Integration Manual

for S32K3 MEM_EXFLS Driver

Document Number: IM34MEM_EXFLS ASRR21-11 Rev0000R3.0.0 Rev. 1.0

Chapter 1

Revision History

Revision	Date	Author	Description
1.0	31.03.2023	NXP RTD Team	S32K3 Real-Time Drivers AUTOSAR 4.4 & R21-11 Version 3.0.0

Chapter 2

Introduction

This integration manual describes the integration requirements for Mem_43_ExFls driver for S32K3XX microcontrollers.

AUTOSAR Flash Driver configuration parameters description can be found in the configuration_parameters section. Deviations from the specification are described in the additional_requirements section.

2.1 Supported Derivatives

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

- s32k310_mqfp100
- s32k310_lqfp48
- s32k311_mqfp100 / MWCT2015S_mqfp100
- s32k311_lqfp48
- s32k312_mqfp100 / MWCT2016S_mqfp100
- s32k312_mqfp172 / MWCT2016S_mqfp172
- s32k314_mqfp172
- s32k314_mapbga257
- s32k322_mqfp100 / MWCT2D16S_mqfp100

- s32k322_mqfp172 / MWCT2D16S_mqfp172
- s32k324_mqfp172 / MWCT2D17S_mqfp172
- s32k324_mapbga257
- s32k341_mqfp100
- s32k341_mqfp172
- s32k342_mqfp100
- s32k342_mqfp172
- s32k344_mqfp172
- s32k344_mapbga257
- s32k394_mapbga289
- s32k396_mapbga289
- s32k358_mqfp172
- s32k358_mapbga289
- s32k328_mqfp172
- s32k328_mapbga289
- s32k338_mqfp172
- s32k338_mapbga289
- s32k348_mqfp172
- s32k348_mapbga289
- s32m274_lqfp64
- s32m276_lqfp64

All of the above microcontroller devices are collectively named as S32K3.

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
AUTOSAR	Automotive Open System Architecture
DET	Default Error Tracer
ECC	Error Correcting Code
VLE	Variable Length Encoding
N/A	Not Available
MCU	Microcontroller Unit
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
FEE	Flash EEPROM Emulation
FLS	Flash
RTD	Real Time Drivers
XML	Extensible Markup Language

2.5 Reference List

#	Title	Version
1	Specification of MEM Driver	AUTOSAR Release R21-11
2	Reference Manual	S32K3xx Reference Manual, Rev.6, Draft B, 01/2023
		S32K39 and S32K37 Reference Manual, Rev. 2 Draft A, 11/2022
3	Datasheet	S32K3xx Data Sheet, Rev. 6, 11/2022
		S32K396 Data Sheet, Rev. 1.1 — 08/2022
4	Errata	S32K358_0P14E Mask Set Errata — Rev. 28, 9/2022
		S32K396_0P40E Mask Set Errata, Rev. DEC2022, 12/2022
		S32K311_0P98C Mask Set Errata, Rev. 6/March/2023, 3/2023
		S32K312: Mask Set Errata for Mask 0P09C, Rev. 25/April/2022
		S32K342: Mask Set Errata for Mask 0P97C, Rev. 10, 11/2022
		S32K3x4: Mask Set Errata for Mask 0P55A/1P55A, Rev. 14/Oct/2022

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

The RTD driver files are compiled using:

- NXP GCC 10.2.0 20200723 (Build 1728 Revision g5963bc8)
- Wind River Diab Compiler 7.0.4
- Compiler Versions: Green Hills Multi 7.1.6d / Compiler 2021.1.4
- Compiler Versions: IAR ANSI C/C++ Compiler V8.50.10 (safety version)

The compiler, assembler, and linker flags used for building the driver are explained below.

The TS_T40D34M30I0R0 part of the plugin name is composed as follows:

- T = Target_Id (e.g. T40 identifies Cortex-M architecture)
- D = Derivative_Id (e.g. D34 identifies S32K3 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-m7	Targeted ARM processor for which GCC should tune the performance of the code
-mthumb	Generates code that executes in Thumb state
-mlittle-endian	Generate code for a processor running in little-endian mode
-mfpu=fpv5-sp-d16	Specifies the floating-point hardware available on the target
-mfloat-abi=hard	Specifies the floating-point ABI to use. "hard" allows generation of floating-point instructions and uses FPU-specific calling conventions
-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.
-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

Compiler Option	Description
-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
-DS32K3XX	Predefine S32K3XX as a macro, with definition 1
-D $ (DERIVATIVE)	Predefine S32K3's derivative as a macro, with definition 1. For example: Predefine for S32K344 will be -DS32K344.
-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
-DD_CACHE_ENABLE	Predefine D_CACHE_ENABLE as a macro, with definition 1. Enables data cache initalization in source file system.↩ c under the Platform driver
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initalization in source file system.c under the Platform driver
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver
-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=cortexm7	Targeted ARM processor for which GCC should tune the performance of the code
-mfpu=fpv5-sp-d16	Specifies the floating-point hardware available on the target
-mfloat-abi=hard	Specifies the floating-point ABI to use. "hard" allows generation of floating-point instructions and uses FPU-specific calling conventions
-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=cortexm7	Targeted ARM processor for which GCC should tune the performance of the code
-mthumb	Generates code that executes in Thumb state
-mfpu=fpv5-sp-d16	Specifies the floating-point hardware available on the target
-mfloat-abi=hard	Specifies the floating-point ABI to use. "hard" allows generation of floating-point instructions and uses FPU-specific calling conventions
-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
-specs=nano.specs	Use Newlib nano specs
-specs=nosys.specs	Do not use printf/scanf

3.1.2 DIAB Compiler/Assembler/Linker Options

3.1.2.1 DIAB Compiler Options

Compiler Option	Description
-tARMCORTEXM7MG:simple	Selects target processor (hardware single-precision, software double-precision floating-point)
-mthumb	Selects generating code that executes in Thumb state
-std=c99	Follows the C99 standard for C
-Oz	Like -O2 with further optimizations to reduce code size
-g	Generates DWARF 4.0 debug information
-fstandalone-debug	Emits full debug info for all types used by the program
-Wstrict-prototypes	Warn if a function is declared or defined without specifying the argument types
-Wsign-compare	Produce warnings when comparing signed type with unsigned type
-Wdouble-promotion	Give a warning when a value of type float is implicitly promoted to double
-Wunknown-pragmas	Issues a warning for unknown pragmas
-Wundef	Warns if an undefined identifier is evaluated in an #if directive. Such identifiers are replaced with zero

Compiler Option	Description
-Wextra	Enables some extra warning flags that are not enabled by '-Wall'
-Wall	Enables all of the most useful warnings (for historical reasons this option does not literally enable all warnings)
-pedantic	Emits a warning whenever the standard specified by the -std option requires a diagnostic
-Werror=implicit-function-declaration	Generates an error whenever a function is used before being declared
-fno-common	Compile common globals like normal definitions
-fno-signed-char	Char is unsigned
-fno-trigraphs	Do not process trigraph sequences
-V	Displays the current version number of the tool suite
-c	Stop after assembly and produce an object file for each source file
-DS32K3XX	Predefine S32K3XX as a macro, with definition 1
-D $ (DERIVATIVE)	Predefine S32K3's derivative as a macro, with definition 1
-DDIAB	Predefine DIAB 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
-DD_CACHE_ENABLE	Predefine D_CACHE_ENABLE as a macro, with definition 1. Enables data cache initalization in source file system.↩ c under the Platform driver
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initalization in source file system.c under the Platform driver
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver
-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 DIAB Assembler Options

Assembler Option	Description
-mthumb	Selects generating code that executes in Thumb state
-Xpreprocess-assembly	Invokes C preprocessor on assembly files before running the assembler
-Xassembly-listing	Produces an .lst assembly listing file
-c	Stop after assembly and produce an object file for each source file
-tARMCORTEXM7MG:simple	Selects target processor (hardware single-precision, software double-precision floating-point)

3.1.2.3 DIAB 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
linker_script_file.dld	Use linker_script_file.dld as the linker script. This script replaces the default linker script (rather than adding to it)
-m30	m2 + m4 + m8 + m16
-Xstack-usage	Gathers and display stack usage at link time
-Xpreprocess-lecl	Perform pre-processing on linker scripts
-Llibrary_path	Points to the libraries location for ARMV7EMMG to be used for linking
-lc	Links with the standard C library
-lm	Links with the math library
-tARMCORTEXM7MG:simple	Selects target processor (hardware single-precision, software double-precision floating-point)

3.1.3 GHS Compiler/Assembler/Linker Options

3.1.3.1 GHS Compiler Options

Compiler Option	Description
-cpu=cortexm7	Selects target processor: Arm Cortex M7
-thumb	Selects generating code that executes in Thumb state
-fpu=vfpv5_d16	Specifies hardware floating-point using the v5 version of the VFP instruction set, with 16 double-precision floating-point registers
-fsingle	Use hardware single-precision, software double-precision FP instructions
-C99	Use (strict ISO) C99 standard (without extensions)
–ghstd=last	Use the most recent version of Green Hills Standard mode (which enables warnings and errors that enforce a stricter coding standard than regular C and C++)
-Osize	Optimize for size
–gnu_asm	Enables GNU extended asm syntax support
-dual_debug	Generate DWARF 2.0 debug information
-G	Generate debug information
-keeptempfiles	Prevents the deletion of temporary files after they are used. If an assembly language file is created by the compiler, this option will place it in the current directory instead of the temporary directory
-Wimplicit-int	Produce warnings if functions are assumed to return int
-Wshadow	Produce warnings if variables are shadowed
-Wtrigraphs	Produce warnings if trigraphs are detected
-Wundef	Produce a warning if undefined identifiers are used in #if preprocessor statements
–unsigned_chars	Let the type char be unsigned, like unsigned char
–unsigned_fields	Bitfelds declared with an integer type are unsigned

Compiler Option	Description
–no_commons	Allocates uninitialized global variables to a section and initializes them to zero at program startup
–no_exceptions	Disables C++ support for exception handling
–no_slash_comment	C++ style // comments are not accepted andgenerate errors
–prototype_errors	Controls the treatment of functions referenced or called when no prototype has been provided
–incorrect_pragma_warnings	Controls the treatment of valid #pragma directives that use the wrong syntax
-c	Stop after assembly and produce an object file for each source file
-DS32K3XX	Predefine S32K3XX as a macro, with definition 1
-D $ (DERIVATIVE)	Predefine S32K3's derivative as a macro, with definition 1. For example: Predefine for S32K344 will be -DS32K344.
-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
-DD_CACHE_ENABLE	Predefine D_CACHE_ENABLE as a macro, with definition 1. Enables data cache initalization in source file system.↩ c under the Platform driver
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initalization in source file system.c under the Platform driver
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver
-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=cortexm7	Selects target processor: Arm Cortex M7
-fpu=vfpv5_d16	Specifies hardware floating-point using the v5 version of the VFP instruction set, with 16 double-precision floating-point registers
-fsingle	Use hardware single-precision, software double-precision FP instructions
-preprocess_assembly_files	Controls whether assembly files with standard extensions such as .s and .asm are preprocessed
-list	Creates a listing by using the name and directory of the object file with the .lst extension
-c	Stop after assembly and produce an object file for each source file

3.1.3.3 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 themodules in this library is provided in the src/libstartup directory
-lind_sd	Link language-independent library, containing support routines for features such as software floating point, run-time error checking, C99 complex numbers, and some general purpose routines of the ANSI C library
-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

3.1.4 IAR Compiler/Assembler/Linker Options

3.1.4.1 IAR Compiler Options

Compiler Option	Description
–cpu Cortex-M7	Targeted ARM processor for which IAR should tune the performance of the code
–cpu_mode thumb	Generates code that executes in Thumb state
–endian little	Generate code for a processor running in little-endian mode
–fpu VFPv5-SP	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). Single-precision variant.
-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

Compiler Option	Description
–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
–do_explicit_zero_opt_in_named_sections	Disable the exception for variables in user-named sections, and thus treat explicit initializations to zero as zero initializations, not copy 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
-DS32K3XX	Predefine S32K3XX as a macro, with definition 1
-D $ (DERIVATIVE)	Predefine S32K3's derivative as a macro, with definition 1. For example: Predefine for S32K344 will be -DS32K344.
-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.
-DD_CACHE_ENABLE	Predefine D_CACHE_ENABLE as a macro, with definition 1. Enables data cache initalization in source file system.↩ c under the Platform driver
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initalization in source file system.c under the Platform driver
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initalization in source file system.c under the Platform driver
-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.

3.1.4.2 IAR Assembler Options

Assembler Option	Description
–cpu Cortex-M7	Targeted ARM processor for which IAR should generate the instruction set
–fpu VFPv5-SP	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). Single-precision variant.
–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

3.1.4.3 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-M7	Selects the ARM processor variant to link the application for
–fpu VFPv5-SP	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). Single-precision variant.
–entry _start	Treats _start as a root symbol and start label
–enable_stack_usage	Enables stack usage analysis. If a linker map file is produced, a stack usage chapter is included in the map file
–skip_dynamic_initialization	Dynamic initialization (typically initialization of C++ objects with static storage duration) will not be performed automatically during application startup
–no_wrap_diagnostics	Does not wrap long lines in diagnostic messages

3.2 Files required for compilation

- This section describes the include files required to compile, assemble and link the AUTOSAR Mem External Flash Driver for S32N2RT 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_ExFls Files

- Include file:
  - Mem_43_ExFls_TS_T40D34M30I0R0\include\Mem_43_ExFls.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\include\Mem_43_ExFls_IPW.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\include\Mem_43_ExFls_Types.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\include\Qspi_Ip.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\include\Qspi_Ip_Common.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\include\Qspi_Ip_Controller.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\include\Qspi_Ip_HwAccess.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\include\Qspi_Ip_Hyperflash.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\include\Qspi_Ip_HyperflashRegs.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\include\Qspi_Ip_HyperflashTypes.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\include\Qspi_Ip_TrustedFunctions.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\include\Qspi_Ip_Types.h
- Source files:
  - Mem_43_ExFls_TS_T40D34M30I0R0\src\Mem_43_ExFls.c
  - Mem_43_ExFls_TS_T40D34M30I0R0\src\Mem_43_ExFls_IPW.c
  - Mem_43_ExFls_TS_T40D34M30I0R0\src\Qspi_Ip.c
  - Mem_43_ExFls_TS_T40D34M30I0R0\src\Qspi_Ip_Controller.c
  - Mem_43_ExFls_TS_T40D34M30I0R0\src\Qspi_Ip_Hyperflash.c
  - Mem_43_ExFls_TS_T40D34M30I0R0\src\Qspi_Ip_Sfdp.c

Note

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

- Mem_43_ExFls_Cfg.h
- Mem_43_ExFls_CfgDefines.h
- Qspi_Ip_Cfg.h
- Qspi_Ip_CfgDefines.h
- Qspi_Ip_Features.h
- Mem_43_ExFls_Cfg.c
- Qspi_Ip_Cfg.c

3.2.2 Other includes files

3.2.2.1 Files from MemIf folder

- MemIf_TS_T40D34M30I0R0\include\MemIf_Types.h

3.2.2.2 Files from Base common folder

- BaseNXP_TS_T40D34M30I0R0\include\Compiler.h
- BaseNXP_TS_T40D34M30I0R0\include\Compiler_Cfg.h
- BaseNXP_TS_T40D34M30I0R0\include\ComStack_Types.h
- BaseNXP_TS_T40D34M30I0R0\include\Mem_43_ExFls_MemMap.h
- BaseNXP_TS_T40D34M30I0R0\include\Mcal.h
- BaseNXP_TS_T40D34M30I0R0\include\Platform_Types.h
- BaseNXP_TS_T40D34M30I0R0\include\Std_Types.h
- BaseNXP_TS_T40D34M30I0R0\include\Reg_eSys.h
- BaseNXP_TS_T40D34M30I0R0\include\Soc_Ips.h
- BaseNXP_TS_T40D34M30I0R0\include\Reg_Macros.h
- BaseNXP_TS_T40D34M30I0R0\include\SilRegMacros.h
- BaseNXP_TS_T40D34M30I0R0\header\S32N2RT.h
- BaseNXP_TS_T40D34M30I0R0\header\S32N2RT.h

3.2.2.3 Files from Det folder

- Det_TS_T40D34M30I0R0\include\Det.h

3.2.2.4 Files from Platform folder:

- Platform_TS_T40D34M30I0R0\include\Platform.h

3.2.2.5 Files from Rte folder

- Rte_TS_T40D34M30I0R0\include\SchM_Mem_43_ExFls.h

3.2.2.6 Files from Os folder

- Os_TS_T40D34M30I0R0\include\Os.h

3.2.2.7 Files from Rm folder

- Rm_TS_T40D34M30I0R0\include\CDD_Rm.h

3.3 Setting up the plugins

The MEM_43_EXFLS 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_ExFls module folder:

- VSMD (Vendor Specific Module Definition) file in EB Tresos Studio XDM format:
  - Mem_43_ExFls_TS_T40D34M30I0R0\config\Mem_43_ExFls.xdm
- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:
  - Mem_43_ExFls_TS_T40D34M30I0R0\autosar\Mem_43_ExFls<subderivative_name>.epd
- Code Generation Templates for parameters without variation points:
  - Mem_43_ExFls_TS_T40D34M30I0R0\generate_PC\include\Mem_43_ExFls_Cfg.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\generate_PC\include\Mem_43_ExFls_CfgDefines.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\generate_PC\include\Qspi_Ip_Cfg.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\generate_PC\include\Qspi_Ip_CfgDefines.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\generate_PC\include\Qspi_Ip_Features.h
  - Mem_43_ExFls_TS_T40D34M30I0R0\generate_PC\src\Mem_43_ExFls_Cfg.c
  - Mem_43_ExFls_TS_T40D34M30I0R0\generate_PC\src\Qspi_Ip_Cfg.c

3.3.2 Steps to generate the configuration:

1. Copy the module folders:
   - BaseNXP_TS_T40D34M30I0R0
   - Det_TS_T40D34M30I0R0
   - EcuM_TS_T40D34M30I0R0
   - Mem_43_ExFls_TS_T40D34M30I0R0
   - MemIf_TS_T40D34M30I0R0
   - Os_TS_T40D34M30I0R0
   - Platform_TS_T40D34M30I0R0
   - Resource_TS_T40D34M30I0R0
   - Rm_TS_T40D34M30I0R0
   - Rte_TS_T40D34M30I0R0 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

Mem_43_ExFls shall be initialized during STARTUP phase of EcuM initialization.

The API to be called for this is Mem_ExFls_Init().

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

MEM_EXCLUSIVE_AREA_10 is used in function Mem_43_ExFls_Erase to protect the updates for Mem_43_ExFls_eJobRuntimeInfo[InstanceIndex]

MEM_EXCLUSIVE_AREA_11 is used in function Mem_43_ExFls_Write to protect the updates for Mem_43_ExFls_eJobRuntimeInfo[InstanceIndex]

MEM_EXCLUSIVE_AREA_12 is used in function Mem_43_ExFls_Read to protect the updates for Mem_43_ExFls_eJobRuntimeInfo[InstanceIndex]

MEM_EXCLUSIVE_AREA_14 is used in function Mem_43_ExFls_BlankCheck to protect the updates for Mem_43_ExFls_eJobRuntimeInfo[InstanceIndex]

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_ExFls driver.
- If there is an “X” in a table, it means that those 2 critical regions cannot interrupt each other.

EXCLUSIVE_AREA	AREA_10	AREA_11	AREA_12	AREA_14
AREA_10		X	X	X
AREA_11	X		X	X
AREA_12	X	X		X
AREA_14	X	X	X	

5.2 Exclusive areas not available on this platform

None.

5.3 Peripheral Hardware Requirements

The Mem_43_ExFls driver uses the external flash memory peripheral (QuadSPI). For more details about peripherals and their structure refer to the S32K3XX user manual.

The Mem_43_ExFls driver will configure the QuadSPI IP. Prior to Mem_43_ExFls driver initialization, the Quad↩SPI IP should be enabled (clock, power) as required by the application, to a state in which the the Mem_43_ExFls driver can configure and use it. All platform settings or specific external memory settings are out of Mem_43_ExFls driver scope and are expected to be completed before any job is attempted.

Given the large diversity of implementation solutions in the external memories, the following operations are not fully implemented at the Mem_43_ExFls driver level for external sectors:

- External memory calibration/data learning
- External memory error check.

In order to aid implementation, callout functions and external APIs are provided in which the application could choose the best approach adapted to attached external memory. More details are provided in the following sections and User 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

- **BaseNXP**: 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 "Mem_43_ExFls_DevErrorDetect" configuration parameter.
- **Rte**: The RTE module is needed for implementing data consistency of exclusive areas that are used by Mem_43_ExFls module.
- **MemIf**: This module allows the NVRAM manager to access several memory abstraction modules.
- **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.
- **Os**: The OS module is used for OS configuration. RTD modules need OS to retrieve the application information.

5.7 Data Cache Restrictions

None.

5.8 User Mode support

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

5.8.1 User Mode configuration in the module

The Mem_43_ExFls 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 Qspi_Ip_Sfp_Configure_Privileged(QuadSPI_Type * baseAddr, Qspi_Ip_ControllerConfigType const * userConfigPtr)	Configure the SFP registers	Qspi_Ip_Trusted↩ Functions.h
void Qspi_Ip_Sfp_ClearLatchedErrors_↩ Privileged(QuadSPI_Type * BaseAddr)	Clear the errors latched in SFP registers	Qspi_Ip_Trusted↩ Functions.h
void Qspi_Ip_ResetPrivilegedRegisters_↩ Privileged(QuadSPI_Type * BaseAddr)	Reset the registers the require privilege access for programming	Qspi_Ip_Trusted↩ Functions.h
uint16 Qspi_Ip_WriteLuts_Privileged(uint32 Instance, uint8 StartLutRegister, const uint32 *Data, uint8 Size)	Configure pairs of LUT commands from the specified LUT register	Qspi_Ip_Trusted↩ Functions.h
void Qspi_Ip_SetAhbSeqId_Privileged(uint32 instance, uint8 seqID)	Sets sequence ID for AHB operations	Qspi_Ip_Trusted↩ Functions.h

5.8.2 User Mode configuration in AutosarOS

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

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

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

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

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

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

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

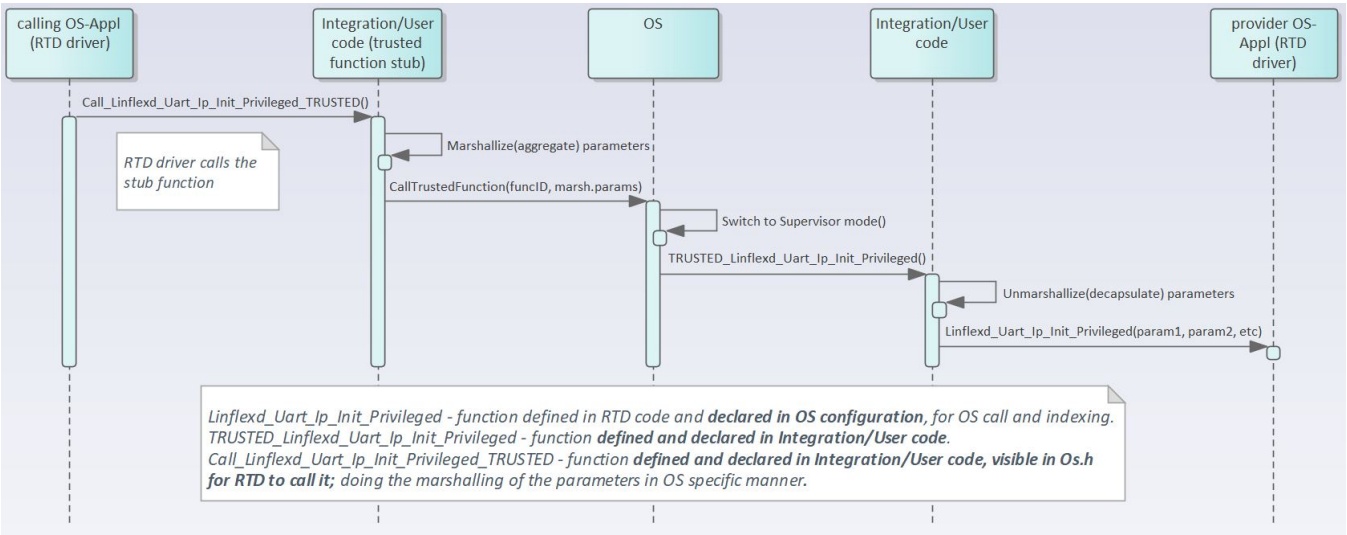

Figure 5.1 Example sequence chart for calling `Linflexd_Uart_Ip_Init_Privileged` as trusted function

5.9 Multicore support

- The Mem_43_ExFls driver does not support for multicore feature.

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

Mem_43_ExFls_MainFunction (call rate depends on target application, i.e. how fast the data needs to be read/written/compared into Flash memory).

6.2 API Requirements

None.

6.3 Calls to Notification Functions, Callbacks, Callouts

The Mem_43_ExFls driver provides notifications that are user configurable:

Notification	Usage
MemQspiInitCallout	Perform additional checks and configurations on the external memory at the end of MEM_EXFLS initialization
MemQspiResetCallout	Perform reset or cancel on external memory, depending on the available reset mechanisms
MemQspiErrorCheckCallout	Check any errors occurred in external memories during erase or program operations
MemQspiEccCheckCallout	Interrogate the ECC status of the memory after each read operation

Chapter 7

Memory allocation

- Sections to be defined in Mem_43_ExFls_MemMap.h
- Linker command file

7.1 Sections to be defined in Mem_43_ExFls_MemMap.h

Index	Section name	Type of section	Description
1	MEM_43_EXFLS_START_SEC_↩ CODE	Code	Start of memory Section for Code.
	MEM_43_EXFLS_STOP_SEC_↩ CODE		End of above section.
2	MEM_43_EXFLS_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_EXFLS_STOP_SEC_↩ CONFIG_DATA_UNSPECIFIED		End of above section.
3	MEM_43_EXFLS_START_SEC_↩ CONST_UNSPECIFIED	Constant Data	The parameters that are not variant aware shall be stored in memory section for constants.
	MEM_43_EXFLS_STOP_SEC_↩ CONST_UNSPECIFIED		End of above section.
4	MEM_43_EXFLS_START_SEC_↩ VAR_INIT_8	Variables	Start of memory Section for Variable declaration that have to be aligned to 8 bit.
	MEM_43_EXFLS_STOP_SEC_↩ VAR_INIT_8		End of above section.
5	MEM_43_EXFLS_START_SEC_↩ VAR_INIT_BOOLEAN	Variables	Start of memory Section for Variable declaration that have to be aligned type boolean
	MEM_43_EXFLS_STOP_SEC_↩ VAR_INIT_BOOLEAN		End of above section.

Index	Section name	Type of section	Description
6	MEM_43_EXFLS_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_EXFLS_STOP_SEC_↩ VAR_INIT_UNSPECIFIED		End of above section.
7	MEM_43_EXFLS_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_EXFLS_STOP_SEC_↩ VAR_CLEARED_8		End of above section.
8	MEM_43_EXFLS_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_EXFLS_STOP_SEC_↩ VAR_CLEARED_16		End of above section.
9	MEM_43_EXFLS_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_EXFLS_STOP_SEC_↩ VAR_CLEARED_32		End of above section.
10	MEM_43_EXFLS_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.
	MEM_43_EXFLS_STOP_SEC_↩ VAR_CLEARED_BOOLEAN		End of above section.
11	MEM_43_EXFLS_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_EXFLS_STOP_SEC_↩ VAR_CLEARED_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

None.

9.1 Application's tasks

- It is responsibility of integrator/application to ensure that MCU-wide parameters like voltage supply etc. are according to and in limits specified in MCU documentation.
- Integrator/application is responsible to implement additional functionality that cancel any on-going erase/write Qspi jobs if MCU conditions are not in such limits.

9.2 External memory

Mem_43_ExFls driver shall support all Flash external memories that can be used via the QuadSPI Controller. These memories should:

- Fit to the LUT programming model
- Be electrically compatible with the QuadSPI IP
- The driver shall support initialization of such memories as well as configuring the command sequences for basic functionality (Flash Writes, Reads, Erase, etc)
- Rationale: In order to allow a wide array of Flash external memories that might be on customer boards, we allow the configuration of any memory that can be configured through the QuadSpi Controller
- To support various Flash external memories, the driver has callouts to allow configuring device specific features, for example the **MemQspiInitCallout** field allows configuration of memory specific features during Init
  The section presents requirements that must be complied with when integrating the MEM_43_EXFLS 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
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

External assumptions for driver

External Assumption Req ID	External Assumption Text
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 Main↩ Function service. 4: GetJobResult service pointer: Function pointer to Mem driver GetJobResult 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: HwSpecificService 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. Note: Image feature
SWS_Mem_00051	The value of the delimiter field shall be the ones' complement of the unique identifier value. Note: Image feature
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>_De↩ Init() 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

Tips for MEM_43_EXFLS integration

10.1 ECC Management on Flash

The section presents requirements that must be complied with when integrating the Mem_43_ExFls driver into the application.

10.1.1 ECC Management on Qspi Flash

The ECC management on QSPI sectors is dependent on the external memory specific implementation.

The driver offers MemQspiEccCheckCallout, called at the end of each read job to offer the possibility to the application to interrogate the external memory error status.

Depending on the hardware resources available on the external memory, the application can enable and implement the ECC callout in order to check any error bits available and return a failed value in order to mark the current job as failed.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support