Integration Manual

for S32K1_S32M24X CAN_43_FLEXCAN Driver

Document Number: IM2CAN_43_FLEXCANASRR21-11 Rev0000R2.0.0 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 . . . 7
3.1.2 IAR Compiler/Assembler/Linker Options . . . 11
3.1.3 GHS Compiler/Assembler/Linker Options . . . 13
3.2 Files required for compilation . . . 15
3.3 Setting up the plugins . . . 16
3.3.1 Location of various files inside the CAN module folder . . . 16

4 Function calls to module 17

4.1 Function Calls during Start-up . . . 17
4.2 Function Calls during Shutdown . . . 17
4.3 Function Calls during Wake-up . . . 17

5 Module requirements 18

5.1 Exclusive areas to be defined in BSW scheduler . . . 18
5.2 Exclusive areas not available on this platform . . . 27
5.3 Peripheral Hardware Requirements . . . 27
5.4 ISR to configure within AutosarOS - dependencies . . . 27
5.5 ISR Macro . . . 30
5.5.1 Without an Operating System . . . 30
5.5.2 With an Operating System . . . 30
5.6 Other AUTOSAR modules - dependencies . . . 31
5.7 Data Cache Restrictions . . . 31
5.8 User Mode support . . . 31
5.8.1 User Mode configuration in the module . . . 31
5.8.2 User Mode configuration in AutosarOS . . . 34
5.9 Multicore support . . . 35

6 Main API Requirements 36

6.1 Main function calls within BSW scheduler . . . 36
6.2 API Requirements . . . 37
6.3 Calls to Notification Functions, Callbacks, Callouts . . . 37

7 Memory allocation **38**
7.1 Sections to be defined in Can_43_FLEXCAN_MemMap.h 38
7.2 Linker command file 39

8 Integration Steps **40**

9 External assumptions for driver **41**

Chapter 1

Revision History

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

Chapter 2

Introduction

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

This Integration Manual describes NXP Semiconductor AUTOSAR CAN for S32K1_M24X. AUTOSAR CAN driver configuration parameters and deviations from the specification are described in Driver chapter of this document. A↩ UTOSAR 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:

- 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 CAN Driver	AUTOSAR Release R21-11
2	Reference Manual	S32K1xx Series Reference Manual, Rev. 14, 09/2021
		S32M24x Reference Manual, Rev. 2 Draft A, 05/2023
3	Errata	S32K116_0N96V Rev. 22/OCT/2021
		S32K118_0N97V Rev. 22/OCT/2021
		S32K142_0N33V Rev. 22/OCT/2021
		S32K144_0N57U Rev. 22/OCT/2021
		S32K144W_0P64A Rev. 22/OCT/2021
		S32K146_0N73V Rev. 22/OCT/2021
		S32K148_0N20V Rev. 22/OCT/2021
		S32M244_P64A+P73G, Rev. 0
		S32M242_N33V+P73G, Rev. 0, 6/2023
4	Datasheet	S32K1xx Data Sheet, Rev. 14, 08/2021
		S32M2xx Data Sheet, Rev. 3 DraftA, 05/2023

Chapter 3

Building the driver

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

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

3.1 Build Options

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

The RTD driver files are compiled using:

- NXP GCC 10.2.0 20200723 (Build 1728 Revision g5963bc8)
- IAR ANSI C/C++ Compiler V8.40.3.228/W32 for ARM Functional Safety
- Green Hills Multi 7.1.6d / Compiler 2020.1.4

The compiler, assembler, and linker flags used for building the driver are explained below.
The TS_T40D2M20I0R0 part of the plugin name is composed as follows:

- T = Target_Id (e.g. T40 identifies Cortex-M architecture)
- D = Derivative_Id (e.g. D2 identifies S32K1 platform)
- M = SW_Version_Major and SW_Version_Minor
- I = SW_Version_Patch
- R = Reserved

3.1.1 GCC Compiler/Assembler/Linker Options

3.1.1.1 GCC Compiler Options

Compiler Option	Description
-mcpu=cortex-m4	Targeted ARM processor for which GCC should tune the performance of the code (for S32K14x or S32M24x devices)
-mcpu=cortex-m0plus	Targeted ARM processor for which GCC should tune the performance of the code (for S32K11x devices)
-mthumb	Generates code that executes in Thumb state
-mlittle-endian	Generate code for a processor running in little-endian mode
-mfpu=fpv4-sp-d16	Specifies the floating-point hardware available on the target (for S32K14x or S32M24x devices)
-mfloat-abi=hard	Specifies the floating-point ABI to use. "hard" allows generation of floating-point instructions and uses FPU-specific calling conventions (for S32K14x or S32M24x devices)
-mfpu=auto	Specifies the floating-point hardware available on the target (for S32K11x devices)
-mfloat-abi=soft	Specifies the floating-point ABI to use. Specifying "soft" causes GCC to generate output containing library calls for floating-point operations (for S32K11x devices)
-std=c99	Specifies the ISO C99 base standard
-Os	Optimize for size. Enables all -O2 optimizations except those that often increase code size
-ggdb3	Produce debugging information for use by GDB using the most expressive format available, including GDB extensions if at all possible. Level 3 includes extra information, such as all the macro definitions present in the program
-Wall	Enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros
-Wextra	This enables some extra warning flags that are not enabled by -Wall
-pedantic	Issue all the warnings demanded by strict ISO C. Reject all programs that use forbidden extensions. Follows the version of the ISO C standard specified by the aforementioend -std option
-Wstrict-prototypes	Warn if a function is declared or defined without specifying the argument types
-Wundef	Warn if an undefined identifier is evaluated in an #if directive. Such identifiers are replaced with zero
-Wunused	Warn whenever a function, variable, label, value, macro is unused
-Werror=implicit-function-declaration	Make the specified warning into an error. This option throws an error when a function is used before being declared
-Wsign-compare	Warn when a comparison between signed and unsigned values could produce an incorrect result when the signed value is converted to unsigned.
-Wdouble-promotion	Give a warning when a value of type float is implicitly promoted to double
-fno-short-enums	Specifies that the size of an enumeration type is at least 32 bits regardless of the size of the enumerator values.

Compiler Option	Description
-funsigned-char	Let the type char be unsigned by default, when the declaration does not use either signed or unsigned
-funsigned-bitfields	Let a bit-field be unsigned by default, when the declaration does not use either signed or unsigned
-fomit-frame-pointer	Omit the frame pointer in functions that don't need one. This avoids the instructions to save, set up and restore the frame pointer; on many targets it also makes an extra register available.
-fno-common	Makes the compiler place uninitialized global variables in the BSS section of the object file. This inhibits the merging of tentative definitions by the linker so you get a multiple-definition error if the same variable is accidentally defined in more than one compilation unit
-fstack-usage	This option is only used to build test for generation Ram/↩ Stack size report. Makes the compiler output stack usage information for the program, on a per-function basis
-fdump-ipa-all	This option is only used to build test for generation Ram/↩ Stack size report. Enables all inter-procedural analysis dumps
-c	Stop after assembly and produce an object file for each source file
-DS32K1XX	Predefine S32K1XX as a macro, with definition 1
-DS32K148	Predefine S32K148 as a macro, with definition 1. S32↩ K148 can be replaced according to derivatives name S32K116,S32K118,S32K142,S32K142W,S32K144,S32↩ K144W,S32K146,S32K148,S32M244,S32M242.
-DGCC	Predefine GCC as a macro, with definition 1
-DUSE_SW_VECTOR_MODE	Predefine USE_SW_VECTOR_MODE as a macro, with definition 1. By default, the drivers are compiled to handle interrupts in Software Vector Mode
-DI_CACHE_ENABLE	Predefine I_CACHE_ENABLE as a macro, with definition 1. Enables instruction cache initialization in source file system.c under the Platform driver (for S32K14x or S32↩ M24x devices)
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initialization in source file system.c under the Platform driver (for S32K14x or S32M24x devices)
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_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 (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_<asr_package_name>\src\Can_43_FLEXCAN.c
- Can_43_FLEXCAN_<asr_package_name>\src\Can_43_FLEXCAN_Ipw.c
- Can_43_FLEXCAN_<asr_package_name>\src\Can_43_FLEXCAN_Ipw_Irq.c
- Can_43_FLEXCAN_<asr_package_name>\src\FlexCAN_Ip.c
- Can_43_FLEXCAN_<asr_package_name>\src\FlexCAN_Ip_Irq.c
- Can_43_FLEXCAN_<asr_package_name>\src\FlexCAN_Ip_Hw_Access.c
- Can_43_FLEXCAN_<asr_package_name>\include\Can_43_FLEXCAN.h
- Can_43_FLEXCAN_<asr_package_name>\include\Can_43_FLEXCAN_Flexcan_Types.h
- Can_43_FLEXCAN_<asr_package_name>\include\Can_43_FLEXCAN_Ipw.h
- Can_43_FLEXCAN_<asr_package_name>\include\Can_43_FLEXCAN_Ipw_Irq.h
- Can_43_FLEXCAN_<asr_package_name>\include\Can_43_FLEXCAN_Ipw_Types.h
- Can_43_FLEXCAN_<asr_package_name>\include\Can_43_FLEXCAN_Irq.h
- Can_43_FLEXCAN_<asr_package_name>\include\FlexCAN_Ip.h
- Can_43_FLEXCAN_<asr_package_name>\include\FlexCAN_Ip_DeviceReg.h
- Can_43_FLEXCAN_<asr_package_name>\include\FlexCAN_Ip_Hw_Access.h
- Can_43_FLEXCAN_<asr_package_name>\include\FlexCAN_Ip_Irq.h
- Can_43_FLEXCAN_<asr_package_name>\include\FlexCAN_Ip_Types.h
- Can_43_FLEXCAN_<asr_package_name>\include\FlexCAN_Ip_Wrapper.h
- Can_43_FLEXCAN_<asr_package_name>\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_<asr_package_name>\config\Can_43_FLEXCAN.xdm

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

- Can_43_FLEXCAN_<asr_package_name>\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 CA↩ N_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_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 in FlexCAN_Ip_SetBitrate.

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

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

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

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

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

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

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

CAN_EXCLUSIVE_AREA_16 is used in function Can_43_FLEXCAN_SetControllerMode to protect the register CAN_FDCTRL 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_0_15_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN0_ORED_16_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_47_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN0_ORED_48_63_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN1_ORED_0_15_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN1_ORED_16_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_47_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN1_ORED_48_63_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN2_ORED_0_15_MB_IRQHandler) to protect the variable g_FlexCAN_u32ImaskBuff in FlexCAN_SetMsgBuffIntCmd.

CAN_EXCLUSIVE_AREA_18 is used in function ISR(CAN2_ORED_16_31_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.

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 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_12 is used in function FlexCAN_Ip_ConfigPN_Privileged to protect the updates for:

- CAN_MCR register

  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 register

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

- CAN_MCR, CAN_FDCTRL register

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

- CAN_FDCTRL 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_Ip_ConfigTimeStamp_Privileged 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_12	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		x						

#	ARE_A_00	ARE_A_01	ARE_A_02	ARE_A_03	ARE_A_04	ARE_A_05	ARE_A_06	ARE_A_07	ARE_A_08	ARE_A_09	ARE_A_10	ARE_A_11	ARE_A_12	ARE_A_13	ARE_A_14	ARE_A_15	ARE_A_16	ARE_A_17	ARE_A_18	ARE_A_19	ARE_A_20	ISR Critical Regions (composed diagram)
ARE_A_03			x		x	x	x	x	x				x	x		x						
ARE_A_04			x	x		x	x	x	x				x	x		x						
ARE_A_05			x	x	x		x	x	x				x	x		x						
ARE_A_06			x	x	x	x		x	x		x		x	x	x	x		x		x	x	
ARE_A_07			x	x	x	x	x		x				x	x		x						
ARE_A_08			x	x	x	x	x	x					x	x		x						

#	ARE_A_00	ARE_A_01	ARE_A_02	ARE_A_03	ARE_A_04	ARE_A_05	ARE_A_06	ARE_A_07	ARE_A_08	ARE_A_09	ARE_A_10	ARE_A_11	ARE_A_12	ARE_A_13	ARE_A_14	ARE_A_15	ARE_A_16	ARE_A_17	ARE_A_18	ARE_A_19	ARE_A_20	ISR Critical Regions (composed diagram)
ARE_A_09																x	x					
ARE_A_10							x								x						x	
ARE_A_11																			x			
ARE_A_12			x	x	x	x	x	x	x					x		x						
ARE_A_13			x	x	x	x	x	x	x				x			x						
ARE_A_14							x				x					x					x	

#	A↩R↩E↩A↩_↩00	A↩R↩E↩A↩_↩01	A↩R↩E↩A↩_↩02	A↩R↩E↩A↩_↩03	A↩R↩E↩A↩_↩04	A↩R↩E↩A↩_↩05	A↩R↩E↩A↩_↩06	A↩R↩E↩A↩_↩07	A↩R↩E↩A↩_↩08	A↩R↩E↩A↩_↩09	A↩R↩E↩A↩_↩10	A↩R↩E↩A↩_↩11	A↩R↩E↩A↩_↩12	A↩R↩E↩A↩_↩13	A↩R↩E↩A↩_↩14	A↩R↩E↩A↩_↩15	A↩R↩E↩A↩_↩16	A↩R↩E↩A↩_↩17	A↩R↩E↩A↩_↩18	A↩R↩E↩A↩_↩19	A↩R↩E↩A↩_↩20	I↩S↩Rs Critical Regions (composed diagram)
A↩R↩E↩A↩_↩15			x	x	x	x	x	x	x	x			x	x	x		x					
A↩R↩E↩A↩_↩16										x						x						
A↩R↩E↩A↩_↩17							x													x		
A↩R↩E↩A↩_↩18												x										x
A↩R↩E↩A↩_↩19							x											x				
A↩R↩E↩A↩_↩20							x				x				x							

#	ARE_A_00	ARE_A_01	ARE_A_02	ARE_A_03	ARE_A_04	ARE_A_05	ARE_A_06	ARE_A_07	ARE_A_08	ARE_A_09	ARE_A_10	ARE_A_11	ARE_A_12	ARE_A_13	ARE_A_14	ARE_A_15	ARE_A_16	ARE_A_17	ARE_A_18	ARE_A_19	ARE_A_20	ISR Rs Critical Regions (composed diagram)
ISR Rs Critical Regions (composed diagram)																			x			

5.2 Exclusive areas not available on this platform

None.

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 S32K11X

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	**26**
CAN0_ORED_0_31_MB_IRQHandler	**27**

- **Table with interrupts for S32K142**

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	94
CAN0_Error_IRQHandler	95
CAN0_Wake_Up_IRQHandler	96
CAN0_ORED_0_15_MB_IRQHandler	97
CAN0_ORED_16_31_MB_IRQHandler	98
CAN1_ORED_IRQHandler	101
CAN1_Error_IRQHandler	102
CAN1_ORED_0_15_MB_IRQHandler	104

- **Table with interrupts for S32K144**

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	94
CAN0_Error_IRQHandler	95
CAN0_Wake_Up_IRQHandler	96
CAN0_ORED_0_15_MB_IRQHandler	97
CAN0_ORED_16_31_MB_IRQHandler	98
CAN1_ORED_IRQHandler	101
CAN1_Error_IRQHandler	102
CAN1_ORED_0_15_MB_IRQHandler	104
CAN2_ORED_IRQHandler	108
CAN2_Error_IRQHandler	109
CAN2_ORED_0_15_MB_IRQHandler	111

- **Table with interrupts for S32K144W**

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	94
CAN0_Error_IRQHandler	95
CAN0_Wake_Up_IRQHandler	96
CAN0_ORED_0_15_MB_IRQHandler	97
CAN0_ORED_16_31_MB_IRQHandler	98
CAN0_ORED_32_47_MB_IRQHandler	99
CAN0_ORED_48_63_MB_IRQHandler	100
CAN1_ORED_IRQHandler	101
CAN1_Error_IRQHandler	102
CAN1_ORED_0_15_MB_IRQHandler	104
CAN1_ORED_16_31_MB_IRQHandler	105
CAN1_ORED_32_47_MB_IRQHandler	106
CAN1_ORED_48_63_MB_IRQHandler	107

- **Table with interrupts for S32K146**

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	94
CAN0_Error_IRQHandler	95
CAN0_Wake_Up_IRQHandler	96
CAN0_ORED_0_15_MB_IRQHandler	97
CAN0_ORED_16_31_MB_IRQHandler	98
CAN1_ORED_IRQHandler	101
CAN1_Error_IRQHandler	102
CAN1_ORED_0_15_MB_IRQHandler	104
CAN1_ORED_16_31_MB_IRQHandler	105
CAN2_ORED_IRQHandler	108
CAN2_Error_IRQHandler	109
CAN2_ORED_0_15_MB_IRQHandler	111

- **Table with interrupts for S32K148**

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	94
CAN0_Error_IRQHandler	95
CAN0_Wake_Up_IRQHandler	96
CAN0_ORED_0_15_MB_IRQHandler	97
CAN0_ORED_16_31_MB_IRQHandler	98
CAN1_ORED_IRQHandler	101
CAN1_Error_IRQHandler	102
CAN1_ORED_0_15_MB_IRQHandler	104
CAN1_ORED_16_31_MB_IRQHandler	105
CAN2_ORED_IRQHandler	108
CAN2_Error_IRQHandler	109
CAN2_ORED_0_15_MB_IRQHandler	111
CAN2_ORED_16_31_MB_IRQHandler	112

- **Table with interrupts for S32M241 and S32M242**

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	78
CAN0_Error_IRQHandler	79
CAN0_Wake_Up_IRQHandler	80
CAN0_ORED_0_15_MB_IRQHandler	81
CAN0_ORED_16_31_MB_IRQHandler	82

- **Table with interrupts for S32M243 and S32M244**

ISR Name	Hardware Interrupt Vector
CAN0_ORED_IRQHandler	78
CAN0_Error_IRQHandler	79

ISR Name	Hardware Interrupt Vector
CAN0_Wake_Up_IRQHandler	80
CAN0_ORED_0_15_MB_IRQHandler	81
CAN0_ORED_16_31_MB_IRQHandler	82
CAN0_ORED_32_47_MB_IRQHandler	83
CAN0_ORED_48_63_MB_IRQHandler	84

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

Function syntax	Description	Available via
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_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
Flexcan_Ip_StatusType Flex↩ CAN_Ip_EnterFreezeMode_↩ Privileged(uint8 instance)	Enter FlexCAN Module in Freeze Mode	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType Flex↩ CAN_Ip_ExitFreezeMode_↩ Privileged(uint8 instance)	Exit FlexCAN Module from Freeze Mode	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType FlexC↩ AN_Ip_SetRxFifoGlobalMask_↩ Privileged(uint8 instance, uint32 mask)	Set RxFifo Global Mask	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType FlexCA↩ N_Ip_Deinit_Privileged(uint8 instance)	DeInitilize the FlexCAN instance driver	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType Flex↩ CAN_Ip_SetStartMode_↩ Privileged(uint8 instance)	Set the FlexCAN instance in STA↩ RT mode	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType Flex↩ CAN_Ip_SetStopMode_↩ Privileged(uint8 instance)	Set the FlexCAN instance in STOP mode	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType Flex↩ CAN_Ip_SetRxMaskType↩ _Privileged(uint8 instance, Flexcan_Ip_RxMaskType type)	Set RX masking type	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType Flex↩ CAN_Ip_SetRxMb14Mask_↩ Privileged(uint8 instance, uint32 mask)	Set Rx14Mask filter for message buffer 14	FlexCAN_Ip_Trusted↩ Functions.h

Function syntax	Description	Available via
Flexcan_Ip_StatusType Flex↩ CAN_Ip_SetRxMb15Mask_↩ Privileged(uint8 instance, uint32 mask)	Set Rx14Mask filter for message buffer 15	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType FlexC↩ AN_Ip_SetRxIndividualMask↩ _Privileged(uint8 instance, uint8 mb_idx, uint32 mask)	Sets the FlexCAN Rx individual mask	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType FlexC↩ AN_Ip_SetRxMbGlobalMask_↩ Privileged(uint8 instance, uint32 mask)	Sets the FlexCAN Rx Message Buffer Global mask	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType FlexCA↩ N_Ip_SetBitrate_Privileged(uint8 instance, const Flexcan_Ip_Time↩ SegmentType ∗ bitrate, boolean enhExt)	Sets the FlexCAN bit rate for standard frames or the arbitration phase of FD frames	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType Flex↩ CAN_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_Trusted↩ Functions.h
Flexcan_Ip_StatusType FlexC↩ AN_Ip_SetTxArbitrationStart↩ Delay_Privileged(uint8 instance, uint8 value)	This function will set how many C↩ AN bits the Tx arbitration process start point can	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType Flex↩ CAN_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_Trusted↩ Functions.h
Flexcan_Ip_StatusType Flex↩ CAN_Ip_EnableInterrupts_↩ Privileged(uint8 u8Instance)	Enable all mb interrupts configured	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType Flex↩ CAN_Ip_DisableInterrupts_↩ Privileged(uint8 u8Instance)	Disable all mb interrupts configured	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType Flex↩ CAN_Ip_SetErrorInt_↩ Privileged(uint8 u8Instance, Flexcan_Ip_ErrorIntType type, boolean enable)	Enable\Disable Error or BusOff Interrupt	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType Flex↩ CAN_Ip_SetListenOnlyMode↩ _Privileged(uint8 instance, const boolean enable)	Set FlexCAN Listen Only	FlexCAN_Ip_Trusted↩ Functions.h
Flexcan_Ip_StatusType Flex↩ CAN_Ip_ConfigTimeStamp_↩ Privileged(uint8 instance, const Flexcan_Ip_TimeStampConfig↩ Type ∗ time_stamp)	Set FlexCAN Config Timestamp	FlexCAN_Ip_Trusted↩ Functions.h

Function syntax	Description	Available via
Flexcan_Ip_StatusType FlexCA↩ N_Ip_ConfigPN_Privileged(uint8 u8Instance, boolean bEnable, const Flexcan_Ip_PnConfigType * pPnConfig)	Configures Pretended Networking settings	FlexCAN_Ip_Trusted↩ Functions.h

5.8.2 User Mode configuration in AutosarOS

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

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

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

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

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

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

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

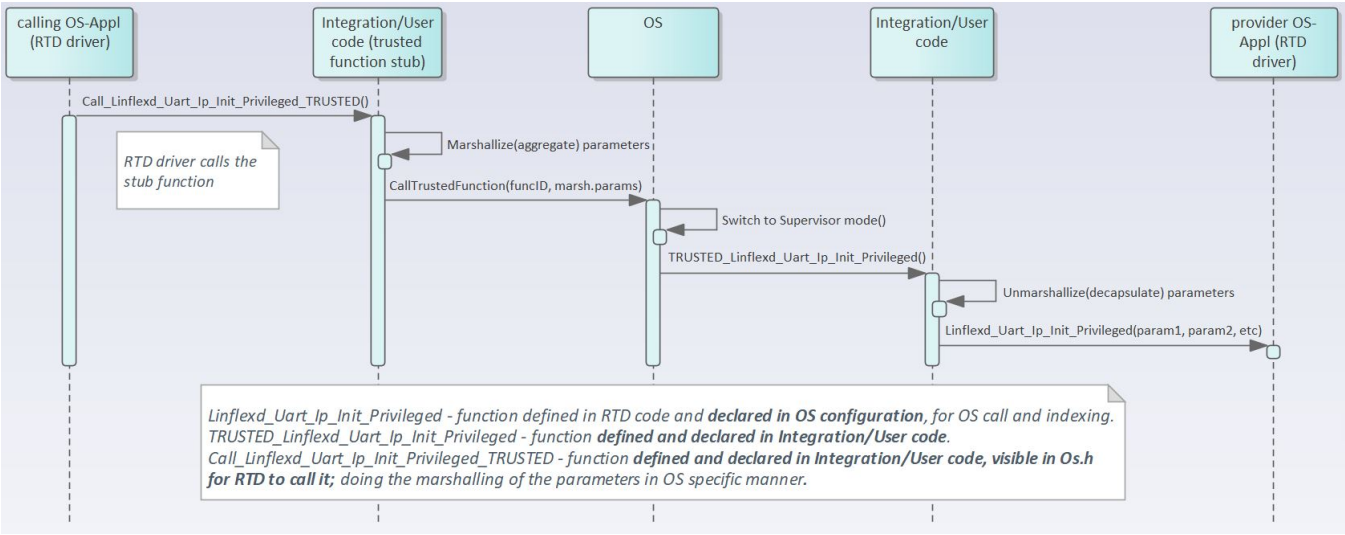

Figure 5.1 Example sequence chart for calling `Linflexd_Uart_Ip_Init_Privileged` as trusted function

5.9 Multicore support

None.

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_CONFIG_DATA_UNSPECIFIED	Configuration Data	Start of Memory Section for Config Data
CAN_43_FLEXCAN_STOP_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	End of Memory Section for Config Data
CAN_43_FLEXCAN_START_SEC_CODE	Code	Start of memory Section for Code
CAN_43_FLEXCAN_STOP_SEC_CODE	Code	End of memory Section for Code
CAN_43_FLEXCAN_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 initialized with values after every reset.
CAN_43_FLEXCAN_STOP_SEC_VAR_CLEARED_UNSPECIFIED	Variables	End of above section.
CAN_43_FLEXCAN_START_SEC_VAR_CLEARED_UNSPECIFIED	Variables	These variables are never cleared and never initialized by start-up code.
CAN_43_FLEXCAN_STOP_SEC_VAR_CLEARED_UNSPECIFIED	Variables	End of above section.
CAN_43_FLEXCAN_START_SEC_CONST_UNSPECIFIED	Constant Data	Used for constants
CAN_43_FLEXCAN_STOP_SEC_CONST_UNSPECIFIED	Constant Data	End of above section.
CAN_43_FLEXCAN_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. Normally, this section is used to store descriptors and data buffers. This section must also be cache inhibited
CAN_43_FLEXCAN_STOP_SEC_VAR_CLEARED_UNSPECIFIED_NO_CACHEABLE	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 assumptions for driver

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
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.
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.

External Assumption Req ID	External Assumption Text
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.
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.

External assumptions for driver

External Assumption Req ID	External Assumption Text
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_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

External Assumption Req ID	External Assumption Text
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.