Integration Manual

for S32K1_S32M24x MCU Driver

Document Number: IM2MCUASRR21-11 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 NXP Semiconductors' AUTOSAR Mcu Driver for S32K1_S32M24x.

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 Mcu Driver	AUTOSAR Release R21-11
2	S32K1xx Reference Manual	S32K1xx Series Reference Manual, Rev. 14, 09/2021
3	S32K1xx Data Sheet	S32K1xx Data Sheet, Rev. 14, 08/2021
4	S32M24x Reference Manual	S32M24x Reference Manual, Rev. 2 Draft A, 05/2023
5	S32M24x Data Sheet	Supports S32M24x and S32M27x, Rev. 3 DraftA, 05/2023
6	S32M244 Errata Document	S32M244_P64A+P73G, Rev. 0
6	S32M242 Errata Document	S32M242_N33V+P73G, Rev. 0, 6/2023
7	S32K116 Errata Document	S32K116_0N96V Rev. 22/OCT/2021
8	S32K118 Errata Document	S32K118_0N97V Rev. 22/OCT/2021
9	S32K142 Errata Document	S32K142_0N33V Rev. 22/OCT/2021
10	S32K144 Errata Document	S32K144_0N57U Rev. 22/OCT/2021
11	S32K144W Errata Document	S32K144W_0P64A Rev. 22/OCT/2021
12	S32K146 Errata Document	S32K146_0N73V Rev. 22/OCT/2021
13	S32K148 Errata Document	S32K148_0N20V Rev. 22/OCT/2021

Chapter 3

Building the driver

- Build Options
- Files required for compilation
- Setting up the plugins

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

3.1 Build Options

- GCC Compiler/Assembler/Linker Options
- 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. DWARF debug information will contain references to deleted functions that may break some third-party debuggers
-Llibrary_path	Points to library_path (the libraries location) for thumb2 to be used for linking
-larch	Link architecture specific library
-lstartup	Link run-time environment startup routines. The source code for the modules in this library is provided in the src/libstartup directory
-lind_sd	Link language-independent library, containing support routines for features such as software floating point, run-time error checking, C99 complex numbers, and some general purpose routines of the ANSI C library (for S32K14x or S32M24x devices)
-lind_sf	Link language-independent library, containing support routines for features such as software floating point, run-time error checking, C99 complex numbers, and some general purpose routines of the ANSI C library (for S32K11x devices)
-v	Prints verbose information about the activities of the linker, including the libraries it searches to resolve undefined symbols
-keep=C40_Ip_AccessCode	Avoid linker remove function C40_Ip_AccessCode from Fls module because it is not referenced explicitly
-nostartfiles	Controls the start files to be linked into the executable

3.2 Files required for compilation

This section describes the include files required to compile, assemble and link the AUTOSAR Mcu Driver for S32K1_S32M24x microcontrollers.

To avoid integration of incompatible files, all the include files from other modules shall have the same AR_MAJOR_VERSION and AR_MINOR_VERSION, i.e. only files with the same AUTOSAR major and minor versions can be compiled.

3.2.0.0.1 Mcu Driver Files:

- Mcu_TS_T40D2M20I0R0\src\Clock_Ip.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_Data.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_Divider.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_DividerTrigger.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_ExtOsc.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_FracDiv.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_Frequency.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_Gate.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_IntOsc.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_Irq.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_Monitor.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_Pll.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_ProgFreqSwitch.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_Selector.c
- Mcu_TS_T40D2M20I0R0\src\Clock_Ip_Specific.c
- Mcu_TS_T40D2M20I0R0\src\Mcu_Dem_Wrapper.c
- Mcu_TS_T40D2M20I0R0\src\Mcu_Ipw.c
- Mcu_TS_T40D2M20I0R0\src\Mcu_Ipw_Irq.c
- Mcu_TS_T40D2M20I0R0\src\Mcu.c
- Mcu_TS_T40D2M20I0R0\src\Power_Ip.c
- Mcu_TS_T40D2M20I0R0\src\Power_Ip_AEC.c
- Mcu_TS_T40D2M20I0R0\src\Power_Ip_CortexM4.c
- Mcu_TS_T40D2M20I0R0\src\Power_Ip_PMC.c
- Mcu_TS_T40D2M20I0R0\src\Power_Ip_PMC_Irq.c
- Mcu_TS_T40D2M20I0R0\src\Power_Ip_Private.c
- Mcu_TS_T40D2M20I0R0\src\Power_Ip_RCM.c
- Mcu_TS_T40D2M20I0R0\src\Power_Ip_RCM_Irq.c
- Mcu_TS_T40D2M20I0R0\src\Power_Ip_SIM.c
- Mcu_TS_T40D2M20I0R0\src\Power_Ip_SMC.c
- Mcu_TS_T40D2M20I0R0\src\Ram_Ip.c
- Mcu_TS_T40D2M20I0R0\include\Clock_Ip.h
- Mcu_TS_T40D2M20I0R0\include\Clock_Ip_Private.h

- Mcu_TS_T40D2M20I0R0\include\Clock_Ip_Specific.h
- Mcu_TS_T40D2M20I0R0\include\Clock_Ip_Types.h
- Mcu_TS_T40D2M20I0R0\include\Mcu_Dem_Wrapper.h
- Mcu_TS_T40D2M20I0R0\include\Mcu_EnvCfg.h
- Mcu_TS_T40D2M20I0R0\include\Mcu_Ipw_Types.h
- Mcu_TS_T40D2M20I0R0\include\Mcu_Ipw.h
- Mcu_TS_T40D2M20I0R0\include\Mcu.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip_AEC.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip_AEC_Types.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip_CortexM4.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip_PMC.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip_PMC_Types.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip_Private.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip_RCM.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip_RCM_Types.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip_SIM.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip_SMC.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip_SMC_Types.h
- Mcu_TS_T40D2M20I0R0\include\Power_Ip_Types.h
- Mcu_TS_T40D2M20I0R0\include\Ram_Ip_Types.h
- Mcu_TS_T40D2M20I0R0\include\Ram_Ip.h

3.2.0.0.2 Mcu Driver Generated Files (must be generated by the user using a configuration tool):

- Clock_Ip_Cfg_Defines.h
- Clock_Ip_Cfg.h
- Mcu_Cfg.h
- Power_Ip_Cfg_Defines.h
- Power_Ip_Cfg.h
- Ram_Ip_Cfg_Defines.h
- Ram_Ip_Cfg.h
- Clock_Ip_Cfg.c
- Mcu_Cfg.c
- Power_Ip_Cfg.c
- Ram_Ip_Cfg.c

Note

As a deviation from the standard:

- Mcu_[VariantName]_PBcfg.c, Clock_Ip_[VariantName]_PBcfg.c, Power_Ip_[VariantName]_PBcfg.c, Ram_Ip_[VariantName]_PBcfg.c, - These files will contain the definition for all parameters that are variant aware, independent of the configuration class that will be selected (PC, PB)
- Mcu_Cfg.c, Clock_Ip_Cfg.c, Power_Ip_Cfg.c, Ram_Ip_Cfg.c - These files will contain the definition for all configuration structures containing only variables that are not variant aware, configured and generated only once. These files alone do not contain the whole structure needed by Mcu_Init function to configure the driver. Based on the number of variants configured in the EcuC, there can be more than one configuration structure for one module even for VariantPreCompile.

3.2.0.0.3 BaseNXP Files:

- BaseNXP_TS_T40D2M20I0R0\header\S32K116_COMMON.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K116_CMU.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K116_FTM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K116_PCC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K116_PMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K116_RCM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K116_SCG.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K116_SIM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K116_SMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K116_SYSTICK.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K118_COMMON.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K118_CMU.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K118_FTM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K118_PCC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K118_PMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K118_RCM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K118_SCG.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K118_SIM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K118_SMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K118_SYSTICK.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142_COMMON.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142_FTM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142_PCC.h

- BaseNXP_TS_T40D2M20I0R0\header\S32K142_PMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142_RCM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142_SCG.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142_SIM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142_SMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142_SYSTICK.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142W_COMMON.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142W_FTM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142W_PCC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142W_PMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142W_RCM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142W_SCG.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142W_SIM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142W_SMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K142W_SYSTICK.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144_COMMON.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144_FTM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144_PCC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144_PMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144_RCM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144_SCG.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144_SIM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144_SMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144_SYSTICK.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144W_COMMON.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144W_FTM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144W_PCC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144W_PMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144W_RCM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144W_SCG.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144W_SIM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144W_SMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K144W_SYSTICK.h

- BaseNXP_TS_T40D2M20I0R0\header\S32K146_COMMON.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K146_FTM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K146_PCC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K146_PMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K146_RCM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K146_SCG.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K146_SIM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K146_SMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K146_SYSTICK.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K148_COMMON.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K148_FTM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K148_PCC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K148_PMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K148_QUADSPI.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K148_RCM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K148_SCG.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K148_SIM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K148_SMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32K148_SYSTICK.h
- BaseNXP_TS_T40D2M20I0R0\header\S32M24x_COMMON.h
- BaseNXP_TS_T40D2M20I0R0\header\S32M24x_FTM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32M24x_PCC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32M24x_PMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32M24x_PMC_142.h
- BaseNXP_TS_T40D2M20I0R0\header\S32M24x_PMC_AE.h
- BaseNXP_TS_T40D2M20I0R0\header\S32M24x_RCM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32M24x_SCG.h
- BaseNXP_TS_T40D2M20I0R0\header\S32M24x_SIM.h
- BaseNXP_TS_T40D2M20I0R0\header\S32M24x_SMC.h
- BaseNXP_TS_T40D2M20I0R0\header\S32M24x_SYSTICK.h
- BaseNXP_TS_T40D2M20I0R0\header\S32M24x_AEC_AE.h
- BaseNXP_TS_T40D2M20I0R0\include\Mcal.h
- BaseNXP_TS_T40D2M20I0R0\include\StandardTypes.h

- BaseNXP_TS_T40D2M20I0R0\include\Devassert.h
- BaseNXP_TS_T40D2M20I0R0\include\Platform_Types.h
- BaseNXP_TS_T40D2M20I0R0\include\Std_Types.h
- BaseNXP_TS_T40D2M20I0R0\include\RegLockMacros.h
- BaseNXP_TS_T40D2M20I0R0\include\OsIf.h
- BaseNXP_TS_T40D2M20I0R0\src\OsIf_Timer.c
- BaseNXP_TS_T40D2M20I0R0\src\OsIf_Timer_System.c

3.2.0.0.4 DEM Files:

- Dem_TS_T40D2M20I0R0\include\Dem.h
- Dem_TS_T40D2M20I0R0\src\Dem.c

3.2.0.0.5 DET Files:

- Det_TS_T40D2M20I0R0\include\Det.h
- Det_TS_T40D2M20I0R0\src\Det.c

3.2.0.0.6 RTE Files:

- Rte_TS_T40D2M20I0R0\include\SchM_Mcu.h
- Rte_TS_T40D2M20I0R0\src\SchM_Mcu.c

3.3 Setting up the plugins

The Mcu Driver was designed to be configured by using the EB Tresos Studio (version 29.0.0 or later)

3.3.0.0.1 Location of various files inside the MCU module folder:

- VSMD (Vendor Specific Module Definition) file in EB Tresos Studio XDM format:
  - Mcu_TS_T40D2M20I0R0\config\Mcu.xdm
- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:
  - Mcu_TS_T40D2M20I0R0\autosar\Mcu_<subderivative_name>.epd
- Code Generation Templates for variant aware parameters:
  - Mcu_TS_T40D2M20I0R0\generate_PB\src\Clock_Ip_PBcfg.c
  - Mcu_TS_T40D2M20I0R0\generate_PB\src\Mcu_PBcfg.c
  - Mcu_TS_T40D2M20I0R0\generate_PB\src\Power_Ip_PBcfg.c
  - Mcu_TS_T40D2M20I0R0\generate_PB\src\Ram_Ip_PBcfg.c
  - Mcu_TS_T40D2M20I0R0\generate_PB\include\Clock_Ip_PBcfg.h
  - Mcu_TS_T40D2M20I0R0\generate_PB\include\Mcu_PBcfg.h
  - Mcu_TS_T40D2M20I0R0\generate_PB\include\Power_Ip_PBcfg.h
  - Mcu_TS_T40D2M20I0R0\generate_PB\include\Ram_Ip_PBcfg.h
  - Mcu_TS_T40D2M20I0R0\generate_PB\Clock_Ip_RegOperations.m
  - Mcu_TS_T40D2M20I0R0\generate_PB\Mcu_RegOperations.m
  - Mcu_TS_T40D2M20I0R0\generate_PB\Power_Ip_RegOperations.m
  - Mcu_TS_T40D2M20I0R0\generate_PB\Ram_Ip_RegOperations.m
- Code Generation Templates for parameters without variation points:
  - Mcu_TS_T40D2M20I0R0\generate_PC\src\Clock_Ip_Cfg.c
  - Mcu_TS_T40D2M20I0R0\generate_PC\src\Mcu_Cfg.c
  - Mcu_TS_T40D2M20I0R0\generate_PC\src\Power_Ip_Cfg.c
  - Mcu_TS_T40D2M20I0R0\generate_PC\src\Ram_Ip_Cfg.c
  - Mcu_TS_T40D2M20I0R0\generate_PC\include\Clock_Ip_Cfg.h
  - Mcu_TS_T40D2M20I0R0\generate_PC\include\Clock_Ip_Cfg_Defines.h
  - Mcu_TS_T40D2M20I0R0\generate_PC\include\Mcu_Cfg.h
  - Mcu_TS_T40D2M20I0R0\generate_PC\include\Power_Ip_Cfg.h
  - Mcu_TS_T40D2M20I0R0\generate_PC\include\Power_Ip_Cfg_Defines.h
  - Mcu_TS_T40D2M20I0R0\generate_PC\include\Ram_Ip_Cfg.h
  - Mcu_TS_T40D2M20I0R0\generate_PC\include\Ram_Ip_Cfg_Defines.h
  - Mcu_TS_T40D2M20I0R0\generate_PC\Clock_Ip_RegOperations.m
  - Mcu_TS_T40D2M20I0R0\generate_PC\Mcu_RegOperations.m
  - Mcu_TS_T40D2M20I0R0\generate_PC\Power_Ip_RegOperations.m
  - Mcu_TS_T40D2M20I0R0\generate_PC\Ram_Ip_RegOperations.m

3.3.0.0.2 Steps to generate the configuration:

1. Copy the following module folders into the Tresos plugins folder:
   - BaseNXP_TS_T40D2M20I0R0
   - Dem_TS_T40D2M20I0R0
   - Det_TS_T40D2M20I0R0
   - EcuC_TS_T40D2M20I0R0
   - Os_TS_T40D2M20I0R0
   - Platform_TS_T40D2M20I0R0
   - Resource_TS_T40D2M20I0R0
   - Rte_TS_T40D2M20I0R0
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 first BSW module to be initialized after Power on shall be MCU. The MCU shall be initialized in the following sequence.

1. Mcu_Init()
2. Mcu_InitClock()
3. Mcu_GetPllStatus() - Till PLL is locked.
4. Mcu_DistributePllClock()
5. Mcu_SetMode()
6. Mcu_InitRamSection() - If required

4.2 Function Calls during Shutdown

Mcu_SetMode (sleep mode) API shall be called during GO SLEEP phase of the EcuM's Shutdown state to configure the hardware for Sleep mode. This shall be called after ICU & GPT are set to sleep.

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

Exclusive Areas implemented in High level driver layer (HLD)
MCU_EXCLUSIVE_AREA_00 is used in function Mcu_SetMode() to protect against itself in the context of multicore usage of it and ISR event.
MCU_EXCLUSIVE_AREA_01 is used in function Mcu_DisableCmu() to protect against itself in the context of multicore usage of it and ISR event.
Exclusive Areas implemented in Low level driver layer (IPL)
MCU_EXCLUSIVE_AREA_01 is used in function Clock_Ip_InitClock() to protect against itself in the context of multicore usage of it and ISR event.
MCU_EXCLUSIVE_AREA_01 is used in function Clock_Ip_DisableClockMonitor() to protect against itself in the context of multicore usage of it and ISR event.
Below is the table depicting the exclusivity between different critical region IDs from the MCU driver. If there is an "X" in the table, it means that those 2 critical regions cannot interrupt each other.

Table 5.1 Critical Region Exclusive Matrix

MCU_EXCLUSIVE_AREA	MCU_EA_00	MCU_EA_01	Interrupt Service Routines Critical Regions(composed diagram)
MCU_EA_00	X		X
MCU_EA_01		X	X

Note
MCU_EA_xx means MCU_EXCLUSIVE_AREA_xx

5.2 Exclusive areas not available on this platform

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

MCU_EXCLUSIVE_AREA_02 is used in function Mcu_EmiosConfigureGpren() to protect against itself in the context of multicore usage of it and ISR event.

5.3 Peripheral Hardware Requirements

None.

5.4 ISR to configure within AutosarOS - dependencies

The following ISR's are used by the MCU driver:

Table 5.3 MCU ISRs

Module Name	ISR Name	Vector Number	ISR Number
PMC	Power_Ip_PMC_Low↩ Voltage_ISR	36(for S32K14x, S32M24x)/37(for S32K11x)	20(for S32K14x, S32M24x)/21(for S32K11x)
RCM	Power_Ip_RCM_↩ AlternateResetIsr	39	23
CMU(for S32K11x)	Mcu_Cmu_ClockFail_↩ IRQHandler	37	21
CMU(for S32K11x)	Mcu_PMC_SCG_CM↩ U_Isr	37	21
PMC	Mcu_PMC_SCG_CM↩ U_Isr	36(for S32K14x, S32M24x)/37(for S32K11x)	20(for S32K14x, S32M24x)/21(for S32K11x)

Module Name	ISR Name	Vector Number	ISR Number
SCG	Mcu_PMC_SCG_CM↩ U_Isr	73(for S32K14x)/37(for S32K11x)	57(for S32K14x)/21(for S32K11x)
PMC_AE(for S32M24x)	Power_Ip_PMC_AE_↩ VoltageDetectHvdOn↩ VddintVdd15_ISR	NA	13
PMC_AE(for S32M24x)	Power_Ip_PMC_AE_↩ VoltageDetectHvdOn↩ Vdd_ISR	NA	12
PMC_AE(for S32M24x)	Power_Ip_PMC_AE_↩ VoltageDetectLvdOn↩ Vddc_ISR	NA	7
PMC_AE(for S32M24x)	Power_Ip_PMC_AE_↩ VoltageDetectLvdOn↩ Vls_ISR	NA	4

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

- Platform: This module is used for configures platform specific settings, managing the interrupt requests and other system wide settings as defined in each hardware implementation.
- Det: The DET module is used for enabling Development error detection. The API function used is Det_↩ ReportError(). The activation / deactivation of Development error detection is configurable using the 'Mcu↩ DevErrorDetect' configuration parameter.
- Dem: This module is necessary for enabling reporting of production relevant error status. The API function used is Dem_SetEventStatus().
- EcuC: The ECUC module is used for ECU configuration. MCAL modules need ECUC to retrieve the variant information.
- Rte: The RTE module is needed for implementing data consistency of exclusive areas that are used by MCU module. The module is the realization (for a particular ECU) of the interfaces of the AUTOSAR Virtual Function Bus (VFB) and thus provides the infrastructure services for communication between Application Software Components as well as facilitating access to basic software components including the OS
- BaseNXP: The BaseNXP module contains the common files/definitions needed by the MCAL. This means that it is a dependency for all other MCAL modules.
- Resource: Resource module is used to select microcontroller's derivatives.

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

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

Function syntax	Description	Available via
void Clock_Ip_SetScgAsyncDiv1↩ _TrustedCall(Clock_Ip_Divider↩ ConfigType const* Config)	Write Config to SCG_SIRCDIV1 register	Clock_Ip_TrustedFunction.h

Function syntax	Description	Available via
void Clock_Ip_SetScgAsyncDiv2_TrustedCall(Clock_Ip_DividerConfigType const* Config)	Write Config to SCG_SIRCDIV2 register	Clock_Ip_TrustedFunction.h
void Clock_Ip_SetScgRunDivcore_TrustedCall(Clock_Ip_DividerConfigType const* Config)	Write Config to SCG_RCCR register	
void Clock_Ip_SetScgRunDivbus_TrustedCall(Clock_Ip_DividerConfigType const* Config)	Write Config to SCG_RCCR register	
void Clock_Ip_SetScgRunDivslow_TrustedCall(Clock_Ip_DividerConfigType const* Config)	Write Config to SCG_RCCR register	
void Clock_Ip_SetScgVlprDivcore_TrustedCall(Clock_Ip_DividerConfigType const* Config)	Write Config to SCG_VCCR register	
void Clock_Ip_SetScgVlprDivslow_TrustedCall(Clock_Ip_DividerConfigType const* Config)	Write Config to SCG_VCCR register	
void Clock_Ip_SetScgHsrunDivcore_TrustedCall(Clock_Ip_DividerConfigType const* Config)	Write Config to SCG_HCCR register	
void Clock_Ip_SetScgHsrunDivbus_TrustedCall(Clock_Ip_DividerConfigType const* Config)	Write Config to SCG_HCCR register	
void Clock_Ip_SetScgHsrunDivslow_TrustedCall(Clock_Ip_DividerConfigType const* Config)	Write Config to SCG_HCCR register	
void Clock_Ip_SetSimClkoutDiv_TrustedCall(Clock_Ip_DividerConfigType const *Config)	Write Config to IP_SIM::CHIPCTL register	
void Clock_Ip_SetPccPcdDivFrac_TrustedCall(Clock_Ip_DividerConfigType const* Config)	Write Config to PCC register	
void Clock_Ip_SetSimTraceDivMul_TrustedCall(Clock_Ip_DividerConfigType const* Config)	Write Config to IP_SIM::CLKDIV4 register	
void Clock_Ip_ResetSOSC_TrustedCall(Clock_Ip_XoscConfigType const* Config)	Reset IP_SCG::SOSCCSR register	
void Clock_Ip_SetSOSC_TrustedCall(Clock_Ip_XoscConfigType const* Config)	Write Config of SOSC to register	
void Clock_Ip_DisableSOSC_TrustedCall(Clock_Ip_NameType XoscName)	Disabele SOSC	
void Clock_Ip_EnableSOSC_TrustedCall(Clock_Ip_XoscConfigType const* Config)	Enable SOSC	

Module requirements

Function syntax	Description	Available via
void Clock_Ip_SetSirc_Trusted↩ Call(Clock_Ip_IrcoscConfigType const∗ Config)	Write Config to SIRC register	Clock_Ip_TrustedFunction.h
void Clock_Ip_EnableSirc_Trusted↩ Call(Clock_Ip_IrcoscConfigType const∗ Config)	Enable SIRC	
void Clock_Ip_DisableSirc_↩ TrustedCall(Clock_Ip_NameType Name)	Disable SIRC	
void Clock_Ip_SetSircVlp_Trusted↩ Call(Clock_Ip_IrcoscConfigType const∗ Config)	Write Config for SIRC in Low power mode to register	
void Clock_Ip_EnableSircVlp↩ _TrustedCall(Clock_Ip_Ircosc↩ ConfigType const∗ Config)	Enable Sirc in Low power mode	
void Clock_Ip_DisableSircVlp_↩ TrustedCall(Clock_Ip_NameType Name)	Disable Sirc in Low power mode	
void Clock_Ip_DisableSircVlp_↩ TrustedCall(Clock_Ip_NameType Name)	Disable Sirc in Low power mode	
void Clock_Ip_EnableSircStop↩ _TrustedCall(Clock_Ip_Ircosc↩ ConfigType const∗ Config)	Enable SIRC in Stop mode to register	
void Clock_Ip_DisableSircStop_↩ TrustedCall(Clock_Ip_NameType Name)	Disable SIRC in Stop mode to register	
void Clock_Ip_SetFirc_Trusted↩ Call(Clock_Ip_IrcoscConfigType const∗ Config)	Write Config for Firc to register	
void Clock_Ip_EnableFirc_↩ TrustedCall(Clock_Ip_Ircosc↩ ConfigType const∗ Config)	Enable Firc to register	
void Clock_Ip_DisableFirc_↩ TrustedCall(Clock_Ip_NameType Name)	Disable Firc to register	
void Clock_Ip_ResetSpll_Trusted↩ Call(Clock_Ip_PllConfigType const∗ Config)	Reset Spll register	
void Clock_Ip_SetSpll_Trusted↩ Call(Clock_Ip_PllConfigType const∗ Config)	Write Config to Spll register	
void Clock_Ip_DisableSpll_↩ TrustedCall(Clock_Ip_NameType PllName)	Disable Spll	
void Clock_Ip_EnableSpll_Trusted↩ Call(Clock_Ip_PllConfigType const∗ Config)	Enable Spll	

Function syntax	Description	Available via
void Clock_Ip_ResetScgRunSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Reset SCG_RCCR selector register	Clock_Ip_TrustedFunction.h
void Clock_Ip_SetScgRunSel_↩ TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Write Config to SCG_RCCR selector register	
void Clock_Ip_SetScgVlprSel_↩ TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Write Config to SCG_VCCR selector register	
void Clock_Ip_ResetScgHsrunSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Reset SCG_HCCR selector register	
void Clock_Ip_SetScgHsrunSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Write Config to SCG_HCCR selector register	
void Clock_Ip_ResetSimRtcSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Reset SIM_LPOCLKS_RTCCLKS↩ EL register	
void Clock_Ip_SetSimRtcSel_↩ TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Write Config to RTC_CLK register	
void Clock_Ip_ResetSimLpoSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Reset LPO_CLK selector	
void Clock_Ip_SetSimLpoSel_↩ TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Write Config to LPO_CLK selector	
void Clock_Ip_ResetScgClkoutSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Reset CLKOUT_CLK selector	
void Clock_Ip_SetScgClkoutSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Write Config to LPO_CLK register	
void Clock_Ip_ResetSimFtmoptSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Reset SIM->FTMOPT0 selector register	
void Clock_Ip_SetSimFtmoptSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Write Config to SIM->FTMOPT0 register	
void Clock_Ip_SetSimFtmoptSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Write Config to SIM->FTMOPT0 register	
void Clock_Ip_SetSimClkoutSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Write Config to SIM_CHIPCTL_C↩ LKOUTSEL register	
void Clock_Ip_ResetPccPcsSelect↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const *Config)	Reset register of PCC peripheral clock	

Module requirements

Function syntax	Description	Available via
void Clock_Ip_SetPccPcsSelect↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const ∗Config)	Write Config to PCC register	Clock_Ip_TrustedFunction.h
void Clock_Ip_ResetSimTraceSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const ∗Config)	Reset selector register of TRACE_↩ CLK	
void Clock_Ip_SetSimTraceSel↩ _TrustedCall(Clock_Ip_Selector↩ ConfigType const ∗Config)	Write Config to TRACE_CLK selector register	
void SetFircToResetValue_Trusted↩ Call(void)	Reset FIRC	
void SetSimLpoclksRegister_↩ TrustedCall(Clock_Ip_ClockConfig↩ Type const ∗Config)	Write Confgig whole SIM_LPOCLKS register	
void Clock_Ip_EnableCmu0Gate_↩ TrustedCall(void)	Enable Cmu clock gate	
void Clock_Ip_EnableCmu1Gate_↩ TrustedCall(void)	Enable Cmu clock gate	
void Power_Ip_CM4_EnableSleep↩ OnExit(void)	The function enables SLEEPONEXIT bit	Power_Ip_TrustedFunction.h
void Power_Ip_CM4_DisableSleep↩ OnExit(void)	The function disables SLEEPONEXIT bit	
void Power_Ip_CM4_EnableDeep↩ Sleep(void)	The function enables DEEPSLEEP	
void Power_Ip_CM4_DisableDeep↩ Sleep(void)	The function disables DEEPSLEEP	
void Power_Ip_CM4_System↩ Reset(void)	The function initiates a system reset request to reset the SoC	
void Power_Ip_PMC_Power↩ Init(const Power_Ip_PMC_Config↩ Type ∗ ConfigPtr)	This function configure the Power Management Controller	
void Power_Ip_RCM_Reset↩ Init(const Power_Ip_RCM_Config↩ Type ∗ ConfigPtr)	This function initializes the Reset parameters	
uint32 Power_Ip_RCM_GetReset↩ Reason(void)	This function returns the Reset reason	
Power_Ip_RawResetType Power_↩ Ip_RCM_GetResetRawValue(void)	This function returns the Raw Reset value	
void Power_Ip_SCG_DisableClock↩ Monitors(void)	This function will disable all clock monitors	
void Power_Ip_SIM_SRAM↩ RetentionConfig(Power_Ip_SRAM↩ RetenConfigType SRAMRetenConfig)	Configuration for SRAM retention	
void Power_Ip_SMC_Allowed↩ ModesConfig(const Power_Ip_SM↩ C_ConfigType ∗ ConfigPtr)	This function will configure the allowed modes	

Function syntax	Description	Available via
uint32 Power_Ip_SMC_Mode↩ Config(const Power_Ip_Mode↩ ConfigType * ModeConfigPtr)	This function switches the mode by writing SMC_PMCTRL and SMC_↩ STOPCTRL	

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.

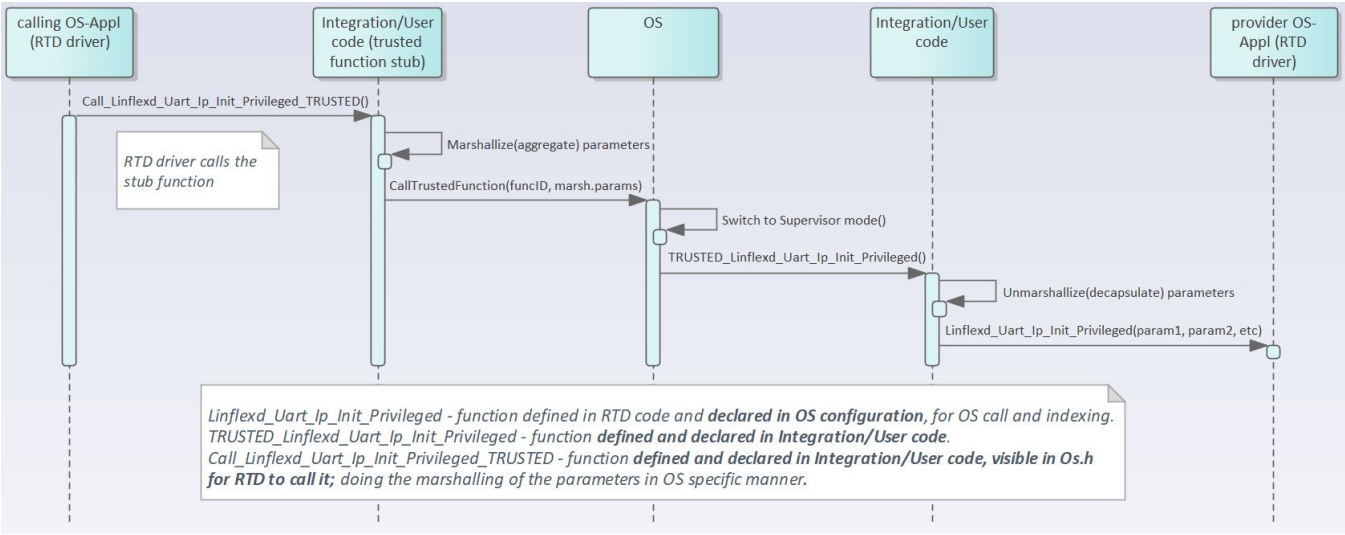

Figure 5.1 Example sequence chart for calling `Linflexd_Uart_Ip_Init_Privileged` as trusted function

5.9 Multicore support

The Mcu 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

```
- McuPerformResetCallout called by MCU right before Mcu_PerformReset() .
- McuErrorIsrNotification The callout configured by the user for error ISR notifications.
- McuCmuNotification The callout configured by the user for CMU notifications.
- McuPrepareMemoryConfig The callout configured by the user for preparing flash and ram
controllers configuration.
```

Chapter 7

Memory allocation

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

7.1 Sections to be defined in Mcu_MemMap.h

Section name	Type of section	Description
MCU_START_SEC_CONFIG_DATA↩ _UNSPECIFIED	Configuration Data	Start of Memory Section for Config Data
MCU_STOP_SEC_CONFIG_DATA_↩ UNSPECIFIED	Configuration Data	End of Memory Section for Config Data
MCU_START_SEC_CONST_UNSPE↩ CIFIED	Configuration Data	Start of Memory Section for Config Data that is not variant aware
MCU_STOP_SEC_CONST_UNSPECI↩ FIED	Configuration Data	End of Memory Section for Config Data that is not variant aware
MCU_START_SEC_CODE	Code	Start of memory Section for Code
MCU_STOP_SEC_CODE	Code	End of memory Section for Code
MCU_START_SEC_CODE_AC	Code	Start of code relative addressing mode to ensure Position-independent Code
MCU_STOP_SEC_CODE_AC	Code	End of above section.
MCU_START_SEC_RAMCODE	Code	Start of memory Section for Code to be located in Ram
MCU_STOP_SEC_RAMCODE	Code	End of memory Section for Code to be located in Ram
MCU_START_SEC_VAR_CLEARED↩ _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 cleared to zero by start-up code.
MCU_STOP_SEC_VAR_CLEARED_↩ UNSPECIFIED	Variables	End of above section.
MCU_START_SEC_VAR_CLEARED↩ _UNSPECIFIED_NO_CACHEABLE	Variables	Used for variables, structures, arrays when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit. These variables are cleared to zero by start-up code (no cacheable)

Section name	Type of section	Description
MCU_STOP_SEC_VAR_CLEARED_↩ UNSPECIFIED_NO_CACHEABLE	Variables	End of above section.
MCU_START_SEC_VAR_INIT_32	Variables	Used for variables which have to be aligned to 32 bit. For instance used for variables of size 32 bit or used for composite data types↩ :arrays ,structs containing elements of maximum 32 bits. These variables are initialized with values after every reset.
MCU_STOP_SEC_VAR_INIT_32	Variables	End of above section.
MCU_START_SEC_VAR_INIT_UNS↩ PECIFIED	Variables	Used for variables, structures, arrays, when the SIZE (alignment) does not fit the criterian of 8,16 or 32 bit. These variables are initialized with values after every reset.
MCU_STOP_SEC_VAR_INIT_UNSP↩ ECIFIED	Variables	End of above section.

7.2 Linker command file

Memory shall be allocated for every section defined in the driver's "<Module>"_MemMap.h.

Chapter 8

Integration Steps

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

1. Generate the required module configuration(s). For more details refer to section Files Required for Compilation
2. Allocate the proper memory sections in the driver's memory map header file ("<Module>"_MemMap.h) and linker command file. For more details refer to section Sections to be defined in <Module>_MemMap.h
3. Compile & build the module with all the dependent modules. For more details refer to section Building the Driver

Chapter 9

External assumptions for driver

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

External Assumption Req ID	External Assumption Text
SWS_Mcu_00244	If the register can affect several hardware modules and if it is an I/O register, it shall be initialised by the PORT driver. Note: These registers are not unde MCU's coverage
SWS_Mcu_00246	One-time writable registers that require initialisation directly after reset shall be initialised by the startup code. Note: This requierement refers to the start-up code
SWS_Mcu_00247	All other registers not mentioned before shall be initialised by the start-up code. Note: This requierement refers to the start-up code
SWS_Mcu_00136	The MCU module's environment shall call the function Mcu_InitRam↩ Section only after the MCU module has been initialized using the function Mcu_Init.
SWS_Mcu_00139	The MCU module's environment shall only call the function Mcu_InitClock after the MCU module has been initialized using the function Mcu_Init.
SWS_Mcu_00141	The function Mcu_DistributePllClock shall remove the current clock source (for example internal oscillator clock) from MCU clock distribution.
SWS_Mcu_00142	If the function Mcu_DistributePllClock is called before PLL has locked, this function shall return E_NOT_OK immediately, without any further action.
SWS_Mcu_00145	The MCU module's environment shall only call the function Mcu_Perform↩ Reset after the MCU module has been initialized by the function Mcu_Init.
SWS_Mcu_00148	The MCU module's environment shall only call the function Mcu_SetMode after the MCU module has been initialized by the function Mcu_Init.
EA_RTD_00071	If interrupts are locked, a centralized function pair to lock and unlock interrupts shall be used.
EA_RTD_00080	The integrator shall assure the execution of code from system RAM when flash memory configurations need to be change (i.e. PFCR control fields of PFLASH memory need to be change) .
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_00086	The integrator shall ensure that the following Mcu functions (Mcu_Init↩ Clock, Mcu_DistributePllClock, Mcu_InitRamSection) are not interrupted during their execution.

External assumptions for driver

External Assumption Req ID	External Assumption Text
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.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.