUSIVE_AREA_09
EEP_EXCLUSIVE_AREA_01		x	x	x	x	x			
EEP_EXCLUSIVE_AREA_02	x		x	x	x	x			
EEP_EXCLUSIVE_AREA_03	x	x		x	x	x			
EEP_EXCLUSIVE_AREA_04	x	x	x		x	x			
EEP_EXCLUSIVE_AREA_05	x	x	x	x		x			
EEP_EXCLUSIVE_AREA_06	x	x	x	x	x				
EEP_EXCLUSIVE_AREA_07									
EEP_EXCLUSIVE_AREA_08									
EEP_EXCLUSIVE_AREA_09									

5.2 Peripheral Hardware Requirements

The EEP driver uses/controls the FLEXRAM, FTFC peripheral. For more details about peripheral and its structure refer to hardware reference manual.

Attempts to launch an FTFC command in VLP and HSRUN mode is not supported. For more details, please refer to the reference manual.

5.3 ISR to configure within OS – dependencies

None.

5.4 ISR Macro

None.

5.5 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 'CanDevErrorDetect' configuration parameter.
- **EcuC:** (only for ASR 4.2) The ECUC module is used for ECU configuration.MCAL modules need ECUC to retrieve the variant information.
- **Rte:** Exclusive areas implementations.
- **MemIf:** Memory Interface

5.6 Data Cache Restriction

None.

5.7 User Mode Support

The Eep driver can run in both supervisor mode and user mode. For this platform, there is not any specific measure required for running in user mode.

Chapter 6
Main API Requirements

6.1 Main functions calls within Rte module

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

6.2 API Requirements

None

6.3 Calls to Notification Functions, Callbacks, Callouts

The EEP driver provides notifications that are user configurable:

- EepAcCallback (usually routed to Wdg module)
- EepJobEndNotification (usually routed to EA module)
- EepJobErrorNotification (usually routed to EA module)
- EepStartEepromAccessNotif (used, if needed, to mark the start of a eeprom program access)
- EepFinishedEepromAccessNotif (used, if needed, to mark the end of a eeprom program access)

6.4 Tips for EEP integration

Synchronous vs. Asynchronous write mode

Asynchronous write mode (set up by parameter EepPageAsynchBehaviorEn) works in the way, that Eep_MainFunction() just schedules the HW write operation and does not wait for its completion. In the next Eep_MainFunction() it is checked if the write operation is finished. If yes (depends on how often the Eep_MainFunction() is called), another write operation is scheduled. This process is repeated until all data is written. In this mode, EepMaxWriteFastMode/EepMaxWriteNormalMode values are ignored, data is written just by page(1,2 or 4 bytes) length.

When synchronous write mode is used, Eep_MainFunction() initializes write operation and also waits for its completion.

So the main differences between these two modes are in the time consumption and number of calls of the Eep_MainFunction(). The Eep_MainFunction() takes less time in asynchronous mode, but the whole write operation uses more Eep_MainFunction() executions.

In case EEP is used in the same application with FLS or CSEC, then the asynchronous mode must not be used.

Usage of Eep_QuickWrite API

The NonAutosar interface Eep_QuickWrite is available if parameter EepEnableQuickWriteApi is set to true. Eep_QuickWrite will initiate a write job which will be executed by Eep_MainFunction. Processing of a quick write job by Eep_MainFunction will: set up the hardware in the quick write mode with the specified number of u16QuickWritesLength, executes the write, then sets up the hardware for the normal mode. In case changing the mode to quick writes or normal mode fails, this is not reported to the user as the error is considered irrelevant for the user: only the actual status of the write operation is considered relevant and it will be reported via DEM error/ JobErrorNotification/Eep_GetJobResult. The hardware has no feature to inform about which mode is actually set up(quick/normal) in case there were errors. So if Eep_QuickWrite is called inside the application and the mode change has errors, then the jobs of Eep_Write might be processed in quick write mode In the quick write mode, the hardware will write as fast as possible and postpone the management operations which are time consuming. A use-case for Eep_QuickWrite is in case of storing crash data. The u16QuickWritesLength parameter has the following restrictions: must be larger than 16, smaller than 512 and multiple of 4. Also, the data length specified by u32Length parameter must be a multiple of u16QuickWritesLength.

Note: The quick write mode should be used only if necessary as it has the following disadvantage: if a reset occurs before the entire u16QuickWritesLength number of bytes was written to FLEXRAM, then all bytes are lost. For more details about quick write mode please see the description of "Set FlexRAM Function command" from the hardware manual.

Alignment and endurance consideration

The maximum endurance specified in the data sheet will be achieved if only 4 bytes FLEXRAM writes are done. For writing to FLEXRAM, EEP uses 1 byte page for unaligned accesses and 2 or 4 bytes page for aligned accesses. To achieve the highest endurance, the EEP integrator must use only source and destination buffers which are start and end aligned for the write jobs triggered by Eep_Write, Eep_Erase and Eep_QuickWrite. For example: when calling Eep_Write, make sure that u32TargetAddress and pSourceAddressPtr are 4 bytes start and end aligned (and u32Length is multiple of 4).

EEP, FLS and CSEC driver integration

EEP, FLS and CSEC drivers share some hardware resources of the FLEXNVM memory, so several aspects need to be considered when using them together in the same application.

1.If CSEC is used, then the parameter EepSizeUsedForCSEC must be configured to the actual size of FLEXRAM used by the CSEC driver to store the keys.

2.If EEP is used with CSEC in the same application, then the EEP asynchronous mode must not be used.

3.If EEP is used with FLS in the same application, then the EEP and FLS asynchronous modes must not be used.

4.If EEP, FLS and CSEC are used together, then the integrator must ensure that EEP operation is not interrupted by the other drivers (for example call their APIs from the same task and make sure no interrupt running CSEC will interrupt EEP or map the dedicated exclusive area from each driver on the same OS resource, etc)

Memory protection

The integrator shall ensure that ERAM and EFLASH memory used by the EEP driver is unprotected before calling Eep_Write, Eep_QuickWrite and Eep_Erase APIs.

Mode restriction

Attempts to launch an FTFC command in VLP and HSRUN mode is not supported. The integrator shall ensure that the microcontroller runs in a mode which supports FTFC operations before calling Eep_Init, Eep_Write, Eep_QuickWrite and Eep_Erase APIs.

Chapter 7
Memory Allocation

7.1 Sections to be defined in MemMap.h

For Post Build data:

```
#ifdef EEP_START_SEC_CONFIG_DATA_8
#undef EEP_START_SEC_CONFIG_DATA_8
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
#ifdef EEP_STOP_SEC_CONFIG_DATA_8
#undef EEP_STOP_SEC_CONFIG_DATA_8
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif



#ifdef EEP_START_SEC_CONFIG_DATA_UNSPECIFIED
#undef EEP_START_SEC_CONFIG_DATA_UNSPECIFIED
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
#ifdef EEP_STOP_SEC_CONFIG_DATA_UNSPECIFIED
#undef EEP_STOP_SEC_CONFIG_DATA_UNSPECIFIED
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
```

For Code:

```
#ifdef EEP_START_SEC_CODE
#undef EEP_START_SEC_CODE
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
#ifdef EEP_STOP_SEC_CODE
#undef EEP_STOP_SEC_CODE
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
```

For Variables:

```
#ifdef EEP_START_SEC_VAR_INIT_BOOLEAN
#undef EEP_START_SEC_VAR_INIT_BOOLEAN
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
#ifdef EEP_STOP_SEC_VAR_INIT_BOOLEAN
#undef EEP_STOP_SEC_VAR_INIT_BOOLEAN
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif



#ifdef EEP_START_SEC_VAR_INIT_8
#undef EEP_START_SEC_VAR_INIT_8
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
#ifdef EEP_STOP_SEC_VAR_INIT_8
#undef EEP_STOP_SEC_VAR_INIT_8
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif



#ifdef EEP_START_SEC_VAR_INIT_32
#undef EEP_START_SEC_VAR_INIT_32
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
#ifdef EEP_STOP_SEC_VAR_INIT_32
#undef EEP_STOP_SEC_VAR_INIT_32
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif



#ifdef EEP_START_SEC_VAR_INIT_UNSPECIFIED
#undef EEP_START_SEC_VAR_INIT_UNSPECIFIED
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
#ifdef EEP_STOP_SEC_VAR_INIT_UNSPECIFIED
#undef EEP_STOP_SEC_VAR_INIT_UNSPECIFIED
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
```

For Constant data:

```
#ifdef EEP_START_SEC_CONST_32
#undef EEP_START_SEC_CONST_32
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
#ifdef EEP_STOP_SEC_CONST_32
#undef EEP_STOP_SEC_CONST_32
#undef MEMMAP_ERROR
/*no definition -> default compiler settings are used */
#endif
```

```
   #ifdef EEP_START_SEC_CONST_UNSPECIFIED
   #undef EEP_START_SEC_CONST_UNSPECIFIED
   #undef MEMMAP_ERROR
   /*no definition -> default compiler settings are used */
   #endif
   #ifdef EEP_STOP_SEC_CONST_UNSPECIFIED
   #undef EEP_STOP_SEC_CONST_UNSPECIFIED
   #undef MEMMAP_ERROR
   /*no definition -> default compiler settings are used */
   #endif
```

For Ram Code:

For position-independent Access Code:

```
  #ifdef EEP_START_SEC_CODE_AC
    #undef EEP_START_SEC_CODE_AC
    #undef MEMMAP_ERROR
    /* use code relative addressing mode to ensure Position-independent Code */
    #pragma section CODE ".aceep_code_rom" far-code /* Diab example */
#endif
#ifdef EEP_STOP_SEC_CODE_AC
    #undef EEP_STOP_SEC_CODE_AC
    #undef MEMMAP_ERROR
    #pragma section CODE
#endif
```

7.2 Linker command file

Memory shall be allocated for every section defined in EEP_MemMap.h

7.3 Linker command file Access Code section

The "Eep_Eeprom_AccessCode" function executes the actual hardware write/erase operations or Set FlexRam command. The function Eep_Eeprom_AccessCode is used by Eep_Init, Eep_Write, Eep_QuickWrite and Eep_Wrase.

The "Eep_Eeprom_AccessCode" function is placed in the driver code inside a specific linker section(".aceep_code_rom"), which can be used in the linker to specifically position this code section.

The "Eep_Eeprom_AccessCode" function must be executed from a different partition than the ones which contain the current written/erased sector, or it has to be executed from RAM, in order to meet the Read-While-Write restrictions. For more details about access code, see also the User Manual, "6.1 Avoiding RWW problem" chapter.

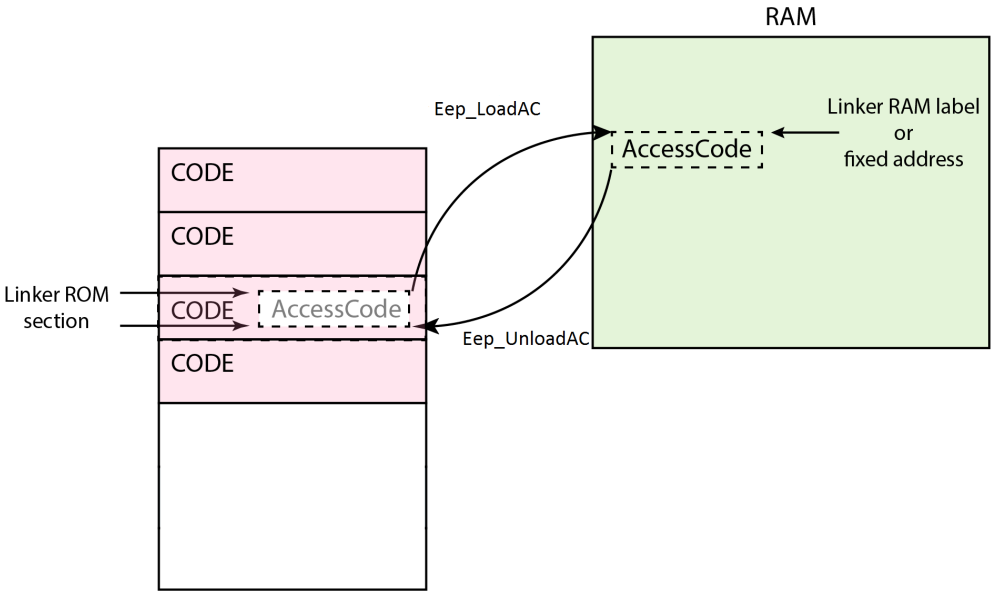

Figure 7-1. "Eep_Eeprom_AccessCode" function used from flash

If "Eep_Eeprom_AccessCode" function is executed from flash, configuration parameter "EepAcLoadOnJobStart" must be cleared and the Read-While-Write restrictions apply when configuring the used flash sectors.

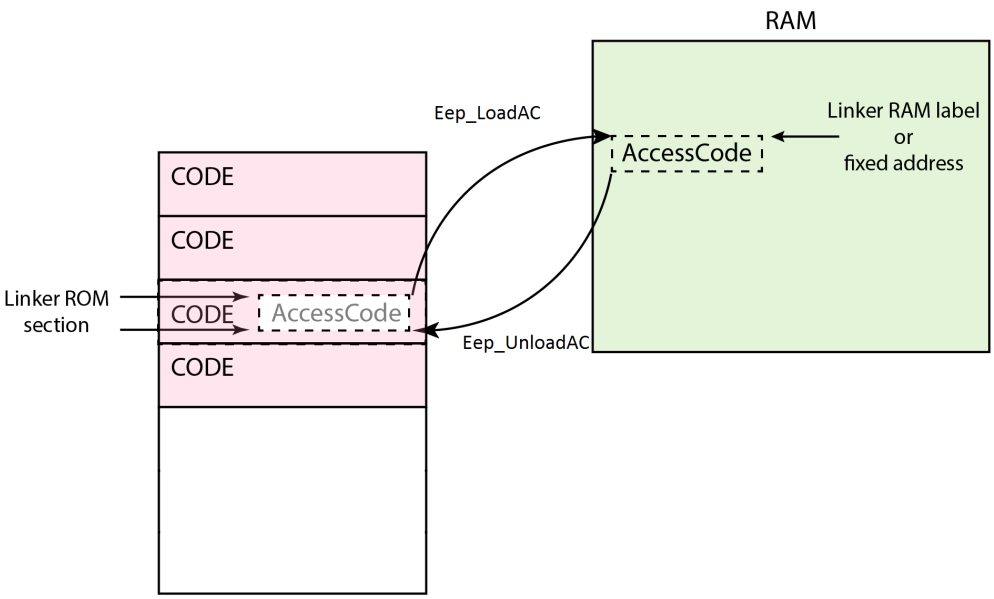

Figure 7-2. "Eep_Eeprom_AccessCode" function used from RAM

If executed from RAM, configuration parameter "EepAcLoadOnJobStart" must be set and there have to be defined in the linker file the following symbols:

- Eep_ACWriteRomStart - start address of the section .aceep_code_rom
- Eep_ACWriteSize - size of .aceep_code_rom (word-aligned, in words)

and at least 4-bytes aligned space reserved in RAM at locations defined by configuration parameters:

- EepAcWrite of space corresponding to Eep_ACWriteSize (see above)

Alternatively, using the following configuration parameters

- EepAcWritePointer

it is possible to use symbolic name instead of absolute addresses, but in this case the linker should define them.

Note that the linker shall be prevented from .aceep_code_rom section removal, esp. when dead code stripping is enabled, e.g. by using keep directive as shown in the examples below.

DIAB linker command file example:

```
    ....

      GROUP : {
            /* ... */
            .aceep_code_rom(TEXT)  ALIGN(4) : {KEEP (*(.aceep_code_rom))}  /* see
EEP_START_SEC_CODE_AC */

            /* ... */
    }>eeprom_memory

      GROUP : {
            /* ... */
            .aceep_code_ram(TEXT) ALIGN(4): {}
            . += SIZEOF(.aceep_code_rom);      /* reserved RAM space */
            .aceep_code_ram_end ALIGN(4): {}   /* make sure the size of the reserved space
is 4-bytes aligned */
            /* ... */
    }>int_sram

    /* Eep module access code support */

    Eep_ACWriteRomStart         = ADDR(.aceep_code_rom);
    Eep_ACWriteSize             = (SIZEOF(.aceep_code_rom)+3) / 4; /* Copy 4 bytes at a
time*/

    _ERASE_FUNC_ADDRESS_        = ADDR(.aceep_code_ram);  /* AC Erase Ptr */
    _WRITE_FUNC_ADDRESS_        = ADDR(.aceep_code_ram);  /* AC Write Ptr */
```

GHS linker command file example:

```
//...
SECTIONS
{
//
// RAM SECTIONS
    .intc_vector                                ABS : > int_sram
    //...
    // reserve space for .aceep_code_ram
    .aceep_code_ram                        ALIGN(4) : {. += SIZEOF(.aceep_code_rom);} > .
    // make sure the size of the reserved space is 4-bytes aligned
    .aceep_code_ram_end                    ALIGN(4) : > .
//
// ROM SECTIONS
//
```

Linker command file Access Code section

```
    //...
    .aceep_code_rom                         ALIGN(4) : > eeprom_memory

    /* Eep module access code support */

    Eep_ACWriteRomStart          = ADDR(.aceep_code_rom);
    Eep_ACWriteSize              = (SIZEOF(.aceep_code_rom)+3) / 4; /* Copy 4 bytes at a
time*/

    _ERASE_FUNC_ADDRESS_         = ADDR(.aceep_code_ram);
    _WRITE_FUNC_ADDRESS_         = ADDR(.aceep_code_ram);
}

OPTION ("-keep=Eep_LLD_AccessCode")
```

Chapter 8
Configuration parameters considerations

Configuration parameter class for Autosar EEP driver fall into the following variants as defined below:

8.1 Configuration Parameters

Specifies whether the configuration parameter shall be of configuration class Post Build.

Table 8-1. Configuration Parameters

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
Eep	IMPLEMENTATION_CONFIG_VARIANT		
EepInitConfiguration	EepSize	Variant PC or Variant PB	Pre Compile
	EepBaseAddress	Variant PC or Variant PB	Pre Compile
	EepAcWrite	Variant PC or Variant PB	Post Build
	EepAcWritePointer	Variant PC or Variant PB	Post Build
	EepJobCallCycle	Variant PC or Variant PB	Post Build
	EepDefaultMode	Variant PC or Variant PB	Post Build
	EepACCallback	Variant PC or Variant PB	Post Build
	EepJobEndNotification	Variant PC or Variant PB	Post Build
	EepJobErrorNotification	Variant PC or Variant PB	Post Build
	EepStartEepromAccessNotif	Variant PC or Variant PB	Post Build
	EepFinishedEepromAccessNotif	Variant PC or Variant PB	Post Build
	EepFastReadBlockSize	Variant PC or Variant PB	Post Build
	EepNormalReadBlockSize	Variant PC or Variant PB	Post Build
	EepFastWriteBlockSize	Variant PC or Variant PB	Post Build
	EepNormalWriteBlockSize	Variant PC or Variant PB	Post Build
EepInitConfiguration/ EepDemEventParameterRefs	EEP_E_COMPARE_FAILED	Variant PC or Variant PB	Post Build
	EEP_E_ERASE_FAILED	Variant PC or Variant PB	Post Build
	EEP_E_READ_FAILED	Variant PC or Variant PB	Post Build

Table continues on the next page...

Table 8-1. Configuration Parameters (continued)

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
	EEP_E_WRITE_FAILED	Variant PC or Variant PB	Post Build
EEP_E_BO_MAINTENANCE	Variant PC or Variant PB	Post Build	
EEP_E_BO_QUICK_WRITES	Variant PC or Variant PB	Post Build	
EEP_E_BO_NORMAL_WRITES	Variant PC or Variant PB	Post Build	
EepInitConfiguration/ EepExternalDriverEepSpiReference	EepSpiReference	Pre Compile parameter for all Variants of Configuration	Pre Compile
NonAutosar	EepAcLoadOnJobStart	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepCancelApi	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepCompareApi	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepGetJobResultApi	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepGetStatusApi	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepSetModeApi	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepPageAsynchBehaviorEn	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepEnableQuickWriteApi	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepSizeUsedForCSEC	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepDisableDemReportErrorStatus	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepEnableUserModeSupport	Pre Compile parameter for all Variants of Configuration	Pre Compile
EepGeneral	EepDevErrorDetect	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepDriverIndex	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepUseInterrupts	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepVersionInfoApi	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepWriteCycleReduction	Pre Compile parameter for all Variants of Configuration	Pre Compile
EepGeneral/EepTimeouts	EepAsyncWriteTimeout	Pre Compile parameter for all Variants of Configuration	Pre Compile
	EepSyncWriteTimeout	Pre Compile parameter for all Variants of Configuration	Pre Compile

Table continues on the next page...

Table 8-1. Configuration Parameters (continued)

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
	EepAbortTimeout	Pre Compile parameter for all Variants of Configuration	Pre Compile
EepPublishedInformation	EepEraseUnitSize	Variant PC or Variant PB	Published information
	EepEraseTime	Variant PC or Variant PB	Published information
	EepEraseValue	Variant PC or Variant PB	Published information
	EepMinimumAddressType	Variant PC or Variant PB	Published information
	EepMinimumLengthType	Variant PC or Variant PB	Published information
	EepReadUnitSize	Variant PC or Variant PB	Published information
	EepTotalSize	Variant PC or Variant PB	Published information
	EepAllowedWriteCycles	Variant PC or Variant PB	Published information
	EepSpecifiedEraseCycles	Variant PC or Variant PB	Published information
	EepWriteTime	Variant PC or Variant PB	Published information
	EepWriteUnitSize	Variant PC or Variant PB	Published information
CommonPublishedInformation	ArReleaseMajorVersion	Variant PC or Variant PB	Published information
	ArReleaseMinorVersion	Variant PC or Variant PB	Published information
	ArReleaseRevisionVersion	Variant PC or Variant PB	Published information
	ModuleId	Variant PC or Variant PB	Published information
	SwMajorVersion	Variant PC or Variant PB	Published information
	SwMinorVersion	Variant PC or Variant PB	Published information
	SwPatchVersion	Variant PC or Variant PB	Published information
	VendorApiInfix	Variant PC or Variant PB	Published information
	VendorId	Variant PC or Variant PB	Published information

Chapter 9
Integration Steps

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

- Generate the required EEP configurations. For more details refer to section Files required for Compilation
- Allocate proper memory sections in EEP_MemMap.h and linker command file. For more details refer to section Sections to be defined in MemMap.h
- Compile & build the EEP with all the dependent modules. For more details refer to section Building the Driver

Chapter 10
ISR Reference

None

Chapter 11
External Assumptions for EEP driver

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

[SMCAL_CPR_EXT183]

The option "Eep Hardware Timeout Handling" shall be enabled in the EEP driver configuration and the timeout value of parameters EepSyncWriteTimeout (in case synchronuous EEP operation is used) and EepAbortTimeout shall be configured to fit the application timing.

[EEP161]

For variants with no postbuild multiple selectable configuration parameters (Variant PC), the EEP module's environment shall pass a NULL pointer to the function Eep_Init().

[EEP115]

The Eep's user shall not call the function Eep_Init during a running operation.

[EEP116]

The Eep's user shall not call the function Eep_SetMode during a running operation.

[EEP117]

The Eep's user shall only call Eep_Read after the Eep module has been been initialized.

[EEP118]

The Eep's user shall not call the function Eep_Read during a running Eep module job (read/write/erase/compare).

[EEP119]

The Eep module's user shall only call the function Eep_Write after the Eep module has been initialized.

[EEP120]

The Eep module's user shall not call the function Eep_Write during a running Eep module job (read/write/erase/compare).

[EEP121]

The Eep module's user shall only call the function Eep_Erase after the Eep module has been initialized.

[EEP122]

The Eep module's user shall not call the function Eep_Erase during a running Eep job (read/write/erase/compare).

[EEP123]

The Eep module's user shall only call the function Eep_Compare after the Eep module has been initialized.

[EEP124]

The Eep module's user shall not call the function Eep_Compare during a running Eep job (read/write/erase/compare).

[EEP136]

The Eep module's user shall not call the Eep_Cancel() function during a running Eep_MainFunction() function.

[EEP126]

The callback function defined in the configuration parameter EepJobEndNotification shall be callable on interrupt level.

[EEP127]

The callback function defined in the configuration parameter EepJobErrorNotification shall be callable on interrupt level.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number IM2EEPASR4.3 Rev0001R1.0.1
Revision 1.0