Integration Manual

for S32K1_S32M24X FEE Driver

Document Number: IM2FEEASR21.11 Rev0000R2.0.0 Rev. 1.0

Chapter 1

Revision History

Revision	Date	Author	Description
1.0	04.08.2023	NXP RTD Team	S32K1_S32M24X Real-Time Drivers AUTOSAR 4.4 & R21-11 Version 2.0.0

Chapter 2

Introduction

- Supported Derivatives

- Overview

- About This Manual

- Acronyms and Definitions

- Reference List

This integration manual describes the integration requirements for Flash EEPROM Emulation (FEE) Driver for S32K1_S32M24X 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 / MWCT1014S_lqfp64

- s32k144_lqfp100 / MWCT1014S_lqfp100

- s32k144_mapbga100

- s32k144w_lqfp48

- s32k144w_lqfp64

- s32k146_lqfp64

- s32k146_lqfp100 / MWCT1015S_lqfp100

- s32k146_mapbga100 / MWCT1015S_mapbga100

- s32k146_lqfp144

- s32k148_lqfp100

- s32k148_mapbga100 / MWCT1016S_mapbga100

- s32k148_lqfp144

- s32k148_lqfp176

- s32m241_lqfp64

- s32m242_lqfp64

- s32m243_lqfp64

- s32m244_lqfp64

All of the above microcontroller devices are collectively named as S32K1_S32M24X. Note: MWCT part numbers contain NXP confidential IP for Qi Wireless Power

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
ASM	Assembler
AUTOSAR	Automotive Open System Architecture
DET	Development Error Tracer
ECU	Electronic Control Unit
MCU	Micro Controller Unit
OS	Operating System
MSB	Most Significant Bit
N/A	Not Applicable
SWS	Software Specification
VLE	Variable Length Encoding
XML	Extensible Markup Language
ISR	Interrupt Service Routine
IVOR	Interrupt Vector Offset Register
ECC	Error Correcting Code
DFO	Data Flash Optimized
DW	Double Word
EEPROM	Electrically Erasable Programmable Read-Only Memory
MemAcc	Memory Access
FEE	Flash EEPROM Emulation
RTD	Real Time Drivers

2.5 Reference List

#	Title	Version
1	S32K1xx Series Reference Manual	Rev. 14, 09/2021
2	Errata S32K116_0N96V	Rev. 22/OCT/2021
3	Errata S32K118_0N97V	Rev. 22/OCT/2021
4	Errata S32K142_0N33V	Rev. 22/OCT/2021
5	Errata S32K144_0N57U	Rev. 22/OCT/2021
6	Errata S32K144W_0P64A	Rev. 22/OCT/2021
7	Errata S32K146_0N73V	Rev. 22/OCT/2021
8	Errata S32K148_0N20V	Rev. 22/OCT/2021
9	S32K1xx Data Sheet	Rev. 14, 08/2021
10	S32M24x Reference Manual	Rev. 2 Draft A, 05/2023
11	Errata S32M244_P64A+P73G	Rev. 0
12	Errata S32M242_N33V+P73G	Rev. 0, 06/2023
13	S32M2xx Data Sheet	Rev. 3 DraftA, 05/2023

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

The RTD driver files are compiled using:

- NXP GCC 10.2.0 20200723 (Build 1728 Revision g5963bc8)
- 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_T40D2M20I0R0 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 or S32M24x 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 or S32M24x 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 or S32M24x 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 don't 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	This option is only used to build test for generation Ram/↩ Stack size report. Makes the compiler output stack usage information for the program, on a per-function basis
-fdump-ipa-all	This option is only used to build test for generation Ram/↩ Stack size report. 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. S32↩ K148 can be replaced according to derivatives name S32K116,S32K118,S32K142,S32K142W,S32K144,S32↩ K144W,S32K146,S32K148,S32M244,S32M242.
-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 initialization in source file system.c under the Platform driver (for S32K14x or S32↩ M24x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initialization in source file system.c under the Platform driver (for S32K14x or S32M24x devices)
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPORT as a macro, with definition 1. Allows drivers to be configured in user mode.
–sysroot=	Specifies the path to the sysroot, for Cortex-M7 it is /arm-none-eabi/newlib
-specs=nano.specs	Use Newlib nano specs
-specs=nosys.specs	Do not use printf/scanf

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 or S32M24x devices)
-mcpu=cortex-m0plus	Targeted ARM processor for which GCC should tune the performance of the code (for S32K11x devices)
-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)
-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 or S32M24x 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 or S32M24x 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 or S32M24x 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

Linker Option	Description
–sysroot=	Specifies the path to the sysroot, for Cortex-M7 it is /arm-none-eabi/newlib
-specs=nano.specs	Use Newlib nano specs
-specs=nosys.specs	Do not use printf/scanf

3.1.2   IAR Compiler/Assembler/Linker Options

3.1.2.1   IAR Compiler Options

Compiler Option	Description
–cpu=Cortex-M4	Targeted ARM processor for which IAR should tune the performance of the code (for S32K14x or S32M24x 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 or S32M24x 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. S32↩K148 can be replaced according to derivatives name S32K116,S32K118,S32K142,S32K142W,S32K144,S32↩K144W,S32K146,S32K148,S32M244,S32M242.

Compiler Option	Description
-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.
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initialization in source file system.c under the Platform driver (for S32K14x or S32↩M24x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initialization in source file system.c under the Platform driver (for S32K14x or S32M24x devices)
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPORT as a macro, with definition 1. Allows drivers to be configured in user mode.

3.1.2.2 IAR Assembler Options

Assembler Option	Description
–cpu=Cortex-M4	Targeted ARM processor for which IAR should tune the performance of the code (for S32K14x or S32M24x 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)
–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

3.1.2.3 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 or S32M24x 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 or S32M24x 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)

Linker Option	Description
–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

3.1.3   GHS Compiler/Assembler/Linker Options

3.1.3.1   GHS Compiler Options

Compiler Option	Description
-cpu=cortexm4	Selects target processor: Arm Cortex M4 (for S32K14x or S32M24x 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 or S32M24x devices)
-fsingle	Use hardware single-precision, software double-precision FP instructions (for S32K14x or S32M24x devices)
-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

Compiler Option	Description
–unsigned_fields	Bitfields 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 and generate 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. S32↩K148 can be replaced according to derivatives name S32K116,S32K118,S32K142,S32K142W,S32K144,S32↩K144W,S32K146,S32K148,S32M244,S32M242.
-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 initialization in source file system.c under the Platform driver (for S32K14x or S32↩M24x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initialization in source file system.c under the Platform driver (for S32K14x or S32M24x devices)
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPORT as a macro, with definition 1. Allows drivers to be configured in user mode

3.1.3.2 GHS Assembler Options

Assembler Option	Description
-cpu=cortexm4	Selects target processor: Arm Cortex M4 (for S32K14x or S32M24x devices)
-cpu=cortexm0plus	Selects target processor: Arm Cortex M0+ (for S32K11x devices)
-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 S32↩K14x devices)
-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)
-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

Assembler Option	Description
-c	Stop after assembly and produce an object file for each source file

3.1.3.3   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. DWARF 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 the modules 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 or S32M24x 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
-nostartfiles	Controls the start files to be linked into the executable

3.2   Files required for compilation

This section describes the include files required to compile, assemble (if assembler code) and link the Fee driver for S32K1_S32M24X 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.

Fee Files

```
..\Fee_TS_T40D2M20I0R0\src\Fee.c
..\Fee_TS_T40D2M20I0R0\include\Fee_Cbk.h
..\Fee_TS_T40D2M20I0R0\include\Fee.h
..\Fee_TS_T40D2M20I0R0\include\Fee_InternalTypes.h
..\Fee_TS_T40D2M20I0R0\include\Fee_Types.h
Fee_Cfg.h - this file should be generated by the user using a configuration/generation tool
Fee_Cfg.c - this file should be generated by the user using a configuration/generation tool
Fee_Features.h - this file should be generated by the user using a configuration/generation tool
```

Other includes files:

Files from the MemIf folder:

```
..\MemIf_TS_T40D2M20I0R0\include\MemIf_Types.h
```

Files from the Base common folder

```
..\Base_TS_T40D2M20I0R0\include\Compiler.h
..\Base_TS_T40D2M20I0R0\include\Compiler_Cfg.h
..\Base_TS_T40D2M20I0R0\include\Fee_MemMap.h
..\Base_TS_T40D2M20I0R0\include\Platform_Types.h
..\Base_TS_T40D2M20I0R0\include\Mcal.h
..\Base_TS_T40D2M20I0R0\include\Std_Types.h
```

Files from the Det folder:

```
..\Det_TS_T40D2M20I0R0\include\Det.h
```

Files from the MemAcc folder:

```
..\MemAcc_TS_T40D2M20I0R0\include\MemAcc.h
```

Files from the Mem_43_EXFLS folder:

```
..\Mem_43_EXFLS_TS_T40D2M20I0R0\include\Mem_43_EXFLS.h
```

Files from the Mem_43_INFLS folder:

```
..\Mem_43_INFLS_TS_T40D2M20I0R0\include\Mem_43_INFLS.h
```

Files from the Mem_43_EEP folder:

```
..\Mem_43_EEP_TS_T40D2M20I0R0\include\Mem_43_EEP.h
```

Files from the Rte folder:

```
..\Rte_TS_T40D2M20I0R0\include\SchM_Fee.h
```

3.3   Setting up the plugins

The Fee driver was designed to be configured by using the EB tresos® (version 29.0.0 or later.)

VSMD (Vendor Specific Module Definition) file in EB tresos® XDM format:

```
Fee_TS_T40D2M20I0R0\config\Fee.xdm
```

VSMD (Vendor Specific Module Definition) file in AUTOSAR compliant EPD format:

```
Fee_TS_T40D2M20I0R0\autosar\Fee.epd
```

Code generation templates for pre-compile time configuration parameters:

```
Fee_TS_T40D2M20I0R0\generate\include\Fee_Cfg.h
Fee_TS_T40D2M20I0R0\generate\include\Fee_Features.h
Fee_TS_T40D2M20I0R0\generate\src\Fee_Cfg.c
```

Steps to generate the configuration:

1. Copy the module folders

   - Base_TS_T40D2M20I0R0
   - Det_TS_T40D2M20I0R0
   - EcuC_TS_T40D2M20I0R0
   - MemAcc_TS_T40D2M20I0R0
   - Mem_43_EXFLS_TS_T40D2M20I0R0
   - Mem_43_INFLS_TS_T40D2M20I0R0
   - Mem_43_EEP_TS_T40D2M20I0R0
   - Fee_TS_T40D2M20I0R0
   - MemIf_TS_T40D2M20I0R0
   - Os_TS_T40D2M20I0R0
   - Platform_TS_T40D2M20I0R0
   - Resource_TS_T40D2M20I0R0
   - Rte_TS_T40D2M20I0R0 into the EB tresos® **plugins** folder.

2. Set the desired output location folder for the generated sources and header files.

3. Use the EB tresos® GUI to modify ECU configuration parameters values.

4. Generate the configuration files.

Chapter 4

Function calls to module

- Function Calls during Start-up
- Function Calls during Shutdown
- Function Calls during Wake-up

4.1   Function Calls during Start-up

Fee shall be initialized during STARTUP2 phase of EcuM initialization. The API member to be called to accomplish this is Fee_Init.

Notes:

```
Fee module is the upper layer module which works on MemAcc module.
MemAcc_Init function must be called before calling the Fee_Init.
Fee_MainFunction and MemAcc_MainFunction routines must be called repeatedly for the FEE module
initialization and its operation.
When an operation (initialization or standard one) finishes, the Fee_GetStatus returns MEMIF_IDLE.
```

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, Fee is using the services of Schedule Manager (SchM) for entering and exiting the exclusive areas. The following critical regions are used in the Fee driver:

FEE_EXCLUSIVE_AREA_00 is used in function Fee_Read to protect the updates for:

- Fee_pJobDataDestPtr

- Fee_uJobBlockIndex

- Fee_uJobBlockOffset

- Fee_uJobBlockLength

- Fee_eModuleStatus

- Fee_eJobResult

- Fee_eJob

FEE__EXCLUSIVE__AREA__01 is used in function Fee_Write to protect the updates for:

- Fee_pJobDataDestPtr

- Fee_uJobBlockIndex

- Fee_eModuleStatus

- Fee_eJobResult

- Fee_eJob

FEE__EXCLUSIVE__AREA__02 is used in function Fee_InvalidateBlock to protect the updates for:

- Fee_uJobBlockIndex

- Fee_eModuleStatus

- Fee_eJobResult

- Fee_eJob

FEE__EXCLUSIVE__AREA__03 is used in function Fee_EraseImmediateBlock to protect the updates for:

- Fee_uJobBlockIndex

- Fee_eModuleStatus

- Fee_eJobResult

- Fee_eJob

The critical regions from interrupts are grouped in "Interrupt Service Routines Critical Regions (composed diagram)". If an exclusive area is "exclusive" with the composed "Interrupt Service Routines Critical Regions (composed diagram)" group, it means that it is exclusive with each one of the ISR critical regions.

- Critical Region Exclusive Matrix

5.1.1   Critical Region Exclusive Matrix

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

Table 5.1 Critical Region Exclusive Matrix

Exclusive Areas	FEE__EA__00	FEE__EA__01	FEE__EA__02	FEE__EA__03
FEE__EA__00		X	X	X
FEE__EA__01	X		X	X
FEE__EA__02	X	X		X
FEE__EA__03	X	X	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).

5.3 Peripheral Hardware Requirements

The FEE module is hardware independent module and depends on the underlying MemAcc module and its configuration only.

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:** The BASE module contains the common files/definitions needed by all RTD modules.

- **Det:** The DET module is used for enabling Development error detection. The API function used is Det_↩ ReportError(). The activation / deactivation of Development error detection is configurable using the 'Fee↩ DevErrorDetect' configuration parameter.

- **MemIf:** This module allows the NVRAM manager to access several memory abstraction modules.

- **Resource:** Resource module is used to select microcontroller's derivatives.

- **MemAcc:** The Memory access provides services for reading, writing and erasing flash memory and a configuration interface for modifying the write/erase protection for Fee if supported by the underlying hardware.

- **Mem_Exfls/Mem_Infls:** The Memory Driver provides services for reading, writing and erasing flash memory and a configuration interface for modifying the write/erase protection for MemAcc if supported by the underlying hardware.

- **EcuC:** The ECUC module is used for ECU configuration. RTD modules need ECUC to retrieve the variant information

- **Rte:** The RTE module is needed for implementing data consistency of exclusive areas that are used by FEE module.

5.7   Data Cache Restrictions

None

5.8   User Mode support

- User Mode config in the module

- User Mode configuration in AutosarOS

5.8.1   User Mode config in the module

No special measures need to be taken to run Fee module from user mode. The Fee 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 have 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 config 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.
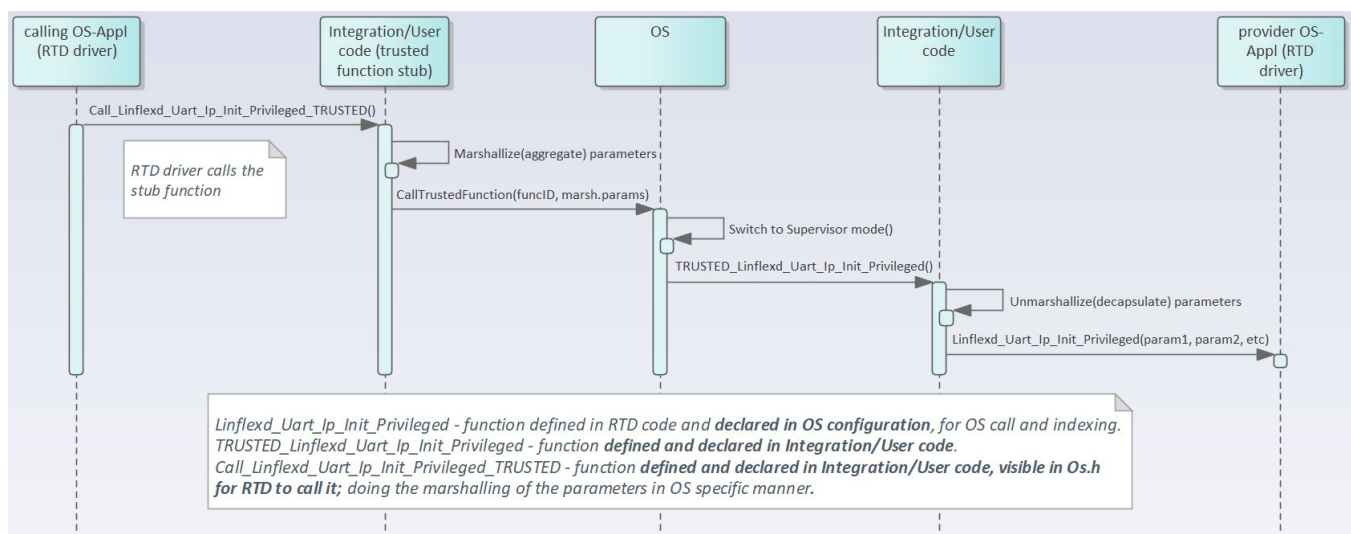
Figure 5.1 Example sequence chart for calling `Linflexd_Uart_Ip_Init_Privileged` as trusted function

5.9 Multicore support

- No implement multicore parallel in this release.

Chapter 6

Main API Requirements

- Main function calls within BSW scheduler
- API Requirements
- Calls to Notification Functions, Callbacks, Callouts

6.1 Main function calls within BSW scheduler

- Fee_MainFunction()
- MemAcc_MainFunction()

Call rate depends on target application, i.e. how fast the data must be read/written/compared.

6.2 API Requirements

Before calling the Fee_Write() function for immediate data, the function Fee_EraseImmediateBlock() must be called to pre-erase the flash area.

6.3 Calls to Notification Functions, Callbacks, Callouts

The Fee module provides user-configurable notifications

- FeeNvMJobEndNotification,
- FeeNvMJobErrorNotification,
- FeeClusterFormatNotification.

FeeNvMJobEndNotification and FeeNvMJobErrorNotification are usually routed to the NvM. FeeClusterFormat↩ Notification is called by Fee to inform the user in case a cluster format is triggered during the Fee initialization. Additionally, the Fee module publishes two APIs

- Fee_JobEndNotification: This callback notification is used by the underlying MemAcc module to report the succeeded/failed end of an MemAcc operation.

Both callbacks must be configured in the Fee module (notifications) regardless of its operation mode (synchronous or asynchronous).

Chapter 7

Memory allocation

- Sections to be defined in Fee_MemMap.h

- Linker command file

7.1   Sections to be defined in Fee_MemMap.h

Section name	Type of section	Description
FEE_START_SEC_CODE	Code	Start of memory Section for Code.
FEE_STOP_SEC_CODE	Code	End of above section.
FEE_START_SEC_VAR_CLEARED_8	Variables	Start of Memory Section for Variable 8 bits.These variables are cleared to zero by start-up code.
FEE_STOP_SEC_VAR_CLEARED_8	Variables	End of above section.
FEE_START_SEC_VAR_CLEARED_16	Variables	Start of Memory Section for Variable 16 bits.These variables are cleared to zero by start-up code.
FEE_STOP_SEC_VAR_CLEARED_16	Variables	End of above section.
FEE_START_SEC_VAR_CLEARED_32	Variables	Start of Memory Section for Variable 32 bits.These variables are cleared to zero by start-up code.
FEE_STOP_SEC_VAR_CLEARED_32	Variables	End of above section.
FEE_START_SEC_VAR_INIT_↩ 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
FEE_STOP_SEC_VAR_INIT_↩ UNSPECIFIED	Variables	End of above section.
FEE_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.

Section name	Type of section	Description
FEE_STOP_SEC_VAR_CLEARED_↵ UNSPECIFIED	Variables	End of above section.
FEE_START_SEC_VAR_CLEARED_↵ BOOLEAN	Variables	Start of memory Section for Variables with type boolean. These variables are cleared to zero by start-up code.
FEE_STOP_SEC_VAR_CLEARED_↵ BOOLEAN	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 FEE 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_00113	When RTD drivers are integrated with AutosarOS and User mode support is enabled, the integrator shall assure that the definition and declaration of all RTD functions needed to be called as trusted functions follow the naming convention Call<Function_Name>TRUSTED(parameter1,parameter2,...) in Integration/User code. They need to visible in Os.h for the driver to call them. They will call RTD <Function_Name>() as trusted functions in OS specific manner.