

Integration Manual

for S32K3 SPI Driver

Document Number: IM34SPIASRR21-11 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	8
3.1 Build Options	8
3.1.1 GCC Compiler/Assembler/Linker Options	9
3.1.2 DIAB Compiler/Assembler/Linker Options	11
3.1.3 GHS Compiler/Assembler/Linker Options	13
3.1.4 IAR Compiler/Assembler/Linker Options	15
3.2 Files required for compilation	17
3.3 Setting up the plugins	20
3.3.1 DMA configuration	21
4 Function calls to module	28
4.1 Function Calls during Start-up	28
4.2 Function Calls during Shutdown	28
4.3 Function Calls during Wake-up	28
5 Module requirements	29
5.1 Exclusive areas to be defined in BSW scheduler	29
5.2 Exclusive areas not available on this platform	35
5.3 Peripheral Hardware Requirements	35
5.4 ISR to configure within AutosarOS - dependencies	35
5.5 ISR Macro	36
5.5.1 Without an Operating System	36
5.5.2 With an Operating System	36
5.6 Other AUTOSAR modules - dependencies	37
5.7 Data Cache Restrictions	37
5.8 User Mode support	38
5.8.1 User Mode configuration in the module	38
5.8.2 User Mode configuration in AutosarOS	38
5.9 Multicore support	39
6 Main API Requirements	41
6.1 Main function calls within BSW scheduler	41
6.2 API Requirements	41

6.3 Calls to Notification Functions, Callbacks, Callouts	41
7 Memory allocation	42
7.1 Sections to be defined in Spi_MemMap.h	42
7.2 Linker command file	43
8 Integration Steps	44
9 External assumptions for driver	45



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 the integration requirements for SPI Driver for S32K3XX microcontrollers.

2.1 Supported Derivatives

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

- s32k310_mqfp100
- s32k310_lqfp48
- s32k311_mqfp100 / MWCT2015S_mqfp100
- s32k311_lqfp48
- s32k312_mqfp100 / MWCT2016S_mqfp100
- s32k312_mqfp172 / MWCT2016S_mqfp172
- s32k314_mqfp172
- s32k314_mapbga257
- s32k322_mqfp100 / MWCT2D16S_mqfp100
- s32k322_mqfp172 / MWCT2D16S_mqfp172
- s32k324_mqfp172 / MWCT2D17S_mqfp172
- s32k324_mapbga257

- s32k341_mqfp100
- s32k341_mqfp172
- s32k342_mqfp100
- s32k342_mqfp172
- s32k344_mqfp172
- s32k344_mapbga257
- s32k394_mapbga289
- s32k396_mapbga289
- s32k358_mqfp172
- s32k358_mapbga289
- s32k328_mqfp172
- s32k328_mapbga289
- s32k338_mqfp172
- s32k338_mapbga289
- s32k348_mqfp172
- s32k348_mapbga289
- s32m274_lqfp64
- s32m276_lqfp64

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

Note: MWCT part numbers contain NXP confidential IP for Qi Wireless Power.

2.2 Overview

AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership working to establish standards for software interfaces and software modules for automobile electronic control systems.

AUTOSAR:

- paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness.
- is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation".
- is a key enabling technology to manage the growing electrics/electronics complexity. It aims to be prepared for the upcoming technologies and to improve cost-efficiency without making any compromise with respect to quality.
- facilitates the exchange and update of software and hardware over the service life of the vehicle.

2.3 About This Manual

This Technical Reference employs the following typographical conventions:

- **Boldface** style: Used for important terms, notes and warnings.
- *Italic* style: Used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

Note

This is a note.

Warning

This is a warning

2.4 Acronyms and Definitions

Term	Definition
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSMI	Basic Software Make file Interface
CS	Chip Select
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DMA	Direct Memory Access
ECU	Electronic Control Unit
FIFO	First In First Out
LSB	Least Significant 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
SPI	Serial Peripheral Interface
XML	Extensible Markup Language
BSW	Basic Software
ISR	Interrupt Service Routine
OS	Operating System
GUI	Graphical User Interface
PB Variant	Post Build Variant
PC Variant	Pre Compile Variant
LT Variant	Link Time Variant

2.5 Reference List

#	Title	Version
1	S32K3XX Reference Manual	S32K3xx Reference Manual, Rev.6 Draft B, 01/2023
2	S32K396 Reference Manual	S32K39 and S32K37 Reference Manual, Rev. 2 Draft A, 11/2022
3	S32M27x Reference Manual	S32M27x Reference Manual, Rev.2, Draft A - 02/2023
4	S32K3XX Datasheet	S32K3xx Data Sheet, Rev. 6, Draft B. 01/2023
5	S32K396 Datasheet	S32K396 Data Sheet, Rev. 1.1, 08/2022
6	S32M2xx Datasheet	S32M2xx Data Sheet, Rev. 2 RC — 12/2022
7	S32K358 Errata	S32K358 Mask Set Errata for Mask 0P14E, Rev. 28, 9/2022
8	S32K311 Errata	S32K311 Mask Set Errata S32K311_0P98C, Rev. 6/March/2023, 3/2023
9	S32K396 Errata	SS32K396 Mask Set Errata for Mask 0P40E, Rev. DEC2022, 12/2022
10	S32K312 Errata	S32K312 Mask Set Errata for Mask 0P09C, Rev. 25/April/2022

#	Title	Version
11	S32K342 Errata	S32K342 Mask Set Errata for Mask 0P97C, Rev. 10 11/2022
12	S32K3x4 Errata	S32K3x4 Mask Set Errata for Mask 0P55A/1P55A, Rev. 14/Oct/2022
13	Specification of SPI Driver	AUTOSAR Release R21-11

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
-mfpv=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 aforementioned -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 initialization 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 initialization in source file system.c under the Platform driver
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initialization in source file system.c under the Platform driver
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPORT↔ 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
-mfpv=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 <code>#if</code> 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 initialization 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 initialization in source file system.c under the Platform driver
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initialization in source file system.c under the Platform driver
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPORT as a macro, with definition 1. Allows drivers to be configured in user mode

3.1.2.2 DIAB Assembler Options

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

3.1.2.3 DIAB Linker Options

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

3.1.3 GHS Compiler/Assembler/Linker Options

3.1.3.1 GHS Compiler Options

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

Compiler Option	Description
-unsigned_chars	Let the type char be unsigned, like unsigned char
-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
-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 initialization 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 initialization in source file system.c under the Platform driver
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initialization in source file system.c under the Platform driver
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPORT as a macro, with definition 1. Allows drivers to be configured in user mode

3.1.3.2 GHS Assembler Options

Assembler Option	Description
-cpu=cortexm7	Selects target processor: Arm Cortex M7
-fpu=vfpv5_d16	Specifies hardware floating-point using the v5 version of the VFP instruction set, with 16 double-precision floating-point registers
-fsingle	Use hardware single-precision, software double-precision FP instructions
-preprocess_assembly_files	Controls whether assembly files with standard extensions such as .s and .asm are preprocessed
-list	Creates a listing by using the name and directory of the object file with the .lst extension
-c	Stop after assembly and produce an object file for each source file

3.1.3.3 GHS Linker Options

Linker Option	Description
-e Reset_Handler	Make the symbol Reset_Handler be treated as a root symbol and the start label of the application
-T linker_script_file.ld	Use linker_script_file.ld as the linker script. This script replaces the default linker script (rather than adding to it)
-map	Produce a map file
-keepmap	Controls the retention of the map file in the event of a link error
-Mn	Generates a listing of symbols sorted alphabetically/numerically by address
-delete	Instructs the linker to remove functions that are not referenced in the final executable. The linker iterates to find functions that do not have relocations pointing to them and eliminates them
-ignore_debug_references	Ignores relocations from DWARF debug sections when using -delete. DWARF debug information will contain references to deleted functions that may break some third-party debuggers
-Llibrary_path	Points to library_path (the libraries location) for thumb2 to be used for linking
-larch	Link architecture specific library
-lstartup	Link run-time environment startup routines. The source code for 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
-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
-Osz	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 initialization 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 initialization in source file system.c under the Platform driver
-DENABLE_FPU	Predefine ENABLE_FPU as a macro, with definition 1. Enables FPU initialization in source file system.c under the Platform driver
-DMCAL_ENABLE_USER_MODE_SUPPORT	Predefine MCAL_ENABLE_USER_MODE_SUPPORT as a macro, with definition 1. Allows drivers to be configured in user mode.

3.1.4.2 IAR Assembler Options

Assembler Option	Description
-cpu Cortex-M7	Targeted ARM processor for which IAR should generate the instruction set
-fpu VFPv5-SP	Use this option to generate code that performs floating-point operations using a Floating Point Unit (FPU). Single-precision variant.
-cpu_mode thumb	Selects the thumb mode for the assembler directive CODE
-g	Disables the automatic search for system include files
-r	Generates debug information

3.1.4.3 IAR Linker Options

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

3.2 Files required for compilation

This section describes the include files required to compile, assemble and link the AUTOSAR Spi Driver for S32K3XX microcontrollers.

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

Spi Driver Files:

- Spi_TS_T40D34M30I0R0\src\Lpspi_Ip.c
- Spi_TS_T40D34M30I0R0\src\Lpspi_Ip_Irq.c
- Spi_TS_T40D34M30I0R0\src\Flexio_Spi_Ip.c
- Spi_TS_T40D34M30I0R0\src\Flexio_Spi_Ip_Irq.c
- Spi_TS_T40D34M30I0R0\src\Spi.c
- Spi_TS_T40D34M30I0R0\src\Spi_IPW.c
- Spi_TS_T40D34M30I0R0\inc\Lpspi_Ip.h
- Spi_TS_T40D34M30I0R0\inc\Lpspi_Ip_Types.h
- Spi_TS_T40D34M30I0R0\inc\Flexio_Spi_Ip.h
- Spi_TS_T40D34M30I0R0\inc\Flexio_Spi_Ip_Types.h
- Spi_TS_T40D34M30I0R0\inc\Spi.h
- Spi_TS_T40D34M30I0R0\inc\Spi_IPW.h
- Spi_TS_T40D34M30I0R0\inc\Spi_IPW_Types.h

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

Building the driver

- Spi_[VariantName]_PBcfg.c (For PB Variant) - The file contains the definition of the initialization pointer for the respective variant.
- Spi_Ipw_[VariantName]_PBcfg.c - The file contains the definition of the initialization pointer for the respective variant.
- Lpspi_Ip_[VariantName]_PBcfg.c - The file contains the definition of the initialization pointer for the respective variant.
- Flexio_Spi_Ip_[VariantName]_PBcfg.c - The file contains the definition of the initialization pointer for the respective variant.
- Spi_[VariantName]_PBcfg.h - The file contains externing definitions of configuration structures.
- Spi_Ipw_[VariantName]_PBcfg.h - The file externing definitions of configuration structures.
- Spi_Ipw_Cfg.h - The file contains the definitions according to configuration.
- Spi_Cfg.h - The file contains the definitions according to configuration.
- Lpspi_Ip_[VariantName]_PBcfg.h - The file externing definitions of configuration structures.
- Flexio_Spi_Ip_[VariantName]_PBcfg.h - The file externing definitions of configuration structures.
- Lpspi_Ip_Cfg.h - The file contains the definitions according to configuration.
- Flexio_Spi_Ip_Cfg.h - The file contains the definitions according to configuration.

Note

As a deviation from standard:

- Spi_[VariantName]_PBcfg.c, Spi_Ipw_[VariantName]_PBcfg.c, Lpspi_Ip_[VariantName]_PBcfg.c, Flexio_Spi_Ip_[VariantName]_PBcfg.c, Spi_[VariantName]_PBcfg.h, Spi_Ipw_[VariantName]_PBcfg.h, Lpspi_Ip_[VariantName]_PBcfg.h, Flexio_Spi_Ip_[VariantName]_PBcfg.h - These files will contain the definition for all parameters that are variant aware, independent of the configuration class that will be selected (PC, LT, PB).

Base Files:

- BaseNXP_TS_T40D34M30I0R0\include\Compiler.h
- BaseNXP_TS_T40D34M30I0R0\include\Compiler_Cfg.h
- BaseNXP_TS_T40D34M30I0R0\include\ComStack_Cfg.h
- BaseNXP_TS_T40D34M30I0R0\include\ComStack_Types.h
- BaseNXP_TS_T40D34M30I0R0\include\Mcal.h
- BaseNXP_TS_T40D34M30I0R0\include\Spi_MemMap.h
- BaseNXP_TS_T40D34M30I0R0\include\Platform_Types.h
- BaseNXP_TS_T40D34M30I0R0\include\Reg_eSys.h
- BaseNXP_TS_T40D34M30I0R0\include\RegLockMacros.h
- BaseNXP_TS_T40D34M30I0R0\include\Soc_Ips.h

- BaseNXP_TS_T40D34M30I0R0\include\StandardTypes.h
- BaseNXP_TS_T40D34M30I0R0\include\OsIf.h
- BaseNXP_TS_T40D34M30I0R0\include\Devassert.h
- BaseNXP_TS_T40D34M30I0R0\header\[PlatformName]_LPSPI.h
- BaseNXP_TS_T40D34M30I0R0\header\[PlatformName]_FLEXIO.h

DEM Files:

- Dem_TS_T40D34M30I0R0\include\Dem.h
- Dem_TS_T40D34M30I0R0\include\Dem_Types.h
- Dem_TS_T40D34M30I0R0\generate_PC\include\Dem_IntErrId.h
- Dem_TS_T40D34M30I0R0\src\Dem.c

DET Files:

- Det_TS_T40D34M30I0R0\include\Det.h
- Det_TS_T40D34M30I0R0\src\Det.c

MCL Files(when DMA or SPI over FLEXIO is used):

- Mcl_TS_T40D34M30I0R0\include\CDD_Mcl.h
- Mcl_TS_T40D34M30I0R0\src\CDD_Mcl.c

RM Files(configure DMA MUX when DMA is used):

- Rm_TS_T40D34M30I0R0\include\CDD_Rm.h
- Rm_TS_T40D34M30I0R0\src\CDD_Rm.c

RTE Files:

- Rte_TS_T40D34M30I0R0\include\SchM_Spi.h
- Rte_TS_T40D34M30I0R0\src\SchM_Spi.c

3.3 Setting up the plugins

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

Location of various files inside the SPI module folder:

- VSMD (Vendor Specific Module Definition) file in EB Tresos Studio XDM format:
 - Spi_TS_T40D34M30I0R0\config\Spi.xdm
- VSMD (Vendor Specific Module Definition) file(s) in AUTOSAR compliant EPD format:
 - Spi_TS_T40D34M30I0R0\autosar\Spi_<subderivative_name>.epd
- Code Generation Templates for variant aware parameters:
 - Spi_TS_T40D34M30I0R0\generate_PB\src\Spi_PBcfg.c
 - Spi_TS_T40D34M30I0R0\generate_PB\src\Spi_Ipw_PBcfg.c
 - Spi_TS_T40D34M30I0R0\generate_PB\src\Spi_PBcfg.c

 - Spi_TS_T40D34M30I0R0\generate_PB\src\Spi_PBcfg.h

 - Spi_TS_T40D34M30I0R0\generate_PB\src\Spi_Ipw_PBcfg.h
 - Spi_TS_T40D34M30I0R0\generate_PB\src\Lpspi_Ip_PBcfg.h
 - Spi_TS_T40D34M30I0R0\generate_PB\src\Flexio_Spi_Ip_PBcfg.h
- Code Generation Templates for parameters without variation points:
 - Spi_TS_T40D34M30I0R0\generate_PC\include\Spi_Cfg.h
 - Spi_TS_T40D34M30I0R0\generate_PC\include\Spi_Ipw_Cfg.h
 - Spi_TS_T40D34M30I0R0\generate_PC\include\Lpspi_Ip_Cfg.h
 - Spi_TS_T40D34M30I0R0\generate_PC\include\Flexio_Spi_Ip_Cfg.h

Steps to generate the configuration:

1. Copy the following module folders into the Tresos plugins folder:

- BaseNXP_TS_T40D34M30I0R0
- Dem_TS_T40D34M30I0R0
- Det_TS_T40D34M30I0R0
- EcuC_TS_T40D34M30I0R0
- Rte_TS_T40D34M30I0R0
- Resource_TS_T40D34M30I0R0
- Mcu_TS_T40D34M30I0R0
- Spi_TS_T40D34M30I0R0
- Mcl_TS_T40D34M30I0R0
- Rm_TS_T40D34M30I0R0
- Os_TS_T40D34M30I0R0

2. Set the desired Tresos Output location folder for the generated sources and header files.
3. Use the EB Tresos Studio GUI to modify ECU configuration parameters values.
4. Generate the configuration files

Dependencies:

- **MCU** is required to generate baud rate according to Peripheral clock.
- **RESOURCE** is required to select processor derivative.
- **DET** is required for signalling the development error detection (parameters out of range, null pointers, etc).
- **DEM** is required for signalling the production error detection (hardware failure, etc).
- **MCL** is required when DMA option is used or HW Unit is SPI over FLEXIO.
- **RM** is required when DMA option is used.
- **ECUC** is required for configuring the variant handling and partition in Tresos.
- **OS** is required for configuring the core selected in EB Tresos and the Get_coreID API has called in SPI driver.

[DMA configuration](#)

3.3.1 DMA configuration This section applies only to SPI units configured for asynchronous transmission (SpiPhyUnitSync not checked) and which use DMA for serializing/deserializing data between the hardware unit and the TX/RX buffers (SpiPhyUnitAsyncUseDma = true).

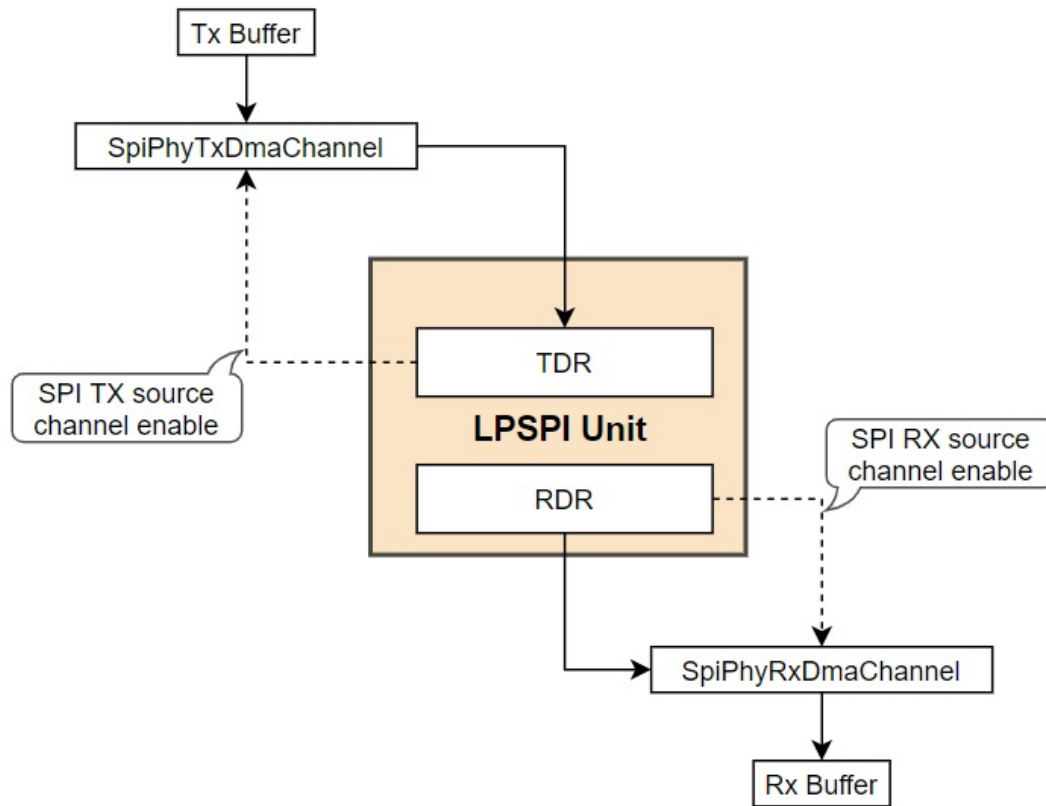


Figure 3.1 DMA transferring mode internal architecture

The mechanism in this diagram is also applied for FLEXIO but SHIFTBUFs used instead of TDR, RDR. Each SPI unit configured in DMA mode requires 2 distinct DMA channels from the same DMA Mux:

- **SpiPhyTxDmaChannel:** The TX DMA channel used for filling TX FIFO with the data in TX Buffer. This channel is triggered by TX SPI unit event and must be “wired” to “SPI TX source” (configured inside the MCL module – "McConfig/Dma Logic Channel" container).
- **SpiPhyRxDmaChannel:** The RX DMA channel used for filling RX buffer with the deserialized data. This channel is triggered by RX SPI unit event and must be “wired” to “SPI RX source” (configured inside the MCL module – "McConfig/Dma Logic Channel" container).

Note

- If DMA uses fixed priority arbitration, then the priority must be SpiPhyRxDmaChannel > SpiPhyTxDmaChannel.
- If DMA uses round robin arbitration, no priority constraints are applied on SpiPhyTxDmaChannel and SpiPhyRxDmaChannel priority.


If the SPI driver is working in interrupt mode, the DMA Tx and DMA Rx notification must be enabled for the specified Tx DMA and Rx DMA channels. The name of the function to be used as a notification is `Lpspi_Ip_LPSPi_X_IrqTxDmaHandler`, `Flexio_Spi_Ip_FLEXIO_SPI_X_IrqTxDmaHandler` or `Lpspi_Ip_LPSPi_X_IrqRxDmaHandler`, `Flexio_Spi_Ip_FLEXIO_SPI_X_IrqRxDmaHandler` where X is the number of SPI unit used.

Table 3.13 SPI DMA Notification


Physical Unit	SPI TX DMA Notification Name	SPI RX DMA Notification Name
LPSPI_0	Lpspi_Ip_LPSPI_0_IrqTxDmaHandler	Lpspi_Ip_LPSPI_0_IrqRxDmaHandler
LPSPI_1	Lpspi_Ip_LPSPI_1_IrqTxDmaHandler	Lpspi_Ip_LPSPI_1_IrqRxDmaHandler
LPSPI_2	Lpspi_Ip_LPSPI_2_IrqTxDmaHandler	Lpspi_Ip_LPSPI_2_IrqRxDmaHandler
LPSPI_3	Lpspi_Ip_LPSPI_3_IrqTxDmaHandler	Lpspi_Ip_LPSPI_3_IrqRxDmaHandler
LPSPI_4	Lpspi_Ip_LPSPI_4_IrqTxDmaHandler	Lpspi_Ip_LPSPI_4_IrqRxDmaHandler
LPSPI_5	Lpspi_Ip_LPSPI_5_IrqTxDmaHandler	Lpspi_Ip_LPSPI_5_IrqRxDmaHandler
FLEXIO_SPI↔_0	Flexio_Spi_Ip_FLEXIO_SPI_0_IrqTx↔DmaHandler	Flexio_Spi_Ip_FLEXIO_SPI_0_IrqRx↔DmaHandler
FLEXIO_SPI↔_1	Flexio_Spi_Ip_FLEXIO_SPI_1_IrqTx↔DmaHandler	Flexio_Spi_Ip_FLEXIO_SPI_1_IrqRx↔DmaHandler


Next figures show an example of DMA configuration for SPI unit.



SpiPhyUnit


Name  SpiPhyUnit_0


General


SpiPhyUnitMapping  LPSPI_2


SpiPhyUnitMode  SPI_MASTER



SpiPhyUnitSync  ☐  ▼



SpiSamplePoint (0 -> 1)  0

SpiPinConfiguration (0 -> 3)  0

SpiPhyUnitClockRef  /Mcu/Mcu/McuModuleConfiguration/Mcu

☒ SpiPhyUnitAlternateClockRef 

SpiPhyUnitAsyncUseDma  ☒  ▼

 SpiPhyTxDmaChannel  /Mcl/Mcl/MclConfig/DMA_LPSPI2_TX




 SpiPhyRxDmaChannel  /Mcl/Mcl/MclConfig/DMA_LPSPI2_RX

Figure 3.2 DMA Configuration sample for SPI Physical Unit - SPI module in EB Tresos

Logic Channel

Name  DMA_LPSPi2_TX



Logic Channel Configuration

Global



Transfer

ScatterGather



Logic Channel Name

 DMA_LOGIC_CH_0 



Hardware Instance

 DMA_IP_HW_INST_0 



Hardware Channel

 DMA_IP_HW_CH_0 


Interrupt Callback

 Lpspi_lp_LPSPi2_IrqTxDmaHandler 



Error Interrupt Callback

 NULL_PTR 



Ecuc Partition Ref



Enable Global Config

 ☒ 

Enable Transfer Config

 ☐ 

Enable Scatter/Gather





 ☐ 

Figure 3.3 DMA TX General configuration - MCL module in EB Tresos


Logic Channel





Name  DMA_LPSPi2_TX

Logic Channel Configuration Global Transfer ScatterGather


Name  dmaLogicChannel_GlobalConfigType





Control



Name  dmaLogicChannelConfig_GlobalControlType



Enable Master Id Replication  ☐  Enable Buffered Writes  ☐ 

Request

Name  dmaLogicChannelConfig_GlobalRequestType

Enable DMAMUX Trigger  ☐  Enable DMAMUX Source  ☒ 

DMAMUX0 Source  DMA_IP_REQ_MUX0_LPSPi2_TX 

DMAMUX1 Source  DMA_IP_REQ_MUX1_DISABLED 



Enable DMA Request  ☐ 

Figure 3.4 DMA TX Global Configuration - MCL module in EB Tresos

Logic Channel

Name DMA_LPSPi2_RX

Logic Channel Configuration

GlobalTransferScatterGather

Logic Channel Name

DMA_LOGIC_CH_1

Hardware Instance

DMA_IP_HW_INST_0

Hardware Channel

DMA_IP_HW_CH_1

Interrupt Callback

Lpspi_lp_LPSPi_2_IrqRxDmaHandler

Error Interrupt Callback

NULL_PTR

Ecuc Partition Ref

Enable Global Config

☒

Enable Transfer Config


☐

Enable Scatter/Gather

☐


Figure 3.5 DMA RX General configuration - MCL module in EB Tresos

Logic Channel


Name  DMA_LPSPi2_RX





Logic Channel Configuration Global Transfer ScatterGather

▼ dmaLogicChannel_GlobalConfigType


Name  dmaLogicChannel_GlobalConfigType





▼ Control



Name  dmaLogicChannelConfig_GlobalControlType



Enable Master Id Replication  ☐  Enable Buffered Writes  ☐ 

▼ Request

Name  dmaLogicChannelConfig_GlobalRequestType

Enable DMAMUX Trigger  ☐  Enable DMAMUX Source  ☒ 

DMAMUX0 Source  DMA_IP_REQ_MUX0_LPSPi2_RX 

DMAMUX1 Source  DMA_IP_REQ_MUX1_DISABLED 



Enable DMA Request  ☐ 

Figure 3.6 DMA RX Global Configuration - MCL module in EB Tresos

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

SPI shall be initialized during STARTUP phase of EcuM initialization. The API to be called for this is `Spi_↔Init()`. The MCU module should be initialized before the SPI is initialized. The API to be called for this purpose is `Spi_Init()`. The PORT and MCL (if the DMA option is used) modules shall be initialized before SPI is initialized.

4.2 Function Calls during Shutdown

SPI can be silenced by calling `Spi_DeInit()`.

4.3 Function Calls during Wake-up

N/A

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

SPI_EXCLUSIVE_AREA_00 is used in function Spi_AsyncTransmit to protect the schedule mechanism for the situation when a scheduling operation determined by a pending Spi_AsyncTransmit() call may be preempted by a job scheduling requested by an ISR event. It also protect concurrent Spi_AsyncTransmit() calls to schedule in the same time different jobs on the same SPI unit.

SPI_EXCLUSIVE_AREA_01 is used in function Spi_AsyncTransmit to protect the schedule mechanism for the situation when a scheduling operation determined by a pending Spi_AsyncTransmit() call may be preempted by a job scheduling requested by an ISR event.

SPI_EXCLUSIVE_AREA_02 is used in function Spi_Cancel to protect the status of the given Job result during Spi_Cancel called and ISR event.

SPI_EXCLUSIVE_AREA_02 is used in function Lpspi_Ip_LPSPi_0_IRQHandler to protect the status of the given Job result during Spi_Cancel called and ISR event.

SPI_EXCLUSIVE_AREA_02 is used in function Lpspi_Ip_LPSPi_1_IRQHandler to protect the status of the given Job result during Spi_Cancel called and ISR event.

Module requirements

SPI_EXCLUSIVE_AREA_02 is used in function `Lpspi_Ip_LPSPI_2_IRQHandler` to protect the status of the given Job result during `Spi_Cancel` called and ISR event.

SPI_EXCLUSIVE_AREA_02 is used in function `Lpspi_Ip_LPSPI_3_IRQHandler` to protect the status of the given Job result during `Spi_Cancel` called and ISR event.

SPI_EXCLUSIVE_AREA_02 is used in function `Lpspi_Ip_LPSPI_4_IRQHandler` to protect the status of the given Job result during `Spi_Cancel` called and ISR event.

SPI_EXCLUSIVE_AREA_02 is used in function `Lpspi_Ip_LPSPI_5_IRQHandler` to protect the status of the given Job result during `Spi_Cancel` called and ISR event.

SPI_EXCLUSIVE_AREA_03 is used in function `Spi_AsyncTransmit` to guarantee the atomicity of locking for the entire set of jobs belonging to an asynchronous sequence.

SPI_EXCLUSIVE_AREA_04 is used in function `Spi_WriteIB` to guarantee the atomicity of updating Channel's flags for transmission default data and all data in Channel's buffer if this function is called concurrent in ISR event.

SPI_EXCLUSIVE_AREA_05 is used in function `Spi_SetupEB` to guarantee the atomicity of updating Channel's flags for transmission default data, discard data and Channel's transmit/receive source pointers if this function is called concurrent in ISR event.

SPI_EXCLUSIVE_AREA_06 is used in function `Spi_SyncTransmit` to protect the status of SPI unit belong to the given sequence. Also it protects the global variable which contains the status of the `Spi_SyncTransmit` service. As stated by the Autosar, this service cannot be called when another sequence is during transmission, using this service.

SPI_EXCLUSIVE_AREA_07 is used in function `Spi_SyncTransmit` to to guarantee the atomicity of release SPI units for the given sequence. Also it protects the global variable which contains the status of the `Spi_↔ SyncTransmit` service. As stated by the Autosar, this service cannot be called when another sequence is during transmission, using this service.

SPI_EXCLUSIVE_AREA_08 is used in function `Spi_SyncTransmit` to guarantee the atomicity of updating SPI unit status in `Lpspi_Ip_SyncTransmit` due to `Lpspi_Ip_AsyncTransmit`, `Lpspi_Ip_AsyncTransmitFast` or `Lpspi_Ip_Cancel` called in ISR event.

SPI_EXCLUSIVE_AREA_09 is used in function `Spi_AsyncTransmit` to guarantee the atomicity of updating SPI unit status in `Lpspi_Ip_AsyncTransmit` due to `Lpspi_Ip_AsyncTransmitFast`, `Lpspi_Ip_SyncTransmit` or `Lpspi_Ip_Cancel` called in ISR event.

SPI_EXCLUSIVE_AREA_10 is used in function `Spi_Cancel` to guarantee the atomicity of updating of CR register and updating SPI unit status in `Lpspi_Ip_Cancel` due to `Lpspi_Ip_SyncTransmit`, `Lpspi_Ip_Async↔ Transmit`, `Lpspi_Ip_AsyncTransmitFast`, `Lpspi_TransmitTxInit`, `Lpspi_Ip_TransferProcess`, `Lpspi_Ip_IrqTx↔ DmaHandler` and `Lpspi_Ip_IrqRxDmaHandler` may be called in ISR event.

SPI_EXCLUSIVE_AREA_11 is used in function `Spi_SyncTransmit` to guarantee the atomicity of updating of CR register in `Lpspi_TransmitTxInit` due to `Lpspi_Ip_AsyncTransmitFast`, `Lpspi_Ip_Cancel`, `Lpspi_Ip_↔ TransferProcess`, `Lpspi_Ip_IrqTxDmaHandler` and `Lpspi_Ip_IrqRxDmaHandler` may be called in ISR event.

SPI_EXCLUSIVE_AREA_11 is used in function `Spi_AsyncTransmit` to guarantee the atomicity of updating of CR register in `Lpspi_TransmitTxInit` due to `Lpspi_Ip_AsyncTransmitFast`, `Lpspi_Ip_Cancel`, `Lpspi_Ip_↔ TransferProcess`, `Lpspi_Ip_IrqTxDmaHandler` and `Lpspi_Ip_IrqRxDmaHandler` may be called in ISR event.

SPI_EXCLUSIVE_AREA_12 is used in function Spi_AsyncTransmit to guarantee the atomicity of updating of CR register in Lpspi_Ip_TransferProcess due to Lpspi_Ip_Cancel, Lpspi_TransmitTxInit, Lpspi_Ip_AsyncTransmitFast, Lpspi_Ip_IrqTxDmaHandler and Lpspi_Ip_IrqRxDmaHandler may be called in ISR event.

SPI_EXCLUSIVE_AREA_13 is used in function Spi_AsyncTransmit to guarantee the atomicity of updating of CR register in Lpspi_Ip_IrqTxDmaHandler due to Lpspi_Ip_Cancel, Lpspi_TransmitTxInit, Lpspi_Ip_AsyncTransmitFast, Lpspi_Ip_TransferProcess and Lpspi_Ip_IrqRxDmaHandler may be called in ISR event.

SPI_EXCLUSIVE_AREA_14 is used in function Spi_AsyncTransmit to guarantee the atomicity of updating of CR register in Lpspi_Ip_IrqRxDmaHandler due to Lpspi_Ip_Cancel, Lpspi_TransmitTxInit, Lpspi_Ip_AsyncTransmitFast, Lpspi_Ip_TransferProcess and Lpspi_Ip_IrqTxDmaHandler may be called in ISR event.

SPI_EXCLUSIVE_AREA_15 is used in function Spi_AsyncTransmit to guarantee the atomicity of updating of CR register and updating SPI unit status in Lpspi_Ip_AsyncTransmitFast due to Lpspi_Ip_SyncTransmit, Lpspi_TransmitTxInit, Lpspi_Ip_Cancel, Lpspi_Ip_TransferProcess, Lpspi_Ip_IrqTxDmaHandler and Lpspi_Ip_IrqRxDmaHandler may be called in ISR event.

SPI_EXCLUSIVE_AREA_16 is used in function Spi_Cancel to guarantee the atomicity updating SPI unit status in Flexio_Spi_Ip_Cancel due to Flexio_Spi_Ip_SyncTransmit or Flexio_Spi_Ip_AsyncTransmit may be called in ISR event.

SPI_EXCLUSIVE_AREA_17 is used in function Spi_SyncTransmit to guarantee the atomicity updating SPI unit status in Flexio_Spi_Ip_SyncTransmit due to Flexio_Spi_Ip_AsyncTransmit or Flexio_Spi_Ip_Cancel may be called in ISR event.

SPI_EXCLUSIVE_AREA_18 is used in function Spi_AsyncTransmit to guarantee the atomicity updating SPI unit status in Flexio_Spi_Ip_AsyncTransmit due to Flexio_Spi_Ip_SyncTransmit or Flexio_Spi_Ip_Cancel may be called in ISR event.

Exclusive Areas implemented in Low level driver layer (IPL)

SPI_EXCLUSIVE_AREA_08 is used in function Lpspi_Ip_SyncTransmit to guarantee the atomicity of updating SPI unit status due to Lpspi_Ip_AsyncTransmit, Lpspi_Ip_AsyncTransmitFast or Lpspi_Ip_Cancel called in ISR event.

SPI_EXCLUSIVE_AREA_09 is used in function Lpspi_Ip_AsyncTransmit to guarantee the atomicity of updating SPI unit status due to Lpspi_Ip_AsyncTransmitFast, Lpspi_Ip_SyncTransmit or Lpspi_Ip_Cancel called in ISR event.

SPI_EXCLUSIVE_AREA_10 is used in function Lpspi_Ip_Cancel to guarantee the atomicity of updating of CR register and updating SPI unit status due to Lpspi_Ip_SyncTransmit, Lpspi_Ip_AsyncTransmit, Lpspi_Ip_AsyncTransmitFast, Lpspi_TransmitTxInit, Lpspi_Ip_TransferProcess, Lpspi_Ip_IrqTxDmaHandler and Lpspi_Ip_IrqRxDmaHandler may be called in ISR event.

SPI_EXCLUSIVE_AREA_11 is used in function Lpspi_Ip_SyncTransmit to guarantee the atomicity of updating of CR register in Lpspi_TransmitTxInit due to Lpspi_Ip_AsyncTransmitFast, Lpspi_Ip_Cancel, Lpspi_Ip_TransferProcess, Lpspi_Ip_IrqTxDmaHandler and Lpspi_Ip_IrqRxDmaHandler may be called in ISR event.

SPI_EXCLUSIVE_AREA_11 is used in function Lpspi_Ip_AsyncTransmit to guarantee the atomicity of updating of CR register in Lpspi_TransmitTxInit due to Lpspi_Ip_AsyncTransmitFast, Lpspi_Ip_Cancel, Lpspi_Ip_TransferProcess, Lpspi_Ip_IrqTxDmaHandler and Lpspi_Ip_IrqRxDmaHandler may be called in ISR event.

Module requirements

SPI_EXCLUSIVE_AREA_12 is used in function Lpspi_Ip_AsyncTransmit to guarantee the atomicity of updating of CR register in Lpspi_Ip_TransferProcess due to Lpspi_Ip_Cancel, Lpspi_TransmitTxInit, Lpspi_Ip_AsyncTransmitFast, Lpspi_Ip_IrqTxDmaHandler and Lpspi_Ip_IrqRxDmaHandler may be called in ISR event.

SPI_EXCLUSIVE_AREA_13 is used in function Lpspi_Ip_AsyncTransmit/Lpspi_Ip_AsyncTransmitFast to guarantee the atomicity of updating of CR register in Lpspi_Ip_IrqTxDmaHandler due to Lpspi_Ip_Cancel, Lpspi_TransmitTxInit, Lpspi_Ip_AsyncTransmitFast, Lpspi_Ip_TransferProcess and Lpspi_Ip_IrqRxDmaHandler may be called in ISR event.

SPI_EXCLUSIVE_AREA_14 is used in function Lpspi_Ip_AsyncTransmit/Lpspi_Ip_AsyncTransmitFast to guarantee the atomicity of updating of CR register in Lpspi_Ip_IrqRxDmaHandler due to Lpspi_Ip_Cancel, Lpspi_TransmitTxInit, Lpspi_Ip_AsyncTransmitFast, Lpspi_Ip_TransferProcess and Lpspi_Ip_IrqTxDmaHandler may be called in ISR event.

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

The critical regions from 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

#	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	I← S← Rs Crit- ical Re- gions (com- posed di- a- gram)
A← R← E← A← _← 00	X	X														X
A← R← E← A← _← 01	X	X														X
A← R← E← A← _← 02			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	I↵ S↵ Rs Crit- ical Re- gions (com- posed di- a- gram)	
A↵ R↵ E↵ A↵ ↵ 03				X													X
A↵ R↵ E↵ A↵ ↵ 04					X												X
A↵ R↵ E↵ A↵ ↵ 05						X											X
A↵ R↵ E↵ A↵ ↵ 06							X	X									X
A↵ R↵ E↵ A↵ ↵ 07							X	X									X
A↵ R↵ E↵ A↵ ↵ 08									X	X	X					X	X

Module requirements

#	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	A↵ R↵ E↵ A↵ ↵	I↵ S↵ Rs Crit- ical Re- gions (com- posed di- a- gram)
	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	
A↵ R↵ E↵ A↵ ↵ 09									X	X	X				X	X
A↵ R↵ E↵ A↵ ↵ 10									X	X	X	X	X	X	X	X
A↵ R↵ E↵ A↵ ↵ 11											X	X	X	X		X
A↵ R↵ E↵ A↵ ↵ 12											X	X	X	X		X
A↵ R↵ E↵ A↵ ↵ 13											X	X	X	X		X
A↵ R↵ E↵ A↵ ↵ 14									X	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	I← S← Rs Crit- ical Re- gions (com- posed di- a- gram)
I← SRs Crit- ical Re- gions (com- posed dia- gram)	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

5.2 Exclusive areas not available on this platform

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

5.3 Peripheral Hardware Requirements

None

5.4 ISR to configure within AutosarOS - dependencies

The following ISRs are used by the SPI driver and need to be assigned to a priority level. The interrupt vector numbers corresponding to INTERRUPT mode is as shown in Table below. For LPSPI, The interrupt occurs each time the TDF or RDF bits in SR register arises. For FLEXIO, The interrupt occurs each time the SSF bits in SHIFTSTAT register arises. (Note: Unused interrupts shouldn't be configured in the OS.)

Table 5.2 SPI ISRs

ISR Name	Cortex-M7 Vector	NVIC Interrupt ID
Lpspi_Ip_LPSPI_0_IRQHandler	181	165
Lpspi_Ip_LPSPI_1_IRQHandler	182	166
Lpspi_Ip_LPSPI_2_IRQHandler	183	167
Lpspi_Ip_LPSPI_3_IRQHandler	184	168
Lpspi_Ip_LPSPI_4_IRQHandler	185	169
Lpspi_Ip_LPSPI_5_IRQHandler	186	170
MCL_FLEXIO_ISR	155	139

Module requirements

Note

In case of `AUTOSAR_OS_NOT_USED`, the compiler option `"-DUSE_SW_VECTOR_MODE"` must be added to the list of compiler options to be used with interrupt controller configured to be in software vector mode.

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 be 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.
- **DEM** : The DEM module is used for enabling reporting of production relevant error status. The API function used is `Dem_SetEventStatus()`.
- **RESOURCE** :Resource module is used to select microcontroller's derivatives.
- **RTE** : The RTE module is needed for implementing data consistency of exclusive areas that are used by SPI module.
- **DET**: The DET module is used for enabling Default Error Tracer detection. The API function used is `Det_ReportError()`. The activation / deactivation of Default Error Tracer detection is configurable using the "SpiDevErrorDetect" configuration parameter.
- **ECUC**: The ECUC module is used for ECU configuration. MCAL modules need ECUC to retrieve the variant information.
- **PORT** : The PORT module is used to configure the port pins with the needed modes, before they are used by the SPI module. For each SPI, the SCK, SOUT, SIN and CSx_y signals need to be configured. In the S32G2XX Reference manual there is an example of the pin configuration. Please refer to the Reference List.
- **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 SPI driver. The SPI reference clock is provided by MCU plugin. The clock frequency may affect the Baudrate, Timing between clock and chip select, Timing between chip select and clock, Timing between chip select assertions. The reference is specified by the parameter `SpiGeneral\SpiPhyUnitClockRef`:



Figure 5.1 Spi reference clock provided by MCU plugin

- **MCL**: For each LPSPI in use, a transmit and a receive DMA channel need to be defined and routed through the DMA Multiplexer using MCL plugin. MCL should be initialized before SPI switch to DMA mode

For each FLEXIO_SPI is used, MCL is needed to configure Flexio channels (MOSI, MISO, CS, CLK) and enable/disable Flexio module and ISR handler

- **OS**: Needed to define a mapping between EcuC partitions and EcuC core ids when multicore support is enabled.

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 `Spi_Memmap`). Otherwise, the SPI driver has some dependencies. User must to put all variables, which were used for transmitter and receiver, to the **NON CACHEABLE** memory section in the RAM zone by the definition `SPI_START_SEC_VAR<INIT_POLICY>__<ALIGNMENT>__NO_CACHEABLE` and `SPI_STOP_SEC_VAR<INIT_POLICY>__<ALIGNMENT>__NO_CACHEABLE`

5.8 User Mode support

- [User Mode configuration in the module](#)
- [User Mode configuration in AutosarOS](#)

Spi module does not include registers protection. So, It is accessible to all registers in any public mode.

5.8.1 User Mode configuration in the module

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

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

Function syntax	Description	Available via
void Lpspi_Ip_SetUserAccess(uint8 Instance)	This function will enable writing all SPI registers under protection in User mode by configuring REG_PROT	Lpspi_Ip_TrustedFunctions.h
void Flexio_Spi_Ip_SetUserAccess(uint8 Instance)	This function will enable writing all FLEXIO_SPI registers under protection in User mode by configuring REG_PROT	Flexio_Spi_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 un-marshalling 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.

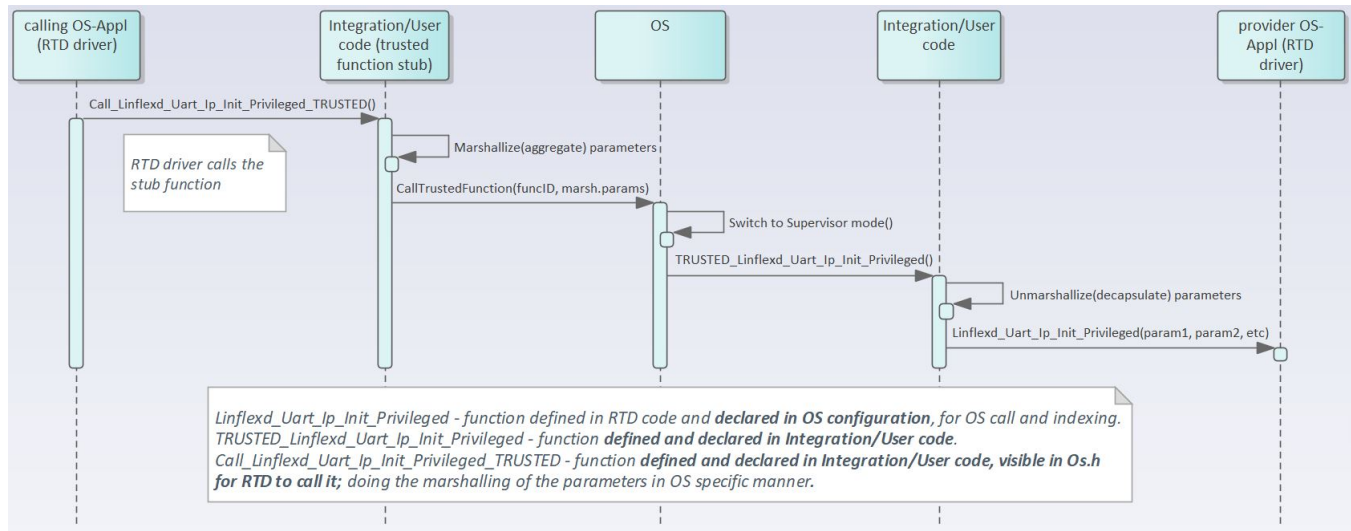


Figure 5.2 Example sequence chart for calling `Linflexd_Uart_Ip_Init_Privileged` as trusted function

5.9 Multicore support

- The SPI implements the "Autosar R21-11 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_BSWDistribution↔ Guide.
- The SPI and the mappable elements can be allocated to one or several ECUC partitions, by means of "↔ SpiEcucPartionRef". If the SPI is mapped to zero ECUC partitions, the SPI behavior reverts to single-core implementation, similar to previous Autosar versions. If the SPI is mapped to one or more ECUC partitions, the SPI enforces the following multicore assumptions:
 - The SPI 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 SPI 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 SPI internal variables and the EcucCoreIDs are used for indexing those variables. (AR-86601 Zero based and dense IDs for OS-Cores and OSApplications)
 - The SPI assumes that initialization is performed on each core, `Spi_Init()` is called separately for each core, using a different configuration structure. (Type II)

Module requirements

- The SPI 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; SPI_Init function will be called with SPI_Config_EcucPartition_1 configuration structure, on Core 1.
- The SPI 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 SPI requires that all variables in NonCacheable MemMap sections be allocated accordingly, to avoid data corruption in multicore context.
- The SPI assumes that RTE module implements the EXCLUSIVE AREAS to be coreaware 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 SPI usecase, to be detailed/removed if some modules require such kind of functionality for critical features which cannot be atomically shared among cores)
- The SPI 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 SPI_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

The function `Spi_MainFunction_Handling()` should be called periodically only if polling mode is enabled for `Spi->_AsyncTransmit()` .

6.2 API Requirements

None.

6.3 Calls to Notification Functions, Callbacks, Callouts

Call-back Notifications: None

User Notification: The SPI Handler & Driver provides notifications per job and sequence in asynchronous mode. The notifications can be configured as pointers to user defined functions. If notification is not desired, the appropriate `EndNotification` field shall be left blank. For asynchronous transmissions, job and sequences notifications are performed before the scheduling of the next job (contrary to the recommendation given by SWS_Spi_00088) . In this way, calls like `Spi_SetupIB()` or `Spi_WriteIB()` can be targeted on the next schedulable jobs, before the starting of the job transfer.

Chapter 7

Memory allocation

- [Sections to be defined in Spi_MemMap.h](#)
- [Linker command file](#)

7.1 Sections to be defined in Spi_MemMap.h

Section name	Type of section	Description
SPI_START_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	Start of Memory Section for Config Data when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit.
SPI_STOP_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	End of Memory Section for Config Data
SPI_START_SEC_CONST_32	Configuration Data	Start of Memory Section for constant data which have to be aligned to 32 bit.
SPI_STOP_SEC_CONST_32	Configuration Data	End of Memory Section for constant data.
SPI_START_SEC_CONST_UNSPECIFIED	Configuration Data	Start of Memory Section for constant data when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit.
SPI_STOP_SEC_CONST_UNSPECIFIED	Configuration Data	End of Memory Section for constant data.
SPI_START_SEC_CODE	Code	Start of memory Section for Code
SPI_STOP_SEC_CODE	Code	End of memory Section for Code
SPI_START_SEC_VAR_Cleared_32	Variables	Used for variables which have to be aligned to 32 bit. For instance used for variables of size 32 bit or used for composite data types: arrays, structures containing elements of maximum 32 bits. These variables are cleared to zero by start-up code.
SPI_STOP_SEC_VAR_Cleared_32	Variables End of above section.	End of above section.

Section name	Type of section	Description
SPI_START_SEC_VAR_CLEARED_32_NO_CACHEABLE	Variables	Used for variables which have to be aligned to 32 bit. For instance used for variables of size 32 bit or used for composite data types: arrays, structures containing elements of maximum 32 bits and that have to be stored in a non-cacheable memory section. These variables are cleared to zero by start-up code.
SPI_STOP_SEC_VAR_CLEARED_32_NO_CACHEABLE	Variables End of above section.	End of above section.
SPI_START_SEC_VAR_CLEARED_UNSPECIFIED	Variables	Used for variables, structures, arrays when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit. These variables are cleared to zero by start-up code.
SPI_STOP_SEC_VAR_CLEARED_UNSPECIFIED	Variables	End of above section.
SPI_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, and that have to be stored in a non-cacheable memory section. These variables are cleared to zero by start-up code.
SPI_STOP_SEC_VAR_CLEARED_UNSPECIFIED_NO_CACHEABLE	Variables	End of above section.
SPI_START_SEC_VAR_INIT_UNSPECIFIED	Variables	Used for variables, structures, arrays when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit.
SPI_STOP_SEC_VAR_INIT_UNSPECIFIED	Variables	End of above section.

7.2 Linker command file

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

Chapter 8

Integration Steps

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

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

Chapter 9

External assumptions for driver

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

External Assumption Req ID	External Assumption Text
SWS_Spi_00244	The SPI Handler/Driver module does not take care of setting the registers which configure the clock, prescaler(s) and PLL in its init function. This has to be done by the MCU module [REF].
SWS_Spi_00052	For the IB Channels, the Handler/Driver shall provide the buffering but it is not able to take care of the consistency of the data in the buffer during transmission. The size of the Channel buffer is fixed.
SWS_Spi_00257	The SPI Handler/Driver is not able to prevent the overwriting of these "transmit" buffers by users during transmissions.
SWS_Spi_00053	For EB Channels the application shall provide the buffering and shall take care of the consistency of the data in the buffer during transmission.
SWS_Spi_00280	The buffer provided by the application for the SPI Handler Driver may have a different size. Note: This refers in the context of External Buffer
SWS_Spi_00084	If different Jobs (and consequently also Sequences) have common Channels, the SPI Handler/Driver' environment shall ensure that read and/or write functions are not called during transmission.
SWS_Spi_00121	The SPI Handler/Driver's environment shall configure the Spi↔InterruptibleSeqAllowed parameter (ON / OFF) in order to select which kind of Sequences the SPI Handler/Driver manages.
SWS_Spi_00080	When using Interruptible Sequences, the caller must be aware that if the multiple Sequences access the same Channels, the data for these Channels may be overwritten by the highest priority Job accessing each Channel. Note:
SWS_Spi_00298	The operation Spi_Init is Non Re-entrant.
SWS_Spi_00300	The operation Std_ReturnType Spi_DeInit() is Non Re-entrant.
SWS_Spi_00173	The SPI Handler/Driver's environment shall call the function Spi↔Transmit after a function call of Spi_SetupEB for EB Channels or a function call of Spi_WriteIB for IB Channels but before the function call Spi_Read↔IB.
SWS_Spi_00027	The SPI Handler/Driver's environment shall call the function Spi_ReadIB after a Transmit method call to have relevant data within IB Channel.
SWS_Spi_00037	The SPI Handler/Driver's environment shall call the Spi_SetupEB function once for each Channel with EB declared before the SPI Handler/Driver's environment calls a Transmit method on them.

External assumptions for driver

External Assumption Req ID	External Assumption Text
SWS_Spi_00038	The SPI Handler/Driver's environment shall call the function Spi_GetJob↔Result to inquire whether the Job transmission has succeeded (SPI_JOB↔_OK) or failed (SPI_JOB_FAILED).
SWS_Spi_00042	The SPI Handler/Driver's environment shall call the function Spi_Get↔SequenceResult to inquire whether the full Sequence transmission has succeeded (SPI_SEQ_OK) or failed (SPI_SEQ_FAILED).
SWS_Spi_00287	The SPI Handler/Driver's environment shall call this function to inquire whether the specified SPI Hardware microcontroller peripheral is SPI_IDLE or SPI_BUSY. Note: This requirement refers to Spi_GetHWUnitStatus()
SWS_Spi_00335	The operation Spi_SetAsyncMode is Non Re-entrant.
SWS_Spi_00265	For implement the call back function other modules are required to provide the routines in the expected manner.
SWS_Spi_00048	The callback notifications Spi_JobEndNotification and Spi_SeqEnd↔Notification shall have no parameters and no return value.
SWS_Spi_00085	It is allowed to use the following API calls within the SPI callback notifications: Spi_ReadIBSpi_WriteIBSpi_SetupEBSpi_GetJob↔ResultSpi_GetSequenceResultSpi_GetHWUnitStatusSpi_CancelAll other SPI Handler/Driver API calls are not allowed.
SWS_Spi_00340	The operation SpiJobEndNotification is Re-entrant.
SWS_Spi_00341	The operation SpiSeqEndNotification is Re-entrant.
SWS_Spi_00077	To transmit a variable number of data, it is mandatory to call the Spi_↔SetupEB function to store new parameters within SPI Handler/Driver before each Spi_AsyncTransmit function call.
SWS_Spi_00078	To transmit a constant number of data, it is only mandatory to call the Spi↔_SetupEB function to store parameters within SPI Handler/Driver before the first Spi_AsyncTransmit function call.
SWS_Spi_00235	If not applicable, the SPI Handler/Driver module's environment shall pass a NULL pointer to the function Spi_Init.
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 Spi is allocated shall be exclusively mapped to a core. Note↔: Internally, the Spi 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 Spi_Init() for each core, using the dedicated configuration pointer for that core.
EA_RTD_00096	The application shall pass the correct initialization pointer, specific to the partition in which the driver is to be used.

External Assumption Req ID	External Assumption Text
EA_RTD_00106	Standalone IP configuration and HL configuration of the same driver shall be done in the same project
EA_RTD_00107	The integrator shall use the IP interface only for hardware resources that were configured for standalone IP usage. Note: The integrator shall not directly use the IP interface for hardware resources that were allocated to be used in HL context.
EA_RTD_00108	The integrator shall use the IP interface to build a CDD, therefore the BSWMD will not contain reference to the IP interface

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.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altivec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. ARM, AMBA, ARM Powered, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and Vision are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. ARM7, ARM9, ARM11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2023 NXP B.V.

