Integration Manual

for S32K3XX CAN Driver

Document Number: IM34CANASRR21-11 Rev0000R3.0.0 Rev. 1.0

1 Revision History 2

2 Introduction 3

2.1 Supported Derivatives . . . 3

2.2 Overview . . . 4

2.3 About This Manual . . . 5

2.4 Acronyms and Definitions . . . 6

2.5 Reference List . . . 6

3 Building the driver 7

3.1 Build Options . . . 7

3.1.1 GCC Compiler/Assembler/Linker Options . . . 8

3.1.2 DIAB Compiler/Assembler/Linker Options . . . 10

3.1.3 GHS Compiler/Assembler/Linker Options . . . 12

3.1.4 IAR Compiler/Assembler/Linker Options . . . 14

3.2 Files required for compilation . . . 16

3.3 Setting up the plugins . . . 17

3.3.1 Location of various files inside the CAN module folder . . . 17

4 Function calls to module 18

4.1 Function Calls during Start-up . . . 18

4.2 Function Calls during Shutdown . . . 18

4.3 Function Calls during Wake-up . . . 18

5 Module requirements 19

5.1 Exclusive areas to be defined in BSW scheduler . . . 19

5.2 Exclusive areas not available on this platform . . . 28

5.3 Peripheral Hardware Requirements . . . 28

5.4 ISR to configure within AutosarOS - dependencies . . . 29

5.5 ISR Macro . . . 32

5.5.1 Without an Operating System . . . 32

5.5.2 With an Operating System . . . 33

5.6 Other AUTOSAR modules - dependencies . . . 33

5.7 Data Cache Restrictions . . . 34

5.8 User Mode support . . . 34

5.8.1 User Mode configuration in the module . . . 34

5.8.2 User Mode configuration in AutosarOS . . . 36

5.9 Multicore support . . . 37

6 Main API Requirements 39

6.1 Main function calls within BSW scheduler . . . 39

6.2 API Requirements . . . 40

6.3 Calls to Notification Functions, Callbacks, Callouts 40

7 Memory allocation **41**

7.1 Sections to be defined in Can_43_FLEXCAN_MemMap.h 41

7.2 Linker command file 42

8 Integration Steps **43**

9 External assumptions for driver **44**

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

- Supported Derivatives
- Overview
- About This Manual
- Acronyms and Definitions
- Reference List

This Integration Manual describes NXP Semiconductor AUTOSAR CAN for S32K3XX. AUTOSAR CAN driver configuration parameters and deviations from the specification are described in Driver chapter of this document. AUTOSAR CAN driver requirements and APIs are described in the AUTOSAR CAN driver software specification document.

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
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 CAN 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
		S32M27x Reference Manual, Rev.2, Draft A, 02/2023
3	Datasheet	S32K3xx Data Sheet, Rev. 6, 11/2022
		S32K396 Data Sheet, Rev. 1.1, 08/2022
		S32M2xx Data Sheet, Rev. 2 RC, 12/2022
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_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=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

Assembler Option	Description
-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

Compiler Option	Description
-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
-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_SUPPO↩ RT 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

Compiler Option	Description
–unsigned_chars	Let the type char be unsigned, like unsigned char
–unsigned_fields	Bitfelds 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 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_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=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. DWA↩ RF debug information will contain references to deleted functions that may break some third-party debuggers
-Llibrary_path	Points to library_path (the libraries location) for thumb2 to be used for linking
-larch	Link architecture specific library
-lstartup	Link run-time environment startup routines. The source code for 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_SUPPO↩ RT 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 (if assembler code) and link the CAN driver. 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 CAN Driver Files:

- Can_43_FLEXCAN_TS_T40D34M30I0R0\src\Can_43_FLEXCAN.c
- Can_43_FLEXCAN_TS_T40D34M30I0R0\src\Can_43_FLEXCAN_Ipw.c
- Can_43_FLEXCAN_TS_T40D34M30I0R0\src\Can_43_FLEXCAN_Ipw_Irq.c
- Can_43_FLEXCAN_TS_T40D34M30I0R0\src\FlexCAN_Ip.c
- Can_43_FLEXCAN_TS_T40D34M30I0R0\src\FlexCAN_Ip_Irq.c
- Can_43_FLEXCAN_TS_T40D34M30I0R0\src\FlexCAN_Ip_Hw_Access.c
- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\Can_43_FLEXCAN.h
- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\Can_43_FLEXCAN_Flexcan_Types.h
- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\Can_43_FLEXCAN_Ipw.h
- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\Can_43_FLEXCAN_Ipw_Irq.h
- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\Can_43_FLEXCAN_Ipw_Types.h
- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\Can_43_FLEXCAN_Irq.h

- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\FlexCAN_Ip.h
- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\FlexCAN_Ip_DeviceReg.h

- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\FlexCAN_Ip_Hw_Access.h
- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\FlexCAN_Ip_Irq.h
- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\FlexCAN_Ip_Types.h
- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\FlexCAN_Ip_Wrapper.h
- Can_43_FLEXCAN_TS_T40D34M30I0R0\include\FlexCAN_Ip_TrustedFunctions.h

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

- Can_43_FLEXCAN_Cfg.h
- FlexCAN_Ip_Cfg.h
- Can_43_FLEXCAN_Ipw_Cfg.h
- Can_43_FLEXCAN_[VariantName]_PBcfg.c
- FlexCAN_Ip_[VariantName]_PBcfg.c
- Can_43_FLEXCAN_Ipw_[VariantName]_PBcfg.c
- Can_43_FLEXCAN_[VariantName]_PBcfg.h
- FlexCAN_Ip_[VariantName]_PBcfg.h
- Can_43_FLEXCAN_Ipw_[VariantName]_PBcfg.h

Note

3.3 Setting up the plugins

The CAN driver was designed to be configured by using the EB Tresos Studio (versionEB tresos Studio 29.0.0 or later.)

3.3.1 Location of various files inside the CAN module folder

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

- Can_43_FLEXCAN_TS_T40D34M30I0R0\config\Can_43_FLEXCAN.xdm

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

- Can_43_FLEXCAN_TS_T40D34M30I0R0\autosar\Can_43_FLEXCAN_<subderivative_name>.epd

Chapter 4

Function calls to module

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

4.1 Function Calls during Start-up

The CAN module shall be initialized by Can_43_FLEXCAN_Init() service call during the start-up. API service Can_43_FLEXCAN_SetControllerMode(Can_Controller, CAN_CS_STARTED) shall be used for setting the C↩ AN controller to running mode.

/note Can driver don't enable FlexCAN instances clocks, in order the user must enable the clocks from Clock Module Configuration for the instances used. Pin settings are not related to Can driver or plugin configuration. GPIO pins used for connection of CAN physical layer have to be properly assigned to the IPV_FlexCAN module prior the CAN initialization.

4.2 Function Calls during Shutdown

The IPV_FlexCAN IP has many Low Power Modes:

• Freeze Mode This low power mode is entered when the HALT and FRZ bits in the MCR Register are asserted. Module ignores the Rx input pin and drives the Tx pin as recessive, stops the prescaler, thus halting all CAN protocol activities and grants write access to the Error Counters Register (ECR), which is read-only in other modes. Exit from this mode is done by negating the FRZ and HALT bits in the MCR Register or when the MCU is removed from Debug Mode /note It is not possible to exit from this mode by receiving a message on the Can bus.

• Module Disable Mode This low power mode is entered when the MDIS bit in the MCR Register is asserted. Module shuts down the clocks to the CAN Protocol Interface and Message Buffer Management sub-modules. Exit from this mode is done by negating the MDIS bit in the MCR Register.

/note It is not possible to exit from this mode by receiving a message on the Can bus.

4.3 Function Calls during Wake-up

Hardware does not support wake-up.

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

CAN_EXCLUSIVE_AREA_00 is used in function Can_43_FLEXCAN_DisableControllerInterrupts to protect the variable Can_u8DisableInterruptLevel.

CAN_EXCLUSIVE_AREA_01 is used in function Can_43_FLEXCAN_EnableControllerInterrupts to protect the variable Can_u8DisableInterruptLevel.

CAN_EXCLUSIVE_AREA_02 is used in function Can_43_FLEXCAN_SetBaudrate to protect the register CAN_MCR in FlexCAN_EnterFreezeMode.

CAN_EXCLUSIVE_AREA_02 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_MCR in FlexCAN_EnterFreezeMode.

CAN_EXCLUSIVE_AREA_02 is used in function Can_43_FLEXCAN_Init to protect the register CAN_MCR in FlexCAN_EnterFreezeMode.

CAN_EXCLUSIVE_AREA_02 is used in function Can_43_FLEXCAN_DeInit to protect the register CAN_MCR in FlexCAN_EnterFreezeMode.

CAN_EXCLUSIVE_AREA_02 is used in function DMA_Can_Callbackx to protect the register CAN_MCR in FlexCAN_EnterFreezeMode.

CAN_EXCLUSIVE_AREA_03 is used in function Can_43_FLEXCAN_ListenOnlyMode to protect the register CAN_MCR in FlexCAN_Enable.

CAN_EXCLUSIVE_AREA_03 is used in function Can_43_FLEXCAN_EnableControllerInterrupts to protect the register CAN_MCR in FlexCAN_Enable.

CAN_EXCLUSIVE_AREA_03 is used in function Can_43_FLEXCAN_DisableControllerInterrupts to protect the register CAN_MCR in FlexCAN_Enable.

CAN_EXCLUSIVE_AREA_03 is used in function Can_43_FLEXCAN_SetClockMode to protect the register CAN_MCR in FlexCAN_Enable.

CAN_EXCLUSIVE_AREA_04 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_MCR in FlexCAN_ExitFreezeMode.

CAN_EXCLUSIVE_AREA_04 is used in function DMA_Can_Callbackx to protect the register CAN_MCR in FlexCAN_ExitFreezeMode.

CAN_EXCLUSIVE_AREA_05 is used in function Can_43_FLEXCAN_ListenOnlyMode to protect the register CAN_MCR in FlexCAN_Disable.

CAN_EXCLUSIVE_AREA_05 is used in function Can_43_FLEXCAN_EnableControllerInterrupts to protect the register CAN_MCR in FlexCAN_Disable.

CAN_EXCLUSIVE_AREA_05 is used in function Can_43_FLEXCAN_DisableControllerInterrupts to protect the register CAN_MCR in FlexCAN_Disable.

CAN_EXCLUSIVE_AREA_05 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_MCR in FlexCAN_Disable.

CAN_EXCLUSIVE_AREA_05 is used in function Can_43_FLEXCAN_SetBaudrate to protect the register CAN_MCR in FlexCAN_Disable.

CAN_EXCLUSIVE_AREA_05 is used in function Can_43_FLEXCAN_Init to protect the register CAN_↩
MCR in FlexCAN_Disable.

CAN_EXCLUSIVE_AREA_05 is used in function Can_43_FLEXCAN_SetClockMode to protect the register CAN_MCR in FlexCAN_Disable.

CAN_EXCLUSIVE_AREA_05 is used in function Can_43_FLEXCAN_DeInit to protect the register CA↩
N_MCR in FlexCAN_Disable.

CAN_EXCLUSIVE_AREA_06 is used in function Can_43_FLEXCAN_EnableControllerInterrupts to protect the register CAN_MCR, CAN_CTRL1, CAN_CTRL2 in FlexCAN_SetErrIntCmd.

CAN_EXCLUSIVE_AREA_06 is used in function Can_43_FLEXCAN_DisableControllerInterrupts to protect the register CAN_MCR, CAN_CTRL1, CAN_CTRL2 in FlexCAN_SetErrIntCmd.

CAN_EXCLUSIVE_AREA_06 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_MCR, CAN_CTRL1, CAN_CTRL2 in FlexCAN_SetErrIntCmd.

CAN_EXCLUSIVE_AREA_07 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_MCR in FlexCAN_Ip_SetStartMode.

CAN_EXCLUSIVE_AREA_08 is used in function Can_43_FLEXCAN_Init to protect the register CAN_↩ MCR in FlexCAN_Ip_SetRxMaskType.

CAN_EXCLUSIVE_AREA_08 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_MCR in FlexCAN_Ip_SetRxMaskType.

CAN_EXCLUSIVE_AREA_10 is used in function Can_43_FLEXCAN_ListenOnlyMode to protect the register CAN_CTRL1 in FlexCAN_Ip_SetListenOnlyMode.

CAN_EXCLUSIVE_AREA_10 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_CTRL1 in FlexCAN_Ip_SetListenOnlyMode.

CAN_EXCLUSIVE_AREA_11 is used in function Can_43_FLEXCAN_AbortMb to protect the variable g_FlexCAN_u32ImaskBuff in FLEXCAN_ClearMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_11 is used in function Can_43_FLEXCAN_SetControllerMode to protect the variable g_FlexCAN_u32ImaskBuff in FLEXCAN_ClearMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_11 is used in function Can_43_FLEXCAN_ErrorIrqCallback to protect the variable g_FlexCAN_u32ImaskBuff in FLEXCAN_ClearMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_13 is used in function Can_43_FLEXCAN_Init to protect the register CAN_↩ MCR in FlexCAN_SetRxFifoFilter.

CAN_EXCLUSIVE_AREA_13 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_MCR in FlexCAN_SetRxFifoFilter.

CAN_EXCLUSIVE_AREA_14 is used in function Can_43_FLEXCAN_SetClockMode to protect the register CAN_CTRL1, CAN_EPRS in FlexCAN_Ip_SetBitrate.

CAN_EXCLUSIVE_AREA_14 is used in function Can_43_FLEXCAN_SetBaudrate to protect the register CAN_CTRL1, CAN_EPRS in FlexCAN_Ip_SetBitrate.

CAN_EXCLUSIVE_AREA_14 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_CTRL1, CAN_EPRS in FlexCAN_Ip_SetBitrate.

CAN_EXCLUSIVE_AREA_15 is used in function Can_43_FLEXCAN_SetClockMode to protect the register CAN_MCR, CAN_FDCTRL, CAN_EDCBT, CAN_EPRS in FlexCAN_Ip_SetBitrateCbt.

CAN_EXCLUSIVE_AREA_15 is used in function Can_43_FLEXCAN_SetBaudrate to protect the register CAN_MCR, CAN_FDCTRL, CAN_EDCBT, CAN_EPRS in FlexCAN_Ip_SetBitrateCbt.

CAN_EXCLUSIVE_AREA_15 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_MCR, CAN_FDCTRL, CAN_EDCBT, CAN_EPRS in FlexCAN_Ip_SetBitrateCbt.

CAN_EXCLUSIVE_AREA_16 is used in function Can_43_FLEXCAN_Init to protect the register CAN_↩ FDCTRL, CAN_ETDC in FlexCAN_Ip_SetTDCOffset.

CAN_EXCLUSIVE_AREA_16 is used in function Can_43_FLEXCAN_SetBaudrate to protect the register CAN_FDCTRL, CAN_ETDC in FlexCAN_Ip_SetTDCOffset.

CAN_EXCLUSIVE_AREA_16 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_FDCTRL, CAN_ETDC in FlexCAN_Ip_SetTDCOffset.

CAN_EXCLUSIVE_AREA_17 is used in function Can_43_FLEXCAN_Init to protect the register CAN_↩ CTRL2 in FlexCAN_Ip_SetTxArbitrationStartDelay.

CAN_EXCLUSIVE_AREA_17 is used in function Can_43_FLEXCAN_SetBaudrate to protect the register CAN_CTRL2 in FlexCAN_Ip_SetTxArbitrationStartDelay.

CAN_EXCLUSIVE_AREA_17 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_CTRL2 in FlexCAN_Ip_SetTxArbitrationStartDelay.

CAN_EXCLUSIVE_AREA_18 is used in function Can_43_FLEXCAN_Write to protect the variable g_↩ FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN0_ORED_0_31_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN0_ORED_32_63_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN0_ORED_64_95_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN1_ORED_0_31_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN1_ORED_32_63_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN1_ORED_64_95_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN2_ORED_0_31_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN2_ORED_32_63_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN2_ORED_64_95_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN3_ORED_0_31_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN3_ORED_32_63_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN4_ORED_0_31_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN4_ORED_32_63_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN5_ORED_0_31_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN5_ORED_32_63_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function Can_43_FLEXCAN_SetControllerMode to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_19 is used in function Can_43_FLEXCAN_Init to protect the register CAN_↩ CTRL2 in FlexCAN_ConfigTimestamp.

CAN_EXCLUSIVE_AREA_19 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_CTRL2 in FlexCAN_ConfigTimestamp.

Exclusive Areas implemented in Low level driver layer (IPL)
CAN_EXCLUSIVE_AREA_02 is used in function FlexCAN_EnterFreezeMode to protect the updates for:

- CAN_MCR register

  CAN_EXCLUSIVE_AREA_03 is used in function FlexCAN_Enable to protect the updates for:

- CAN_MCR register

  CAN_EXCLUSIVE_AREA_04 is used in function FlexCAN_ExitFreezeMode to protect the updates for:

- CAN_MCR register

  CAN_EXCLUSIVE_AREA_05 is used in function FlexCAN_Disable to protect the updates for:

- CAN_MCR register

  CAN_EXCLUSIVE_AREA_06 is used in function FlexCAN_SetErrIntCmd to protect the updates for:

- CAN_MCR, CAN_CTRL1, CAN_CTRL2 register

  CAN_EXCLUSIVE_AREA_07 is used in function FlexCAN_Ip_SetStartMode to protect the updates for:

- CAN_MCR register

  CAN_EXCLUSIVE_AREA_08 is used in function FlexCAN_Ip_SetRxMaskType to protect the updates for:

- CAN_MCR register

  CAN_EXCLUSIVE_AREA_09 is used in function FlexCAN_Ip_ClearTDCFail to protect the updates for:

- CAN_FDCTRL, CAN_ETDC register

  CAN_EXCLUSIVE_AREA_10 is used in function FlexCAN_Ip_SetListenOnlyMode to protect the updates for:

- CAN_CTRL1 register

  CAN_EXCLUSIVE_AREA_11 is used in function FLEXCAN_ClearMsgBuffIntCmd to protect the updates for:

- g_FlexCAN_u32ImaskBuff variable

  CAN_EXCLUSIVE_AREA_13 is used in function FlexCAN_SetRxFifoFilter to protect the updates for:

- CAN_MCR register

  CAN_EXCLUSIVE_AREA_14 is used in function FlexCAN_Ip_SetBitrate to protect the updates for:

- CAN_CTRL1, CAN_EPRS register

  CAN_EXCLUSIVE_AREA_15 is used in function FlexCAN_Ip_SetBitrateCbt to protect the updates for:

- CAN_MCR, CAN_FDCTRL, CAN_EDCBT, CAN_EPRS register

  CAN_EXCLUSIVE_AREA_16 is used in function FlexCAN_Ip_SetTDCOffset to protect the updates for:

- CAN_FDCTRL, CAN_ETDC register

  CAN_EXCLUSIVE_AREA_17 is used in function FlexCAN_Ip_SetTxArbitrationStartDelay to protect the updates for:

- CAN_CTRL2 register

  CAN_EXCLUSIVE_AREA_18 is used in function FlexCAN_SetMsgBuffIntCmd to protect the updates for:

- g_FlexCAN_u32ImaskBuff variable

  CAN_EXCLUSIVE_AREA_19 is used in function FlexCAN_ConfigTimestamp to protect the updates for:

- CAN_CTRL2 register

  CAN_EXCLUSIVE_AREA_20 is used in function FlexCAN_Ip_ManualBusOffRecovery to protect the updates for:

- CAN_CTRL1 register

Critical Region Exclusive Matrix Below is the table depicting the exclusivity between different critical region IDs from the CAN driver. If there is an “X” in a table, it means that those 2 critical regions cannot interrupt each other.

The critical regions from Tx/Rx interrupts are grouped in “ISRs Critical Regions (composed diagram)”. If an exclusive area is “exclusive” with the composed “ISRs Critical Regions (composed diagram)” group, it means that it is exclusive with each one of the ISR critical regions.

Table 5.1 Exclusive Areas

#	AREA_00	AREA_01	AREA_02	AREA_03	AREA_04	AREA_05	AREA_06	AREA_07	AREA_08	AREA_09	AREA_10	AREA_11	AREA_13	AREA_14	AREA_15	AREA_16	AREA_17	AREA_18	AREA_19	AREA_20	ISRs Critical Regions (composed diagram)
AREA_00		x																			
AREA_01	x																				
AREA_02				x	x	x	x	x	x				x		x						
AREA_03			x		x	x	x	x	x				x		x						
AREA_04			x	x		x	x	x	x				x		x						
AREA_05			x	x	x		x	x	x				x		x						

#	AREA_00	AREA_01	AREA_02	AREA_03	AREA_04	AREA_05	AREA_06	AREA_07	AREA_08	AREA_09	AREA_10	AREA_11	AREA_13	AREA_14	AREA_15	AREA_16	AREA_17	AREA_18	AREA_19	AREA_20	ISRs Critical Regions (composed diagram)
AREA_06			x	x	x	x		x	x		x		x	x	x		x		x	x	
AREA_07			x	x	x	x	x		x				x		x						
AREA_08			x	x	x	x	x	x					x		x						
AREA_09															x	x					
AREA_10							x							x						x	
AREA_11																		x			

#	AREA_00	AREA_01	AREA_02	AREA_03	AREA_04	AREA_05	AREA_06	AREA_07	AREA_08	AREA_09	AREA_10	AREA_11	AREA_13	AREA_14	AREA_15	AREA_16	AREA_17	AREA_18	AREA_19	AREA_20	ISRs Critical Regions (composed diagram)
AREA_13			x	x	x	x	x	x	x						x						
AREA_14							x				x				x					x	
AREA_15			x	x	x	x	x	x	x	x			x	x		x					
AREA_16										x					x						
AREA_17							x												x		
AREA_18												x									x

#	**AREA_00**	**AREA_01**	**AREA_02**	**AREA_03**	**AREA_04**	**AREA_05**	**AREA_06**	**AREA_07**	**AREA_08**	**AREA_09**	**AREA_10**	**AREA_11**	**AREA_13**	**AREA_14**	**AREA_15**	**AREA_16**	**AREA_17**	**AREA_18**	**AREA_19**	**AREA_20**	**ISRs Critical Regions (composed diagram)**
AREA_19							x										x				
AREA_20							x				x			x							
ISRs Critical Regions (composed diagram)																		x			

5.2 Exclusive areas not available on this platform

CAN_EXCLUSIVE_AREA_12 is not available.

5.3 Peripheral Hardware Requirements

peripheral hardware requirements template.

5.4 ISR to configure within AutosarOS - dependencies

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

- Table with interrupts for S32K310 and S32K311

ISR Name	Hardware Interrupt Vector
CAN0__ORED__IRQHandler	**109**
CAN0__ORED__0__31__MB__IRQHandler	**110**
CAN0__ORED__32__63__MB__IRQHandler	**111**
CAN1__ORED__IRQHandler	**113**
CAN1__ORED__0__31__MB__IRQHandler	**114**
CAN1__ORED__32__63__MB__IRQHandler	**115**
CAN2__ORED__IRQHandler	**116**
CAN2__ORED__0__31__MB__IRQHandler	**117**
CAN2__ORED__32__63__MB__IRQHandler	**118**

- **Table with interrupts for S32K312**

ISR Name	Hardware Interrupt Vector
CAN0__ORED__IRQHandler	**109**
CAN0__ORED__0__31__MB__IRQHandler	**110**
CAN0__ORED__32__63__MB__IRQHandler	**111**
CAN1__ORED__IRQHandler	**113**
CAN1__ORED__0__31__MB__IRQHandler	**114**
CAN1__ORED__32__63__MB__IRQHandler	**115**
CAN2__ORED__IRQHandler	**116**
CAN2__ORED__0__31__MB__IRQHandler	**117**
CAN2__ORED__32__63__MB__IRQHandler	**118**
CAN3__ORED__IRQHandler	**119**
CAN3__ORED__0__31__MB__IRQHandler	**120**
CAN4__ORED__IRQHandler	**121**
CAN4__ORED__0__31__MB__IRQHandler	**122**
CAN5__ORED__IRQHandler	**123**
CAN5__ORED__0__31__MB__IRQHandler	**124**

- **Table with interrupts for S32K314, S32K324, S32K344**

ISR Name	Hardware Interrupt Vector
CAN0__ORED__IRQHandler	**109**
CAN0__ORED__0__31__MB__IRQHandler	**110**
CAN0__ORED__32__63__MB__IRQHandler	**111**
CAN0__ORED__64__95__MB__IRQHandler	**112**
CAN1__ORED__IRQHandler	**113**

ISR Name	Hardware Interrupt Vector
CAN1_ORED_0_31_MB_IRQHandler	114
CAN1_ORED_32_63_MB_IRQHandler	115
CAN2_ORED_IRQHandler	116
CAN2_ORED_0_31_MB_IRQHandler	117
CAN2_ORED_32_63_MB_IRQHandler	118
CAN3_ORED_IRQHandler	119
CAN3_ORED_0_31_MB_IRQHandler	120
CAN4_ORED_IRQHandler	121
CAN4_ORED_0_31_MB_IRQHandler	122
CAN5_ORED_IRQHandler	123
CAN5_ORED_0_31_MB_IRQHandler	124

- Table with interrupts for S32K322, S32K341, S32K342

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	109
CAN0_ORED_0_31_MB_IRQHandler	110
CAN0_ORED_32_63_MB_IRQHandler	111
CAN1_ORED_IRQHandler	113
CAN1_ORED_0_31_MB_IRQHandler	114
CAN1_ORED_32_63_MB_IRQHandler	115
CAN2_ORED_IRQHandler	116
CAN2_ORED_0_31_MB_IRQHandler	117

- Table with interrupts for S32K394 and S32K396

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	109
CAN0_ORED_0_31_MB_IRQHandler	110
CAN0_ORED_32_63_MB_IRQHandler	111
CAN0_ORED_64_95_MB_IRQHandler	112
CAN1_ORED_IRQHandler	113
CAN1_ORED_0_31_MB_IRQHandler	114
CAN1_ORED_32_63_MB_IRQHandler	115
CAN1_ORED_64_95_MB_IRQHandler	129
CAN2_ORED_IRQHandler	116
CAN2_ORED_0_31_MB_IRQHandler	117
CAN2_ORED_32_63_MB_IRQHandler	118
CAN2_ORED_64_95_MB_IRQHandler	130
CAN3_ORED_IRQHandler	119
CAN3_ORED_0_31_MB_IRQHandler	120
CAN3_ORED_32_63_MB_IRQHandler	131
CAN4_ORED_IRQHandler	121
CAN4_ORED_0_31_MB_IRQHandler	122

ISR Name	Hardware Interrupt Vector
CAN4_ORED_32_63_MB_IRQHandler	132
CAN5_ORED_IRQHandler	123
CAN5_ORED_0_31_MB_IRQHandler	124
CAN5_ORED_32_63_MB_IRQHandler	133

- Table with interrupts for S32K328, S32K338, S32K348, S32K358

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	109
CAN0_ORED_0_31_MB_IRQHandler	110
CAN0_ORED_32_63_MB_IRQHandler	111
CAN0_ORED_64_95_MB_IRQHandler	112
CAN1_ORED_IRQHandler	113
CAN1_ORED_0_31_MB_IRQHandler	114
CAN1_ORED_32_63_MB_IRQHandler	115
CAN1_ORED_64_95_MB_IRQHandler	129
CAN2_ORED_IRQHandler	116
CAN2_ORED_0_31_MB_IRQHandler	117
CAN2_ORED_32_63_MB_IRQHandler	118
CAN2_ORED_64_95_MB_IRQHandler	130
CAN3_ORED_IRQHandler	119
CAN3_ORED_0_31_MB_IRQHandler	120
CAN3_ORED_32_63_MB_IRQHandler	131
CAN4_ORED_IRQHandler	121
CAN4_ORED_0_31_MB_IRQHandler	122
CAN4_ORED_32_63_MB_IRQHandler	132
CAN5_ORED_IRQHandler	123
CAN5_ORED_0_31_MB_IRQHandler	124
CAN5_ORED_32_63_MB_IRQHandler	133
CAN6_ORED_IRQHandler	125
CAN6_ORED_0_31_MB_IRQHandler	126
CAN6_ORED_32_63_MB_IRQHandler	134
CAN7_ORED_IRQHandler	127
CAN7_ORED_0_31_MB_IRQHandler	128
CAN7_ORED_32_63_MB_IRQHandler	135

- Table with interrupts for S32K388

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	109
CAN0_ORED_0_31_MB_IRQHandler	110
CAN0_ORED_32_63_MB_IRQHandler	111
CAN0_ORED_64_95_MB_IRQHandler	112
CAN1_ORED_IRQHandler	113

ISR Name	Hardware Interrupt Vector
CAN1_ORED_0_31_MB_IRQHandler	114
CAN1_ORED_32_63_MB_IRQHandler	115
CAN1_ORED_64_95_MB_IRQHandler	129
CAN2_ORED_IRQHandler	116
CAN2_ORED_0_31_MB_IRQHandler	117
CAN2_ORED_32_63_MB_IRQHandler	118
CAN2_ORED_64_95_MB_IRQHandler	130
CAN3_ORED_IRQHandler	119
CAN3_ORED_0_31_MB_IRQHandler	120
CAN3_ORED_32_63_MB_IRQHandler	131
CAN4_ORED_IRQHandler	121
CAN4_ORED_0_31_MB_IRQHandler	122
CAN4_ORED_32_63_MB_IRQHandler	132
CAN5_ORED_IRQHandler	123
CAN5_ORED_0_31_MB_IRQHandler	124
CAN5_ORED_32_63_MB_IRQHandler	133
CAN6_ORED_IRQHandler	125
CAN6_ORED_0_31_MB_IRQHandler	126
CAN6_ORED_32_63_MB_IRQHandler	134
CAN7_ORED_IRQHandler	127
CAN7_ORED_0_31_MB_IRQHandler	128
CAN7_ORED_32_63_MB_IRQHandler	135

- **Table with interrupts for S32M274 and S32M276**

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	109
CAN0_ORED_0_31_MB_IRQHandler	110
CAN0_ORED_32_63_MB_IRQHandler	111
CAN1_ORED_IRQHandler	113
CAN1_ORED_0_31_MB_IRQHandler	114
CAN1_ORED_32_63_MB_IRQHandler	115
CAN2_ORED_IRQHandler	116
CAN2_ORED_0_31_MB_IRQHandler	117
CAN2_ORED_32_63_MB_IRQHandler	118

5.5 ISR Macro

RTD drivers use the ISR macro to define the functions that will process hardware interrupts. Depending on whether the OS is used or not, this macro can have different definitions.

5.5.1 Without an Operating System

The macro *USING_OS_AUTOSAROS* must not be defined.

5.5.1.1 Using Software Vector Mode

The macro *USE_SW_VECTOR_MODE* must be defined and the ISR macro is defined as:

#define ISR(IsrName) void IsrName(void)

In this case, the drivers' interrupt handlers are normal C functions and their prologue/epilogue will handle the context save and restore.

5.5.1.2 Using Hardware Vector Mode

The macro *USE_SW_VECTOR_MODE* must not defined and the ISR macro is defined as:

#define ISR(IsrName) INTERRUPT_FUNC void IsrName(void)

In this case, the drivers' interrupt handlers must also handle the context save and restore.

5.5.2 With an Operating System

Please refer to your OS documentation for description of the ISR macro.

5.6 Other AUTOSAR modules - dependencies

- **Base**: The Base module contains the common files/definitions needed by all MCAL modules.
- **Mcu**: The Mcu driver provides services for basic microcontroller initialization, power down functionality, reset and microcontroller specific functions required by other MCAL software modules. The clocks need to be initialized prior to using the Can driver. The clock frequency may affect the Can bit rate, the transmitting or receiving a Can frame process.\P
- **Port**: The Port module is used to configure the port pins with the needed modes, before they are used by the Can module.
- **EcuC** : The ECUC module is used for ECU configuration. Can modules need ECUC to retrieve the variant information.
- **Det** The Det module is used for enabling Default error detection. The API function used is Det_Report↩ Error(). The activation / deactivation of Default error detection is configurable using the 'CanDevErrorDetect' configuration parameter.
- **Resource**: Sub-Derivative model is selected from Resource configuration.
- **Rte**: The Rte module is needed for implementing data consistency of exclusive areas that are used by Can module.
- **EcuM**: This module is used for processing the Wakeup notifications of CAN. Whenever the module is in 'Sleep' mode and a wakeup event occurs, it is reported to EcuM through the EcuM_CheckWakeupEvent() API.
- **Mcl**: This module is used for enabling DMA channels for Can controllers.

5.7 Data Cache Restrictions

In the DMA transfer mode, DMA transfers may issue cache coherency problems. To avoid possible coherency issues when D-CACHE is enabled, the user shall ensure that the buffers used as TCD source and destination are allocated in the NON-CACHEABLE area (by means of Can_43_FLEXCAN_Memmap).

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 Can Driver can be run in user mode as long as the configuration parameter CanEnableUserModeSupport is enabled and MCAL_ENABLE_USER_MODE_SU↩PPORT is defined and call the following functions as trusted functions:

Function syntax	Description	Available via
void FlexCAN_ClearOutput↩LegacyFIFO(FLEXCAN_Type * base)	Clears Legacy RxFifo message buffers	FlexCAN_Ip_Trusted↩Functions.h
void FlexCAN_ClearOutput↩EnhanceFIFO(FLEXCAN_Type * base)	Clears Enhanced RxFifo message buffers	FlexCAN_Ip_Trusted↩Functions.h
Flexcan_Ip_StatusType Flex↩CAN_Ip_Init_Privileged(uint8 Flexcan_Ip_u8Instance, Flexcan↩_Ip_StateType * Flexcan_Ip_p↩State, const Flexcan_Ip_Config↩Type * Flexcan_Ip_pData)	Initializes the FlexCAN peripheral	FlexCAN_Ip_Trusted↩Functions.h
Flexcan_Ip_StatusType Flex↩CAN_Ip_ConfigRxFifo_↩Privileged(uint8 instance, Flexcan_Ip_RxFifoIdElement↩FormatType id_format, const Flexcan_Ip_IdTableType * id_↩filter_table)	FlexCAN Rx Legacy FIFO filter configuration	FlexCAN_Ip_Trusted↩Functions.h
Flexcan_Ip_StatusType FlexC↩AN_Ip_ConfigEnhancedRxFifo↩_Privileged(uint8 instance, const Flexcan_Ip_EnhancedIdTable↩Type * id_filter_table)	FlexCAN Enhanced Rx FIFO filter configuration	FlexCAN_Ip_Trusted↩Functions.h
Flexcan_Ip_StatusType FlexC↩AN_Ip_MainFunctionBusOff_↩Privileged(uint8 instance)	Check a bus-off event	FlexCAN_Ip_Trusted↩Functions.h
boolean FlexCAN_Ip_GetStop↩Mode_Privileged(uint8 instance)	Check if the FlexCAN instance is STOPPED	FlexCAN_Ip_Trusted↩Functions.h
boolean FlexCAN_Ip_GetStart↩Mode_Privileged(uint8 instance)	Check if the FlexCAN instance is STARTED	FlexCAN_Ip_Trusted↩Functions.h

Function syntax	Description	Available via
Flexcan_Ip_StatusType FlexCAN_Ip_EnterFreezeMode_Privileged(uint8 instance)	Enter FlexCAN Module in Freeze Mode	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_ExitFreezeMode_Privileged(uint8 instance)	Exit FlexCAN Module from Freeze Mode	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetRxFifoGlobalMask_Privileged(uint8 instance, uint32 mask)	Set RxFifo Global Mask	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_Deinit_Privileged(uint8 instance)	DeInitilize the FlexCAN instance driver	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetStartMode_Privileged(uint8 instance)	Set the FlexCAN instance in START mode	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetStopMode_Privileged(uint8 instance)	Set the FlexCAN instance in STOP mode	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetRxMaskType_Privileged(uint8 instance, Flexcan_Ip_RxMaskType type)	Set RX masking type	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetRxMb14Mask_Privileged(uint8 instance, uint32 mask)	Set Rx14Mask filter for message buffer 14	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetRxMb15Mask_Privileged(uint8 instance, uint32 mask)	Set Rx14Mask filter for message buffer 15	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetRxIndividualMask_Privileged(uint8 instance, uint8 mb_idx, uint32 mask)	Sets the FlexCAN Rx individual mask	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetRxMbGlobalMask_Privileged(uint8 instance, uint32 mask)	Sets the FlexCAN Rx Message Buffer Global mask	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetBitrate_Privileged(uint8 instance, const Flexcan_Ip_TimeSegmentType * bitrate, boolean enhExt)	Sets the FlexCAN bit rate for standard frames or the arbitration phase of FD frames	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetBitrateCbt_Privileged(uint8 instance, const Flexcan_Ip_TimeSegmentType * bitrate, boolean bitRateSwitch)	Sets the FlexCAN bit rate for the data phase of FD frames	FlexCAN_Ip_TrustedFunctions.h

Function syntax	Description	Available via
Flexcan_Ip_StatusType FlexCAN_Ip_SetTxArbitrationStartDelay_Privileged(uint8 instance, uint8 value)	This function will set how many CAN bits the Tx arbitration process start point can	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetTDCOffset_Privileged(uint8 instance, boolean enable, uint8 offset)	Enables/Disables the Transceiver Delay Compensation feature and sets the Transceiver Delay Compensation Offset	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_EnableInterrupts_Privileged(uint8 u8Instance)	Enable all mb interrupts configured	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_DisableInterrupts_Privileged(uint8 u8Instance)	Disable all mb interrupts configured	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetErrorInt_Privileged(uint8 u8Instance, Flexcan_Ip_ErrorIntType type, boolean enable)	Enable\Disable Error or BusOff Interrupt	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_SetListenOnlyMode_Privileged(uint8 instance, const boolean enable)	Set FlexCAN Listen Only	FlexCAN_Ip_TrustedFunctions.h
Flexcan_Ip_StatusType FlexCAN_Ip_ConfigTimeStamp_Privileged(uint8 instance, const Flexcan_Ip_TimeStampConfigType * time_stamp)	Set FlexCAN Config Timestamp	FlexCAN_Ip_TrustedFunctions.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.

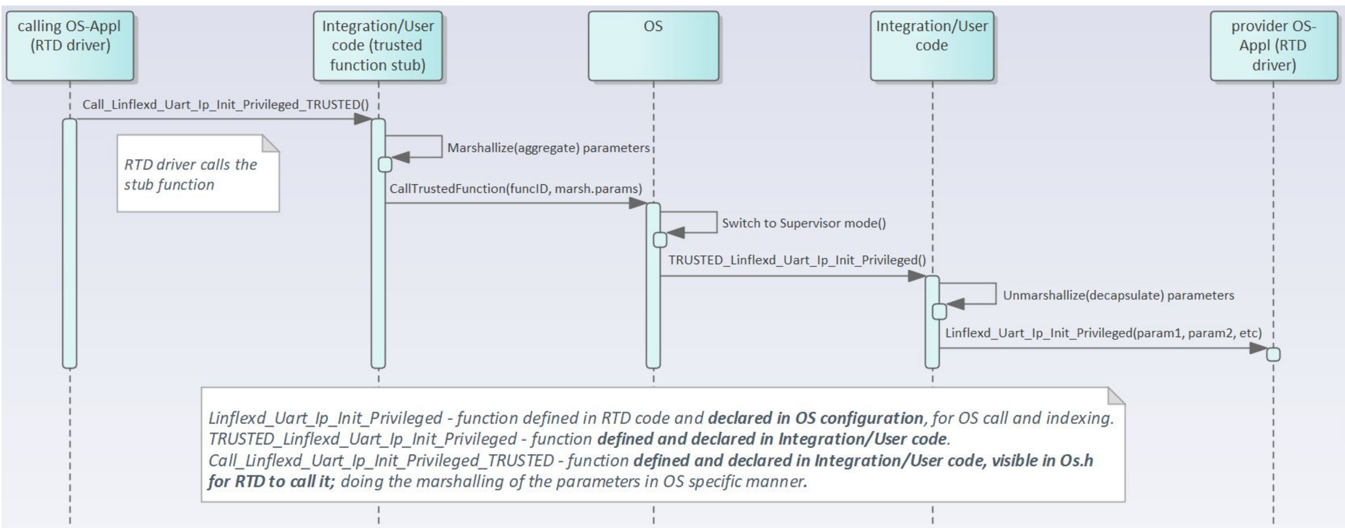

Figure 5.1 Example sequence chart for calling `Linflexd_Uart_Ip_Init_Privileged` as trusted function

5.9 Multicore support

The Can implements the "Autosar 4.7 MCAL Multicore Distribution" according to type II, in which the mappable element is set to HW Unit. For additional details, please refer to AUTOSAR_EXP_BSWDistributionGuide.
The Can and the mappable elements can be allocated to zero, one or several ECUC partitions, by means of "↩ CanEcucPartionRef". If the Can is mapped to zero ECUC partitions, the Can behavior reverts to single-core implementation, similar to previous Autosar versions. If the Can is mapped to one or more ECUC partitions, the Can enforces the following multi-core assumptions:
The Can assumes there is a single EcucPartition allocated per core. Internally, the module will use the Core ID returned by GetCoreID API to reference the appropriate global data and configuration elements.
The Can assumes the EcucCoreIDs are defined in a compact/consecutive order, starting from zero. The rationale is that the number of EcucPartitions is used for dimensioning the Can internal variables and the EcucCoreIDs are used for indexing those variables. (AR-86601 Zero based and dense IDs for OS-Cores and OSApplications) The Can assumes that initialization is performed on each core, Can_43_FLEXCAN_Init() is called separately for each core, using a different configguration structure. (Type II) The Can initialization expects the upper layer will pass the correct initialization pointer, specific to the partition in which the driver is to be used. For example: EcucPartition_1

is assigned to CoreID 1; Can_43_FLEXCAN_Init function will be called with Can_43_FLEXCAN_Config_Ecuc↩
Partition_1 configuration structure, on Core 1.
The Can will check upon each API call if the requested resource is configured to be available on the current core, if DET error reporting is enabled.
The Can requires that all variables in NonCacheable MemMap sections be allocated accordingly, to avoid data corruption in multicore context.
The Can assumes that RTE module implements the EXCLUSIVE AREAS to be core-aware only. The rationale is that the module implementation ensures data integrity by separating the mappable elements for different cores already, thus implementing the EXCLUSIVE AREAS in a blocking manner (ex: spin-lock) on a multicore scope, might affect the performance of the drivers on the two cores, although they might access separate HW elements. For single-core scope, the EXCLUSIVE AREAS keep the same purpose as on previous AUTOSAR implementations. (∗ - to be updated per Can usecase, to be detailed/removed if some modules require such kind of functionality for critical features which cannot be atomically shared among cores).
The Can assumes that each interrupt is routed by the system only to the core on which is supposed to be serviced.
The configuration structure name shall be available in the caller scope of Can_43_FLEXCAN_Init function by being declared with EXTERN, according to its generated name.

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

CAN Driver support 4 main functions that can be configured to be scheduled by BSW scheduler: • *void Can_43↩ _FLEXCAN_MainFunction_Write(void)*
- *void Can_43_FLEXCAN_MainFucntion_Read(void)*
- *void Can_43_FLEXCAN_MainFunction_BusOff(void)*
- *void Can_43_FLEXCAN_MainFunction_Mode(void)*

These Autosar APIs are scheduled if these 3 events are configured to be in “Polling” mode by the following parameters: • CanTxProcessing
#define CAN_43_FLEXCAN_TX_POLLING_SUPPORT (STD_ON)
- CanRxProcessing

#define CAN_43_FLEXCAN_RX_POLLING_SUPPORT (STD_ON)
- CanBusoffProcessing

#define CAN_43_FLEXCAN_BUSOFF_POLLING_SUPPORT (STD_ON)
The period for polling is configured by the following 4 parameters: • CanMainFunctionWritePeriod
#define CAN_43_FLEXCAN_MAINFUNCTION_PERIOD_WRITE (uint32)0.0010U
- CanMainFunctionReadPeriod

#define CAN_43_FLEXCAN_MAINFUNCTION_PERIOD_READ (uint32)0.0010U
- CanMainFunctionBusoffPeriod

#define CAN_43_FLEXCAN_MAINFUNCTION_PERIOD_BUSOFF (uint32)0.0010U
- CanMainFunctionModePeriod

#define CAN_43_FLEXCAN_MAINFUNCTION_MODE_PERIOD (uint32)0.0010U

Note

> A configuration for an hardware unit can be possible in such a way that one controller will handle events by interrupts and another by polling method.

6.2 API Requirements

SWS_Can_00360,SWS_Can_00361,SWS_Can_00362,SWS_Can_00363,SWS_Can_00228,SWS_Can_00112,SWS_Can_00185,SWS_Can_00235,

6.3 Calls to Notification Functions, Callbacks, Callouts

Call-back Notifications
The CAN stack provides the following call-back notifications:
- *CanIf_TxConfirmation*: This CAN Interface call-back function is called when a CAN message has been transmitted. *void CanIf_TxConfirmation(PduIdType CanTxPduId)*
- *CanIf_RxIndication*: This CAN Interface call-back function is called when valid CAN message is received. *void CanIf_RxIndication(const Can_HwType* Mailbox, const PduInfoType* PduInfoPtr)*
- *CanIf_ControllerBusOff*: This CAN Interface call-back function is called when the CAN controller reached the bus-off state (see CAN specification for further details). *void CanIf_ControllerBusOff(uint8 Controller)*
- *CanIf_ControllerErrorStatePassive*: This CAN Interface call-back function is called when the CAN driver detected the error state passive of CAN controller (see CAN specification for further details). *void CanIf_ControllerErrorStatePassive(uint8 ControllerId, uint16 RxErrorCounter, uint16 TxErrorCounter)*
- *CanIf_ErrorNotification*: This CAN Interface call-back function is called when the CAN driver detected the error state of CAN controller which is predefined in Can_ErrorType (see CAN specification for further details). *void CanIf_ErrorNotification(uint8 ControllerId, Can_ErrorType Can_ErrorType)*

User Notification

Chapter 7

Memory allocation

- Sections to be defined in Can_43_FLEXCAN_MemMap.h
- Linker command file

7.1 Sections to be defined in Can_43_FLEXCAN_MemMap.h

Section name	Type of section	Description
CAN_43_FLEXCAN_START_SEC_C↩ONFIG_DATA_UNSPECIFIED	Configuration Data	Start of Memory Section for Config Data
CAN_43_FLEXCAN_STOP_SEC_CO↩NFIG_DATA_UNSPECIFIED	Configuration Data	End of Memory Section for Config Data
CAN_43_FLEXCAN_START_SEC_C↩ODE	Code	Start of memory Section for Code
CAN_43_FLEXCAN_STOP_SEC_CO↩DE	Code	End of memory Section for Code
CAN_43_FLEXCAN_START_SEC_V↩AR_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 initialized with values after every reset.
CAN_43_FLEXCAN_STOP_SEC_VA↩R_CLEARED_UNSPECIFIED	Variables	End of above section.
CAN_43_FLEXCAN_START_SEC_V↩AR_CLEARED_UNSPECIFIED	Variables	These variables are never cleared and never initialized by start-up code.
CAN_43_FLEXCAN_STOP_SEC_VA↩R_CLEARED_UNSPECIFIED	Variables	End of above section.
CAN_43_FLEXCAN_START_SEC_C↩ONST_UNSPECIFIED	Constant Data	Used for constants
CAN_43_FLEXCAN_STOP_SEC_CO↩NST_UNSPECIFIED	Constant Data	End of above section.
CAN_43_FLEXCAN_START_SEC_V↩AR_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. Normally, this section is used to store descriptors and data buffers. This section must also be cache inhibited
CAN_43_FLEXCAN_STOP_SEC_VA↩R_CLEARED_UNSPECIFIED_NO_C↩ACHEABLE	Variables	End of the 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 CAN driver into the application.

External Assumption Req ID	External Assumption Text
SWS_Can_00436	Can_GeneralTypes.h shall contain all types and constants that are shared among the AUTOSAR CAN modules Can, CanIf and CanTrcv. Note↩ : Implemented in base
SWS_Can_00415	Name: Can_PduType Kind: Structure Elements: swPduHandle Type↩ : PduIdType Comment: – length Type: uint8 Comment: – id Type: Can↩ _IdType Comment: – sdu Type: uint8 Comment: – Description: This type unites PduId (swPduHandle), SduLength (length), SduData (sdu), and CanId (id) for any CAN L-SDU. Available via: Can_GeneralTypes.h Note: defined into Can_GeneralTypes.h included into Base module
SWS_Can_00416	Name: Can_IdType Kind: Type Derived from: Basetype: Variation uint32 Range: Standard: 0..0x400007FF: 0..0x400007FF Extended: 0..0xDFFFF↩ FFF: 0..0xDFFFFFFF Description: Represents the Identifier of an L-PDU. The two most significant bits specify the frame type: 00 CAN message with Standard CAN ID 01 CAN FD frame with Standard CAN ID 10 CAN message with Extended CAN ID 11 CAN FD frame with Extended CAN ID Available via: Can_GeneralTypes.h Note: defined into Can_General↩ Types.h included into Base module
SWS_Can_00429	Name: Can_HwHandleType Kind: Type Derived from: Basetype: Variation uint16: – uint8: – Range: Standard: 0..0x0FF: 0..0x0FF Extended: 0..0xFFFF: 0..0xFFFF Description: Represents the hardware object handles of a CAN hardware unit. For CAN hardware units with more than 255 HW objects use extended range. Available via: Can_GeneralTypes.h Note: defined into Can_GeneralTypes.h included into Base module
SWS_Can_00039	Range: CAN_BUSY: 0x02: transmit request could not be processed because no transmit object was available Description: Overlayed return value of Std_ReturnType for CAN driver API Can_Write() Available via: Can_↩ GeneralTypes.h Note: defined into Can_GeneralTypes.h included into Base module
SWS_Can_00024	The valid values that can be configured are hardware dependent. Therefore the rules and constraints can't be given in the standard. The configuration tool is responsible to do a static configuration checking, also regarding dependencies between modules (i.e. Port driver, MCU driver etc.)

External Assumption Req ID	External Assumption Text
SWS_Can_91013	Name: Can_ControllerStateType Kind: Enumeration Range: CAN_C↩ S_UNINIT: 0x00: CAN controller state UNINIT. CAN_CS_STARTED: 0x01: CAN controller state STARTED. CAN_CS_STOPPED: 0x02: C↩ AN controller state STOPPED. CAN_CS_SLEEP: 0x03: CAN controller state SLEEP. Description: States that are used by the several Controller↩ Mode functions. Available via: Can_GeneralTypes.h Note: defined into Can_GeneralTypes.h included into Base module
SWS_Can_91003	Name: Can_ErrorStateType Kind: Enumeration Range: CAN_ERRO↩ RSTATE_ACTIVE: –: The CAN controller takes fully part in communication. CAN_ERRORSTATE_PASSIVE: –: The CAN controller takes part in communication, but does not send active error frames. CAN_↩ ERRORSTATE_BUSOFF: –: The CAN controller does not take part in communication. Description: Error states of a CAN controller. Available via: Can_GeneralTypes.h Note: defined into Can_GeneralTypes.h included into Base module
SWS_CAN_00496	Name: Can_HwType Kind: Structure Elements: CanId Type: Can_Id↩ Type Comment: Standard/Extended CAN ID of CAN L-PDU Hoh Type: Can_HwHandleType Comment: ID of the corresponding Hardware Object Range ControllerId Type: uint8 Comment: ControllerId provided by CanIf clearly identify the corresponding controller Description: This type defines a data structure which clearly provides an Hardware Object Handle including its corresponding CAN Controller and therefore CanDrv as well as the specific CanId. Available via: Can_GeneralTypes.h
SWS_Can_91021	Name: Can_ErrorType Kind: Enumeration Range: CAN_ERROR_B↩ IT_MONITORING1: 0x01: A 0 was transmitted and a 1 was read back CAN_ERROR_BIT_MONITORING0: 0x02: A 1 was transmitted and a 0 was read back CAN_ERROR_BIT: 0x03: The HW reports a CAN bit error but cannot report distinguish between CAN_ERROR_BIT_M↩ ONITORING1 and CAN_ERROR_BIT_MONITORING0 CAN_ERR↩ OR_CHECK_ACK_FAILED: 0x04: Acknowledgement check failed CA↩ N_ERROR_ACK_DELIMITER: 0x05: Acknowledgement delimiter check failed CAN_ERROR_ARBITRATION_LOST: 0x06: The sender lost in arbitration. CAN_ERROR_OVERLOAD: 0x07: CAN overload detected via an overload frame. Indicates that the receive buffers of a receiver are full. CAN_ERROR_CHECK_FORM_FAILED: 0x08: Violations of the fixed frame format CAN_ERROR_CHECK_STUFFING_FAILED↩ : 0x09: Stuffing bits not as expected CAN_ERROR_CHECK_CRC_FA↩ ILED: 0xA: CRC failed CAN_ERROR_BUS_LOCK: 0xB: Bus lock (Bus is stuck to dominant level) Description: The enumeration represents a superset of CAN Error Types which typical CAN HW is able to report. That means not all CAN HW will be able to support the complete set. Available via: Can_GeneralTypes.h
SWS_CAN_91029	Name: Can_TimeStampType (draft) Kind: Structure Elements: nanoseconds Type: uint32 Comment: Nanoseconds part of the time seconds Type: uint32 Comment: Seconds part of the time Description: Variables of this type are used to express time stamps based on relative time.: Value range: ∗ Seconds: 0 .. 4.294.967.295 s (circa 136 years) ∗ Nanoseconds: 0 .. 999.↩ 999.999 ns: Tags: atp.Status=draft Available via: Can_GeneralTypes.h
EA_RTD_00001	The external application shall call Can_Init function only when the driver state is CAN_UNINIT and the state of all controllers is UNINIT.
EA_RTD_00002	The external application shall call Can_MainFunction_Write only after driver initialization.

External Assumption Req ID	External Assumption Text
EA_RTD_00003	The external application shall call Can_MainFunction_Read only after driver initialization.
EA_RTD_00004	The external application shall call Can_MainFunction_BusOff only after driver initialization.
EA_RTD_00005	The external application shall call Can_MainFunction_Wakeup only after driver initialization.
EA_RTD_00006	The external application shall call Can_SetControllerMode only after driver initialization.
EA_RTD_00007	The external application shall call Can_DisableControllerInterrupts function only after driver initialization.
EA_RTD_00008	The external application shall call Can_EnableControllerInterrupts function only after driver initialization.
EA_RTD_00009	The external application shall call Can_Write function only after driver initialization.
EA_RTD_00010	The external application shall assure that Can_Init does not preempt and is not preempted by any other CAN driver API excepting Can_GetVersion↩ Info. The external application shall assure that Can_Init does not preempt itself.
EA_RTD_00011	The external application shall assure that Can_MainFunction_Write does not preempt and is not preempted by any other CAN driver API exception Can_GetVersionInfo.The external application shall assure that Can↩ _MainFunction_Write does not preempt itself.
EA_RTD_00012	The external application shall assure that Can_MainFunction_Read does not preempt and is not preempted by any other CAN driver API exception Can_GetVersionInfo.The external application shall assure that Can↩ _MainFunction_Read does not preempt itself.
EA_RTD_00013	The external application shall assure that Can_MainFunction_BusOff does not preempt and is not preempted by any other CAN driver API exception Can_GetVersionInfo.The external application shall assure that Can↩ _MainFunction_BusOff does not preempt itself.
EA_RTD_00014	The external application shall assure that Can_SetControllerMode does not preempt and is not preempted by any other CAN driver API using the same controller parameter. The external application shall assure that Can_Set↩ ControllerMode does not preempt itself.
EA_RTD_00015	The external application shall assure that Can_DisableControllerInterrupts does not preempt and is not preempted by any other CAN driver API using the same controller parameter.
EA_RTD_00016	The external application shall assure that Can_EnableControllerInterrupts does not preempt and is not preempted by any other CAN driver API using the same controller parameter.
EA_RTD_00017	The external application shall assure that Can_Write does not preempt and is not preempted by any other CAN driver API using the same controller as the hardware handle parameter.The external application shall assure that Can_Write does not preempt itself for the same hardware handle parameter.
EA_RTD_00018	The external application shall call Can_ChangeBaudrate only when the CAN controller is in state STOPPED.
EA_RTD_00019	The external application shall call Can_Init function only when the driver state is CAN_UNINIT and the state of all controllers is UNINIT.

External Assumption Req ID	External Assumption Text
EA_RTD_00020	The external application shall assure that Can_CheckWakeup does not preempt and is not preempted by any other CAN driver API using the same controller parameter. The external application shall assure that Can_CheckWakeup does not preempt itself.
EA_RTD_00021	The external application shall call Can_CheckWakeup function only after driver initialization.
EA_RTD_00022	The external application shall call Can_MainFunction_Mode function only after driver initialization.
EA_RTD_00023	The external application shall call Can_Init function only when the driver state is CAN_UNINIT and the state of all controllers is UNINIT.
EA_RTD_00024	The external application shall call Can_SetControllerMode(CAN_T_START) only when the CAN controller is in state STOPPED.
EA_RTD_00025	The external application shall call Can_SetControllerMode(CAN_T_STOP) only when the CAN controller is in state STARTED or STOPPED.
EA_RTD_00026	The external application shall call Can_SetControllerMode(CAN_T_SLEEP) only when the CAN controller is in state SLEEP or STOPPED.
EA_RTD_00027	The external application shall call Can_SetControllerMode(CAN_T_WAKEUP) only when the CAN controller is in state SLEEP or STOPPED.
EA_RTD_00028	The external application shall assure that Can_ChangeBaudrate does not preempt and is not preempted by any other CAN driver API using the same controller parameter.The external application shall assure that Can_ChangeBaudrate does not preempt itself.The external application shal call Can_ChangeBaudrate only when the controller parameter is STOPPED.
EA_RTD_00029	The external application shall call Can_ChangeBaudrate only after driver initialization and when the configured controller is in the STOPPED state.
EA_RTD_00030	The external application shall assure that Can_CheckBaudrate does not preempt and is not preempted by any other CAN driver API using the same controller parameter.
EA_RTD_00031	The external application shall call Can_CheckBaudrate only after driver initialization.
EA_RTD_00071	If interrupts are locked, a centralized function pair to lock and unlock interrupts shall be used.
EA_RTD_00081	The integrator shall assure that <MSN>_Init() and <MSN>_DeInit() functions do not interrupt each other.
EA_RTD_00082	When caches are enabled and data buffers are allocated in cacheable memory regions the buffers involved in DMA transfer shall be aligned with both start and end to cache line size. Note: **Rationale**: This ensures that no other buffers/variables compete for the same cache lines.
EA_RTD_00092	The integrator shall allocate a single EcucPartition per core or the partition in which the Can is allocated shall be exclusively mapped to a core. Note: Internally, the Can will use the Core ID returned by GetCoreID API to reference the appropriate global data and configuration elements, that is why a core should reference only one configured partition.
EA_RTD_00093	The application shall define EcucCoreIDs in a compact/consecutive order, starting from zero.
EA_RTD_00094	When multicore support is enabled, the application shall call Can_Init() for each core, using the dedicated configuration pointer for that core.

External Assumption Req ID	External Assumption Text
EA_RTD_00096	The application shall pass the correct initialization pointer, specific to the partition in which the driver is to be used.
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_00109	Name: Can_TimeStampType Type: Structure Element: uint32 nanoseconds Nanoseconds part of the time uint32 seconds Seconds part of the time Description: Shall define time stamps types based on relative time. Value range: - Seconds: 0 .. 4.294.967.295 s (136 years) - Nanoseconds: 0 .. 999.999.999 ns Available via Can_GeneralTypes.h
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