# Integration Manual

for S32K1_S32M24X MEM_43_INFLS Driver

Document Number: IM2MEM_43_INFLSASRR21-11 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

This integration manual describes the integration requirements for Mem_43_INFLS 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 |
| BSMI | Basic Software Make file Interface |
| CAN | Controller Area Network |
| C/CPP | C and C++ Source Code |
| CS | Chip Select |
| CTU | Cross Trigger Unit |
| DEM | Diagnostic Event Manager |
| DET | Development Error Tracer |
| DMA | Direct Memory Access |
| ECU | Electronic Control Unit |
| FIFO | First In First Out |
| LSB | Least Signifigant Bit |
| MCU | Micro Controller Unit |
| MIDE | Multi Integrated Development Environment |
| MSB | Most Significant Bit |
| N/A | Not Applicable |
| RAM | Random Access Memory |
| SIU | Systems Integration Unit |
| SWS | Software Specification |
| VLE | Variable Length Encoding |
| XML | Extensible Markup Language |

## 2.5 Reference List

| # | Title | Version |
|---|---|---|
| 1 | Specification of Mem Driver | AUTOSAR Release R21-11 |
| 2 | Reference Manual | S32K1xx Series Reference Manual, Rev. 14, 09/2021 |
| | | S32M24x Reference Manual, Rev. 2 Draft A, 05/2023 |
| 3 | Datasheet | S32K1xx Data Sheet, Rev. 14, 08/2021 |
| | | S32M2xx Data Sheet, Supports S32M24x and S32M27x, Rev. 3 Draft A, 05/2023 |
| 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 |
| | | S32M244_P64A+P73G, Rev. 0 |

# 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_SUPPO↩ RT 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 |
| –sysroot= | Specifies the path to the sysroot, for Cortex-M7 it is /arm-none-eabi/newlib |

| Linker Option | Description |
|---|---|
| -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. |
| -DIAR | Predefine IAR as a macro, with definition 1 |

| Compiler Option | Description |
|---|---|
| -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_SUPPO↩ RT 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_SUPPO↩RT 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. 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 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 and link the AUTOSAR Mem_43_INFLS driver for S32K1XX 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.

### 3.2.1 Mem_43_INFLS Files

- Mem_43_INFLS_TS_T40D2M20I0R0\include\Ftfc_Mem_InFls_Ip.h
- Mem_43_INFLS_TS_T40D2M20I0R0\include\Ftfc_Mem_InFls_Ip_Types.h
- Mem_43_INFLS_TS_T40D2M20I0R0\include\Ftfc_Mem_InFls_Ip_Ac.h
- Mem_43_INFLS_TS_T40D2M20I0R0\include\Ftfc_Mem_InFls_Ip_TrustedFunctions.h
- Mem_43_INFLS_TS_T40D2M20I0R0\include\Mem_43_INFLS.h
- Mem_43_INFLS_TS_T40D2M20I0R0\include\Mem_43_INFLS_Ipw.h
- Mem_43_INFLS_TS_T40D2M20I0R0\include\Mem_43_INFLS_Types.h
- Mem_43_INFLS_TS_T40D2M20I0R0\src\Ftfc_Mem_InFls_Ip.c
- Mem_43_INFLS_TS_T40D2M20I0R0\src\Ftfc_Mem_InFls_Ip_Ac.c
- Mem_43_INFLS_TS_T40D2M20I0R0\src\Mem_43_INFLS.c
- Mem_43_INFLS_TS_T40D2M20I0R0\src\Mem_43_INFLS_Ipw.c

Note

These files should be generated by the user using a configuration/generation tool

- Ftfc_Mem_InFls_Ip_Cfg.h
- Ftfc_Mem_InFls_Ip_Cfg.c
- Mem_43_INFLS_Cfg.h
- Mem_43_INFLS_CfgDefines.h
- Mem_43_INFLS_Cfg.c

### 3.2.2 Files from Base common folder

- BaseNXP_TS_T40D2M20I0R0\include\Compiler.h
- BaseNXP_TS_T40D2M20I0R0\include\Compiler_Cfg.h
- BaseNXP_TS_T40D2M20I0R0\include\ComStack_Types.h
- BaseNXP_TS_T40D2M20I0R0\include\Mem_43_INFLS_MemMap.h
- BaseNXP_TS_T40D2M20I0R0\include\Mcal.h
- BaseNXP_TS_T40D2M20I0R0\include\Platform_Types.h
- BaseNXP_TS_T40D2M20I0R0\include\Std_Types.h
- BaseNXP_TS_T40D2M20I0R0\include\Reg_eSys.h
- BaseNXP_TS_T40D2M20I0R0\include\Soc_Ips.h
- BaseNXP_TS_T40D2M20I0R0\include\Reg_Macros.h
- BaseNXP_TS_T40D2M20I0R0\include\SilRegMacros.h

### 3.2.3 Files from Det folder

- Det_TS_T40D2M20I0R0\include\Det.h

### 3.2.4 Files from Rte folder

- Rte_TS_T40D2M20I0R0\include\SchM_Mem_43_INFLS.h

### 3.2.5 Files from Mcl folder

- Mcl_TS_T40D2M20I0R0\include\Mcl.h

## 3.3 Setting up the plugins

The Mem_43_INFLS driver was designed to be configured by using the EB Tresos Studio (version 29.0.0 b220329-0119 or later.)

### 3.3.1 Location of various files inside the Mem_43_INFLS module folder

- VSMD (Vendor Specific Module Definition) file in EB Tresos Studio XDM format:
  - Mem_43_INFLS_TS_T40D2M20I0R0\config\Mem_43_INFLS.xdm
- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:
  - Mem_43_INFLS_TS_T40D2M20I0R0\autosar\Mem_43_INFLS_<subderivative_name>.epd
- Code Generation Templates for parameters without variation points:
  - Mem_43_INFLS_TS_T40D2M20I0R0\generate_PC\include\Ftfc_Mem_InFls_Ip_Cfg.h
  - Mem_43_INFLS_TS_T40D2M20I0R0\generate_PC\include\Mem_43_INFLS_Cfg.h
  - Mem_43_INFLS_TS_T40D2M20I0R0\generate_PC\include\Mem_43_INFLS_CfgDefines.h
  - Mem_43_INFLS_TS_T40D2M20I0R0\generate_PC\src\Mem_43_INFLS_Cfg.c

### 3.3.2 Steps to generate the configuration:

1. Copy the module folders:
   - BaseNXP_TS_T40D2M20I0R0
   - Det_TS_T40D2M20I0R0
   - EcuC_TS_T40D2M20I0R0
   - Mem_43_INFLS_TS_T40D2M20I0R0
   - Platform_TS_T40D2M20I0R0
   - Resource_TS_T40D2M20I0R0
   - Rte_TS_T40D2M20I0R0
   - Mcl_TS_T40D2M20I0R0 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

The Mem_43_INFLS driver shall be initialized during STARTUP phase of EcuM initialization. The API to be called for this is Mem_43_INFLS_Init().

The MCU module should be initialized before the FLS is initialized.

If the Mem_43_INFLS driver is used in User Mode, be sure that the Flash memory controller registers are accessible and that accessed Flash 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 Flash memory physical sectors that are going to be modified by Mem_43_INFLS driver (i.e. erase and write operations) have to be unprotected for a successful operation.

## 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, Mem_43_INFLS is using the services of Schedule Manager (SchM) for entering and exiting the exclusive areas. The following critical regions are used in the Mem_43_INFLS driver:

**MEM_EXCLUSIVE_AREA_01** is used in function Mem_43_INFLS_Erase to protect the updates for:

- Mem_43_INFLS_eJobRuntimeInfo

**MEM_EXCLUSIVE_AREA_02** is used in function Mem_43_INFLS_Write to protect the updates for:

- Mem_43_INFLS_eJobRuntimeInfo

**MEM_EXCLUSIVE_AREA_03** is used in function Mem_43_INFLS_Read to protect the updates for:

- Mem_43_INFLS_eJobRuntimeInfo

**MEM_EXCLUSIVE_AREA_04** is used in function Mem_43_INFLS_BlankCheck to protect the updates for:

- Mem_43_INFLS_eJobRuntimeInfo

**MEM_EXCLUSIVE_AREA_05** is used in function Mem_43_INFLS_HwSpecificService to protect the updates for:

- Mem_43_INFLS_eJobRuntimeInfo

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.

### 5.1.1 Critical Region Exclusive Matrix

- Below is the table depicting the exclusivity between different critical region IDs from the Mem_43_INFLS driver .
- If there is an "X" in a table, it means that those 2 critical regions cannot interrupt each other.

| MEM_EXCLUSIVE_AREA | AREA_01 | AREA_02 | AREA_03 | AREA_04 | AREA_05 |
|---|---|---|---|---|---|
| AREA_01 | | X | X | X | X |
| AREA_02 | X | | X | X | X |
| AREA_03 | X | X | | X | X |
| AREA_04 | X | X | X | | X |
| AREA_05 | X | X | X | X | |

## 5.2 Exclusive areas not available on this platform

None.

## 5.3 Peripheral Hardware Requirements

The Mem_43_INFLS driver uses the internal flash memory peripheral (FTFC). For more details about peripherals and their structure refer to the 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.

## 5.4 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 are Det_↩ ReportError() or Det_ReportRuntimeError() or Det_ReportTransientFault(). The activation/deactivation of Development error detection is configurable using the "MemDevErrorDetect" configuration parameter.
- **Rte**: The RTE module is needed for implementing data consistency of exclusive areas that are used by Mem↩ _43_INFLS module.
- **Resource**: Resource module is used to select microcontroller's derivatives.
- **EcuC**: The ECUC module is used for ECU configuration. RTD modules need ECUC to retrieve the variant information.
- **Mcl**: This module provides service for Cache operation.

## 5.7 Data Cache Restrictions

The Mem_43_INFLS driver needs to maintain the memory coherency by means of three methods:

1. Disable data cache
2. Configure the flash region upon which the driver operates, as non-cacheable
3. Enable the MemSynchronizeCache feature

Depending on the application configuration and requirements, one option may be more beneficial than other.

If MemSynchronizeCache parameter is enabled in the configuration, then the FLS driver will call Mcl cache A↩ PI functions in order to invalidate the cache after each high voltage operation (write, erase) and before each read operation in order to ensure that the cache and the modified flash memory are in sync. The driver will attempt to invalidate only the modified lines from the cache. If the size of the region to be invalidated is greater than half of the cache size, then the entire cache is invalidated.

If MemSynchronizeCache parameter is disabled, the upper layers have to ensure that the flash region upon which the driver operates is not cached. This can be obtained by either disabling the data cache or by configuring the memory region as non-cacheable.

The cache settings apply to internal flash operations.

Note:

- Failure to properly inhibit or disable data cache on the flash sectors which the driver operates on, will most likely lead to flash jobs failing. This situation is more likely to be visible when all the check functionality (Fls Erase Blank Check, Fls Write Blank Check, Fls Write Verify Check) is enabled, because more read/program sequences occur and the cache is less likely to be self cleared.
- On S32K148 derivative, the cache region allocated to D-Flash memory space has to be set as cache-inhibit, otherwise any access to that space could lead to cache corruption. The FLS driver will check and only invalidate cache for P-Flash sectors, the D-Flash ones will not be touched.
- Additionally, PCCRMR[R2] should be programmed as 00b (NonCacheable) as S32K148 FlexNVMs region is not cacheable. The device may show unexpected behavior when cache tries to access FlexNVM region. An example can be found in the examples of S32K148. For more information, please refer to the LMEM chapter in the Reference manual

## 5.8 User Mode support

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

### 5.8.1 User Mode configuration in the module

The Mem_43_INFLS can be run in user mode if the following steps are performed:

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

| Function syntax | Description | Available via |
|---|---|---|
| void Ftfc_Mem_InFls_Ip_InvalidPrefetchBuff_Ram(void) | Invalidate prefetch buffer before reading to make sure that the driver always reads the new data from flash | Ftfc_Fls_Ip_TrustedFunctions.h |

### 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 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,...)
```

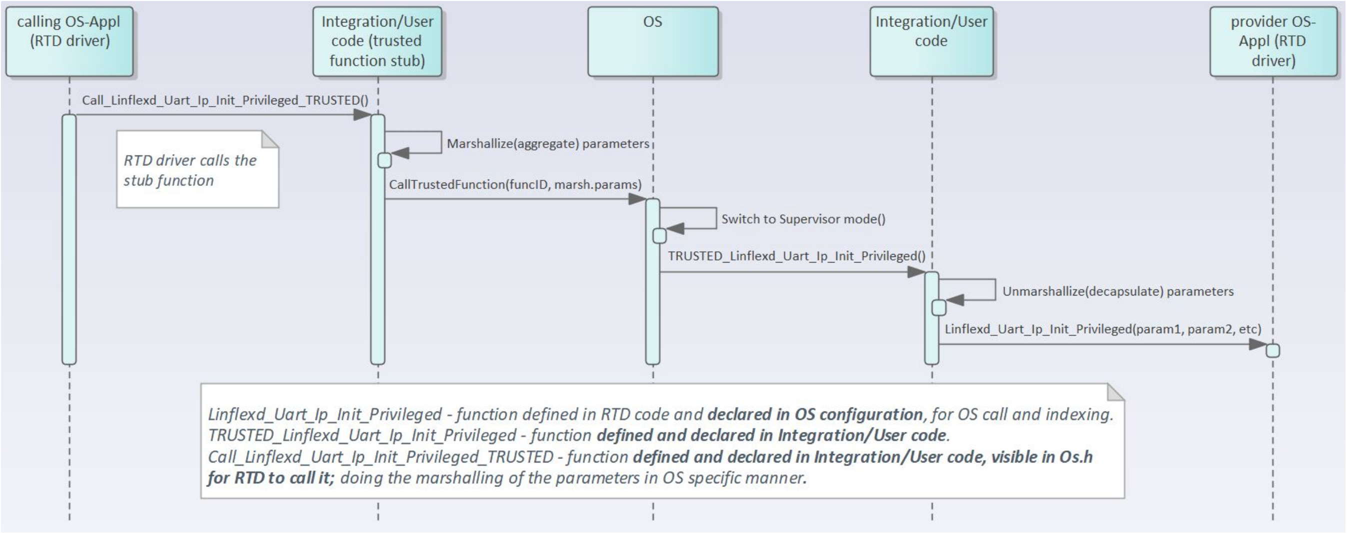
That is the result of macro expansion `OsIf_Trusted_Call` in driver code:

```
#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.


calling OS-Appl (RTD driver)
Integration/User code (trusted function stub)
OS
Integration/User code
provider OS-Appl (RTD driver)
Call_Linflexd_Uart_Ip_Init_Privileged_TRUSTED()
RTD driver calls the stub function
Marshallize(aggregate) parameters
CallTrustedFunction(funcID, marsh.params)
Switch to Supervisor mode()
TRUSTED_Linflexd_Uart_Ip_Init_Privileged()
Unmarshallize(decapsulate) parameters
Linflexd_Uart_Ip_Init_Privileged(param1, param2, etc)

*Linflexd_Uart_Ip_Init_Privileged - function defined in RTD code and **declared in OS configuration**, for OS call and indexing.*
*TRUSTED_Linflexd_Uart_Ip_Init_Privileged - function **defined and declared in Integration/User code**.*
*Call_Linflexd_Uart_Ip_Init_Privileged_TRUSTED - function **defined and declared in Integration/User code, visible in Os.h for RTD to call it;** doing the marshalling of the parameters in OS specific manner.*

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

## 5.9 Multicore support

The Mem_43_INFLS driver does not support Multicore.

# Chapter 6

# Main API Requirements

## 6.1 Main function calls within BSW scheduler

None.

## 6.2 API Requirements

None.

## 6.3 Calls to Notification Functions, Callbacks, Callouts

The Mem_43_INFLS driver provides notifications that are user configurable:

| Notification | Usage |
|---|---|
| MemAcCallback | Usually routed to Wdg module |
| MemStartFlashAccessNotif | Mark the start of a flash read, program access |
| MemFinishedFlashAccessNotif | Mark the end of a flash read, program access |

- In case the Wdg (watchdog) counter has been enabled before, but the erase/write process takes a long time leading to wdg reset (reset cpu). To prevent this, use "MemAcCallback" to reset the Wdg counter

# Chapter 7

# Memory allocation

## 7.1 Sections to be defined in Mem_43_INFLS_MemMap.h

| Index | Section name | Type of section | Description |
|---|---|---|---|
| 1 | MEM_43_INFLS_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 |
| | MEM_43_INFLS_STOP_SEC_CONFIG_DATA_UNSPECIFIED | | End of above section. |
| 2 | MEM_43_INFLS_START_SEC_CONST_UNSPECIFIED | Constant | The parameters that are not variant aware shall be stored in memory section for constants. |
| | MEM_43_INFLS_STOP_SEC_CONST_UNSPECIFIED | | End of above section. |
| 3 | MEM_43_INFLS_START_SEC_CODE | Code | Start of memory Section for Code. |
| | MEM_43_INFLS_STOP_SEC_CODE | | End of above section. |
| 4 | MEM_43_INFLS_START_SEC_CODE_AC | Code | Start of memory section for Code placed in a specific linker section. |
| | MEM_43_INFLS_STOP_SEC_CODE_AC | | End of above section. |
| 5 | MEM_43_INFLS_START_SEC_VAR_CLEARED_8 | Variables | Start of Memory Section for Variable 8 bits. These variables are cleared to zero by start-up code. |
| | MEM_43_INFLS_STOP_SEC_VAR_CLEARED_8 | | End of above section. |

| Index | Section name | Type of section | Description |
|---|---|---|---|
| 6 | MEM_43_INFLS_START_SEC_VAR_CLEARED_16 | Variables | Start of Memory Section for Variable 16 bits. These variables are cleared to zero by start-up code. |
| | MEM_43_INFLS_STOP_SEC_VAR_CLEARED_16 | | End of above section. |
| 7 | MEM_43_INFLS_START_SEC_VAR_CLEARED_32 | Variables | Start of Memory Section for Variable 32 bits. These variables are cleared to zero by start-up code. |
| | MEM_43_INFLS_STOP_SEC_VAR_CLEARED_32 | | End of above section. |
| 8 | MEM_43_INFLS_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. |
| | MEM_43_INFLS_STOP_SEC_VAR_CLEARED_UNSPECIFIED | | End of above section. |
| 9 | MEM_43_INFLS_START_SEC_VAR_INIT_BOOLEAN | Variables | Start of memory Section for Variables with type boolean. |
| | MEM_43_INFLS_STOP_SEC_VAR_INIT_BOOLEAN | | End of above section. |
| 10 | MEM_43_INFLS_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 |
| | MEM_43_INFLS_STOP_SEC_VAR_INIT_UNSPECIFIED | | 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 MEM_INFLS driver into the application.

| External Assumption Req ID | External Assumption Text |
|---|---|
| SWS_Mem_00039 | If built as a separate image, the Mem driver shall be completely self contained, i.e. it must not call any library or any other external functions. Note: Image feature |
| SWS_Mem_00038 | The Mem driver shall provide two ways for the service function invocation: - Direct service invocation - Indirect service invocation by a function pointer table Note: Image feature |
| SWS_Mem_00040 | Since Mem drivers are always hardware/CPU specific, the byte order of data fields and address information within the Mem driver binary shall follow the standard CPU byte order. Note: Image feature |
| SWS_Mem_00041 | Offset [bytes]: Size [bytes]: Name: Description 0: 8: Unique ID: Mem driver unique identifier - used to validate Mem driver version information, etc. 8: 8: Flags: Flags used for additional development error detection. 16: 4/8: Header address: Start address of Mem driver image header - used to verify consistency of Mem driver RAM buffer/ROM image location. 20/24: 4/8: Delimiter address: Address of Mem driver binary image delimiter pattern - used to validate if the binary is complete. Note: Image feature |
| SWS_Mem_00042 | Offset [bytes]: Size [bytes]: Name: Description 0: 2: ABI version: BCD-encoded Mem driver binary interface version. Any change of the interface version will also require an update of the MemAcc module. 2: 2: Vendor ID: Standard AUTOSAR vendor identification. 4: 4: Driver ID: Vendor specific driver identification. Note: Image feature |
| SWS_Mem_00043 | The ABI version of a Mem driver following this specification shall be 0001. Note: Image feature |
| SWS_Mem_00044 | The header address is used for development error checks to verify the consistency of the linked Mem driver binary image with the location of the RAM buffer which is used for execution of the Mem driver. In case of a relocatable/position independent Mem driver binary, the header address shall be set to zero, otherwise, the header address shall hold the physical start address of the Mem driver binary. Note: Image feature |

| External Assumption Req ID | External Assumption Text |
|---|---|
| SWS_Mem_00045 | The flag part of the Mem driver header is a bit-field which holds additional information for develop error checks. Offset [bits]: Size [bits]: Name: Description 0: 1: Relocatable binary: If this bit is set, the Mem driver binary is relocatable and no address consistency checks can be done. 1↩ : 31: Reserved: Reserved by this specification - shall be 0. 32: 32: Vendor specific: Vendor specific flags. Note: Image feature |
| SWS_Mem_00046 | The delimiter address part of the Mem driver header is used for development error checks to verify that the Mem driver binary is complete by checking the delimiter pattern linked to the end of the Mem driver binary. In case of a relocatable/position independent Mem driver binary, the delimiter address shall be set to zero, otherwise, the delimiter address shall hold the physical address of the delimiter pattern. Note: Image feature |
| SWS_Mem_00073 | The function pointer table is a standardized structure used to reference the Mem driver service functions. Entry: Name: Description 1: Init service pointer: Function pointer to Mem driver Init service. 2: DeInit service pointer: Function pointer to Mem driver DeInit service. 3: MainFunction service pointer: Function pointer to Mem driver MainFunction service. 4: GetJobResult service pointer: Function pointer to Mem driver GetJob↩ Result service. 5: Read service pointer: Function pointer to Mem driver Read service. 6: Write service pointer: Function pointer to Mem driver Write service. 7: Erase service pointer: Function pointer to Mem driver Erase service. 8: PropagateError service pointer: Function pointer to Mem driver PropagateError service. 9: BlankCheck service pointer: Function pointer to Mem driver BlankCheck service. 10: Suspend service pointer↩ : Function pointer to Mem driver Suspend service. 11: Resume service pointer: Function pointer to Mem driver Resume service. 12: HwSpecific↩ Service service pointer: Function pointer to Mem driver HwSpecificService service. Note: Image feature |
| SWS_Mem_00048 | The size of the Mem driver function pointers shall be machine/CPU specific. |
| SWS_Mem_00051 | The value of the delimiter field shall be the ones' complement of the unique identifier value. Note: Image feature |
| SWS_Mem_10020 | Module: Header File: Imported Type MemAcc: MemAcc.h: MemAcc↩ _AddressType (draft) Std: Std_Types.h: Std_ReturnType Std_Types.h: Std_VersionInfoType Note: |

EA_RTD_00071 | If interrupts are locked, a centralized function pair to lock and unlock interrupts shall be used. EA_RTD_00081 | The integrator shall assure that <MSN>_Init() and <MSN>_DeInit() functions do not interrupt each other. 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.

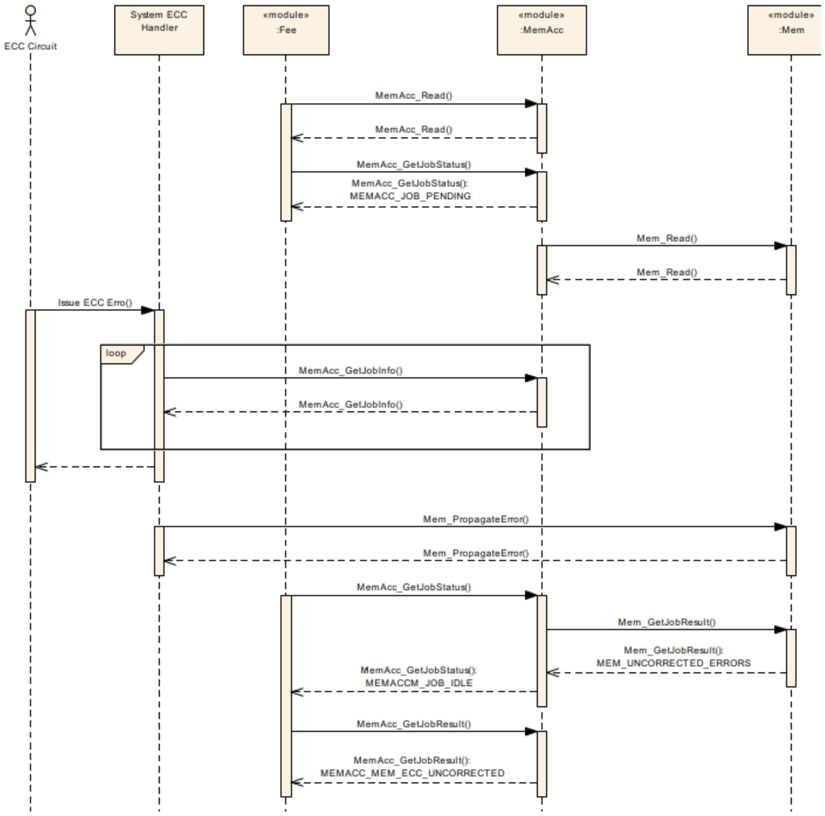
EA_RTD_00106 | Standalone IP configuration and HL configuration of the same driver shall be done in the same project EA_RTD_00107 | The integrator shall use the IP interface only for hardware resources that were configured for standalone IP usage. Note:� The integrator shall not directly use the IP interface 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 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.

# Chapter 10

# ECC Management on Internal Flash

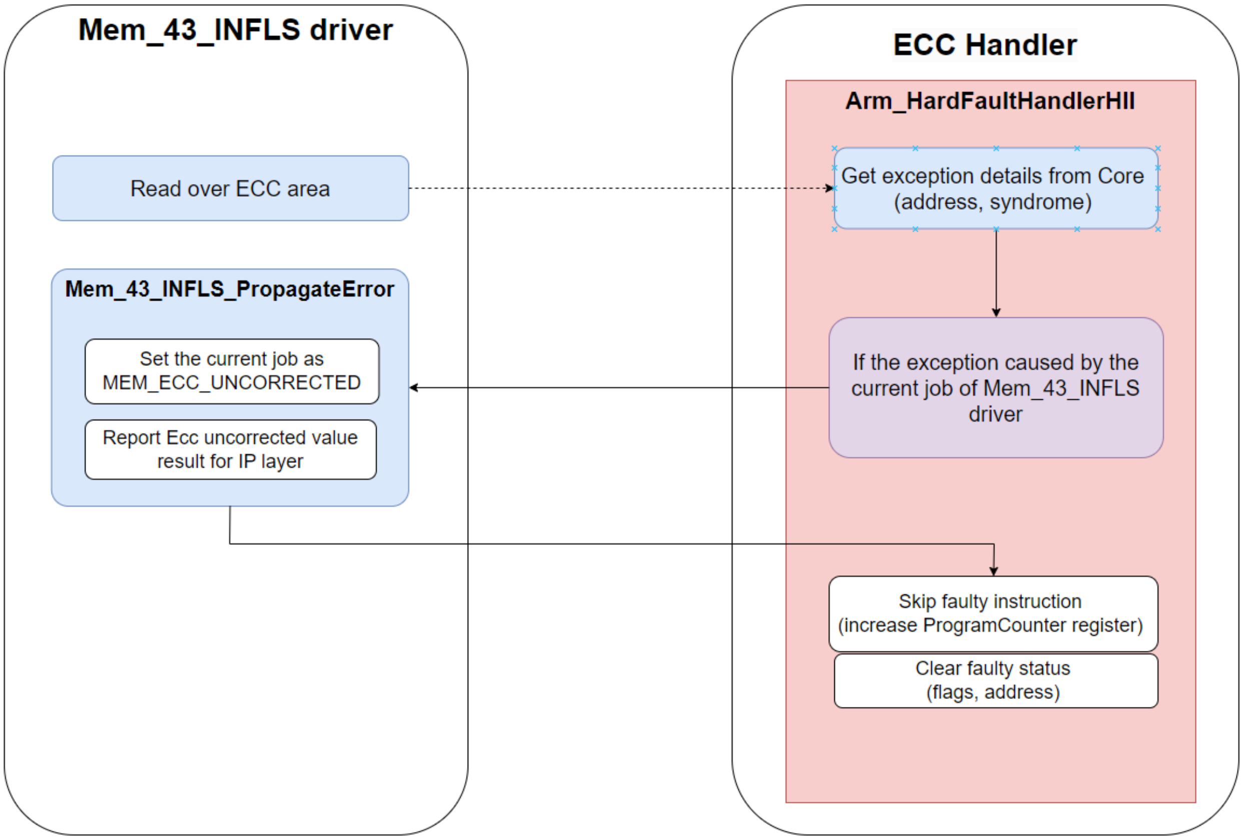
## 10.1 Solution recover from the exception by manually incrementing the program counter (PC register)

- While reading from the Flash, if an ECC exception occurs, a HardFaultHandler will be raised, where current instruction is skipped and the ECC exception will be confirmed.
- A basic flow and an implementation idea for the fault handler is depicted below.



**ECC Handling Example Sequence**

- Below is the actual flow of ECC handling:


Mem_43_INFLS driver
Read over ECC area
Mem_43_INFLS_PropagateError
Set the current job as MEM_ECC_UNCORRECTED
Report Ecc uncorrected value result for IP layer
ECC Handler
Arm_HardFaultHandlerHll
Get exception details from Core (address, syndrome)
If the exception caused by the current job of Mem_43_INFLS driver
Skip faulty instruction (increase ProgramCounter register)
Clear faulty status (flags, address)

1. When Mem_43_INFLS driver read/compare over an ECC erase, an ECC exception occurs, the **Arm_HardFaultHandlerHll** (**EccHandler**) will be raised

2. **Arm_HardFaultHandlerHll** gets some information about the exception:

   - The instruction that generated ECC from **PC** register
   - The data address that caused ECC from **BFAR** register
   - The exception syndrome from **CFSR** register
   - If the exception caused by the current job of Mem_43_INFLS driver,
     - Call Mem_43_INFLS_PropagateError function to report ECC error.
     - It will mark the job as MEM_ECC_UNCORRECTED
   - If not, it will do nothing

3. Based evaluation result, the following recovery strategies may be implemented by the **EccHandler**

   - Skip the instruction that caused the ECC
   - Perform a controlled shutdown of current activity
   - Do nothing (infinite loop), etc

- File: Vector_core.s

```
.section ".intc_vector","ax"
.align 2
.thumb
.globl undefined_handler
.globl undefined_handler
.globl VTABLE
.globl __Stack_start_c0          /* Top of Stack for Initial Stack Pointer */
.globl Reset_Handler             /* Reset Handler */
.globl NMI_Handler               /* NMI Handler */
.globl Arm_FaultHandlerThumb     /* Hard Fault Handler */
.globl MemManage_Handler         /* Reserved */
.globl Arm_FaultHandlerThumb     /* Bus Fault Handler */
.globl UsageFault_Handler        /* Usage Fault Handler */
.globl SVC_Handler               /* SVCall Handler */
.globl DebugMon_Handler          /* Debug Monitor Handler */
.globl PendSV_Handler            /* PendSV Handler */
.globl SysTick_Handler           /* SysTick Handler */ /* 15*/
```

- File: **Arm_FaultHandlerThumb.s**

```
.globl  Arm_FaultHandlerThumb
/*
Step 1: Detect if application is using MSP or PSP
Step 2: Pointer in r0 is provided as an parameter (Arm_HardFaultHandlerHll)to the HLL function
*/
Arm_FaultHandlerThumb:
mov    r0,r14           /* r0 = EXC_RETURN; */
mov    r1,#0x4          /* r1 = 0x4; */
and    r0,r1            /* EXC_RETURN & 0x4; */
beq    label_msp_stack  /* if (EXC_RETURN & 0x4) r0=PSP else r0=MSP; */
mrs    r0,PSP           /* r0 = PSP; (PSP stack used) */
b      label_end_stack
label_msp_stack:
mrs    r0,MSP           /* r0 = MSP; (MSP stack used) */
label_end_stack:
/* Pointer in r0 is provided as an parameter to the HLL function */
add r0,#0x18
/* NOTE: HLL function is called by pure branch (b) without link (bl).
This will cause that, upon HLL exiting, the execution will continue directly
from the location pointed by the address provided in r0. */
LDR   R3,=Arm_HardFaultHandlerHll
bx    R3
```

- File: **Arm_FaultHandlerHll.c**

```
/* check data address */
#define BFAR_ADDR 0xE000ED38
/* check syndrome */
#define CFSR_ADDR 0xE000ED28
void Arm_HardFaultHandlerHll(InstructionAddressType *instr_pt2pt)
{
ExceptionDetailsType  excDetails;
volatile InstructionAddressType instr_pt;
DataAddressType data_pt;
uint32 syndrome;
/* The instruction opcode(or the first 16 bits) value, stored in memory,
for the instruction which caused the fault
*/
uint16 instrOpcode;
/* Size of the instruction opcode stored in memory, 2 or 4 bytes */
uint8 thumbInstrSize;
instr_pt = *instr_pt2pt;
data_pt  = (void const *)(*((uint32 *)BFAR_ADDR));
syndrome = *((uint32 *)CFSR_ADDR);
/* Compute the instruction opcode size for the instruction which caused the hardfault.
The value will be used to compute the address of the following instruction
*/
instrOpcode = *((uint16 *)(*instr_pt2pt));
/* Compute the size of the instruction which caused the fault */
if (((instrOpcode & 0xE800) == 0xE800) || /* 0b11101x... */
((instrOpcode & 0xF000) == 0xF000) || /* 0b11110x... */
((instrOpcode & 0xF800) == 0xF800))   /* 0b11111x... */
{
/* Instruction size is 32 bits, 4 bytes */
thumbInstrSize = 4;
}
else
{
/* Instruction size is 16 bits, 2 bytes */
thumbInstrSize = 2;
}
excDetails.instruction_pt = instr_pt;
excDetails.data_pt        = data_pt;
excDetails.syndrome_u32   = syndrome;
if (FTFC_DSI_EXC_SYNDROME == (excDetails.syndrome_u32 & FTFC_DSI_EXC_SYNDROME))
{
Mem_43_INFLS_PropagateError(MEM_43_INFLS_INSTANCE_0_ID);
/* Exception was handled by one of the functions called above,
continue execution, skipping the causing instruction
In the test code we assume that the exception was caused by 16-bit/32-bit
load Thumb instruction => increment return address by the size of the instruction
*/
instr_pt2pt = instr_pt + thumbInstrSize;
/* clear the flags and address register */
*((volatile uint32 *)CFSR_ADDR) = *((volatile uint32 *)CFSR_ADDR);
*((uint32 *)BFAR_ADDR) = 0x0;
}
}
```

***How to Reach Us:***

**Home Page:**
nxp.com

**Web Support:**
nxp.com/support