Document Title	Specification of SPI Handler / Driver
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	038

Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.4.0

Document Change History

Date	Release	Changed by	Change Description
2018-10-31	4.4.0	AUTOSAR Release Management	• Editorial changes
2017-12-08	4.3.1	AUTOSAR Release Management	• SPI_E_SEQ_IN_PROCESS and SPI_E_SEQ_PENDING are migrated to runtime errors • The notion of prearranged bus is removed to simplify the use • Modified or removed requirements: SWS_Spi_00135, SWS_Spi_00324, SWS_Spi_00039 • Restored requirement: SWS_Spi_00035
2016-11-30	4.3.0	AUTOSAR Release Management	• Requirements removed: SWS_Spi_00339, SWS_Spi_00191, SWS_Spi_00367, SWS_Spi_00239, SWS_Spi_00056, SWS_Spi_00076, SWS_Spi_00148 • Requirements updated: SWS_Spi_00999, SWS_Spi_00092 • Improvement of the traceability with SRS BSW General requirements • Editorial changes
2015-07-31	4.2.2	AUTOSAR Release Management	• Cleanup of requirements chapter • Debugging support marked as obsolete • Editorial changes

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
2014-10-31	4.2.1	AUTOSAR Release Management	• Added SWS_Spi_00383, SWS_Spi_00384, SWS_Spi_00385, SWS_Spi_00386 and ECUC_Spi_00243 • New configuration parameter SpiUserCallbackHeaderFile • SPI hardware error is applicable for sync and async transmits • Editorial changes
2014-03-31	4.1.3	AUTOSAR Release Management	• Description for Spi_AsyncTransmit and Spi_SyncTransmit development errors for already ongoing transmission • Clarification of Spi Channel width and data access type relation
2013-10-31	4.1.2	AUTOSAR Release Management	• ECUC_Spi_00242 (added) • ECUC_Spi_00240 (added) • SWS_Spi_00189 (modified) • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• Added chapter 7.6 and 7.7, table from chapter 7.4 moved to chapter 7.7 • SWS_Spi_00129 removed, SWS_Spi_00128 reformulated • ECUC_Spi_00180, ECUC_Spi_00204 Lengh is in data elements instead of bytes • MemMap header file remname • Added Subchapter 3.x due to SWS General Rollout

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
2011-12-22	4.0.3	AUTOSAR Administration	• Rephrased: requirement SWS_Spi_00002, SWS_Spi_00046, SWS_Spi_00129, SWS_Spi_00233, SWS_Spi_00163, SPI 171, SWS_Spi_00172, SWS_Spi_00289 and SWS_Spi_00290, block 2 in chapter 7.2.2 • Removed: requirement SPI083; SPI132, SPI284 and SPI107 removed from statement • Corrected:Dem_EventStatusType in SWS_Spi_00191, Spi_SyncTransmit Syn/Async changed to Synchronous, SPI_E_PARAM_POINTER in SWS_Spi_00371, • Reference to MCU in SWS_Spi_00244 and SWS_Spi_00342 • Added: requirement SWS_Spi_00140, chapter 10 - SpiCsSelection, SWS_Spi_00194 - SPI_JOB_QUEUED state introduced, SWS_Spi_00195 with error table update • Modified: SWS_Spi_00114 and SWS_Spi_00135, chapter 10 - SpiEnableCs
2010-09-30	3.1.5	AUTOSAR Administration	• Added SWS_Spi_00369, SWS_Spi_00371, SWS_Spi_00370 • Removed SPI190, SPI094 • Updated configuration: base on min-max value for defined parameter; SpiHwUnit belongs to SpiExternalDevice Container; updated SpiTimeClk2Cs

Document Change History			
Date	**Release**	**Changed by**	**Change Description**
2010-02-02	3.1.4	AUTOSAR Administration	• Splitting and refinement of several requirements • Removal of redundant requirements • Introduction of new IDs to allow implementation of debugging concept • Inserted UML diagram in chapter 9 • Updating of Chapter 10 with the inclusion of 2 new container and the definition of the Chip Select configuration • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Administration	• Updated Chapter 10 with the inclusion of CS configuration • Document meta information extended • Small layout adaptations made
2007-01-24	2.1.15	AUTOSAR Administration	• Configuration Specification updating • General rephrasing for clarification • Syntax error • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
2006-05-16	2.0	AUTOSAR Administration	• Document structure adapted to common Release 2.0 SWS Template. • Major changes in chapter 10 • Structure of document changed partly • Other changes see chapter 13
2005-05-31	1.0	AUTOSAR Administration	• Initial Release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview 9
2 Acronyms and abbreviations 10
3 Related documentation 11
3.1 Input documents 11
3.2 Related standards and norms 11
3.3 Related specification 12
4 Constraints and assumptions 12
4.1 Limitations 12
4.2 Applicability to car domains 12
5 Dependencies to other modules 13
6 Requirements traceability 14
7 Functional specification 21
7.1 Overall view of functionalities and features 21
7.2 General behaviour 22
7.2.1 Common configurable feature: Allowed Channel Buffers 25
7.2.2 LEVEL 0, Simple Synchronous behaviour 26
7.2.3 LEVEL 1, Basic Asynchronous behavior 27
7.2.4 Asynchronous configurable feature: Interruptible Sequences 29
7.2.5 LEVEL 2, Enhanced behaviour 31
7.3 Scheduling Advices 32
7.4 Error classification 32
7.4.1 Development Errors 33
7.4.2 Runtime Errors 33
7.4.3 Transient faults 34
7.4.4 Production Errors 34
7.4.5 Extended Production Errors 34
7.5 Error detection 34
7.5.1 API parameter checking 34
7.5.2 SPI state checking 35
8 API specification 36
8.1 Imported types 36
8.2 Type definitions 36
8.2.1 Spi_ConfigType 36
8.2.2 Spi_StatusType 37
8.2.3 Spi_JobResultType 38
8.2.4 Spi_SeqResultType 39
8.2.5 Spi_DataBufferType 39
8.2.6 Spi_NumberOfDataType 40
8.2.7 Spi_ChannelType 40
8.2.8 Spi_JobType 40
8.2.9 Spi_SequenceType 41
8.2.10 Spi_HWUnitType 41
8.2.11 Spi_AsyncModeType 41

8.3 Function definitions 42
8.3.1 Spi_Init 42
8.3.2 Spi_DeInit 43
8.3.3 Spi_WriteIB 44
8.3.4 Spi_AsyncTransmit 45
8.3.5 Spi_ReadIB 47
8.3.6 Spi_SetupEB 48
8.3.7 Spi_GetStatus 51
8.3.8 Spi_GetJobResult 51
8.3.9 Spi_GetSequenceResult 52
8.3.10 Spi_GetVersionInfo 53
8.3.11 Spi_SyncTransmit 53
8.3.12 Spi_GetHWUnitStatus 54
8.3.13 Spi_Cancel 55
8.3.14 Spi_SetAsyncMode 56
8.4 Callback notifications 57
8.5 Scheduled functions 57
8.5.1 Spi_MainFunction_Handling 58
8.6 Expected Interfaces 58
8.6.1 Mandatory Interfaces 58
8.6.2 Optional Interfaces 58
8.6.3 Configurable interfaces 58
9 Sequence diagrams 61
9.1 Initialization 61
9.2 Modes transitions 61
9.3 Write/AsyncTransmit/Read (IB) 62
9.3.1 One Channel, one Job then one Sequence 62
9.3.2 Many Channels, one Job then one Sequence 64
9.3.3 Many Channels, many Jobs and one Sequence 65
9.3.4 Many Channels, many Jobs and many Sequences 67
9.4 Setup/AsyncTransmit (EB) 69
9.4.1 Variable Number of Data / Constant Number of Data 69
9.4.2 One Channel, one Job then one Sequence 69
9.4.3 Many Channels, one Job then one Sequence 71
9.4.4 Many Channels, many Jobs and one Sequence 72
9.4.5 Many Channels, many Jobs and many Sequences 74
9.5 Mixed Jobs Transmission 76
9.6 LEVEL 0 SyncTransmit diagrams 76
9.6.1 Write/SyncTransmit/Read (IB): Many Channels, many Jobs and one Sequence 76
9.6.2 Setup/SyncTransmit (EB): Many Channels, many Jobs and one Sequence 77
10 Configuration specification 79
10.1 How to read this chapter 79
10.2 Containers and configuration parameters 79
10.2.1 Spi 79
10.2.2 SpiDemEventParameterRefs 79
10.2.3 SpiGeneral 80
10.2.4 SpiSequence 83

10.2.5 SpiChannel 85
10.2.6 SpiChannelList 87
10.2.7 SpiJob 88
10.2.8 SpiExternalDevice 89
10.2.9 SpiDriver 93
10.2.10 SpiPublishedInformation 94
10.3 Published information 94
10.4 Configuration concept 94
11 Not applicable requirements 97
12 Appendix 98

1 Introduction and functional overview

The SPI Handler/Driver provides services for reading from and writing to devices connected via SPI busses. It provides access to SPI communication to several users (e.g. EEPROM, Watchdog, I/O ASICs). It also provides the required mechanism to configure the onchip SPI peripheral.

This specification describes the API for a monolithic SPI Handler/Driver. This software module includes handling and driving functionalities. Main objectives of this monolithic SPI Handler/Driver are to take the best of each microcontroller features and to allow implementation optimization depending on static configuration to fit as much as possible to ECU needs.

Hence, this specification defines selectable levels of functionalities and configurable features to allow the design of a high scalable module that exploits the peculiarities of the microcontroller.

To configure the SPI Handler/Driver these steps shall be followed:

- SPI Handler/Driver Level of Functionality shall be selected and optional features configured.
- SPI Channels shall be defined according to data usage, and they could be buffered inside the SPI Handler/Driver (IB) or provided by the user (EB).
- SPI Jobs shall be defined according to HW properties (CS), and they will contain a list of channels using those properties.
- As a final step, Sequences of Jobs shall be defined, in order to transmit data in a sorted way (priority sorted).

The general behaviour of the SPI Handler/Driver can be asynchronous or synchronous according to the Level of Functionality selected.

The specification covers the Handler/Driver functionality combined in one single module. One is the SPI handling part that handles multiple access to busses that could be located in the ECU Abstraction layer. The other part is the SPI driver that accesses the microcontroller hardware directly that could be located in the Microcontroller Abstraction layer.

2 Acronyms and abbreviations

Acronyms and abbreviations which have a local scope and therefore are not contained in the AUTOSAR glossary must appear in a local glossary.

Acronym:	***Description:***
DET	Default Error Tracer – module to which errors are reported.
DEM	Diagnostic Event Manager – module to which production relevant errors are reported.
SPI	Serial Peripheral Interface. It is exactly defined hereafter in this document.
CS	Chip Select
MISO	Master Input Slave Output
MOSI	Master Output Slave Input

Abbreviation:	***Description:***
EB	Externally buffered channels. Buffers containing data to transfer are outside the SPI Handler/Driver.
IB	Internally buffered channels. Buffers containing data to transfer are inside the SPI Handler/Driver.
ID	Identification Number of an element (Channel, Job, Sequence).

Definition:	***Description:***
Channel	A Channel is a software exchange medium for data that are defined with the same criteria: Config. Parameters, Number of Data elements with same size and data pointers (Source & Destination) or location.
Job	A Job is composed of one or several Channels with the same Chip Select (is not released during the processing of Job). A Job is considered atomic and therefore cannot be interrupted by another Job. A Job has an assigned priority.
Sequence	A Sequence is a number of consecutive Jobs to transmit but it can be rescheduled between Jobs using a priority mechanism. A Sequence transmission is interruptible (by another Sequence transmission) or not depending on a static configuration.

3 Related documentation

3.1 Input documents

[1] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[2] General Requirements on SPAL
AUTOSAR_SRS_SPALGeneral.pdf

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[4] Specification of Default Error Tracer
AUTOSAR_SWS_DefaultErrorTracer.pdf

[5] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[6] Requirements on SPI Handler/Driver
AUTOSAR_SRS_SPIHandlerDriver.pdf

[7] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager.pdf

[8] Glossary
AUTOSAR_TR_Glossary.pdf

[9] Specification of MCU Driver
AUTOSAR_SWS_MCUDriver .pdf

[10] Specification of PORT Driver
AUTOSAR_SWS_PORTDriver

[11] Basic Software Module Description Template,
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[12] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

[13] Specification of Standard Types,
AUTOSAR_SWS_StandardTypes.pdf

[14] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral.pdf

3.2 Related standards and norms

Not related.

3.3 Related specification

AUTOSAR provides a General Specification on Basic Software modules [14] (SWS BSW General), which is also valid for SPI Handler Driver.

Thus, the specification SWS BSW General shall be considered as additional and required specification for SPI Handler Driver.

4 Constraints and assumptions

4.1 Limitations

[SWS_Spi_00040] ⌈ The SPI Handler/Driver handles only the Master mode.⌋ ()

[SWS_Spi_00050] ⌈ The SPI Handler/Driver only supports full-duplex mode.⌋ ()

[SWS_Spi_00108] ⌈ The LEVEL 2 SPI Handler/Driver is specified for microcontrollers that have to provide, at least, two SPI busses using separated hardware units. Otherwise, using this level of functionality does not make sense.⌋ ()

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

[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 [9].⌋ ()

Note: SPI peripherals may depend on the system clock, prescaler(s) and PLL. Thus, any change of the system clock (e.g. PLL on / PLL off / clock dividers) may also affect the clock settings of the SPI hardware.

[SWS_Spi_00342] ⌈ Depending on microcontrollers, the SPI peripheral could share registers with other peripherals. In this typical case, the SPI Handler/Driver has a relationship with MCU module [9] for initialising and de-initialising those registers.⌋ ()

[SWS_Spi_00343] ⌈ If Chip Selects are done using microcontroller pins the SPI Handler/Driver has a relationship with PORT module [10]. In this case, this specification assumes that these microcontroller pins are directly accessed by the SPI Handler/Driver module without using APIs of DIO module.
Anyhow, the SPI depends on ECU hardware design and for that reason it may depend on other modules.⌋ ()

6 Requirements traceability

Requirement	Description	Satisfied by
SRS_BSW_00005	Modules of the µC Abstraction Layer (MCAL) may not have hard coded horizontal interfaces	SWS_Spi_00999
SRS_BSW_00006	The source code of software modules above the µC Abstraction Layer (MCAL) shall not be processor and compiler dependent.	SWS_Spi_00999
SRS_BSW_00009	All Basic SW Modules shall be documented according to a common standard.	SWS_Spi_00999
SRS_BSW_00010	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	SWS_Spi_00999
SRS_BSW_00101	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	SWS_Spi_00013, SWS_Spi_00015
SRS_BSW_00161	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	SWS_Spi_00999
SRS_BSW_00164	The Implementation of interrupt service routines shall be done by the Operating System, complex drivers or modules	SWS_Spi_00999
SRS_BSW_00168	SW components shall be tested by a function defined in a common API in the Basis-SW	SWS_Spi_00999
SRS_BSW_00170	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	SWS_Spi_00999
SRS_BSW_00172	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	SWS_Spi_00999
SRS_BSW_00301	All AUTOSAR Basic Software Modules shall only import the necessary information	SWS_Spi_00999
SRS_BSW_00302	All AUTOSAR Basic Software Modules shall only export information needed by other modules	SWS_Spi_00999
SRS_BSW_00306	AUTOSAR Basic Software Modules shall be compiler and platform independent	SWS_Spi_00999
SRS_BSW_00307	Global variables naming convention	SWS_Spi_00999

SRS_BSW_00308	AUTOSAR Basic Software Modules shall not define global data in their header files, but in the C file	SWS_Spi_00999
SRS_BSW_00309	All AUTOSAR Basic Software Modules shall indicate all global data with read-only purposes by explicitly assigning the const keyword	SWS_Spi_00999
SRS_BSW_00312	Shared code shall be reentrant	SWS_Spi_00999
SRS_BSW_00323	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	SWS_Spi_00031, SWS_Spi_00032, SWS_Spi_00060
SRS_BSW_00325	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	SWS_Spi_00999
SRS_BSW_00327	Error values naming convention	SWS_Spi_00004
SRS_BSW_00328	All AUTOSAR Basic Software Modules shall avoid the duplication of code	SWS_Spi_00999
SRS_BSW_00330	It shall be allowed to use macros instead of functions where source code is used and runtime is critical	SWS_Spi_00999
SRS_BSW_00331	All Basic Software Modules shall strictly separate error and status information	SWS_Spi_00999
SRS_BSW_00334	All Basic Software Modules shall provide an XML file that contains the meta data	SWS_Spi_00999
SRS_BSW_00335	Status values naming convention	SWS_Spi_00019, SWS_Spi_00061, SWS_Spi_00062, SWS_Spi_00373
SRS_BSW_00336	Basic SW module shall be able to shutdown	SWS_Spi_00021, SWS_Spi_00022
SRS_BSW_00337	Classification of development errors	SWS_Spi_00004
SRS_BSW_00341	Module documentation shall contains all needed informations	SWS_Spi_00999
SRS_BSW_00342	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	SWS_Spi_00999
SRS_BSW_00343	The unit of time for specification and configuration of Basic SW modules shall be preferably in physical time unit	SWS_Spi_00999
SRS_BSW_00344	BSW Modules shall support link-time configuration	SWS_Spi_00009
SRS_BSW_00347	A Naming seperation of different instances of BSW drivers shall be in place	SWS_Spi_00999
SRS_BSW_00359	All AUTOSAR Basic Software Modules callback functions shall avoid return types other than void if possible	SWS_Spi_00048
SRS_BSW_00360	AUTOSAR Basic Software Modules callback functions are allowed to have	SWS_Spi_00048

	parameters	
SRS_BSW_00369	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	SWS_Spi_00048
SRS_BSW_00375	Basic Software Modules shall report wake-up reasons	SWS_Spi_00999
SRS_BSW_00385	List possible error notifications	SWS_Spi_00004
SRS_BSW_00399	Parameter-sets shall be located in a separate segment and shall be loaded after the code	SWS_Spi_00999
SRS_BSW_00400	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	SWS_Spi_00999
SRS_BSW_00401	Documentation of multiple instances of configuration parameters shall be available	SWS_Spi_00999
SRS_BSW_00405	BSW Modules shall support multiple configuration sets	SWS_Spi_00008, SWS_Spi_00013
SRS_BSW_00406	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	SWS_Spi_00015, SWS_Spi_00046, SWS_Spi_00373
SRS_BSW_00413	An index-based accessing of the instances of BSW modules shall be done	SWS_Spi_00999
SRS_BSW_00416	The sequence of modules to be initialized shall be configurable	SWS_Spi_00999
SRS_BSW_00417	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	SWS_Spi_00999
SRS_BSW_00422	Pre-de-bouncing of error status information is done within the DEM	SWS_Spi_00999
SRS_BSW_00423	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	SWS_Spi_00999
SRS_BSW_00424	BSW module main processing functions shall not be allowed to enter a wait state	SWS_Spi_00999
SRS_BSW_00426	BSW Modules shall ensure data consistency of data which is shared between BSW modules	SWS_Spi_00999
SRS_BSW_00427	ISR functions shall be defined and documented in the BSW module description template	SWS_Spi_00999
SRS_BSW_00428	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	SWS_Spi_00999
SRS_BSW_00429	Access to OS is restricted	SWS_Spi_00999
SRS_BSW_00432	Modules should have separate main processing functions for read/receive	SWS_Spi_00999

	and write/transmit data path	
SRS_BSW_00433	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	SWS_Spi_00999
SRS_SPAL_00157	All drivers and handlers of the AUTOSAR Basic Software shall implement notification mechanisms of drivers and handlers	SWS_Spi_00026, SWS_Spi_00038, SWS_Spi_00042, SWS_Spi_00057, SWS_Spi_00071, SWS_Spi_00073, SWS_Spi_00075, SWS_Spi_00324
SRS_SPAL_12056	All driver modules shall allow the static configuration of notification mechanism	SWS_Spi_00009, SWS_Spi_00044, SWS_Spi_00054, SWS_Spi_00064
SRS_SPAL_12057	All driver modules shall implement an interface for initialization	SWS_Spi_00013, SWS_Spi_00015
SRS_SPAL_12063	All driver modules shall only support raw value mode	SWS_Spi_00999
SRS_SPAL_12064	All driver modules shall raise an error if the change of the operation mode leads to degradation of running operations	SWS_Spi_00021, SWS_Spi_00025
SRS_SPAL_12067	All driver modules shall set their wake-up conditions depending on the selected operation mode	SWS_Spi_00999
SRS_SPAL_12068	The modules of the MCAL shall be initialized in a defined sequence	SWS_Spi_00999
SRS_SPAL_12069	All drivers of the SPAL that wake up from a wake-up interrupt shall report the wake-up reason	SWS_Spi_00999
SRS_SPAL_12075	All drivers with random streaming capabilities shall use application buffers	SWS_Spi_00053
SRS_SPAL_12077	All drivers shall provide a non blocking implementation	SWS_Spi_00999
SRS_SPAL_12078	The drivers shall be coded in a way that is most efficient in terms of memory and runtime resources	SWS_Spi_00999
SRS_SPAL_12092	The driver's API shall be accessed by its handler or manager	SWS_Spi_00999
SRS_SPAL_12125	All driver modules shall only initialize the configured resources	SWS_Spi_00008, SWS_Spi_00009, SWS_Spi_00013
SRS_SPAL_12129	The ISRs shall be responsible for resetting the interrupt flags and calling the according notification function	SWS_Spi_00999
SRS_SPAL_12163	All driver modules shall implement an interface for de-initialization	SWS_Spi_00021, SWS_Spi_00022
SRS_SPAL_12265	Configuration data shall be kept constant	SWS_Spi_00999
SRS_SPAL_12267	Wakeup sources shall be initialized by MCAL drivers and/or the MCU driver	SWS_Spi_00999
SRS_Spi_12024	The SPI Handler/Driver shall allow the static configuration of the following options	SWS_Spi_00008, SWS_Spi_00063

SRS_Spi_12025	The SPI Handler/Driver shall allow the static configuration of all software and hardware properties related to SPI	SWS_Spi_00008, SWS_Spi_00009, SWS_Spi_00052, SWS_Spi_00053, SWS_Spi_00063
SRS_Spi_12026	The SPI Handler/Driver shall allow the static configuration of the desired number of SPI channels	SWS_Spi_00009
SRS_Spi_12032	For an SPI channel assigned to an SPI HW Unit the chip select mode "normal" shall be available	SWS_Spi_00009, SWS_Spi_00066
SRS_Spi_12033	For an SPI channel assigned to an SPI HW Unit the chip select mode "hold" shall be available	SWS_Spi_00009, SWS_Spi_00066
SRS_Spi_12037	The SPI Handler/Driver shall allow a priority controlled allocation of the HW SPI unit	SWS_Spi_00014, SWS_Spi_00059, SWS_Spi_00124, SWS_Spi_00127
SRS_Spi_12093	The SPI Handler/Driver shall be able to handle multiple busses of communication	SWS_Spi_00009, SWS_Spi_00010, SWS_Spi_00034, SWS_Spi_00041, SWS_Spi_00135
SRS_Spi_12094	The SPI Handler/Driver shall handle the chip select	SWS_Spi_00009, SWS_Spi_00066
SRS_Spi_12099	The SPI Handler/Driver shall provide an asynchronous read functionality	SWS_Spi_00016, SWS_Spi_00020, SWS_Spi_00162, SWS_Spi_00163
SRS_Spi_12101	The SPI Handler/Driver shall provide an asynchronous write functionality	SWS_Spi_00018, SWS_Spi_00020, SWS_Spi_00162, SWS_Spi_00163
SRS_Spi_12103	The SPI Handler/Driver shall provide an asynchronous read-write functionality	SWS_Spi_00020, SWS_Spi_00053, SWS_Spi_00058, SWS_Spi_00067, SWS_Spi_00162, SWS_Spi_00163
SRS_Spi_12104	The SPI Handler/Driver shall provide a synchronous functionality which returns any transfer status	SWS_Spi_00025, SWS_Spi_00026, SWS_Spi_00324
SRS_Spi_12108	The SPI Handler/Driver shall call the statically configured notification function	SWS_Spi_00057, SWS_Spi_00118, SWS_Spi_00119, SWS_Spi_00120
SRS_Spi_12150	The SPI Handler/Driver shall allow the static configuration of all software and hardware properties related to asynchronous SPI aspects	SWS_Spi_00009, SWS_Spi_00064, SWS_Spi_00093
SRS_Spi_12152	The SPI Handler/Driver shall provide a synchronous read functionality	SWS_Spi_00016, SWS_Spi_00134
SRS_Spi_12153	The SPI Handler/Driver shall provide a synchronous write functionality	SWS_Spi_00018, SWS_Spi_00134
SRS_Spi_12154	The SPI Handler/Driver shall provide a synchronous write-read functionality	SWS_Spi_00134
SRS_Spi_12170	The SPI Handler/Driver shall not provide the ability to prevent a channel data overwrite	SWS_Spi_00042, SWS_Spi_00084
SRS_Spi_12179	The SPI Handler/Driver shall allow linking consecutive SPI channels by static configuration	SWS_Spi_00003, SWS_Spi_00009, SWS_Spi_00064, SWS_Spi_00065
SRS_Spi_12180	The SPI Driver shall access the SPI	SWS_Spi_00003, SWS_Spi_00065

	bus only for the channel	
SRS_Spi_12181	If an SPI access request for a linked channel is performed, the SPI Handler/Driver shall use this SPI channel and all the linked channels	SWS_Spi_00055, SWS_Spi_00065
SRS_Spi_12197	The transmission data width of each SPI channel shall be configurable	SWS_Spi_00063
SRS_Spi_12198	The SPI Handler/Driver shall provide the functionality of transferring one short data sequence with variable data content	SWS_Spi_00053, SWS_Spi_00077
SRS_Spi_12199	The SPI Handler/Driver shall provide the functionality of transferring any data to any devices in one transfer sequence	SWS_Spi_00003, SWS_Spi_00064, SWS_Spi_00065
SRS_Spi_12200	Reading large data sequences from one slave device using dummy send data shall be possible	SWS_Spi_00003, SWS_Spi_00035, SWS_Spi_00053, SWS_Spi_00065, SWS_Spi_00077
SRS_Spi_12201	Reading large data sequences from multiple slave devices using dummy send data shall be possible	SWS_Spi_00003, SWS_Spi_00035, SWS_Spi_00065, SWS_Spi_00077
SRS_Spi_12202	The SPI Handler/Driver shall support data streams to a HW device with variable number of data	SWS_Spi_00053, SWS_Spi_00078
SRS_Spi_12253	The SPI Handler/Driver shall provide the functionality of transferring one short data sequence with constant data content	SWS_Spi_00052, SWS_Spi_00078
SRS_Spi_12256	The SPI Handler/Driver shall support all controller peripherals	SWS_Spi_00008, SWS_Spi_00009, SWS_Spi_00034
SRS_Spi_12257	The SPI Handler/Driver shall support the communication to daisy chained HW devices	SWS_Spi_00008, SWS_Spi_00009, SWS_Spi_00010, SWS_Spi_00034, SWS_Spi_00063, SWS_Spi_00065, SWS_Spi_00066
SRS_Spi_12258	Data shall be accessible from each device individually	SWS_Spi_00003, SWS_Spi_00009, SWS_Spi_00065
SRS_Spi_12259	Different timing and HW parameters shall be supported	SWS_Spi_00009
SRS_Spi_12260	Different priorities of sequences shall be supported	SWS_Spi_00002, SWS_Spi_00009, SWS_Spi_00014, SWS_Spi_00059, SWS_Spi_00064, SWS_Spi_00093
SRS_Spi_12261	Reading large data sequences from one slave device using variable send data shall be possible	SWS_Spi_00003, SWS_Spi_00053, SWS_Spi_00065
SRS_Spi_12262	Reading large data sequences from multiple slave devices using variable send data shall be possible	SWS_Spi_00003, SWS_Spi_00053, SWS_Spi_00065, SWS_Spi_00078
SRS_Spi_13400	The SPI Handler/Driver shall have a scalable functionality to fit the needs of the ECU	SWS_Spi_00110
SRS_Spi_13401	The SPI Handler/Driver functionalities	SWS_Spi_00109, SWS_Spi_00111,

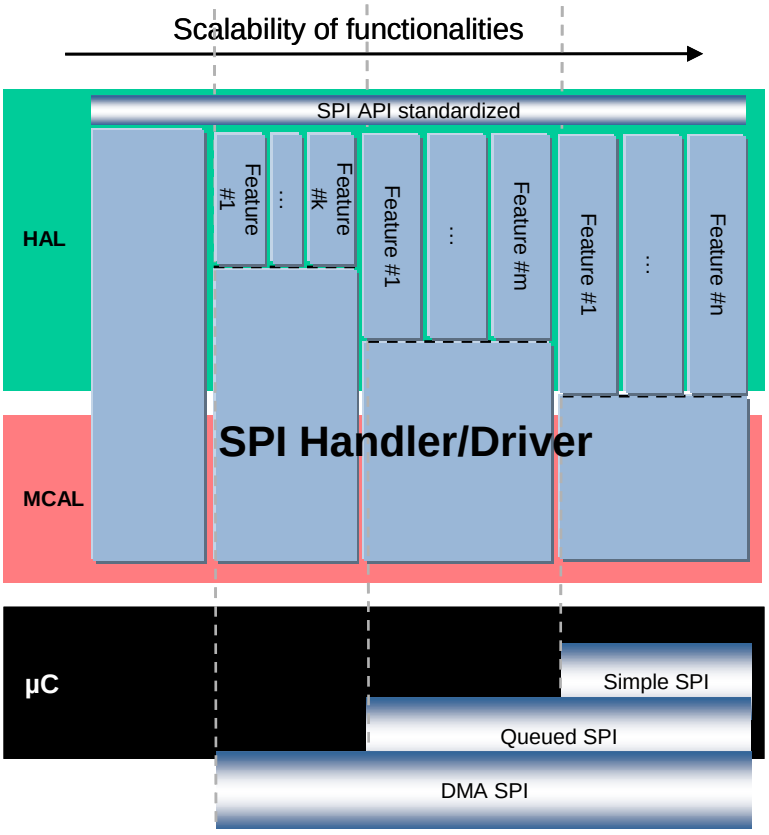
	shall be statically configurable	SWS_Spi_00121, SWS_Spi_00122, SWS_Spi_00125

7 Functional specification

The SPI (Serial Peripheral Interface) has a 4-wire synchronous serial interface. Data communication is enabled with a Chip select wire (CS). Data is transmitted with a 3-wire interface consisting of wires for serial data output (MOSI), serial data input (MISO) and serial clock (CLOCK).

7.1 Overall view of functionalities and features

This specification is based on previous specification experiences and also based on predominant identified use cases. The intention of this section is to summarize how the scalability of this monolithic SPI Handler/Driver allows getting a simple software module that fits simple needs up to a smart software module that fits enhanced needs.



This document specifies the following 3 Levels of Scalable Functionality for the SPI Handler/Driver:

- LEVEL 0, ***Simple Synchronous SPI Handler/Driver:*** the communication is based on synchronous handling with a FIFO policy to handle multiple accesses. Buffer usage is configurable to optimize and/or to take advantage of HW capabilities.

- LEVEL 1, ***Basic Asynchronous SPI Handler/Driver:*** the communication is based on asynchronous behavior and with a Priority policy to handle multiple accesses. Buffer usage is configurable as for “Simple Synchronous” level.
- LEVEL 2, ***Enhanced (Synchronous/Asynchronous) SPI Handler/Driver:*** the communication is based on asynchronous behavior or synchronous handling, using either interrupts or polling mechanism selectable during execution time and with a Priority policy to handle multiple accesses. Buffer usage is configurable as for other levels.

[SWS_Spi_00109] [The SPI Handler/Driver’s level of scalable functionality shall always be statically configurable, i.e. configured at pre-compile time to allow the best source code optimisation.] (SRS_Spi_13401)

[SWS_Spi_00110] [The `SpiLevelDelivered` parameter shall be configured with one of the 3 authorized values according to the described levels (0, 1 or 2) to allow the selection of the SPI Handler/Driver’s level of scalable functionality.] (SRS_Spi_13400)

To improve the scalability, each level has optional features which are configurable (`ON` / `OFF`) or selectable. These are described in detail in the dedicated chapters.

7.2 General behaviour

This chapter, on the one hand, introduces common behavior and configuration for all levels. On the other, it specifies the behavior of each level and also the allowed optional features.

[SWS_Spi_00041] [The SPI Handler/Driver interface configuration shall be based on Channels, Jobs and Sequences as defined in this document (see chapter 2).] (SRS_Spi_12093)

[SWS_Spi_00034] [The SPI Handler/Driver shall support one or more Channels, Jobs and Sequences to drive all kind of SPI compatible HW devices.] (SRS_Spi_12093, SRS_Spi_12256, SRS_Spi_12257)

[SWS_Spi_00255] [Data transmissions shall be done according to Channels, Jobs and Sequences configuration parameters.] ()

[SWS_Spi_00066] [The Chip Select (CS) is attached to the Job definition.] (SRS_Spi_12094, SRS_Spi_12257, SRS_Spi_12032, SRS_Spi_12033)

[SWS_Spi_00263] [Chip Select shall be handled during Job transmission and shall be released at the end of it. This Chip Select handling shall be done according to the Job configuration parameters.] ()

[SWS_Spi_00370] [It shall be possible to define if the Chip Select handling is managed autonomously by the HW peripheral, without explicit chip select control by

the driver, or the SPI driver shall drive the chip select lines explicitly as DIO (see ECUC_Spi_00212).⌋ ()

Example of CS handling: Set the CS active at the beginning of Job transmission; maintain it until the end of transmission of all Channels belonging to this Job afterwards set the CS inactive.

A Channel is defined one time but it could belong to several Jobs according to the user needs and this software specification.

[SWS_Spi_00065] ⌈ A Job shall contain at least one Channel.⌋ (SRS_Spi_12257, SRS_Spi_12179, SRS_Spi_12258, SRS_Spi_12180, SRS_Spi_12181, SRS_Spi_12199, SRS_Spi_12200, SRS_Spi_12261, SRS_Spi_12201, SRS_Spi_12262)

[SWS_Spi_00368] ⌈ Each Channel shall have an associated index which is used for specifying the order of the Channel within the Job.⌋ ()

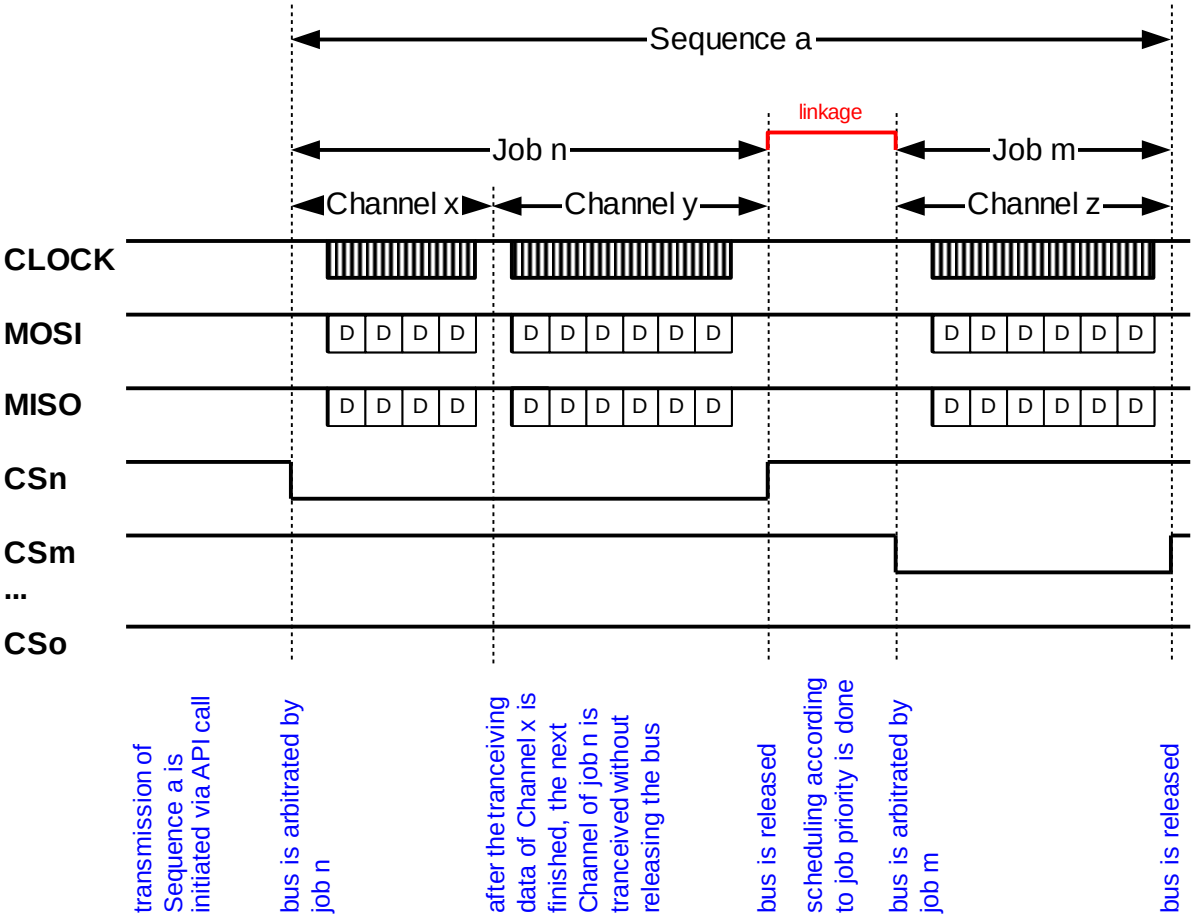
[SWS_Spi_00262] ⌈ If a Job contains more than one Channel, all Channels contained have the same Job properties during transmission and shall be linked together statically.⌋ ()

A Job is defined one time but it could belong to several Sequences according to the user needs and this software specification.

[SWS_Spi_00003] ⌈ A Sequence shall contain at least one Job.⌋ (SRS_Spi_12179, SRS_Spi_12258, SRS_Spi_12180, SRS_Spi_12199, SRS_Spi_12200, SRS_Spi_12261, SRS_Spi_12201, SRS_Spi_12262)

[SWS_Spi_00236] ⌈ If it contains more than one, all Jobs contained have the same Sequence properties during transmission and shall be linked together statically.⌋ ()

A Channel used for a transmission should have its parameters configured but it is allowed to pass Null pointers as source and destination pointers to generate a dummy transmission (See also [SWS_Spi_00028] & [SWS_Spi_00030]).



Channel data may differ from the hardware handled and user (client application) given. On the client side the data is handled in 8, 16 or 32bits mode base on SpiDataWidth (see chapter 8.2.5). On the microcontroller side, the hardware may handle between 1 and 32bits or may handle a fixed value (8 or 16bits) and this width is configurable for each Channel (see SpiDataWidth)..

[SWS_Spi_00149] ⌈ The SPI Handler/Driver shall take care of the differences between the frame width of channel (SpiDataWidth) and data access data type (given by SWS_Spi_00437).⌋ ()

[SWS_Spi_00289] ⌈ If data width (SpiDataWidth) are exactly same (8 or 16 or 32 bits), the SPI Handler/Driver can send and receive data without any bit changes straightforward.⌋ ()

[SWS_Spi_00290] ⌈ If data access casting type is superior to data width (for example SpiDataWidth = 12bits, data access is 16 bits), the data transmitted through the SPI Handler/Driver shall send the lower part, ignore the upper part. Receive the lower part, extend with zero.⌋ ()

This ensures that the user always gets the same interface.

[SWS_Spi_00437] ⌈ Data buffers are accessed as uint8, uint16 or uint32 according to SpiDataWidth

independently to Spi_DataBufferType.

The data access will use following casting:
uint8 for SpiDataWidth < 9
uint16 for 9 =< SpiDataWidth < 17
uint32 for 17 =< SpiDataWidth⌋ ()

7.2.1 Common configurable feature: Allowed Channel Buffers

In order to allow taking advantages of all microcontroller capabilities but also to allow sending/receiving of data to/from a dedicated memory location, all levels have an optional feature with respect to the location of Channel Buffers.

Hence, two main kinds of channel buffering can be used by configuration:

- Internally buffered Channels (IB): The buffer to transmit/receive data is provided by the Handler/Driver.
- Externally buffered Channels (EB): The buffer to transmit/receive is provided by the user (statically and/or dynamically).

Both channel buffering methods may be used depending on the 3 use cases described below:

- Usage 0: the SPI Handler/Driver manages only Internal Buffers.
- Usage 1: the SPI Handler/Driver manages only External Buffers.
- Usage 2: the SPI Handler/Driver manages both buffers types.

[SWS_Spi_00111] ⌈ The SpiChannelBuffersAllowed parameter shall be configured with one of the 3 authorized values (0, 1 or 2) according to the described usage.⌋ (SRS_Spi_13401)

[SWS_Spi_00279] ⌈ The SpiChannelBuffersAllowed parameter shall be configured to select which Channel Buffers the SPI Handler/Driver manages.⌋ ()

7.2.1.1 Behaviour of IB channels

The intention of Internal Buffer channels is to take advantage of microcontrollers including this feature by hardware. Otherwise, this feature should be simulated by software.

[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.⌋ (SRS_Spi_12025, SRS_Spi_12253)

[SWS_Spi_00049] ⌈ The channel data received shall be stored in 1 entry deep internal buffers by channel. The SPI Handler/Driver shall not take care of the overwriting of these “receive” buffers by another transmission on the same channel.⌋ ()

[SWS_Spi_00051] ⌈ The channel data to be transmitted shall be copied in 1 entry deep internal buffers by channel.⌋ ()

[SWS_Spi_00257] ⌈ The SPI Handler/Driver is not able to prevent the overwriting of these “transmit” buffers by users during transmissions.⌋ ()

[SWS_Spi_00438] ⌈ The Handler/Driver shall provide separate buffer for receive and transmit to ensure that transmitted data are not overwritten by the receive data.⌋ ()

7.2.1.2 Behaviour of EB channels

The intention of External Buffer channels is to reuse existing buffers that are located outside. That means the SPI Handler/Driver does not monitor them.

[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.⌋ (SRS_SPAL_12075, SRS_Spi_12025, SRS_Spi_12198, SRS_Spi_12200, SRS_Spi_12261, SRS_Spi_12262, SRS_Spi_12202, SRS_Spi_12103)

[SWS_Spi_00112] ⌈ The size of the Channel buffer is either fixed or variable. A maximum size for the Channel buffer shall be defined by the configuration.⌋ ()

[SWS_Spi_00280] ⌈ The buffer provided by the application for the SPI Handler Driver may have a different size.⌋ ()

7.2.1.3 Buffering channel usage

The following table provides information about the Channel characteristics:

IB Channels	
It provides…	• A more abstracted concept (buffering mechanisms are hidden) • Actual and future optimal implementation taken profit of HW buffer facilities (Given size of 256 bytes covers nowadays requirements).
Suggested use …	• Daisy-chain implementation. • Small data transfer devices (up to 10 Bytes).
EB Channels	
It provides…	• Efficient mechanism to support large stream communication. • Send constant data out of ROM tables and spare RAM size. • Send various data tables each for a different device (highly complex ASICS with several integrated peripheral devices, also mixed signal types, could exceed IB HW buffer size)
Suggested use …	• Large streams communication. • EEPROM communication. • Control of complex HW Chips .

7.2.2 LEVEL 0, Simple Synchronous behaviour

The intention of this functionality level is to provide a Handler/Driver with a reduced set of services to handle only simple synchronous transmissions. This is often the case for ECU including simple SPI networks but also for ECU using high speed external devices.

A simple synchronous transmission means that the function calling the transmission service is blocked during the ongoing transmission until the transmission is finished.

[SWS_Spi_00160] ⌈ The LEVEL 0 SPI Handler/Driver shall offer a synchronous transfer service for SPI busses.⌋ ()

[SWS_Spi_00161] ⌈ For an SPI Handler/Driver operating in LEVEL 0, when there is no on going Sequence transmission, the SPI Handler/Driver shall be in the idle state SPI_IDLE.⌋ ()

[SWS_Spi_00294] ⌈ This monolithic SPI Handler/Driver is able to handle one to n SPI buses according to the microcontroller used.⌋ ()

Then SPI buses are assigned to Jobs and not to Sequences. Consequently, Jobs, on different SPI buses, could belong to the same Sequence. Therefore:

[SWS_Spi_00114] ⌈ The LEVEL 0 SPI Handler/Driver shall accept concurrent Spi_SyncTransmit(), if the sequences to be transmitted use different bus and parameter `SPI_SUPPORT_CONCURRENT_SYNC_TRANSMIT` is enabled. This feature shall be disabled per default. That means during a Sequence on-going transmission, all requests to transmit another Sequence shall be rejected.⌋ ()

[SWS_Spi_00115] ⌈ The LEVEL 0 SPI Handler/Driver behaviour shall include the common feature: Allowed Channel Buffers, which is selected.⌋ ()

[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.⌋ (SRS_Spi_12170)

[SWS_Spi_00384]⌈ When a hardware error is detected, the SPI Handler/Driver shall stop the current sequence, report an error to the DEM as configured and set the state of the Job to SPI_JOB_FAILED and the state of the Sequence to SPI_SEQ_FAILED.⌋ ()

Read and write functions can not guarantee the data integrity while Channel data is being transmitted.

7.2.3 LEVEL 1, Basic Asynchronous behavior

The intention of this functionality level is to provide a Handler/Driver with a reduced set of services to handle asynchronous transmissions only. This is often the case for

ECU with functions related to SPI networks having different priorities but also for ECU using low speed external devices.

An asynchronous transmission means that the user calling the transmission service is not blocked when the transmission is on-going. Furthermore, the user can be notified at the end of transmission[1].

[SWS_Spi_00162] ⌈ The LEVEL 1 SPI Handler/Driver shall offer an asynchronous transfer service for SPI buses. An asynchronous transmission means that the user calling the transmission service is not blocked when the transmission is on going.⌋ (SRS_Spi_12099, SRS_Spi_12101, SRS_Spi_12103)

[SWS_Spi_00295] ⌈ The LEVEL 1 SPI Handler/Driver shall offer an asynchronous transfer service for SPI buses. Furthermore, the user can be notified at the end of transmission.⌋ ()

[SWS_Spi_00163] ⌈ For an SPI Handler/Driver operating in LEVEL 1, when there is no on-going Sequence transmission, the SPI Handler/Driver shall be in the idle state (`SPI_IDLE`).⌋ (SRS_Spi_12099, SRS_Spi_12101, SRS_Spi_12103)

This Handler/Driver will be used by several software modules which may be independent from each other and also may belong to different layers. Therefore, priorities will be assigned to Jobs in order to figure out specific cases of multiple accesses. These cases usually occur within real time systems based on asynchronous mechanisms.

[SWS_Spi_00002] ⌈ Jobs have priorities assigned. Jobs linked in a Sequence shall have same or de-creasing priorities. That means the first Job shall have the equal priority or the highest priority of all Jobs within the Sequence.⌋ (SRS_Spi_12260)

[SWS_Spi_00093] ⌈ Priority order of jobs shall be from the lower to the higher value defined, higher value higher priority (from 0, the lower to 3, the higher, limited to 4 priority levels see [SWS_Spi_00009]).⌋ (SRS_Spi_12260, SRS_Spi_12150)

With reference to Jobs priorities, this Handler/Driver needs rules to make a decision in these specific cases of multiple accesses.

[SWS_Spi_00059] ⌈ The SPI Handler/Driver scheduling method shall schedule Jobs in order to send the highest priority Job first.⌋ (SRS_Spi_12260, SRS_Spi_12037)

This monolithic SPI Handler/Driver is able to handle one to n SPI busses according to the microcontroller used. But SPI busses are assigned to Jobs and not to Sequences. Consequently, Jobs on different SPI buses could belong to the same Sequence. Therefore:

[1] This basic asynchronous behaviour might be implemented either by using interrupt or by polling mechanism. This software design choice is not in the scope of this document, but only solution is required for the LEVEL 1.

[SWS_Spi_00116] ⌈ The LEVEL 1 SPI Handler/Driver may allow transmitting more than one Sequence at the same time. That means during a Sequence transmission, all requests to transmit another Sequence shall be evaluated in order to accept to start a new sequence or to reject it accordingly to the lead Job.⌋ ()

[SWS_Spi_00117] ⌈ The LEVEL 1 SPI Handler/Driver behaviour shall include the common feature: Allowed Channel Buffers, which is selected, and the configured asynchronous feature: Interruptible Sequence (see next chapter).⌋ ()

[SWS_Spi_00267] ⌈ When a hardware error is detected, the SPI Handler/Driver shall stop the current Sequence, report an error to the DEM as configured and set the state of the Job to SPI_JOB_FAILED and the state of the Sequence to SPI_SEQ_FAILED.⌋ ()

[SWS_Spi_00118] ⌈ If Jobs are configured with a specific end notification function, the SPI Handler/Driver shall call this notification function at the end of the Job transmission.⌋ (SRS_Spi_12108)

[SWS_Spi_00281] ⌈ If Sequences are configured with a specific end notification function, the SPI Handler/Driver shall call this notification function at the end of the Sequence transmission.⌋ ()

[SWS_Spi_00119] ⌈ When a valid notification function pointer is configured (see [SWS_Spi_00071]), the SPI Handler/Driver shall call this notification function at the end of a Job transmission regardless of the result of the Job transmission being either `SPI_JOB_FAILED` or `SPI_JOB_OK` (rational: avoid deadlocks or endless loops).⌋ (SRS_Spi_12108)

[SWS_Spi_00120] ⌈ When a valid notification function pointer is configured (see [SWS_Spi_00073]), the SPI Handler/Driver shall call this notification function at the end of a Sequence transmission regardless of the result of the Sequence transmission being either `SPI_SEQ_FAILED`, `SPI_SEQ_OK` or `SPI_SEQ_CANCELLED` (rational: avoid deadlocks or endless loops).⌋ (SRS_Spi_12108)

7.2.4 Asynchronous configurable feature: Interruptible Sequences

In order to allow taking advantages of asynchronous transmission mechanism, level 1 and level 2 of this SPI Handler/Driver have an optional feature with respect to suspending the transmission of Sequences.

Hence two main kinds of sequences can be used by configuration:

- Non-Interruptible Sequences, every Sequence transmission started is not suspended by the Handler/Driver until the end of transmission.
- Mixed Sequences, according to its configuration, a Sequence transmission started may be suspended by the Handler/Driver between two of their consecutives Jobs.

[SWS_Spi_00121] ⌈ The SPI Handler/Driver's environment shall configure the `SpiInterruptibleSeqAllowed` parameter (`ON` / `OFF`) in order to select which kind of Sequences the SPI Handler/Driver manages.⌋ (SRS_Spi_13401)

7.2.4.1 Behavior of Non-Interruptible Sequences

The intention of the Non-Interruptible Sequences feature is to provide a simple software module based on a basic asynchronous mechanism, if only non blocking transmissions should be used.

[SWS_Spi_00122] ⌈ Interruptible Sequences are not allowed within levels 1 and 2 of the SPI/Handler/Driver when the `SpiInterruptibleSeqAllowed` parameter is switched off (i.e. configured with value "OFF"). ⌋ (SRS_Spi_13401)

[SWS_Spi_00123] ⌈ When the SPI Handler/Driver is configured not allowing interruptible Sequences, all Sequences declared are considered as Non-Interruptible Sequences[2].⌋ ()

[SWS_Spi_00282] ⌈ When the SPI Handler/Driver is configured not allowing interruptible Sequences their dedicated parameter SpiInterruptibleSequence can be omitted or the FALSE value should be used as default.⌋ ()

[SWS_Spi_00124] ⌈ According to [SWS_Spi_00116] and [SWS_Spi_00122] requirements, the SPI Handler/Driver is not allowed to suspend a Sequence transmission already started in favour of another Sequence.⌋ (SRS_Spi_12037)

7.2.4.2 Behavior of Mixed Sequences

The intention of the Mixed Sequences feature is to provide a software module with specific asynchronous mechanisms, if, for instance, very long Sequences that could or should be suspended by others with higher priority are used.

[SWS_Spi_00125] ⌈ Interruptible Sequences are allowed within levels 1 and 2 of SPI Handler/Driver when the `SpiInterruptibleSeqAllowed` parameter is switched on (i.e. configured with value "ON").⌋ (SRS_Spi_13401)

[SWS_Spi_00126] ⌈ When the SPI Handler/Driver is configured allowing interruptible Sequences, all Sequences declared shall have their dedicated parameter `SpiInterruptibleSequence` (see SWS_Spi_00064 & SPI106) to identify whether the Sequence can be suspended during transmission.⌋ ()

[2] The intention of this requirement is not to enforce any implementation solution in comparison with another one. But, it is only to ensure that anyhow, all Sequences will be considered as Non Interruptible Sequences.

[SWS_Spi_00014] ⌈ In case of a Sequence configured as Interruptible Sequence and according to [SWS_Spi_00125] requirement, the SPI Handler/Driver is allowed to suspend an already started Sequence transmission in favour of another Sequence with a higher priority Job (see SWS_Spi_00002 & SWS_Spi_00093). That means, at the end of a Job transmission (that belongs to the interruptible sequence) with another Sequence transmit request pending, the SPI Handler/Driver shall perform a re-scheduling in order to elect the next Job to transmit.⌋ (SRS_Spi_12260, SRS_Spi_12037)

[SWS_Spi_00127] ⌈ In case of a Sequence configured as Non-Interruptible Sequence and according to requirement [SWS_Spi_00125], the SPI Handler/Driver is not allowed to suspend this already started Sequence transmission in favour of another Sequence.⌋ (SRS_Spi_12037)

[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.⌋ ()

7.2.5 LEVEL 2, Enhanced behaviour

The intention of this functionality level is to provide a Handler/Driver with a complete set of services to handle synchronous and asynchronous transmissions. This could be the case for ECU with a lot of functions related to SPI networks having different priorities but also for ECU using external devices with different speeds.

Handling asynchronous and synchronous transmissions means that the microcontroller for which this software module is dedicated has to provide more than one SPI bus (see [SWS_Spi_00108]). In fact, the goal is to support SPI buses using a so-called synchronous driver and to support other SPI buses using a so-called asynchronous driver.

[SWS_Spi_00128] ⌈ The LEVEL 2 SPI Handler/Driver shall offer a synchronous transfer service for all SPI HW units configured as synchronous and it shall also offer an asynchronous transfer service for all other SPI buses.⌋ ()

[SWS_Spi_00283] ⌈ In LEVEL 2 if there is no on going Sequence transmission, the SPI Handler/Driver shall be in idle state (SPI_IDLE).⌋ ()

Usually, depending on software design, asynchronous end transmission may be detected by polling or interrupt mechanisms. This level of functionality proposes both mechanisms that are selectable during execution time.

[SWS_Spi_00155] ⌈ The SPI Handler/Driver LEVEL 2 shall implement one polling mechanism mode and one interrupt mechanism mode for SPI busses handled asynchronously.⌋ ()

[SWS_Spi_00156] ⌈ Both the polling mechanism and interrupt mechanism modes for SPI busses shall be selectable during execution time (see [SWS_Spi_00188]).⌋ ()

The requirements for LEVEL 0 apply to synchronous behaviour.
The requirements for LEVEL 1 apply to asynchronous behaviour.

7.3 Scheduling Advices

For asynchronous levels, LEVEL 1 and LEVEL 2, the SPI Handler/Driver can call end notification functions at the end of a Job and/or Sequence transmission (see [SWS_Spi_00118]). In a second time, in case of interruptible Sequences (that could be suspended), if another Sequence transmit request is pending, a rescheduling is also done by the SPI Handler/Driver in order to elect the next Job to transmit (see [SWS_Spi_00014]).

[SWS_Spi_00088] ⌈ For asynchronous levels, LEVEL 1 and LEVEL 2, the SPI Handler/Driver can call end notification functions at the end of a Job.⌋ ()

[SWS_Spi_00268] ⌈ For asynchronous levels, LEVEL 1 and LEVEL 2, the SPI Handler/Driver can call end notification functions at the end of a Sequence transmission.⌋ ()

[SWS_Spi_00269] ⌈ For asynchronous levels, LEVEL 1 and LEVEL 2 in case of interruptible Sequences, if another Sequence transmit request is pending, a rescheduling is also done by the SPI Handler/Driver in order to elect the next Job to transmit.⌋ ()

[SWS_Spi_00270] ⌈ In case call end notification function and rescheduling are fully done by software, the order between these shall be first scheduling and then the call of end notification function executed.⌋ ()

[SWS_Spi_00271] ⌈ In case call end notification function and rescheduling are fully done by hardware, the order could not be configured as required; the order shall be completely documented.⌋ ()

7.4 Error classification

[SWS_Spi_00004] ⌈ SPI Handler/driver shall be able to detect the error SPI_E_PARAM_CHANNEL when API service called with wrong parameter.⌋ (SRS_BSW_00327, SRS_BSW_00337, SRS_BSW_00385)

[SWS_Spi_00237] ⌈ SPI Handler/driver shall be able to detect the error SPI_E_PARAM_JOB when API service called with wrong parameter.⌋ ()

[SWS_Spi_00238] ⌈ SPI Handler/driver shall be able to detect the error SPI_E_PARAM_SEQ when API service called with wrong parameter.⌋ ()

[SWS_Spi_00240] ⌈ SPI Handler/driver shall be able to detect the error SPI_E_PARAM_LENGTH when API service called with wrong parameter.⌋ ()

[SWS_Spi_00241] ⌈ SPI Handler/driver shall be able to detect the error SPI_E_PARAM_UNIT when API service called with wrong parameter.⌋ ()

[SWS_Spi_00242] ⌈ SPI Handler/driver shall be able to detect the error SPI_E_UNINIT when API service used without module initialization.⌋ ()

[SWS_Spi_00243] ⌈ SPI Handler/driver shall be able to detect the error SPI_E_SEQ_PENDING when services called in a wrong sequence.⌋ ()

[SWS_Spi_00245] ⌈ SPI Handler/driver shall be able to detect the error SPI_E_SEQ_IN_PROCESS when synchronous transmission service called at wrong time.⌋ ()

[SWS_Spi_00246] ⌈ SPI Handler/driver shall be able to detect the error SPI_E_ALREADY_INITIALIZED when API SPI_Init service called while the SPI driver has already been initialized time.⌋ ()

[SWS_Spi_00195] ⌈ SPI Handler/driver shall be able to detect the error SPI_E_HARDWARE_ERROR when an hardware error occur during asynchronous or synchronous transmit. Please see also SWS_Spi_00267 and SWS_Spi_00384.⌋ ()

The Sections 7.4.1, 7.4.2 and 7.4.5 summarize the errors that the SPI Handler/Driver shall be able to detect.

7.4.1 Development Errors

Type or error	*Related error code*	*Value[hex]*
API service called with wrong parameter	SPI_E_PARAM_CHANNEL SPI_E_PARAM_JOB SPI_E_PARAM_SEQ SPI_E_PARAM_LENGTH SPI_E_PARAM_UNIT	0x0A 0x0B 0x0C 0x0D 0x0E
APIs called with a Null Pointer	SPI_E_PARAM_POINTER	0x10
API service used without module initialization	SPI_E_UNINIT	0x1A
API SPI_Init service called while the SPI driver has already been initialized	SPI_E_ALREADY_INITIALIZED	0x4A

7.4.2 Runtime Errors

Type or error	*Related error code*	*Value[hex]*
Services called in a wrong sequence	SPI_E_SEQ_PENDING	0x2A
Synchronous transmission service	SPI_E_SEQ_IN_PROCESS	0x3A

called at wrong time		

7.4.3 Transient faults

There are no transient faults.

7.4.4 Production Errors

There are no production errors.

7.4.5 Extended Production Errors

[SWS_Spi_00383]⌈

Error Name:	SPI_E_HARDWARE_ERROR	
Short Description:	An hardware error occurred during asynchronous or synchronous SPI transmit	
Long Description:	This Extended Production Error shall be issued when any error bit inside the SPI hardware transmit status register is raised	
Detection Criteria:	Fail	The SPI transmit status register information shall be reported to DEM as Dem_SetEventStatus (SPI_E_HARDWARE_ERROR, DEM_EVENT_STATUS_FAILED) when any error bit inside the SPI transmit status register is set. (SWS_Spi_00385)
	Pass	The SPI transmit status register information shall be reported to DEM as Dem_SetEventStatus (SPI_E_HARDWARE_ERROR, DEM_EVENT_STATUS_PASSED) when no error bit inside the SPI transmit status register is set. (SWS_Spi_00386)
Secondary Parameters:	N/A	
Time Required:	N/A	
Monitor Frequency	continuous	

⌋ ()

[SWS_Spi_00385]⌈ When any error bit inside the SPI transmit status register is set, the SPI transmit status register information shall be reported to DEM as Dem_SetEventStatus (SPI_E_HARDWARE_ERROR, DEM_EVENT_STATUS_FAILED)⌋ ()

[SWS_Spi_00386]⌈ When no error bit inside the SPI transmit status register is set, the SPI transmit status register information shall be reported to DEM as Dem_SetEventStatus (SPI_E_HARDWARE_ERROR, DEM_EVENT_STATUS_PASSED)⌋ ()

7.5 Error detection

7.5.1 API parameter checking

[SWS_Spi_00031] ⌈ The API parameter `Channel` shall have a value within the defined channels in the initialization data structure, and the correct type of channel (IB or EB) has to be used with services. Related error value: `SPI_E_PARAM_CHANNEL`. Otherwise, the service is not done and the return value shall be `E_NOT_OK`. ⌋ (SRS_BSW_00323)

[SWS_Spi_00032] ⌈ The API parameters Sequence and Job shall have values within the specified range of values. Related errors values: SPI_E_PARAM_SEQ or SPI_E_PARAM_JOB.⌋ (SRS_BSW_00323)

[SWS_Spi_00254] ⌈ If the Sequence and Job related service is not done and, depending on services, either the return value shall be E_NOT_OK or a failed result (SPI_JOB_FAILED or SPI_SEQ_FAILED).⌋ ()

[SWS_Spi_00060] ⌈ The API parameter Length of data shall have a value within the specified buffer maximum value. Related error value: SPI_E_PARAM_LENGTH.⌋ (SRS_BSW_00323)

[SWS_Spi_00258] ⌈ If the API parameter Length related service is not done and the return value shall be E_NOT_OK.⌋ ()

[SWS_Spi_00143] ⌈ The API parameter HWUnit shall have a value within the specified range of values. Related error value: SPI_E_PARAM_UNIT.⌋ ()

[SWS_Spi_00288] ⌈ If HWUnit related service is not done and the return value shall be SPI_UNINIT.⌋ ()

[SWS_Spi_00235] ⌈ If not applicable, the SPI Handler/Driver module's environment shall pass a NULL pointer to the function Spi_Init.⌋ ()

7.5.2 SPI state checking

[SWS_Spi_00046] ⌈ If development error detection for the SPI module is enabled and the SPI Handler/Driver's environment calls any API function before initialization, an error should be reported to the DET with the error value SPI_E_UNINIT according to the configuration.⌋ (SRS_BSW_00406)

[SWS_Spi_00256] ⌈ The SPI Handler/Driver shall not process the invoked function but, depending on the invoked function, shall either return the value E_NOT_OK or a failed result (SPI_JOB_FAILED or SPI_SEQ_FAILED).⌋ ()

[SWS_Spi_00233] ⌈
If development error detection for the SPI module is enabled, the calling of the routine SPI_Init() while the SPI driver is already initialized will cause a development error `SPI_E_ALREADY_INITIALIZED` and the desired functionality shall be left without any action.⌋ ()

8 API specification

8.1 Imported types

In this chapter all types included from the following modules are listed:

Module	*Header File*	*Imported Type*
Dem	Rte_Dem_Type.h	Dem_EventIdType
	Rte_Dem_Type.h	Dem_EventStatusType
Std_Types	StandardTypes.h	Std_ReturnType
	StandardTypes.h	Std_VersionInfoType

8.2 Type definitions

8.2.1 Spi_ConfigType

[SWS_Spi_00372] ⌈

Name:	`Spi_ConfigType`	
Type:	`Structure`	
Range:	`Implementation Specific`	The contents of the initialization data structure are SPI specific.
Description:	This type of the external data structure shall contain the initialization data for the SPI Handler/Driver.	
Available via:	`Spi.h`	

⌋ ()

[SWS_Spi_00344] ⌈ The description of the type Spi_ConfigType is implementation specific and it shall be provided for external use.⌋ ()

[SWS_Spi_00008] ⌈ The type `Spi_ConfigType` is an external data structure and shall contain the initialization data for the SPI Handler/Driver. It shall contain:

- MCU dependent properties for SPI HW units
- Definition of Channels
- Definition of Jobs
- Definition of Sequences⌋ (SRS_BSW_00405, SRS_SPAL_12125, SRS_Spi_12256, SRS_Spi_12257, SRS_Spi_12025, SRS_Spi_12024)

[SWS_Spi_00063] ⌈ For the type `Spi_ConfigType`, the definition for each Channel shall contain:

- Buffer usage with EB/IB Channel
- Transmit data width (1 up to 32 bits)
- Number of data buffers for IB Channels (at least 1) or it is the maximum of data for EB Channels (a value of 0 makes no sense)
- Transfer start LSB or MSB
- Default transmit value⌋ (SRS_Spi_12257, SRS_Spi_12025, SRS_Spi_12197, SRS_Spi_12024)

[SWS_Spi_00009] ⌈ For the type `Spi_ConfigType`, the definition for each Job shall contain:

- Assigned SPI HW Unit
- Assigned Chip Select pin (it is possible to assign no pin)
- Chip select functionality on/off
- Chip select pin polarity high or low
- Baud rate
- Timing between clock and chip select
- Shift clock idle low or idle high
- Data shift with leading or trailing edge
- Priority (4 levels are available from 0, the lower to 3, the higher)
- Job finish end notification function
- MCU dependent properties for the Job (only if needed)
- Fixed link of Channels (at least one)⌋ (SRS_BSW_00344, SRS_SPAL_12056, SRS_SPAL_12125, SRS_Spi_12093, SRS_Spi_12094, SRS_Spi_12256, SRS_Spi_12257, SRS_Spi_12025, SRS_Spi_12179, SRS_Spi_12026, SRS_Spi_12259, SRS_Spi_12258, SRS_Spi_12260, SRS_Spi_12032, SRS_Spi_12033, SRS_Spi_12150)

[SWS_Spi_00064] ⌈ For the type `Spi_ConfigType`, the definition for each Sequence shall contain:

- Collection of Jobs (at least one)
- Interruptible or not interruptible after each Job
- Sequence finish end notification function⌋ (SRS_SPAL_12056, SRS_Spi_12179, SRS_Spi_12260, SRS_Spi_12199, SRS_Spi_12150)

[SWS_Spi_00010] ⌈ For the type `Spi_ConfigType`, the configuration will map the Jobs to the different SPI hardware units and the devices.⌋ (SRS_Spi_12093, SRS_Spi_12257)

8.2.2 Spi_StatusType

[SWS_Spi_00373] ⌈

Name:	`Spi_StatusType`		
Type:	`Enumeration`		
Range:	`SPI_UNINIT`	0x00	The SPI Handler/Driver is not initialized or not usable.
	`SPI_IDLE`	0x01	The SPI Handler/Driver is not currently transmitting any Job.
	`SPI_BUSY`	0x02	The SPI Handler/Driver is performing a SPI Job (transmit).
Description:	This type defines a range of specific status for SPI Handler/Driver.		
Available via:	`Spi.h`		

⌋ (SRS_BSW_00406, SRS_BSW_00335)

[SWS_Spi_00061] ⌈ The type Spi_StatusType defines a range of specific status for SPI Handler/Driver. It informs about the SPI Handler/Driver status or specified SPI Hardware microcontroller peripheral.⌋ (SRS_BSW_00335)

[SWS_Spi_00259] ⌈ The type Spi_StatusType can be obtained calling the API service Spi_GetStatus.⌋ ()

[SWS_Spi_00260] ⌈ The type Spi_StatusType can be obtained calling the API service Spi_GetHWUnitStatus.⌋ ()

[SWS_Spi_00011] ⌈ After reset, the type `Spi_StatusType` shall have the default value `SPI_UNINIT`.⌋ ()

[SWS_Spi_00345] ⌈ API service Spi_GetStatus shall return SPI_UNINIT when the SPI Handler/Driver is not initialized or not usable.⌋ ()

[SWS_Spi_00346] ⌈ API service Spi_GetStatus shall return SPI_IDLE when The SPI Handler/Driver is not currently transmitting any Job.⌋ ()
[SWS_Spi_00347] ⌈ API service Spi_GetStatus shall return SPI_BUSY when The SPI Handler/Driver is performing a SPI Job transmit.⌋ ()

[SWS_Spi_00348] ⌈ Spi_GetHWUnitStatus function shall return SPI_IDLE when The SPI Hardware microcontroller peripheral is not currently transmitting any Job,⌋ ()

[SWS_Spi_00349] ⌈ Spi_GetHWUnitStatus function shall return SPI_BUSYwhen The SPI Hardware microcontroller peripheral is performing a SPI Job transmit.⌋ ()

8.2.3 Spi_JobResultType

[SWS_Spi_00374] ⌈

Name:	`Spi_JobResultType`		
Type:	`Enumeration`		
Range:	`SPI_JOB_OK`	--	The last transmission of the Job has been finished successfully.
	`SPI_JOB_PENDING`	--	The SPI Handler/Driver is performing a SPI Job. The meaning of this status is equal to SPI_BUSY.
	`SPI_JOB_FAILED`	--	The last transmission of the Job has failed.
	`SPI_JOB_QUEUED`	--	An asynchronous transmit Job has been accepted, while actual transmission for this Job has not started yet.
Description:	This type defines a range of specific Jobs status for SPI Handler/Driver.		
Available via:	`Spi.h`		

⌋ ()

[SWS_Spi_00062] ⌈ The type Spi_JobResultType defines a range of specific Jobs status for SPI Handler/Driver.⌋ (SRS_BSW_00335)

[SWS_Spi_00261] ⌈ The type Spi_JobResultType it informs about a SPI Handler/Driver Job status and can be obtained calling the API service Spi_GetJobResult with the Job ID.⌋ ()

[SWS_Spi_00012] ⌈ After reset, the type `Spi_JobResultType` shall have the default value `SPI_JOB_OK`.⌋ ()

[SWS_Spi_00350] ⌈ The function Spi_GetJobResult shall return SPI_JOB_OK when the last transmission of the Job has been finished successfully.⌋ ()

8.2.4 Spi_SeqResultType

[SWS_Spi_00375] ⌈

Name:	Spi_SeqResultType		
Type:	Enumeration		
Range:	SPI_SEQ_OK	--	The last transmission of the Sequence has been finished successfully.
	SPI_SEQ_PENDING	--	The SPI Handler/Driver is performing a SPI Sequence. The meaning of this status is equal to SPI_BUSY.
	SPI_SEQ_FAILED	--	The last transmission of the Sequence has failed.
	SPI_SEQ_CANCELED	--	The last transmission of the Sequence has been canceled by user.
Description:	This type defines a range of specific Sequences status for SPI Handler/Driver.		
Available via:	Spi.h		

⌋ ()

[SWS_Spi_00351] ⌈ The type Spi_SeqResultType defines a range of specific Sequences status for SPI Handler/Driver and can be obtained calling the API service Spi_GetSequenceResult, it shall be provided for external use.⌋ ()

[SWS_Spi_00019] ⌈ The type Spi_SeqResultType defines the range of specific Sequences status for SPI Handler/Driver.⌋ (SRS_BSW_00335)

[SWS_Spi_00251] ⌈ The type Spi_SeqResultType defines about SPI Handler/Driver Sequence status and can be obtained calling the API service Spi_GetSequenceResult with the Sequence ID.⌋ ()

[SWS_Spi_00017] ⌈ After reset, the type Spi_SeqResultType shall have the default value SPI_SEQ_OK.⌋ ()

[SWS_Spi_00352] ⌈ Spi_GetSequenceResult function shall return SPI_SEQ_OK when the last transmission of the Sequence has been finished successfully.⌋ ()

[SWS_Spi_00353] ⌈ Spi_GetSequenceResult function shall return SPI_SEQ_PENDING when the SPI Handler/Driver is performing a SPI Sequence. The meaning of this status is equal to SPI_BUSY.⌋ ()

[SWS_Spi_00354] ⌈ Spi_GetSequenceResult function shall return SPI_SEQ_FAILED when the last transmission of the Sequence has failed.⌋ ()

8.2.5 Spi_DataBufferType

[SWS_Spi_00376] ⌈

Name:	Spi_DataBufferType
Type:	uint8
Description:	Type of application data buffer elements.
Available via:	Spi.h

⌋ ()

[SWS_Spi_00355] ⌈ Spi_DataBufferType defines the type of application data buffer elements. Type is uint8. Access to the data is selected dynamically as is described in SWS_SPI_00437. The data buffer has to be aligned to 32 bits. It shall be provided for external use.⌋ ()

[SWS_Spi_00164] ⌈ The type `Spi_DataBufferType` refers to application data buffer elements.⌋ ()

8.2.6 Spi_NumberOfDataType

[SWS_Spi_00377] ⌈

Name:	`Spi_NumberOfDataType`
Type:	`uint16`
Description:	Type for defining the number of data elements of the type Spi_DataBufferType to send and / or receive by Channel
Available via:	`Spi.h`

⌋ ()

[SWS_Spi_00165] ⌈ The type `Spi_NumberOfDataType` is used for defining the number of data elements of the type specified in [SWS_SPI_00437] to send and / or receive by Channel.⌋ ()

8.2.7 Spi_ChannelType

[SWS_Spi_00378] ⌈

Name:	`Spi_ChannelType`
Type:	`uint8`
Description:	Specifies the identification (ID) for a Channel.
Available via:	`Spi.h`

⌋ ()

[SWS_Spi_00356] ⌈ The type Spi_ChannelType specifies the identification (ID) for a Channel.⌋ ()

[SWS_Spi_00166] ⌈ The type `Spi_ChannelType` is used for specifying the identification (ID) for a Channel.⌋ ()

8.2.8 Spi_JobType

[SWS_Spi_00379] ⌈

Name:	`Spi_JobType`
Type:	`uint16`
Description:	Specifies the identification (ID) for a Job.
Available via:	`Spi.h`

⌋ ()

[SWS_Spi_00357] ⌈ The type Spi_JobType specifies the identification (ID) for a Job.⌋ ()

[SWS_Spi_00167] ⌈ The type `Spi_JobType` is used for specifying the identification (ID) for a Job.⌋ ()

8.2.9 Spi_SequenceType

[SWS_Spi_00380] ⌈

Name:	`Spi_SequenceType`
Type:	`uint8`
Description:	Specifies the identification (ID) for a sequence of jobs.
Available via:	`Spi.h`

⌋ ()

[SWS_Spi_00358] ⌈ The type Spi_SequenceType specifies the identification (ID) for a sequence of jobs.⌋ ()

[SWS_Spi_00168] ⌈ The type `Spi_SequenceType` is used for specifying the identification (ID) for a sequence of jobs.⌋ ()

8.2.10 Spi_HWUnitType

[SWS_Spi_00381] ⌈

Name:	`Spi_HWUnitType`
Type:	`uint8`
Description:	Specifies the identification (ID) for a SPI Hardware microcontroller peripheral (unit).
Available via:	`Spi.h`

⌋ ()

[SWS_Spi_00359] ⌈ The type Spi_HWUnitType specifies the identification (ID) for a SPI Hardware microcontroller peripheral (unit).⌋ ()

[SWS_Spi_00169] ⌈ The type `Spi_HWUnitType` is used for specifying the identification (ID) for a SPI Hardware microcontroller peripheral (unit).⌋ ()

8.2.11 Spi_AsyncModeType

[SWS_Spi_00382] ⌈

Name:	`Spi_AsyncModeType`		
Type:	`Enumeration`		
Range:	`SPI_POLLING_MODE`	--	The asynchronous mechanism is ensured by polling, so interrupts related to SPI busses handled asynchronously are disabled.
	`SPI_INTERRUPT_MODE`	--	The asynchronous mechanism is ensured by interrupt, so interrupts related to SPI busses handled asynchronously are enabled.
Description:	Specifies the asynchronous mechanism mode for SPI busses handled asynchronously in LEVEL 2.		
Available via:	`Spi.h`		

⌋ ()

[SWS_Spi_00360] ⌈ The type Spi_AsyncModeType specifies the asynchronous mechanism mode for SPI buses handled asynchronously in LEVEL 2 and obtained by the API Spi_SetAsyncMode.⌋ ()

[SWS_Spi_00170] ⌈ The type `Spi_AsyncModeType` is used for specifying the asynchronous mechanism mode for SPI busses handled asynchronously in LEVEL 2.⌋ ()

[SWS_Spi_00150] ⌈ The type `Spi_AsyncModeType` is made available or not depending on the pre-compile time parameter: `SpiLevelDelivered`. This is only relevant for LEVEL 2.⌋ ()

[SWS_Spi_00361] ⌈ If API Spi_SetAsyncMode function is called by the parameter value SPI_POLLING_MODE then asynchronous mechanism is ensured by polling. So interrupts related to SPI buses handled asynchronously are disabled.⌋ ()

[SWS_Spi_00362] ⌈ If API Spi_SetAsyncMode function is called by the parameter value SPI_INTERRUPT_MODE asynchronous mechanism is ensured by interrupt, so interrupts related to SPI buses handled asynchronously are enabled.⌋ ()

8.3 Function definitions

8.3.1 Spi_Init

[SWS_Spi_00175] ⌈

Service name:	Spi_Init	

Syntax:	`void Spi_Init(`
	`  const Spi_ConfigType* ConfigPtr`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to configuration set
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	None	
Description:	Service for SPI initialization.	
Available via:	`Spi.h`	

⌋ ()

[SWS_Spi_00298] ⌈ The operation Spi_Init is Non Re-entrant.⌋ ()

[SWS_Spi_00299] ⌈ The function Spi_Init provides the service for SPI initialization.⌋ ()

[SWS_Spi_00013] ⌈ The function `Spi_Init` shall initialize all SPI relevant registers with the values of the structure referenced by the parameter `ConfigPtr`.⌋ (SRS_BSW_00405, SRS_BSW_00101, SRS_SPAL_12057, SRS_SPAL_12125)

[SWS_Spi_00082] ⌈ The function `Spi_Init` shall define default values for required parameters of the structure referenced by the `ConfigPtr`. For example: all buffer pointers shall be initialized as a null value pointer.⌋ ()

[SWS_Spi_00015] ⌈ After the module initialization using the function `Spi_Init`, the SPI Handler/Driver shall set its state to `SPI_IDLE`, the Sequences result to `SPI_SEQ_OK` and the jobs result to `SPI_JOB_OK`.⌋ (SRS_BSW_00406, SRS_BSW_00101, SRS_SPAL_12057)

[SWS_Spi_00151] ⌈ For LEVEL 2 (see chapter 7.2.5 and SPI103), the function `Spi_Init` shall set the SPI Handler/Driver asynchronous mechanism mode to `SPI_POLLING_MODE` by default. Interrupts related to SPI busses shall be disabled.⌋ ()

A re-initialization of a SPI Handler/Driver by executing the Spi_Init() function requires a de-initialization before by executing a Spi_DeInit().

Parameters of the function `Spi_Init` shall be checked as it is explained in section API parameter checking

8.3.2 Spi_DeInit

[SWS_Spi_00176] ⌈

Service name:	Spi_DeInit	
Syntax:	`Std_ReturnType Spi_DeInit( void )`	
Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: de-initialisation command has been accepted E_NOT_OK: de-initialisation command has not been accepted
Description:	Service for SPI de-initialization.	
Available via:	`Spi.h`	

⌋ ()

[SWS_Spi_00300] ⌈ The operation Std_ReturnType Spi_DeInit( ) is Non Reentrant.⌋ ()

[SWS_Spi_00301] ⌈ When the API Spi_DeInit has been accepted the return value of this function shall be E_OK.⌋ ()

[SWS_Spi_00302] ⌈ When the API Spi_DeInit has not been accepted the return value of this function shall be E_NOT_OK.⌋ ()

[SWS_Spi_00303] ⌈ The function Spi_DeInit provides the service for SPI de-initialization.⌋ ()

[SWS_Spi_00021] ⌈ The function Spi_DeInit shall de-initialize SPI Handler/Driver.⌋ (SRS_BSW_00336, SRS_SPAL_12163, SRS_SPAL_12064)

[SWS_Spi_00252] ⌈ In case of the SPI Handler/Driver state is not SPI_BUSY, the deInitialization function shall put all already initialized microcontroller SPI peripherals into the same state such as Power On Reset.⌋ ()

[SWS_Spi_00253] ⌈ The function call Spi_DeInit shall be rejected if the status of SPI Handler/Driver is SPI_BUSY.⌋ ()

[SWS_Spi_00022] ⌈ After the module de-initialization using the function `Spi_DeInit`, the SPI Handler/Driver shall set its state to `SPI_UNINIT`.⌋ (SRS_BSW_00336, SRS_SPAL_12163)

The SPI Handler/Driver shall have been initialized before the function `Spi_DeInit` is called, otherwise see [SWS_Spi_00046].

8.3.3 Spi_WriteIB

[SWS_Spi_00177] ⌈

Service name:	Spi_WriteIB	

Syntax:	`Std_ReturnType Spi_WriteIB(`
	`  Spi_ChannelType Channel,`
	`  const Spi_DataBufferType* DataBufferPtr`
	`)`	

Service ID[hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Channel ID.
	DataBufferPtr	Pointer to source data buffer. If this pointer is null, it is assumed that the data to be transmitted is not relevant and the default transmit value of this channel will be used instead.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: write command has been accepted E_NOT_OK: write command has not been accepted
Description:	Service for writing one or more data to an IB SPI Handler/Driver Channel specified by parameter.	
Available via:	`Spi.h`	

⌋ ()

[SWS_Spi_00304] ⌈ The operation Spi_WriteIB is Re-entrant.⌋ ()

[SWS_Spi_00305] ⌈ When the API Spi_WriteIB command has been accepted the function returns the value E_OK.⌋ ()

[SWS_Spi_00306] ⌈ When the API Spi_WriteIB command has not been accepted the function returns the value E_NOT_OK.⌋ ()

[SWS_Spi_00307] ⌈ The function Spi_WriteIB provides the service for writing one or more data to an IB SPI Handler/Driver Channel by the respective parameter.⌋ ()

[SWS_Spi_00018] ⌈ The function `Spi_WriteIB` shall write one or more data to an IB SPI Handler/Driver Channel specified by the respective parameter.⌋ (SRS_Spi_12101, SRS_Spi_12153)

[SWS_Spi_00024] ⌈ The function `Spi_WriteIB` shall take over the given parameters, and save the pointed data to the internal buffer defined with the function `Spi_Init`.⌋ ()

[SWS_Spi_00023] ⌈ If the given parameter “DataBufferPtr” is null, the function `Spi_WriteIB` shall assume that the data to be transmitted is not relevant and the default transmit value of the given channel shall be used instead.⌋ ()

[SWS_Spi_00137] ⌈ The function `Spi_WriteIB` shall be pre-compile time configurable by the parameter `SpiChannelBuffersAllowed`. This function is only relevant for Channels with IB.⌋ ()

Parameters of the function `Spi_WriteIB` shall be checked as it is explained in section API parameter checking.

The SPI Handler/Driver shall have been initialized before the function `Spi_WriteIB` is called, otherwise see [SWS_Spi_00046].

8.3.4 Spi_AsyncTransmit

[SWS_Spi_00178] ⌈

Service name:	Spi_AsyncTransmit	

Syntax:	`Std_ReturnType Spi_AsyncTransmit(`
	`  Spi_SequenceType Sequence`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	Sequence	Sequence ID.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Transmission command has been accepted E_NOT_OK: Transmission command has not been accepted
Description:	Service to transmit data on the SPI bus.	
Available via:	`Spi.h`	

⌋ ()

[SWS_Spi_00308] ⌈ The operation Std_ReturnType Spi_AsyncTransmit(Spi_SequenceType Sequence) is Re-entrant.⌋ ()

[SWS_Spi_00309] ⌈ When the API Spi_AsyncTransmit command has been accepted the function shall return the value E_OK.⌋ ()

[SWS_Spi_00310] ⌈ When the API Spi_AsyncTransmit command has not been accepted the function shall return the value E_NOT_OK.⌋ ()

[SWS_Spi_00311] ⌈ The function Spi_AsyncTransmit provides service to transmit data on the SPI bus.⌋ ()

[SWS_Spi_00020] ⌈ The function `Spi_AsyncTransmit` shall take over the given parameter, initiate a transmission, set the SPI Handler/Driver status to `SPI_BUSY`, set the sequence result to `SPI_SEQ_PENDING` and return. ⌋ (SRS_Spi_12099, SRS_Spi_12101, SRS_Spi_12103)

[SWS_Spi_00194] ⌈ When the function Spi_AsyncTransmit is called, shall take over the given parameter and set the Job status to SPI_JOB_QUEUED, which can be obtained by calling the API service Spi_GetJobResult.⌋ ()

[SWS_Spi_00157] ⌈ When the function Spi_AsyncTransmit is called, the SPI Handler/Driver shall handle the Job results. Result shall be SPI_JOB_PENDING when the transmission of Jobs is started.⌋ ()

[SWS_Spi_00292] ⌈ When the function Spi_AsyncTransmit is called, the SPI Handler/Driver shall handle the Job results. Result shall be SPI_JOB_OK when the transmission of Jobs is success.⌋ ()

[SWS_Spi_00293] ⌈ When the function Spi_AsyncTransmit is called, the SPI Handler/Driver shall handle the Job results. Result shall be SPI_JOB_FAILED when the transmission of Jobs is failed.⌋ ()

[SWS_Spi_00081] ⌈ When the function Spi_AsyncTransmit is called and the requested Sequence is already in state SPI_SEQ_PENDING, the SPI Handler/Driver shall not take in account this new request and this function shall return with value E_NOT_OK, in this case.⌋ ()

[SWS_Spi_00266] ⌈ When the function Spi_AsyncTransmit is called and the requested Sequence is already in state SPI_SEQ_PENDING the SPI Handler/Driver shall report the SPI_E_SEQ_PENDING error according to [SWS_BSW_00042] and [SWS_BSW_00045].⌋ ()

[SWS_Spi_00086] ⌈ When the function Spi_AsyncTransmit is called and the requested Sequence shares Jobs with another sequence that is in the state SPI_SEQ_PENDING, the SPI Handler/Driver shall not take into account this new request and this function shall return the value E_NOT_OK. In this case and according

to [SWS_BSW_00042] and [SWS_BSW_00045], the SPI Handler/Driver shall report the SPI_E_SEQ_PENDING error.] ()

[SWS_Spi_00035] [When the function Spi_AsyncTransmit is used with EB and the source data pointer has been provided as NULL using the Spi_SetupEB method, the default transmit data configured for each channel shall be transmitted. (See also [SWS_SPI_00028])] (SRS_Spi_12200, SRS_Spi_12201)

[SWS_Spi_00036] [When the function `Spi_AsyncTransmit` is used with EB and the destination data pointer has been provided as NULL using the `Spi_SetupEB` method, the SPI Handler/Driver shall ignore receiving data (See also [SWS_Spi_00030])] ()

[SWS_Spi_00055] [When the function `Spi_AsyncTransmit` is used for a Sequence with linked Jobs, the function shall transmit from the first Job up to the last Job in the sequence.] (SRS_Spi_12181)

[SWS_Spi_00057] [At the end of a sequence transmission initiated by the function `Spi_AsyncTransmit` and if configured, the SPI Handler/Driver shall invoke the sequence notification call-back function after the last Job end notification if this one is also configured.] (SRS_SPAL_00157, SRS_Spi_12108)

[SWS_Spi_00133] [The function `Spi_AsyncTransmit` is pre-compile time selectable by the configuration parameter `SpiLevelDelivered`. This function is only relevant for LEVEL 1 and LEVEL 2.] ()

[SWS_Spi_00173] [The SPI Handler/Driver's environment shall call the function `Spi_AsyncTransmit` 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_ReadIB`.] ()

Parameters of the function `Spi_AsyncTransmit` shall be checked as explained in section API parameter checking

The SPI Handler/Driver shall have been initialized before the function `Spi_AsyncTransmit` is called otherwise see [SWS_Spi_00046].

8.3.5 Spi_ReadIB

[SWS_Spi_00179] [

Service name:	Spi_ReadIB

Syntax:	`Std_ReturnType Spi_ReadIB(`
	`  Spi_ChannelType Channel,`
	`  Spi_DataBufferType* DataBufferPointer`
	`)`

Service ID[hex]:	0x04
Sync/Async:	Synchronous
Reentrancy:	Reentrant

Parameters (in):	Channel	Channel ID.
Parameters (in-out):	None	
Parameters (out):	DataBufferPointer	Pointer to destination data buffer in RAM
Return value:	Std_ReturnType	E_OK: read command has been accepted E_NOT_OK: read command has not been accepted
Description:	Service for reading synchronously one or more data from an IB SPI Handler/Driver Channel specified by parameter.	
Available via:	`Spi.h`	

⌋ ()

[SWS_Spi_00312] ⌈ The operation Spi_ReadIB is Re-entrant.⌋ ()

[SWS_Spi_00313] ⌈ The function Spi_ReadIB return values E_OK: read command has been accepted.⌋ ()

[SWS_Spi_00314] ⌈ The function Spi_ReadIB return values E_NOT_OK: read command has not been accepted.⌋ ()

[SWS_Spi_00315] ⌈ The function Spi_ReadIB provides the service for reading synchronously one or more data from an IB SPI Handler/Driver Channel specified by parameter.⌋ ()

[SWS_Spi_00016] ⌈ The function `Spi_ReadIB` shall read synchronously one or more data from an IB SPI Handler/Driver Channel specified by the respective parameter.⌋ (SRS_Spi_12099, SRS_Spi_12152)

[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_00138] ⌈ The function `Spi_ReadIB` is pre-compile time configurable by the parameter `SpiChannelBuffersAllowed`. This function is only relevant for Channels with IB.⌋ ()

Parameters of the function `Spi_ReadIB` shall be checked as it is explained in section API parameter checking.

The SPI Handler/Driver shall have been initialized before the function `Spi_ReadIB` is called otherwise see [SWS_Spi_00046].

8.3.6 Spi_SetupEB

[SWS_Spi_00180] ⌈

Service name:	Spi_SetupEB

Syntax:	`Std_ReturnType Spi_SetupEB(`
	`  Spi_ChannelType Channel,`
	`  const Spi_DataBufferType* SrcDataBufferPtr,`
	`  Spi_DataBufferType* DesDataBufferPtr,`

	`Spi_NumberOfDataType Length`)	
Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Channel ID.
	SrcDataBufferPtr	Pointer to source data buffer.
	Length	Length (number of data elements) of the data to be transmitted from SrcDataBufferPtr and/or received from DesDataBufferPtr Min.: 1 Max.: Max of data specified at configuration for this channel
Parameters (in-out):	DesDataBufferPtr	Pointer to destination data buffer in RAM.
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Setup command has been accepted E_NOT_OK: Setup command has not been accepted
Description:	Service to setup the buffers and the length of data for the EB SPI Handler/Driver Channel specified.	
Available via:	`Spi.h`	

] ()

[SWS_Spi_00316] ⌈ The operation Spi_SetupEB is Re-entrant.⌋ ()

[SWS_Spi_00317] ⌈ Return values of the function Spi_SetupEB are E_OK: Setup command has been accepted and E_NOT_OK: Setup command has not been accepted.⌋ ()

[SWS_Spi_00318] ⌈ The function Spi_SetupEB provides the service to setup the buffers and the length of data for the EB SPI Handler/Driver Channel specified.⌋ ()

[SWS_Spi_00058] ⌈ The function `Spi_SetupEB` shall set up the buffers and the length of data for the specific EB SPI Handler/Driver Channel.⌋ (SRS_Spi_12103)

[SWS_Spi_00067] ⌈ The function `Spi_SetupEB` shall update the buffer pointers and length attributes of the specified Channel with the provided values.⌋ (SRS_Spi_12103)

As these attributes are persistent, they will be used for all succeeding calls to a Transmit method (for the specified Channel).

[SWS_Spi_00028] ⌈ When the SPI Handler/Driver's environment is calling the function `Spi_SetupEB` with the parameter `SrcDataBufferPtr` being a Null pointer, the function shall transmit the default transmit value configured for the channel after a Transmit method is requested. (See also [SWS_Spi_00035])⌋ ()

[SWS_Spi_00030] ⌈ When the function `Spi_SetupEB` is called with the parameter `DesDataBufferPtr` being a Null pointer, the SPI Handler/Driver shall ignore the received data after a Transmit method is requested.(See also [SWS_Spi_00036])⌋ ()

[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.⌋ ()

[SWS_Spi_00139] ⌈ The function `Spi_SetupEB` is pre-compile time configurable by the parameter `SpiChannelBuffersAllowed`. This function is only relevant for Channels with EB.⌋ ()

Parameters of the function `Spi_SetupEB` shall be checked as it is explained in section API parameter checking.

The SPI Handler/Driver shall have been initialized before the function `Spi_SetupEB` is called otherwise see [SWS_Spi_00046].

8.3.7 Spi_GetStatus

[SWS_Spi_00181] ⌈

Service name:	Spi_GetStatus	

Syntax:	`Spi_StatusType Spi_GetStatus(`
	`  void`
	`)`	

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Spi_StatusType	Spi_StatusType
Description:	Service returns the SPI Handler/Driver software module status.	
Available via:	`Spi.h`	

⌋ ()

[SWS_Spi_00319] ⌈ The operation Spi_GetStatus is Re-entrant.⌋ ()

[SWS_Spi_00320] ⌈ The function Spi_GetStatus returns the SPI Handler/Driver software module status.⌋ ()

[SWS_Spi_00025] ⌈ The function `Spi_GetStatus` shall return the SPI Handler/Driver software module status.⌋ (SRS_SPAL_12064, SRS_Spi_12104)

8.3.8 Spi_GetJobResult

[SWS_Spi_00182] ⌈

Service name:	Spi_GetJobResult	

Syntax:	`Spi_JobResultType Spi_GetJobResult(`
	`  Spi_JobType Job`
	`)`	

Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Job	Job ID. An invalid job ID will return an undefined result.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Spi_JobResultType	Spi_JobResultType
Description:	This service returns the last transmission result of the specified Job.	
Available via:	`Spi.h`	

⌋ ()

[SWS_Spi_00321] ⌈ The operation Spi_GetJobResult is Re-entrant.⌋ ()

[SWS_Spi_00322] ⌈ The function Spi_GetJobResult service returns the last transmission result of the specified Job.⌋ ()

[SWS_Spi_00026] ⌈ The function `Spi_GetJobResult` shall return the last transmission result of the specified Job. ⌋ (SRS_SPAL_00157, SRS_Spi_12104)

[SWS_Spi_00038] ⌈ The SPI Handler/Driver's environment shall call the function `Spi_GetJobResult` to inquire whether the Job transmission has succeeded (`SPI_JOB_OK`) or failed (`SPI_JOB_FAILED`).⌋ (SRS_SPAL_00157)

NOTE: Every new transmit job that has been accepted by the SPI Handler/Driver overwrites the previous job result with `SPI_JOB_QUEUED` or `SPI_JOB_PENDING`.

Parameters of the function `Spi_GetJobResult` shall be checked as it is explained in section API parameter checking.

If SPI Handler/Driver has not been initialized before the function `Spi_GetJobResult` is called, the return value is undefined.

8.3.9 Spi_GetSequenceResult

[SWS_Spi_00183] ⌈

Service name:	Spi_GetSequenceResult	
Syntax:	`Spi_SeqResultType Spi_GetSequenceResult( Spi_SequenceType Sequence )`	
Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Sequence	Sequence ID. An invalid sequence ID will return an undefined result.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Spi_SeqResultType	Spi_SeqResultType
Description:	This service returns the last transmission result of the specified Sequence.	
Available via:	`Spi.h`	

⌋ ()

[SWS_Spi_00323] ⌈ The operation Spi_GetSequenceResult is Re-entrant.⌋ ()

[SWS_Spi_00324] ⌈ The function Spi_GetSequenceResult shall return the last transmission result of the specified Sequence.⌋ (SRS_SPAL_00157, SRS_Spi_12104)

[SWS_Spi_00042] ⌈ The SPI Handler/Driver's environment shall call the function `Spi_GetSequenceResult` to inquire whether the full Sequence transmission has

succeeded (SPI_SEQ_OK) or failed (SPI_SEQ_FAILED).⌋ (SRS_SPAL_00157, SRS_Spi_12170)

Note:

- Every new transmit sequence that has been accepted by the SPI Handler/Driver overwrites the previous sequence result with SPI_SEQ_PENDING.
- If the SPI Handler/Driver has not been initialized before the function Spi_GetSequenceResult is called, the return value is undefined.

Parameters of the function Spi_GetSequenceResult shall be checked as it is explained in section API parameter checking.

8.3.10 Spi_GetVersionInfo

[SWS_Spi_00184] ⌈

Service name:	Spi_GetVersionInfo	

Syntax:	`void Spi_GetVersionInfo(`
	`  Std_VersionInfoType* versioninfo`
	`)`	

Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (in-out):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	This service returns the version information of this module.	
Available via:	Spi.h	

⌋ ()

[SWS_Spi_00371] ⌈ If Det is enabled, the parameter versioninfo shall be checked for being NULL. The error SPI_E_PARAM_POINTER shall be reported in case the value is a NULL pointer.⌋ ()

8.3.11 Spi_SyncTransmit

[SWS_Spi_00185] ⌈

Service name:	Spi_SyncTransmit	

Syntax:	`Std_ReturnType Spi_SyncTransmit(`
	`  Spi_SequenceType Sequence`
	`)`	

Service ID[hex]:	0x0a	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	Sequence	Sequence ID.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Transmission command has been accepted E_NOT_OK: Transmission command has not been accepted

Description:	Service to transmit data on the SPI bus
Available via:	`Spi.h`

] ()

[SWS_Spi_00327] ⌈ The operation Spi_SyncTransmit is Re-entrant.⌋ ()

[SWS_Spi_00328] ⌈ Return value of the function Spi_SyncTransmit is E_OK: when Transmission command has been accepted.⌋ ()

[SWS_Spi_00329] ⌈ Return value of the function Spi_SyncTransmit is E_NOT_OK: When Transmission command has not been accepted.⌋ ()

[SWS_Spi_00330] ⌈ The function Spi_SyncTransmit provides the service to transmit data on the SPI bus.⌋ ()

[SWS_Spi_00134] ⌈ When the function Spi_SyncTransmit is called, shall take over the given parameter and set the SPI Handler/Driver status to SPI_BUSY can be obtained calling the API service SPI_GetStatus.⌋ (SRS_Spi_12152, SRS_Spi_12153, SRS_Spi_12154)

[SWS_Spi_00285] ⌈ When the function Spi_SyncTransmit is called, shall take over the given parameter and set the Sequence status to SPI_SEQ_PENDING can be obtained calling the API service Spi_GetSequenceResult.⌋ ()

[SWS_Spi_00286] ⌈ When the function Spi_SyncTransmit is called, shall take over the given parameter and set the Job status to SPI_JOB_PENDING can be obtained calling the API service Spi_GetJobResult.⌋ ()

[SWS_Spi_00135] ⌈ When the function `Spi_SyncTransmit` is called while a sequence is on transmission and `SPI_SUPPORT_CONCURRENT_SYNC_TRANSMIT` is disabled or another sequence is on transmission on same bus, the SPI Handler/Driver shall not take into account this new transmission request and the function shall return the value `E_NOT_OK` (see [SWS_Spi_00114]). In this case, the SPI Handler/Driver shall report the `SPI_E_SEQ_IN_PROCESS` error according to [SWS_BSW_00042] and [SWS_BSW_00045].⌋ (SRS_Spi_12093)

[SWS_Spi_00136] ⌈ The function `Spi_SyncTransmit` is pre-compile time selectable by the configuration parameter `SpiLevelDelivered`. This function is only relevant for LEVEL 0 and LEVEL 2.⌋ ()

Parameters of the function `Spi_SyncTransmit` shall be checked as it is explained in section API parameter checking

8.3.12 Spi_GetHWUnitStatus

[SWS_Spi_00186] ⌈

Service name:	Spi_GetHWUnitStatus

Syntax:	`Spi_StatusType Spi_GetHWUnitStatus( Spi_HWUnitType HWUnit )`	
Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	HWUnit	SPI Hardware microcontroller peripheral (unit) ID.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Spi_StatusType	Spi_StatusType
Description:	This service returns the status of the specified SPI Hardware microcontroller peripheral.	
Available via:	`Spi.h`	

] ()

[SWS_Spi_00331] [The operation Spi_GetHWUnitStatus is Re-entrant.] ()

[SWS_Spi_00332] [The function Spi_GetHWUnitStatus service returns the status of the specified SPI Hardware microcontroller peripheral.] ()

[SWS_Spi_00141] [The function Spi_GetHWUnitStatus shall return the status of the specified SPI Hardware microcontroller peripheral.] ()

[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.] ()

[SWS_Spi_00142] [The function `Spi_GetHWUnitStatus` is pre-compile time configurable On / Off by the configuration parameter `SpiHwStatusApi`.] ()

Parameters of the function `Spi_GetHWUnitStatus` shall be checked as it is explained in section API parameter checking.

If SPI Handler/Driver has not been initialized before the function `Spi_GetHWUnitStatus` is called, the return value is undefined.

8.3.13 Spi_Cancel

[SWS_Spi_00187] [

Service name:	Spi_Cancel	
Syntax:	`void Spi_Cancel( Spi_SequenceType Sequence )`	
Service ID[hex]:	0x0c	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	Sequence	Sequence ID.
Parameters (in-out):	None	

Parameters (out):	None
Return value:	None
Description:	Service cancels the specified on-going sequence transmission.
Available via:	`Spi.h`

⌋ ()

[SWS_Spi_00333] ⌈ The operation Spi_Cancel is Re-entrant.⌋ ()

[SWS_Spi_00334] ⌈ The function Spi_Cancel service cancels the specified on-going sequence transmission.⌋ ()

[SWS_Spi_00144] ⌈ The function `Spi_Cancel` shall cancel the specified on-going sequence transmission without cancelling any Job transmission and set the sequence result to `SPI_SEQ_CANCELLED`.⌋ ()

With other words, the `Spi_Cancel` function stops a Sequence transmission after a (possible) on transmission Job ended and before a (potential) next Job transmission starts.

[SWS_Spi_00145] ⌈ When the sequence is cancelled by the function `Spi_Cancel` and if configured, the SPI Handler/Driver shall call the sequence notification call-back function instead of starting a potential next job belonging to it.⌋ ()

[SWS_Spi_00146] ⌈ The function `Spi_Cancel` is pre-compile time configurable On / Off by the configuration parameter `SpiCancelApi`.⌋ ()

The SPI Handler/Driver is not responsible on external devices damages or undefined state due to cancelling a sequence transmission. It is up to the SPI Handler/Driver's environment to be aware to what it is doing!

8.3.14 Spi_SetAsyncMode

[SWS_Spi_00188] ⌈

Service name:	Spi_SetAsyncMode	

Syntax:	`Std_ReturnType Spi_SetAsyncMode(`
	`  Spi_AsyncModeType Mode`
	`)`	

Service ID[hex]:	0x0d	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Mode	New mode required.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Setting command has been done E_NOT_OK: setting command has not been accepted
Description:	Service to set the asynchronous mechanism mode for SPI busses handled asynchronously.	
Available via:	`Spi.h`	

⌋ ()

[SWS_Spi_00335] ⌈ The operation Spi_SetAsyncMode is Non Re-entrant.⌋ ()

[SWS_Spi_00336] ⌈ Return value of the function Spi_SetAsyncMode is E_OK: Setting command has been done.⌋ ()

[SWS_Spi_00337] ⌈ Return value of the function Spi_SetAsyncMode is E_NOT_OK: setting command has not been accepted.⌋ ()

[SWS_Spi_00338] ⌈ The function Spi_SetAsyncMode service to set the asynchronous mechanism mode for SPI buses handled asynchronously.⌋ ()

[SWS_Spi_00152] ⌈ The function `Spi_SetAsyncMode` according to the given parameter shall set the asynchronous mechanism mode for SPI channels configured to behave asynchronously.⌋ ()

[SWS_Spi_00171] ⌈ If the function `Spi_SetAsyncMode` is called while the SPI Handler/Driver status is `SPI_BUSY` and an asynchronous transmission is in progress, the SPI Handler/Driver shall not change the AsyncModeType and keep the mode type as it is. The function shall return the value `E_NOT_OK`.⌋ ()

[SWS_Spi_00172] ⌈ If `Spi_SetAsyncMode` is called while a synchronous transmission is in progress, the SPI Handler/Driver shall set the AsyncModeType according to parameter 'Mode', even if the SPI Handler/Driver status is `SPI_BUSY`. The function shall return the value `E_OK`.⌋ ()

[SWS_Spi_00154] ⌈ The function `Spi_SetAsyncMode` is pre-compile time selectable by the configuration parameter `SpiLevelDelivered`. This function is only relevant for LEVEL 2.⌋ ()

8.4 Callback notifications

This chapter lists all functions provided by the SPI module to lower layer modules.

The SPI Handler/Driver module belongs to the lowest layer of AUTOSAR Software Architecture hence this module specification has not identified any callback functions.

8.5 Scheduled functions

This chapter lists all functions provided by the SPI Handler/Driver and called directly by the Basic Software Module Scheduler.

The SPI Handler/Driver module requires a scheduled function for the management of the asynchronous mode managed with polling (see SWS_Spi_00361). The specified functions below exemplify how to implement them if they are needed.

8.5.1 Spi_MainFunction_Handling

[SWS_Spi_00189] ⌈

Service name:	Spi_MainFunction_Handling

Syntax:	`void Spi_MainFunction_Handling(`
	`  void`
	`)`

Service ID[hex]:	0x10
Description:	--
Available via:	`SchM_Spi.h`

⌋ ()

This function shall polls the SPI interrupts linked to HW Units allocated to the transmission of SPI sequences to enable the evolution of transmission state machine.

8.6 Expected Interfaces

This chapter lists all functions that the SPI Handler/Driver requires from other modules.

8.6.1 Mandatory Interfaces

The SPI Handler/Driver module requires some interfaces to fulfill its core functionality.

[SWS_Spi_00389]⌈

API function	***Header File***	***Description***
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.

⌋ ()

8.6.2 Optional Interfaces

This chapter defines all interfaces which are required to fulfill an optional functionality of SPI Handler/Driver module.

[SWS_Spi_00339] ⌈

API function	***Header File***	***Description***
Dem_SetEventStatus	Dem.h	Called by SW-Cs or BSW modules to report monitor status information to the Dem. BSW modules calling Dem_SetEventStatus can safely ignore the return value.
Det_ReportError	Det.h	Service to report development errors.

⌋ ()

8.6.3 Configurable interfaces

In this chapter all interfaces are listed where the target function could be configured. The target function is usually a call-back function. The name of these interfaces is not fixed because they are configurable.

[SWS_Spi_00075] ⌈ The SPI Handler/Driver shall use the callback routines Spi_JobEndNotification to inform other software modules about certain states or state changes.⌋ (SRS_SPAL_00157)

[SWS_Spi_00264] ⌈ The SPI Handler/Driver shall use the callback routines Spi_SeqEndNotification to inform other software modules about certain states or state changes.⌋ ()

[SWS_Spi_00265] ⌈ For implement the call back function other modules are required to provide the routines in the expected manner.⌋ ()

[SWS_Spi_00044] ⌈ The SPI Handler/Driver's implementer must implement the callback notifications `Spi_JobEndNotification` and `Spi_SeqEndNotification` as function pointers defined within the initialization data structure (`Spi_ConfigType`).⌋ (SRS_SPAL_12056)

[SWS_Spi_00048] ⌈ The callback notifications `Spi_JobEndNotification` and `Spi_SeqEndNotification` shall have no parameters and no return value.⌋ (SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00369)

[SWS_Spi_00054] ⌈ If a callback notification is configured as null pointer, no callback shall be executed.⌋ (SRS_SPAL_12056)

[SWS_Spi_00085] ⌈ It is allowed to use the following API calls within the SPI callback notifications:

- Spi_ReadIB
- Spi_WriteIB
- Spi_SetupEB
- Spi_GetJobResult
- Spi_GetSequenceResult
- Spi_GetHWUnitStatus
- Spi_Cancel

All other SPI Handler/Driver API calls are not allowed.⌋ ()

8.6.3.1 Spi_JobEndNotification

[SWS_Spi_00192] ⌈

Service name:	(*Spi_JobEndNotification)

Syntax:	`void (*Spi_JobEndNotification)(`
	`  void`
	`)`

Sync/Async:	Synchronous
Reentrancy:	Reentrant

Parameters (in):	None
Parameters (in-out):	None
Parameters (out):	None
Return value:	None
Description:	Callback routine provided by the user for each Job to notify the caller that a job has been finished.
Available via:	`Spi_Externals.h`

⌋ ()

[SWS_Spi_00340] ⌈ The operation SpiJobEndNotification is Re-entrant.⌋ ()

[SWS_Spi_00071] ⌈ If the `SpiJobEndNotification` is configured (i.e. not a null pointer), the SPI Handler/Driver shall call the configured callback notification at the end of a Job transmission.⌋ (SRS_SPAL_00157)

Note: This routine might be called on interrupt level, depending on the calling function.

8.6.3.2 Spi_SeqEndNotification

[SWS_Spi_00193] ⌈

Service name:	(*Spi_SeqEndNotification)

Syntax:	`void (*Spi_SeqEndNotification)(`
	`  void`
	`)`

Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (in-out):	None
Parameters (out):	None
Return value:	None
Description:	Callback routine provided by the user for each Sequence to notify the caller that a sequence has been finished.
Available via:	`Spi_Externals.h`

⌋ ()

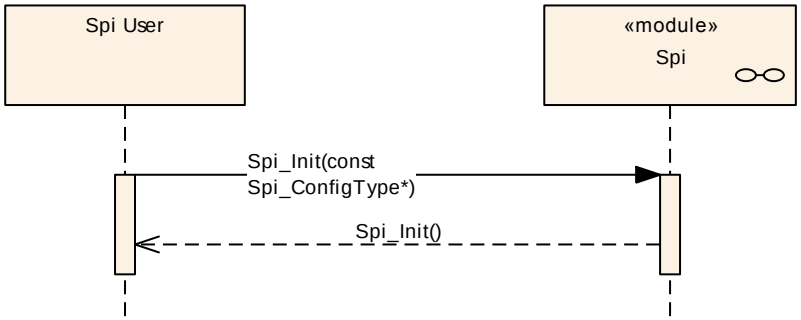
[SWS_Spi_00341] ⌈ The operation SpiSeqEndNotification is Re-entrant.⌋ ()

[SWS_Spi_00073] ⌈ If the `SpiSeqEndNotification` is configured (i.e. not a null pointer), the SPI Handler/Driver shall call the configured callback notification at the end of a Sequence transmission.⌋ (SRS_SPAL_00157)

Note: This routine might be called on interrupt level, depending on the calling function.

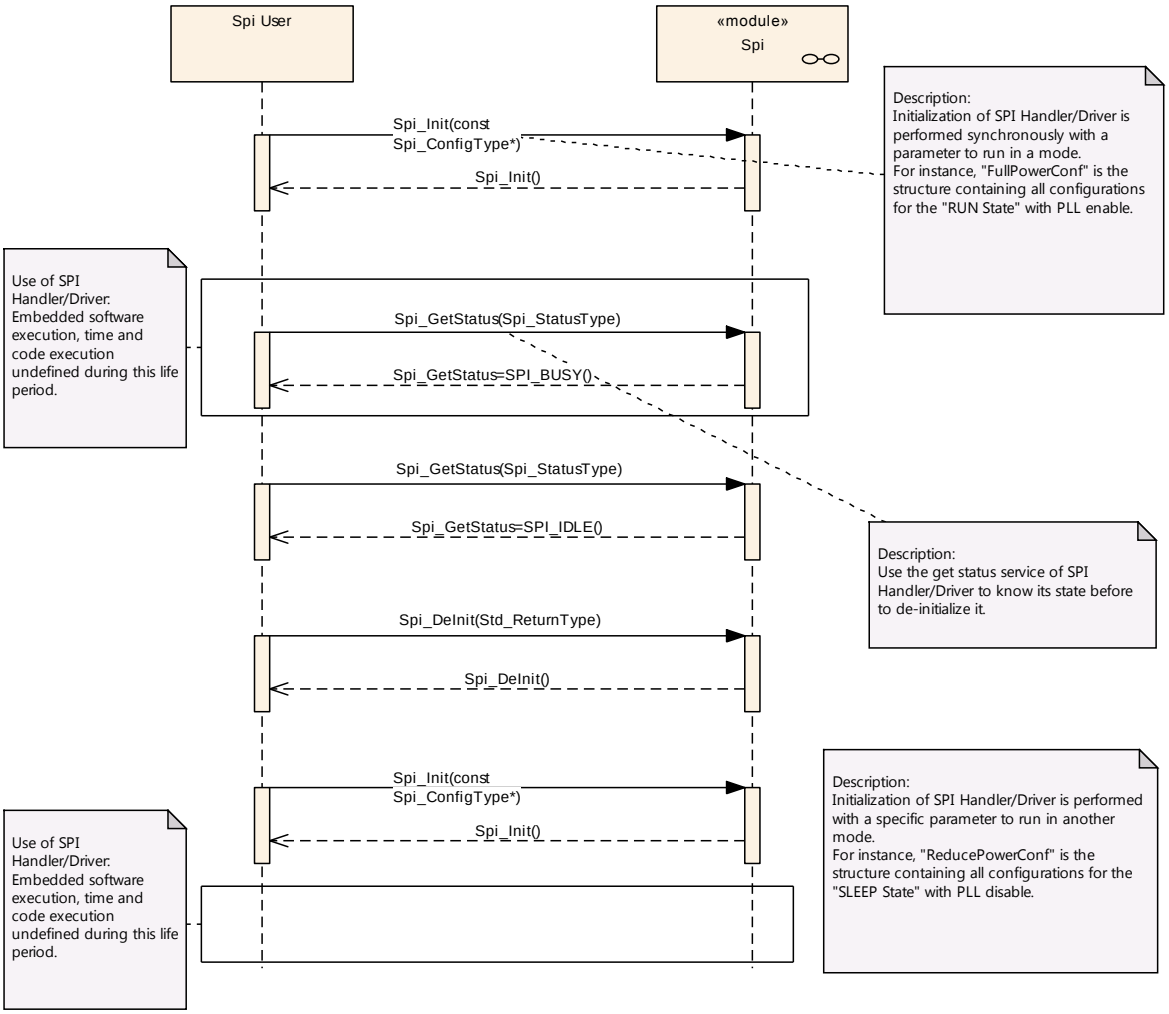
9 Sequence diagrams

9.1 Initialization



9.2 Modes transitions

The following sequence diagram shows an example of an Init / DeInit calls for a running mode transition.



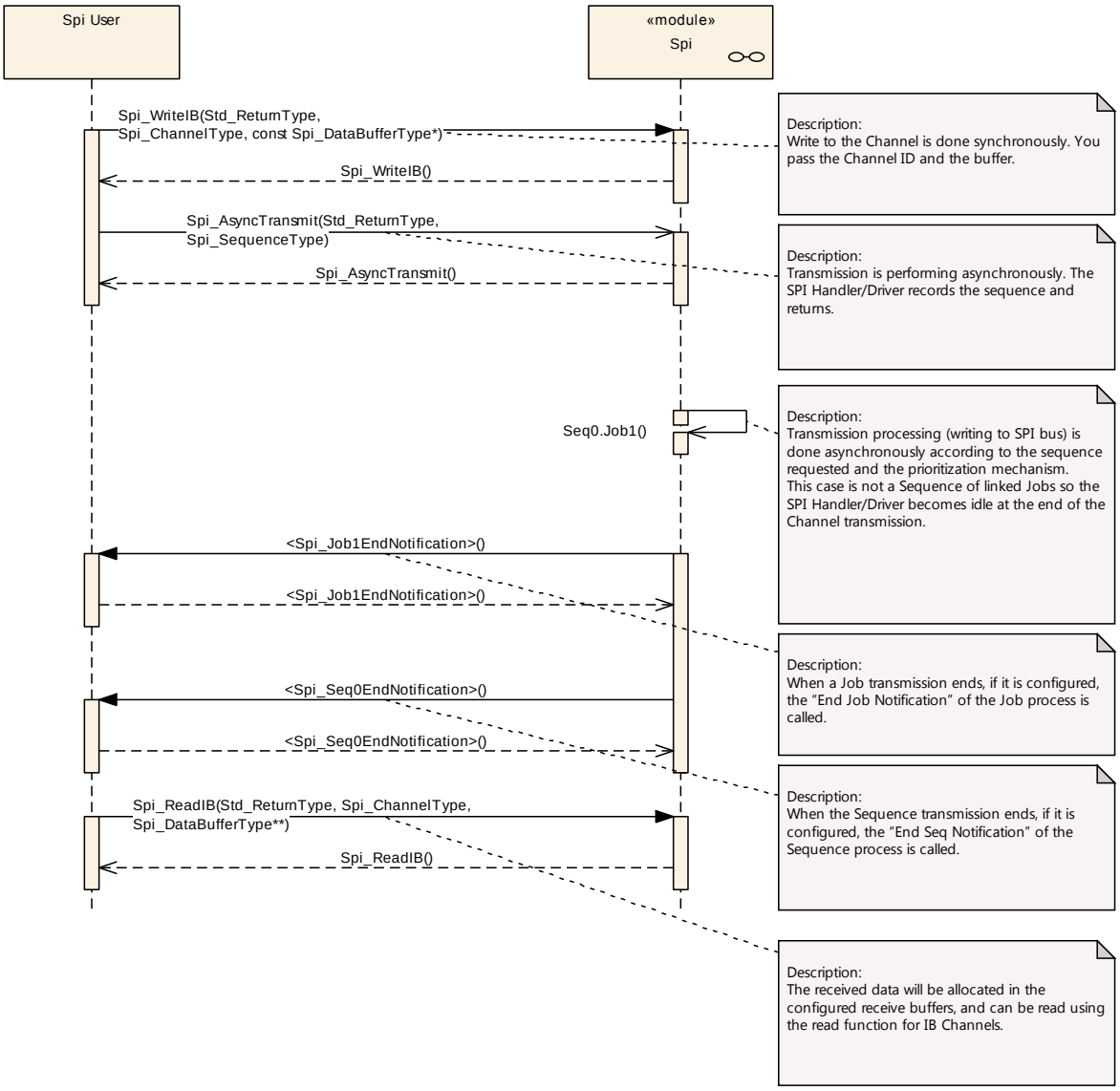
9.3 Write/AsyncTransmit/Read (IB)

9.3.1 One Channel, one Job then one Sequence

The following sequence diagram shows an example of Spi_WriteIB / Spi_AsyncTransmit / Spi_ReadIB calls for a Sequence transmission with only one Job composed of only one Channel. Write or Read step could be skipped when Job is just reading or writing respectively.

Example: Channel ID 2 belongs to Job ID 1 which belongs to Sequence ID 0

Sequence	***Job***	***Channel***
ID0	ID1	ID2

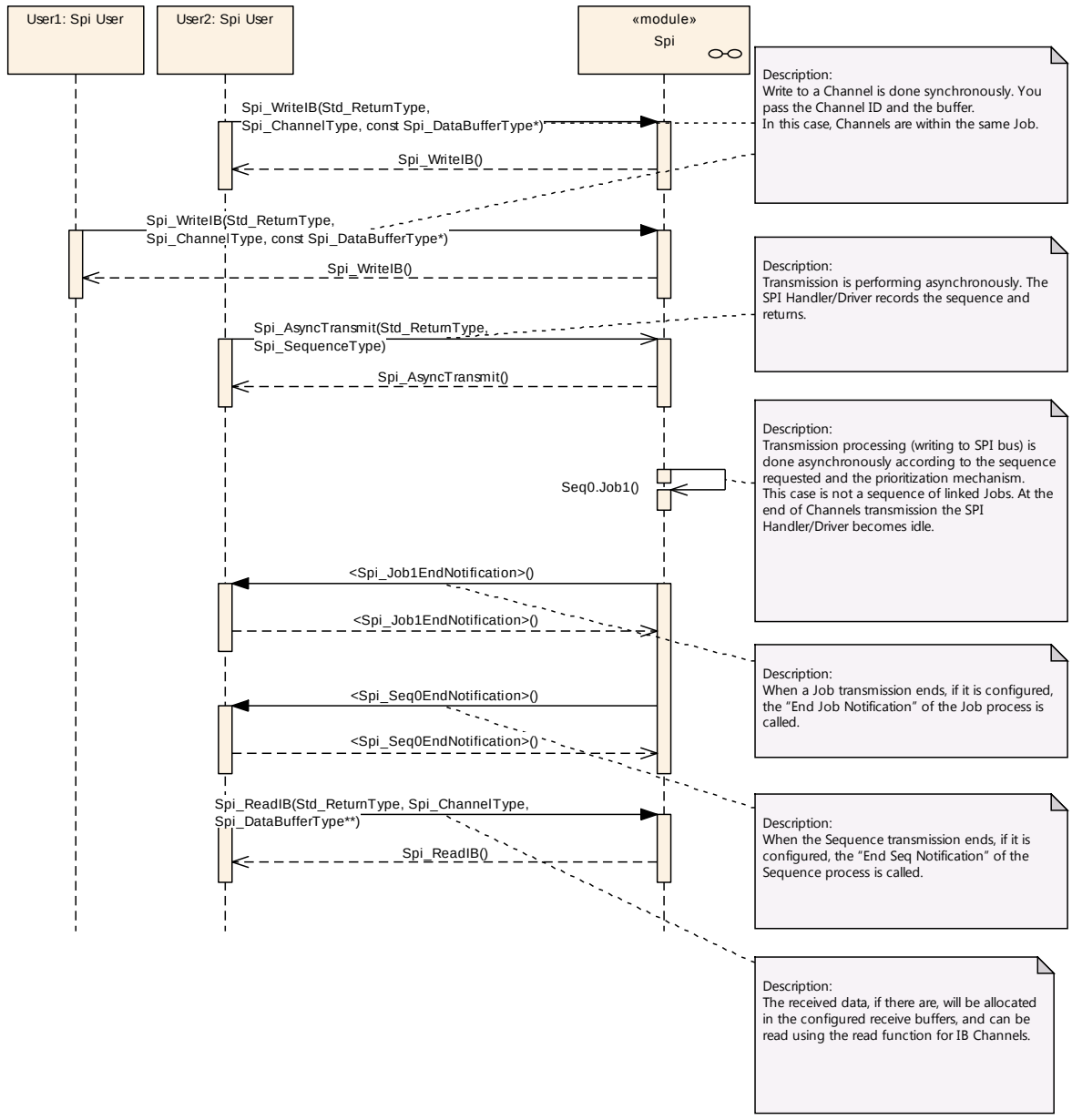

Spi User
«module»
Spi
Spi_WriteIB(Std_ReturnType,
Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
Spi_AsyncTransmit(Std_ReturnType,
Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1()
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()
Spi_ReadIB(Std_ReturnType, Spi_ChannelType,
Spi_DataBufferType**)
Spi_ReadIB()
Description:
Write to the Channel is done synchronously. You pass the Channel ID and the buffer.
Description:
Transmission is performing asynchronously. The SPI Handler/Driver records the sequence and returns.
Description:
Transmission processing (writing to SPI bus) is done asynchronously according to the sequence requested and the prioritization mechanism.
This case is not a Sequence of linked Jobs so the SPI Handler/Driver becomes idle at the end of the Channel transmission.
Description:
When a Job transmission ends, if it is configured, the "End Job Notification" of the Job process is called.
Description:
When the Sequence transmission ends, if it is configured, the "End Seq Notification" of the Sequence process is called.
Description:
The received data will be allocated in the configured receive buffers, and can be read using the read function for IB Channels.

9.3.2 Many Channels, one Job then one Sequence

The following sequence diagram shows an example of Spi_WriteIB / Spi_AsyncTransmit / Spi_ReadIB calls for a Sequence transmission with only one Job composed of many Channels. Write or Read steps could be skipped when Job is just reading or writing respectively.

Example: Channels ID 2 & 3 belong to Job ID 1 which belongs to Sequence ID 0

Sequence	*Job*	*Channel*
ID0	ID1	ID2
		ID3

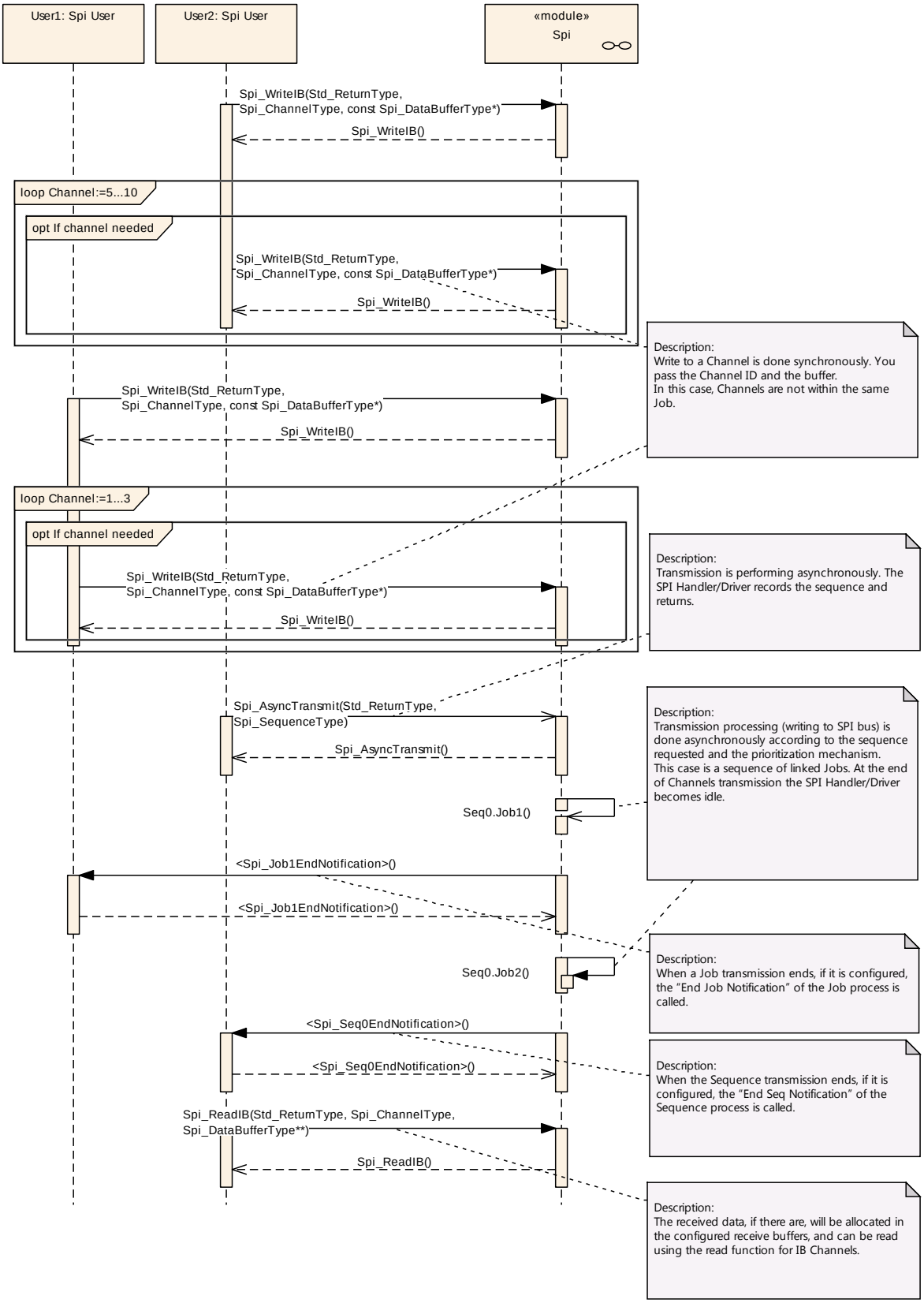


9.3.3 Many Channels, many Jobs and one Sequence

The following sequence diagram shows an example of Spi_WriteIB / Spi_AsyncTransmit / Spi_ReadIB calls for a Sequence transmission of linked Jobs. Write or Read steps could be skipped when Jobs are just reading or writing respectively.

Example: Channels ID 0 to 3 belong to Job ID 1 (higher priority), Channels ID 4 to 10 belong to Job ID 2 (Lower priority) which has not an end notification function. These Jobs belong to the same Sequence ID 0

Sequence	***Job***		***Channel***
	Name	***Priority***	
ID0	ID1	High	ID0…ID3
	ID2	Low	ID4…ID10

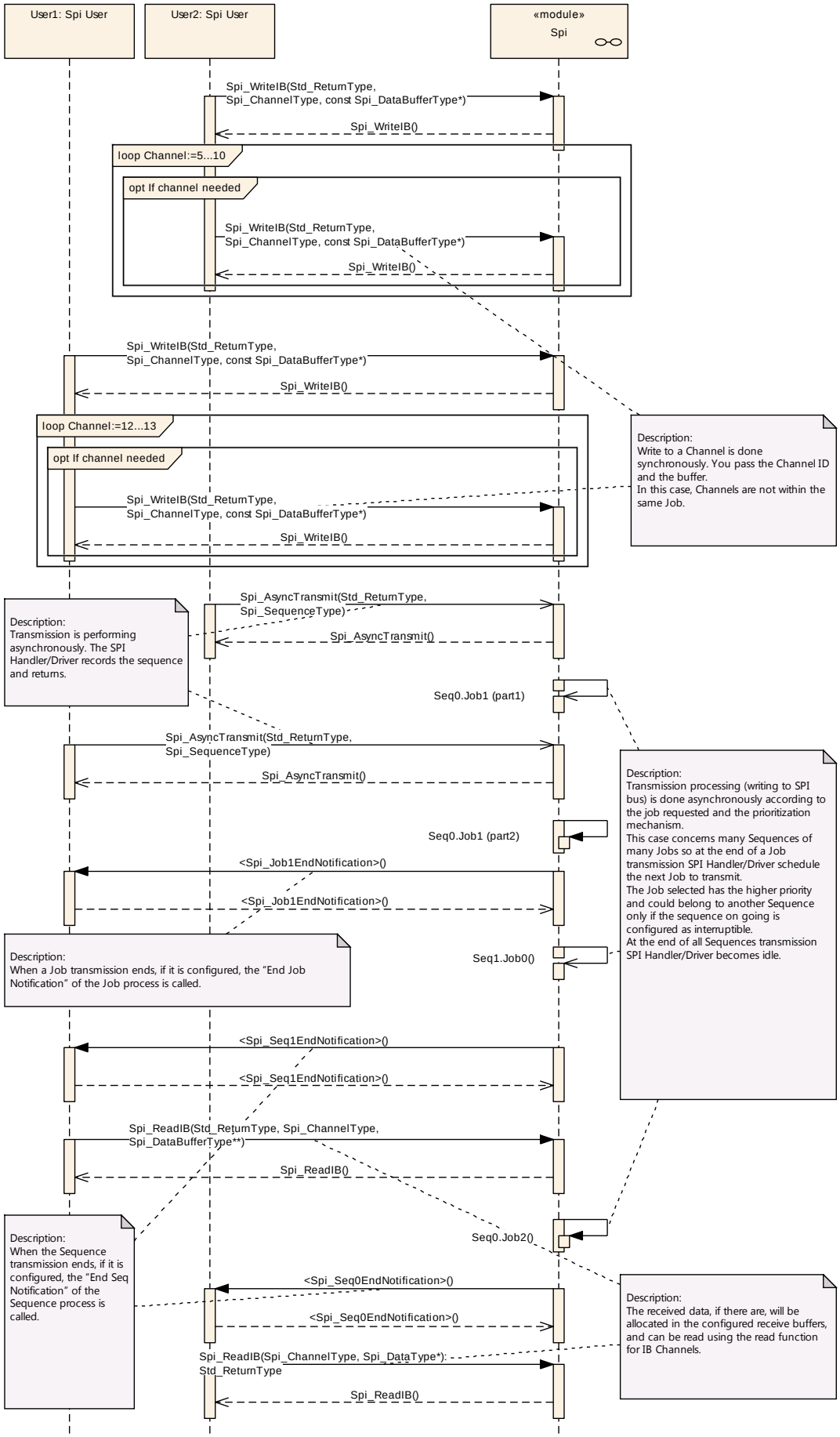

User1: Spi User
User2: Spi User
«module»
Spi
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
loop Channel:=5...10
opt If channel needed
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
Description:
Write to a Channel is done synchronously. You pass the Channel ID and the buffer.
In this case, Channels are not within the same Job.
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
loop Channel:=1...3
opt If channel needed
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
Description:
Transmission is performing asynchronously. The SPI Handler/Driver records the sequence and returns.
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1()
Description:
Transmission processing (writing to SPI bus) is done asynchronously according to the sequence requested and the prioritization mechanism.
This case is a sequence of linked Jobs. At the end of Channels transmission the SPI Handler/Driver becomes idle.
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
Seq0.Job2()
Description:
When a Job transmission ends, if it is configured, the "End Job Notification" of the Job process is called.
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()
Description:
When the Sequence transmission ends, if it is configured, the "End Seq Notification" of the Sequence process is called.
Spi_ReadIB(Std_ReturnType, Spi_ChannelType, Spi_DataBufferType**)
Spi_ReadIB()
Description:
The received data, if there are, will be allocated in the configured receive buffers, and can be read using the read function for IB Channels.

9.3.4 Many Channels, many Jobs and many Sequences

The following sequence diagram shows an example of Spi_WriteIB / Spi_AsyncTransmit / Spi_ReadIB calls for Sequences transmission. Write or Read steps could be skipped when Jobs are just reading or writing respectively.

Example: Channels ID 0 to 3 belong to Job ID 1 (high priority 2), Channels ID 4 to 10 belong to Job ID 2 (Low priority 1) which has not an end notification function. These Jobs belong to the same Sequence ID 0 which is configured as interruptible.
Channels ID 11 to 13 belong to Job ID 0 (higher priority 3) which belongs to Sequence ID 1 which is configured as not interruptible.

Sequence		***Job***		***Channel***
Name	***Interruptible***	***Name***	***Priority***	
ID0	Yes	ID1	2	ID0…ID3
		ID2	1	ID4…ID10
ID1	No	ID0	3	ID11…ID13


User1: Spi User
User2: Spi User
«module»
Spi
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
loop Channel:=5...10
opt If channel needed
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
loop Channel:=12...13
opt If channel needed
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
Description:
Write to a Channel is done synchronously. You pass the Channel ID and the buffer.
In this case, Channels are not within the same Job.
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Description:
Transmission is performing asynchronously. The SPI Handler/Driver records the sequence and returns.
Seq0.Job1 (part1)
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1 (part2)
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
Description:
When a Job transmission ends, if it is configured, the "End Job Notification" of the Job process is called.
Seq1.Job0()
Description:
Transmission processing (writing to SPI bus) is done asynchronously according to the job requested and the prioritization mechanism.
This case concerns many Sequences of many Jobs so at the end of a Job transmission SPI Handler/Driver schedule the next Job to transmit.
The Job selected has the higher priority and could belong to another Sequence only if the sequence on going is configured as interruptible.
At the end of all Sequences transmission SPI Handler/Driver becomes idle.
<Spi_Seq1EndNotification>()
<Spi_Seq1EndNotification>()
Spi_ReadIB(Std_ReturnType, Spi_ChannelType, Spi_DataBufferType**)
Spi_ReadIB()
Seq0.Job2()
Description:
When the Sequence transmission ends, if it is configured, the "End Seq Notification" of the Sequence process is called.
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()
Spi_ReadIB(Spi_ChannelType, Spi_DataType*): Std_ReturnType
Spi_ReadIB()
Description:
The received data, if there are, will be allocated in the configured receive buffers, and can be read using the read function for IB Channels.

9.4 Setup/AsyncTransmit (EB)

9.4.1 Variable Number of Data / Constant Number of Data

[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.⌋ (SRS_Spi_12198, SRS_Spi_12200, SRS_Spi_12201)

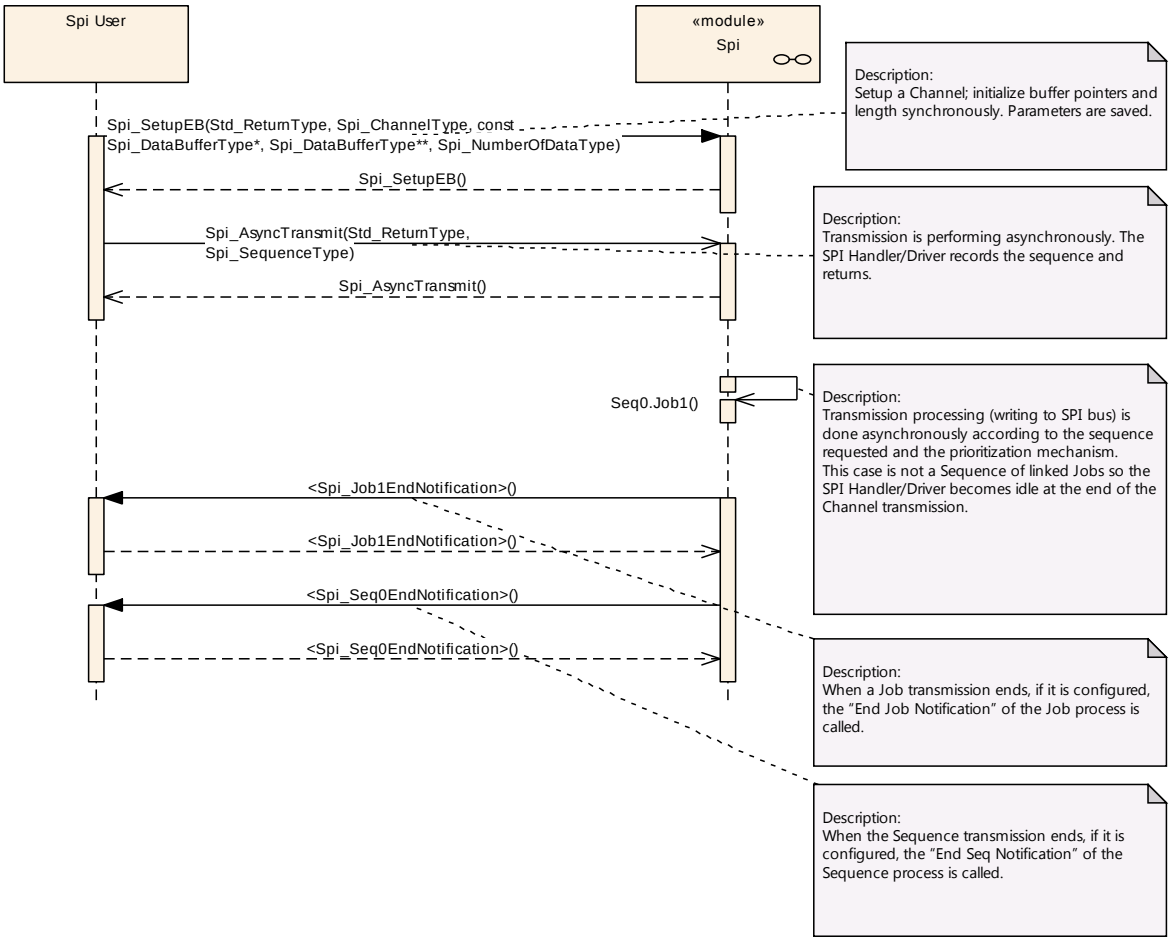
[SWS_Spi_00078] ⌈ To transmit a constant number of data, it is only mandatory to call the `Spi_SetupEB` function to store parameters within SPI Handler/Driver before the first `Spi_AsyncTransmit` function call.⌋ (SRS_Spi_12253, SRS_Spi_12262, SRS_Spi_12202)

9.4.2 One Channel, one Job then one Sequence

The following sequence diagram shows an example of `Spi_SetupEB` / `Spi_AsyncTransmit` calls for a Sequence transmission with only one Job composed of only one Channel. Write or Read accesses are “User Dependant” and could be skipped when Job is just reading or writing respectively.

Example: Channel ID 2 belongs to Job ID 1 which belongs to Sequence ID 0

Sequence	***Job***	***Channel***
ID0	ID1	ID2

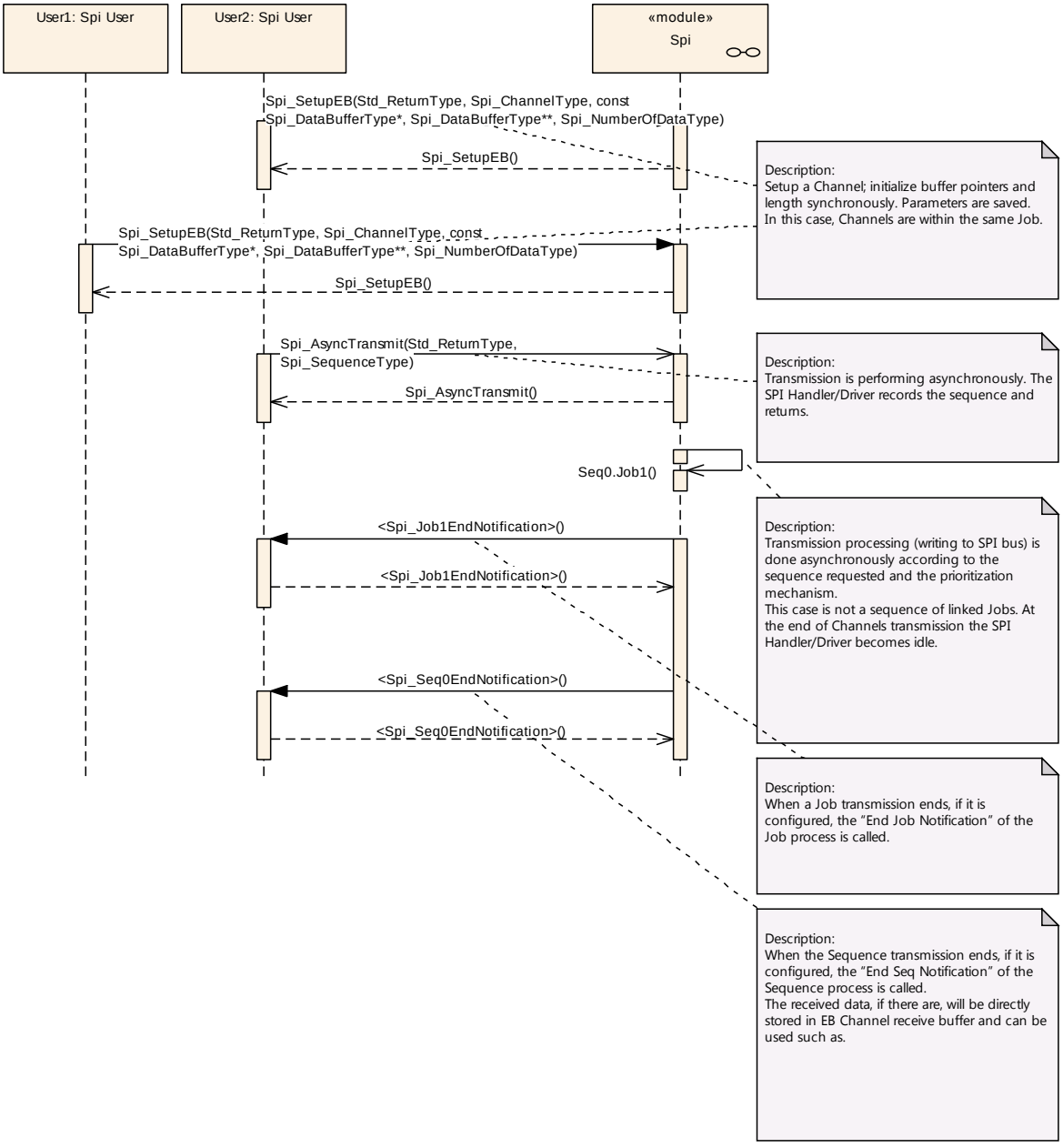

Spi User
«module»
Spi
Description:
Setup a Channel; initialize buffer pointers and length synchronously. Parameters are saved.
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Description:
Transmission is performing asynchronously. The SPI Handler/Driver records the sequence and returns.
Spi_AsyncTransmit()
Seq0.Job1()
Description:
Transmission processing (writing to SPI bus) is done asynchronously according to the sequence requested and the prioritization mechanism.
This case is not a Sequence of linked Jobs so the SPI Handler/Driver becomes idle at the end of the Channel transmission.
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()
Description:
When a Job transmission ends, if it is configured, the "End Job Notification" of the Job process is called.
Description:
When the Sequence transmission ends, if it is configured, the "End Seq Notification" of the Sequence process is called.

9.4.3 Many Channels, one Job then one Sequence

The following sequence diagram shows an example of `Spi_SetupEB` / `Spi_AsyncTransmit` calls for a Sequence transmission with only one Job composed of many Channels. Write or Read accesses are “User Dependant” and could be skipped when Job is just reading or writing respectively.

Example: Channels ID 2 & 3 belong to Job ID 1 which belongs to Sequence ID 0

Sequence	***Job***	***Channel***
ID0	ID1	ID2
		ID3

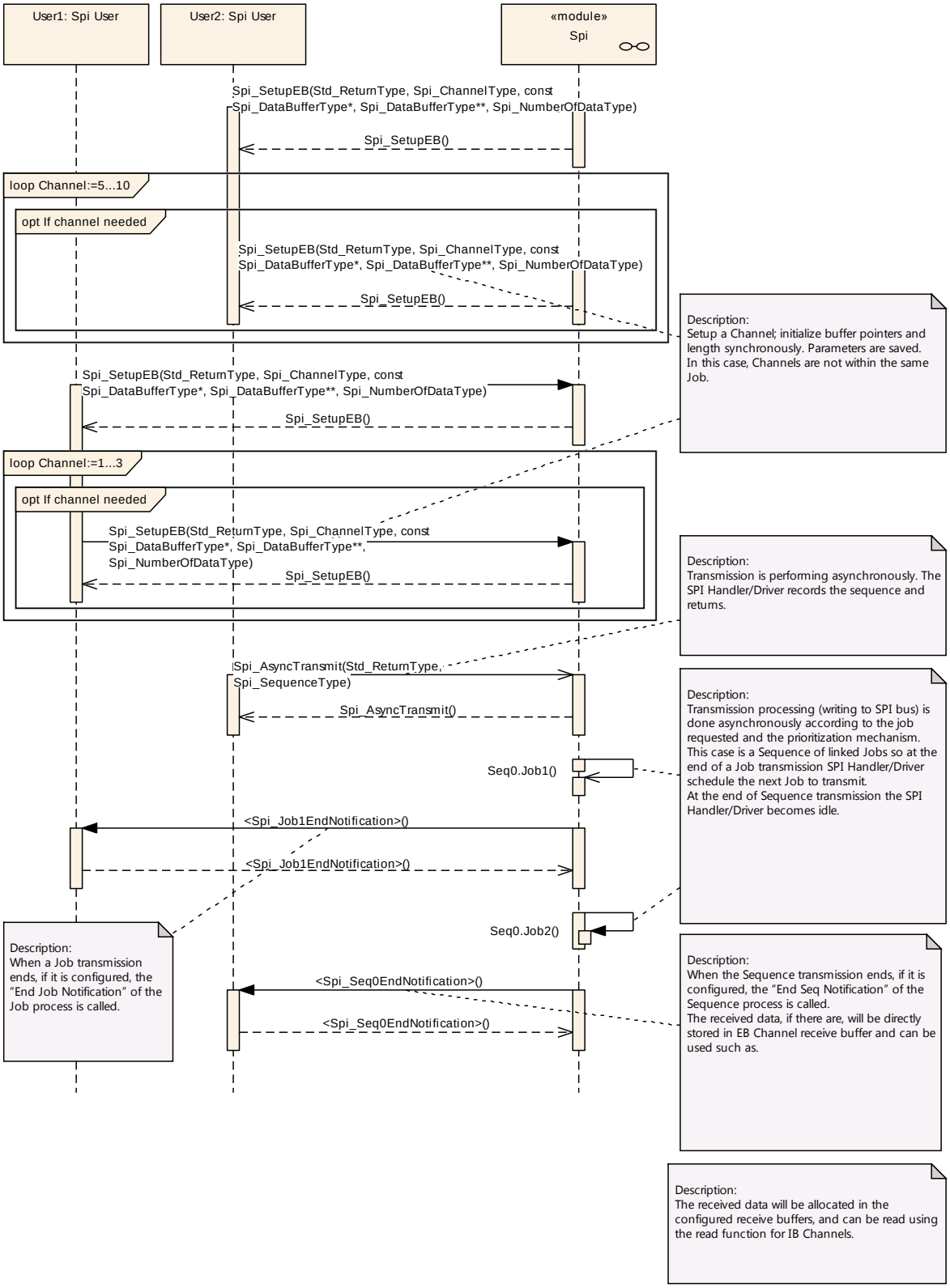


9.4.4 Many Channels, many Jobs and one Sequence

The following sequence diagram shows an example of `Spi_SetupEB` / `Spi_AsyncTransmit` calls for a Sequence transmission of linked Jobs. Write or Read accesses are “User Dependant” and could be skipped when Job is just reading or writing respectively.

Example: Channels ID 0 to 3 belong to Job ID 1 (higher priority), Channels ID 4 to 10 belong to Job ID 2 (Lower priority) which has not an end notification function. These Jobs belong to the same Sequence ID 0

Sequence	***Job***	***Channel***
ID0	ID1	ID0…ID3
	ID2	ID4…ID10

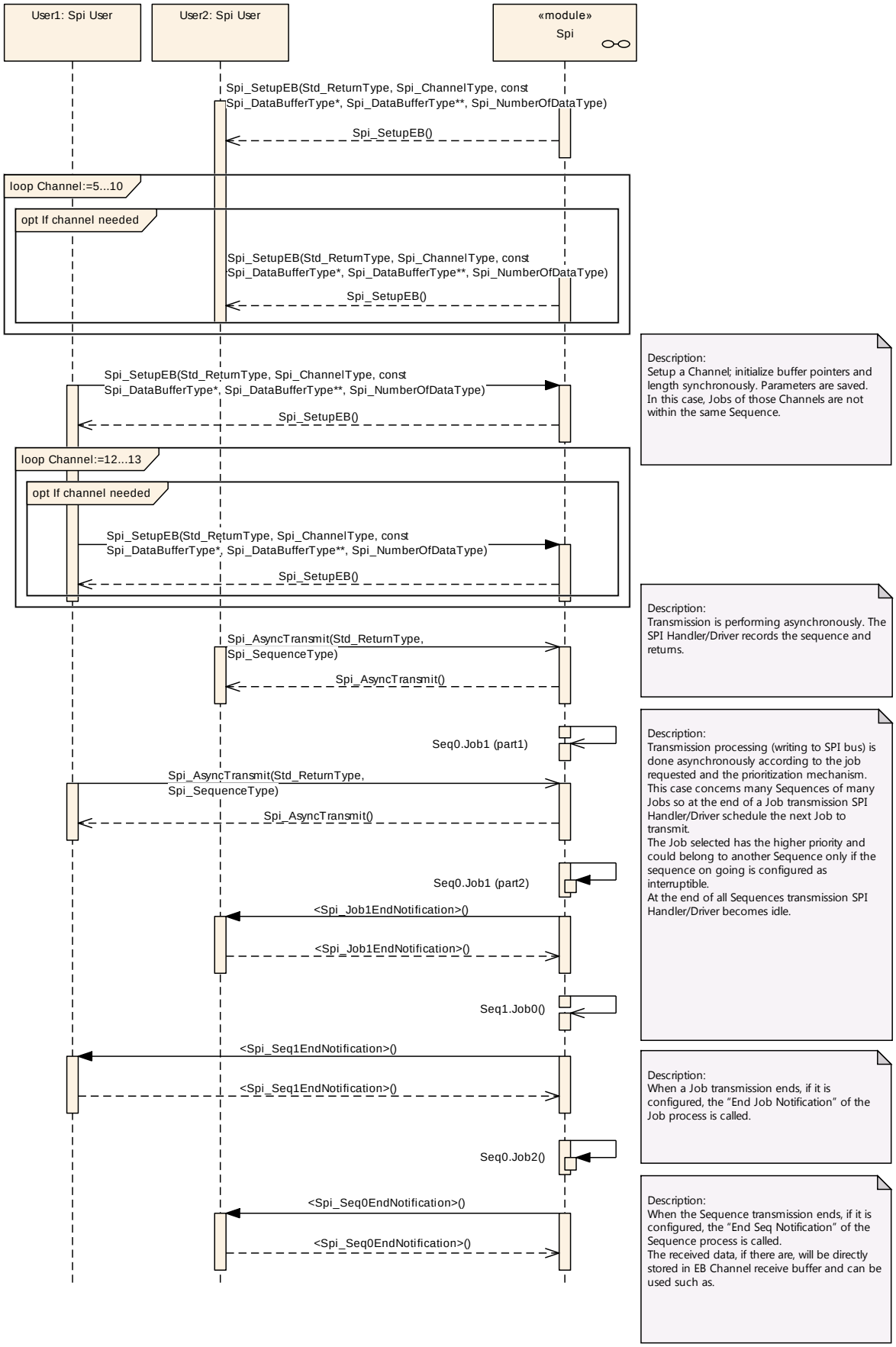

User1: Spi User
User2: Spi User
«module»
Spi
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
loop Channel:=5...10
opt If channel needed
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
Description:
Setup a Channel; initialize buffer pointers and length synchronously. Parameters are saved.
In this case, Channels are not within the same Job.
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
loop Channel:=1...3
opt If channel needed
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
Description:
Transmission is performing asynchronously. The SPI Handler/Driver records the sequence and returns.
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Description:
Transmission processing (writing to SPI bus) is done asynchronously according to the job requested and the prioritization mechanism.
This case is a Sequence of linked Jobs so at the end of a Job transmission SPI Handler/Driver schedule the next Job to transmit.
At the end of Sequence transmission the SPI Handler/Driver becomes idle.
Seq0.Job1()
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
Seq0.Job2()
Description:
When a Job transmission ends, if it is configured, the "End Job Notification" of the Job process is called.
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()
Description:
When the Sequence transmission ends, if it is configured, the "End Seq Notification" of the Sequence process is called.
The received data, if there are, will be directly stored in EB Channel receive buffer and can be used such as.
Description:
The received data will be allocated in the configured receive buffers, and can be read using the read function for IB Channels.

9.4.5 Many Channels, many Jobs and many Sequences

The following sequence diagram shows an example of `Spi_SetupEB` / `Spi_AsyncTransmit` calls for Sequences transmission. Write or Read accesses are “User Dependant” and could be skipped when Job is just reading or writing respectively.

Example: Channels ID 0 to 3 belong to Job ID 1 (high priority 2), Channels ID 4 to 10 belong to Job ID 2 (Low priority 1) which has not an end notification function. These Jobs belong to the same Sequence ID 0 which is configured as interruptible.
Channels ID 11 to 13 belong to Job ID 0 (higher priority 3) which belongs to Sequence ID 1 which is configured as not interruptible.

Sequence		***Job***		***Channel***
Name	***Interruptible***	***Name***	***Priority***	
ID0	Yes	ID1	2	ID0…ID3
		ID2	1	ID4…ID10
ID1	No	ID0	3	ID11…ID13


User1: Spi User
User2: Spi User
«module»
Spi
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
loop Channel:=5...10
opt If channel needed
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
Description:
Setup a Channel; initialize buffer pointers and length synchronously. Parameters are saved. In this case, Jobs of those Channels are not within the same Sequence.
loop Channel:=12...13
opt If channel needed
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Description:
Transmission is performing asynchronously. The SPI Handler/Driver records the sequence and returns.
Seq0.Job1 (part1)
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1 (part2)
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
Description:
Transmission processing (writing to SPI bus) is done asynchronously according to the job requested and the prioritization mechanism.
This case concerns many Sequences of many Jobs so at the end of a Job transmission SPI Handler/Driver schedule the next Job to transmit.
The Job selected has the higher priority and could belong to another Sequence only if the sequence on going is configured as interruptible.
At the end of all Sequences transmission SPI Handler/Driver becomes idle.
Seq1.Job0()
<Spi_Seq1EndNotification>()
<Spi_Seq1EndNotification>()
Description:
When a Job transmission ends, if it is configured, the "End Job Notification" of the Job process is called.
Seq0.Job2()
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()
Description:
When the Sequence transmission ends, if it is configured, the "End Seq Notification" of the Sequence process is called.
The received data, if there are, will be directly stored in EB Channel receive buffer and can be used such as.

9.5 Mixed Jobs Transmission

All kind of mixed Jobs transmission is possible according to the Channels configuration and the priority requirement inside Sequences.

The user knows which Channels are in use. Then, according to the types of these Channels, the appropriate methods shall be called.

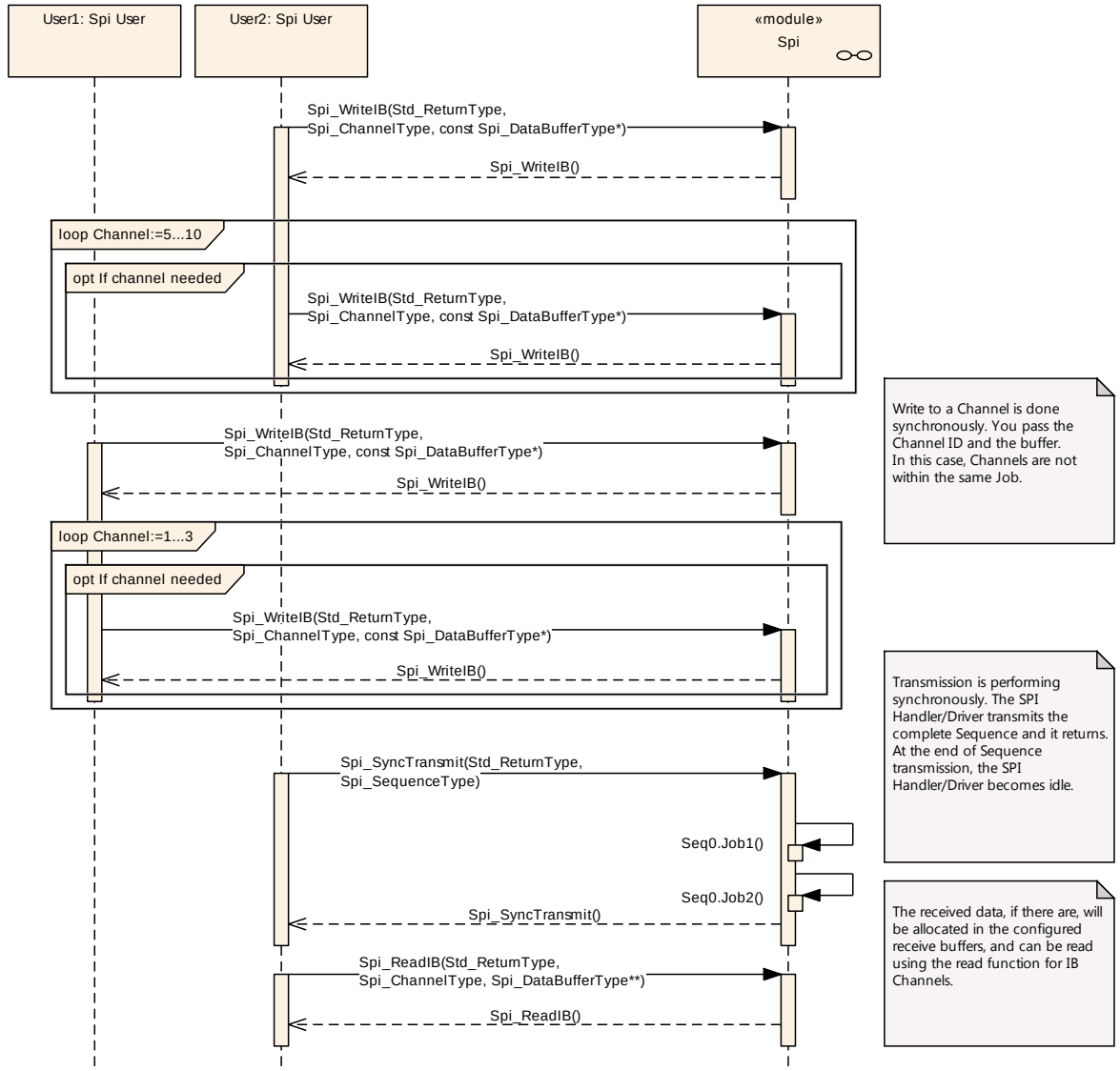
9.6 LEVEL 0 SyncTransmit diagrams

9.6.1 Write/SyncTransmit/Read (IB): Many Channels, many Jobs and one Sequence

The following sequence diagram shows an example of Spi_WriteIB / Spi_SyncTransmit / Spi_ReadIB calls for a Sequence transmission of linked Jobs. Write or Read steps could be skipped when Jobs are just reading or writing respectively.

Example: Channels ID 0 to 3 belong to Job ID 1 (higher priority), Channels ID 4 to 10 belong to Job ID 2 (Lower priority). These Jobs belong to the same Sequence ID 0

Sequence	***Job***		***Channel***
	Name	***Priority***	
ID0	ID1	High	ID0…ID3
	ID2	Low	ID4…ID10

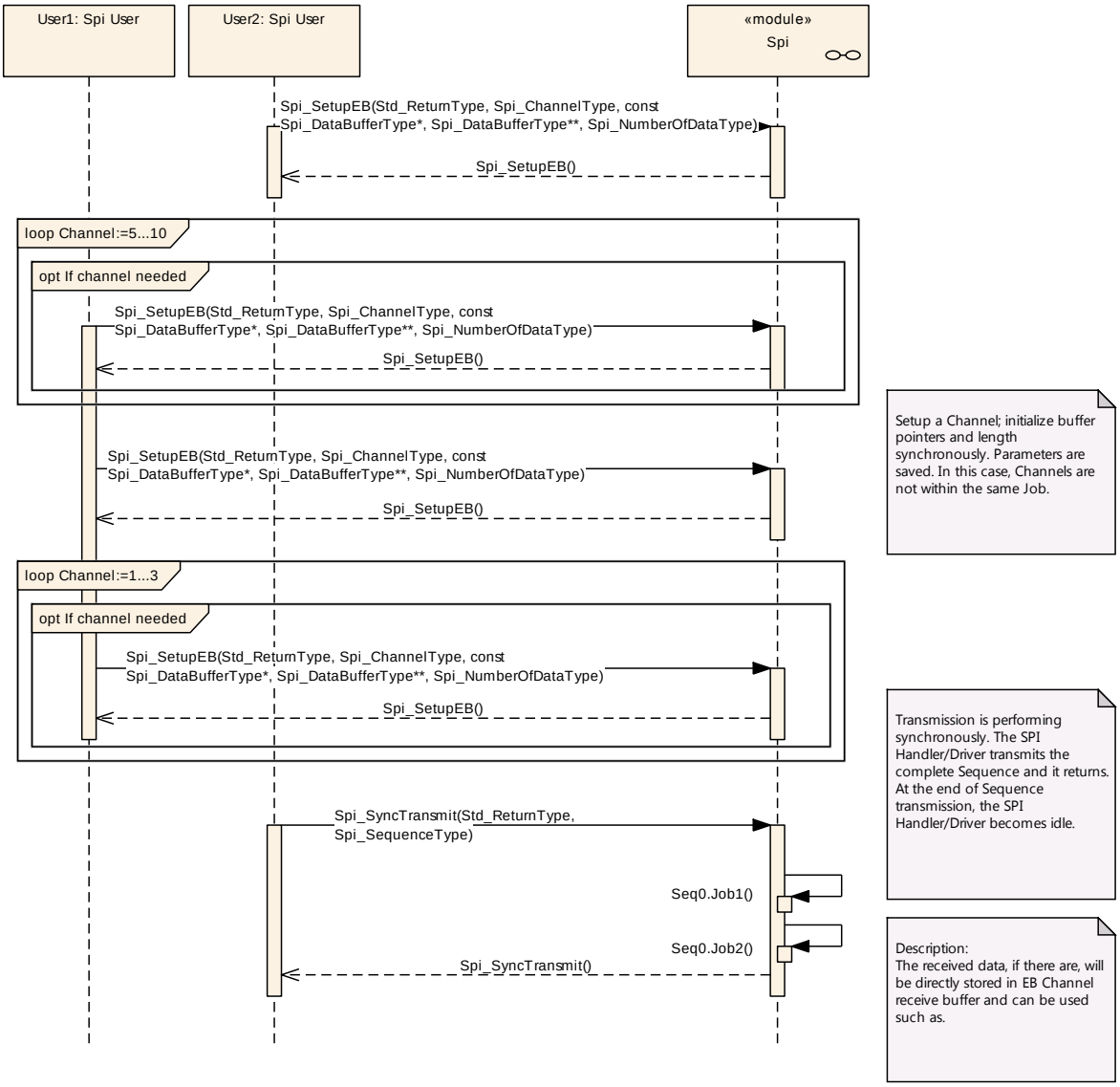


9.6.2 Setup/SyncTransmit (EB): Many Channels, many Jobs and one Sequence

The following sequence diagram shows an example of `Spi_SetupEB` / `Spi_SyncTransmit` calls for a Sequence transmission of linked Jobs. Write or Read accesses are “User Dependant” and could be skipped when Job is just reading or writing respectively.

Example: Channels ID 0 to 3 belong to Job ID 1 (higher priority), Channels ID 4 to 10 belong to Job ID 2 (Lower priority). These Jobs belong to the same Sequence ID 0

Sequence	***Job***	***Channel***
ID0	ID1	ID0…ID3
	ID2	ID4…ID10


User1: Spi User
User2: Spi User
«module»
Spi
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
loop Channel:=5...10
opt If channel needed
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
loop Channel:=1...3
opt If channel needed
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
Spi_SyncTransmit(Std_ReturnType, Spi_SequenceType)
Seq0.Job1()
Seq0.Job2()
Spi_SyncTransmit()
Setup a Channel; initialize buffer pointers and length synchronously. Parameters are saved. In this case, Channels are not within the same Job.
Transmission is performing synchronously. The SPI Handler/Driver transmits the complete Sequence and it returns. At the end of Sequence transmission, the SPI Handler/Driver becomes idle.
Description:
The received data, if there are, will be directly stored in EB Channel receive buffer and can be used such as.

10 Configuration specification

10.1 How to read this chapter

For details refer to the chapter 10.1 “Introduction to configuration specification” in *SWS_BSWGeneral.*

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters are described in Chapter 7 and Chapter 8. Further hardware / implementation specific parameters can be added if necessary.

[SWS_Spi_00390] DRAFT ⌈ The SPI module shall reject configurations with partition mappings which are not supported by the implementation. ⌋ ()

10.2.1 Spi

SWS Item	**ECUC_Spi_00103 :**
Module Name	*Spi*
Module Description	Configuration of the Spi (Serial Peripheral Interface) module.
Post-Build Variant Support	true
Supported Config Variants	VARIANT-LINK-TIME, VARIANT-POST-BUILD, VARIANT-PRE-COMPILE

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
SpiDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
SpiDriver	1	This container contains the configuration parameters and sub containers of the AUTOSAR Spi module.
SpiGeneral	1	General configuration settings for SPI-Handler
SpiPublishedInformation	1	Container holding all SPI specific published information parameters

10.2.2 SpiDemEventParameterRefs

SWS Item	**ECUC_Spi_00240 :**
Container Name	SpiDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
Configuration Parameters	

SWS Item	**ECUC_Spi_00241 :**
Name	SPI_E_HARDWARE_ERROR
Parent Container	SpiDemEventParameterRefs

Description	Reference to configured DEM event to report "Hardware failure". If the reference is not configured the error shall not be reported.		
Multiplicity	0..1		
Type	Symbolic name reference to [DemEventParameter]		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

No Included Containers

10.2.3 SpiGeneral

SWS Item	**ECUC_Spi_00225 :**
Container Name	SpiGeneral
Description	General configuration settings for SPI-Handler
Configuration Parameters	

SWS Item	**ECUC_Spi_00226 :**		
Name	SpiCancelApi		
Parent Container	SpiGeneral		
Description	Switches the Spi_Cancel function ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00227 :**		
Name	SpiChannelBuffersAllowed		
Parent Container	SpiGeneral		
Description	Selects the SPI Handler/Driver Channel Buffers usage allowed and delivered. IB = 0; EB = 1; IB/EB = 2;		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 2		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00228 :**

Name	SpiDevErrorDetect		
Parent Container	SpiGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00229 :**		
Name	SpiHwStatusApi		
Parent Container	SpiGeneral		
Description	Switches the Spi_GetHWUnitStatus function ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00230 :**		
Name	SpiInterruptibleSeqAllowed		
Parent Container	SpiGeneral		
Description	Switches the Interruptible Sequences handling functionality ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local dependency: This parameter depends on SPI_LEVEL_DELIVERED value. It is only used for SPI_LEVEL_DELIVERED configured to 1 or 2.		

SWS Item	**ECUC_Spi_00231 :**	
Name	SpiLevelDelivered	
Parent Container	SpiGeneral	
Description	Selects the SPI Handler/Driver level of scalable functionality that is available and delivered.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 2	
Default value	--	
Post-Build Variant Value	false	

Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00242 :**		
Name	SpiMainFunctionPeriod		
Parent Container	SpiGeneral		
Description	This parameter defines the cycle time of the function Spi_MainFunction_Handling in seconds. The parameter is not used by the driver it self, but it is used by upper layer.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default value	0.01		
Post-Build Variant Multi-plicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00237 :**		
Name	SpiSupportConcurrentSyncTransmit		
Parent Container	SpiGeneral		
Description	Specifies whether concurrent Spi_SyncTransmit() calls for different sequences shall be configurable.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00232 :**		
Name	SpiVersionInfoApi		
Parent Container	SpiGeneral		
Description	Switches the Spi_GetVersionInfo function ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	false		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00244 :**		
Name	SpiEcucPartitionRef		
Parent Container	SpiGeneral		

Description	Maps the SPI driver to zero or multiple ECUC partitions to make the driver API available in the according partition. **Tags:** atp.Status=draft		
Multiplicity	0..*		
Type	Reference to [EcucPartition]		
Post-Build Variant Multi-plicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Spi_00245 :**		
Name	SpiKernelEcucPartitionRef		
Parent Container	SpiGeneral		
Description	Maps the SPI kernel to zero or one ECUC partitions to assign the driver kernel to a certain core. The ECUC partition referenced is a subset of the ECUC partitions where the SPI driver is mapped to. **Tags:** atp.Status=draft		
Multiplicity	0..1		
Type	Reference to [EcucPartition]		
Post-Build Variant Multi-plicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

[SWS_Spi_CONSTR_00001] DRATF [The ECUC partitions referenced by SpiKernelEcucPartitionRef shall be a subset of the ECUC partitions referenced by SpiEcucPartitionRef.] ()

10.2.4 SpiSequence

SWS Item	**ECUC_Spi_00106 :**
Container Name	SpiSequence
Description	All data needed to configure one SPI-sequence
Configuration Parameters	

SWS Item	**ECUC_Spi_00222 :**
Name	SpiInterruptibleSequence
Parent Container	SpiSequence

Description	This parameter allows or not this Sequence to be suspended by another one.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: This SPI_INTERRUPTIBLE_SEQ_ALLOWED parameter as to be configured as ON.		

SWS Item	**ECUC_Spi_00223 :**		
Name	SpiSeqEndNotification		
Parent Container	SpiSequence		
Description	This parameter is a reference to a notification function.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Multi-plicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00224 :**		
Name	SpiSequenceId		
Parent Container	SpiSequence		
Description	SPI Sequence ID, used as parameter in SPI API functions.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ECUC_Spi_00221 :**
Name	SpiJobAssignment
Parent Container	SpiSequence
Description	A sequence references several jobs, which are executed during a communication sequence
Multiplicity	1..*
Type	Reference to [SpiJob]

Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.2.5 SpiChannel

SWS Item	**ECUC_Spi_00104 :**
Container Name	SpiChannel
Description	All data needed to configure one SPI-channel
Configuration Parameters	

SWS Item	**ECUC_Spi_00200 :**		
Name	SpiChannelId		
Parent Container	SpiChannel		
Description	SPI Channel ID, used as parameter in SPI API functions.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00201 :**		
Name	SpiChannelType		
Parent Container	SpiChannel		
Description	Buffer usage with EB/IB channel.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	EB	External Buffer	
	IB	Internal Buffer	
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: SPI_CHANNEL_BUFFERS_ALLOWED		

SWS Item	**ECUC_Spi_00202 :**
Name	SpiDataWidth
Parent Container	SpiChannel
Description	This parameter is the width of a transmitted data unit.
Multiplicity	1

Type	EcucIntegerParamDef		
Range	1 .. 32		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00203 :**		
Name	SpiDefaultData		
Parent Container	SpiChannel		
Description	The default data to be transmitted when (for internal buffer or external buffer) the pointer passed to Spi_WriteIB (for internal buffer) or to Spi_SetupEB (for external buffer) is NULL.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967295		
Default value	--		
Post-Build Variant Multi-plicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00204 :**		
Name	SpiEbMaxLength		
Parent Container	SpiChannel		
Description	This parameter contains the maximum size (number of data elements) of data buffers in case of EB Channels and only.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: The SPI_CHANNEL_TYPE parameter has to be configured as EB for this Channel. The SPI_CHANNEL_BUFFERS_ALLOWED parameter has to be config-ured as 1 or 2.		

SWS Item	**ECUC_Spi_00205 :**
Name	SpiIbNBuffers
Parent Container	SpiChannel
Description	This parameter contains the maximum number of data buffers in case of IB Channels and only.
Multiplicity	1
Type	EcucIntegerParamDef

Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: The SPI_CHANNEL_TYPE parameter has to be configured as IB for this Channel. The SPI_CHANNEL_BUFFERS_ALLOWED parameter has to be configured as 0 or 2.		

SWS Item	**ECUC_Spi_00206 :**		
Name	SpiTransferStart		
Parent Container	SpiChannel		
Description	This parameter defines the first starting bit for transmission.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	LSB	Transmission starts with the Least Significant Bit first	
	MSB	Transmission starts with the Most Significant Bit first	
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.2.6 SpiChannelList

SWS Item	**ECUC_Spi_00233 :**
Container Name	SpiChannelList
Description	References to SPI channels and their order within the Job.
Configuration Parameters	

SWS Item	**ECUC_Spi_00234 :**		
Name	SpiChannelIndex		
Parent Container	SpiChannelList		
Description	This parameter specifies the order of Channels within the Job.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00215 :**
Name	SpiChannelAssignment
Parent Container	SpiChannelList
Description	A job reference to a SPI channel.

Multiplicity	1		
Type	Reference to [SpiChannel]		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

No Included Containers

10.2.7 SpiJob

SWS Item	**ECUC_Spi_00105 :**
Container Name	SpiJob
Description	All data needed to configure one SPI-Job, amongst others the connection between the internal SPI unit and the special settings for an external device is done.
Configuration Parameters	

SWS Item	**ECUC_Spi_00218 :**		
Name	SpiJobEndNotification		
Parent Container	SpiJob		
Description	This parameter is a reference to a notification function.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00219 :**		
Name	SpiJobId		
Parent Container	SpiJob		
Description	SPI Job ID, used as parameter in SPI API functions.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00220 :**

Name	SpiJobPriority		
Parent Container	SpiJob		
Description	Priority set accordingly to SPI093: 0, lowest, 3, highest priority		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 3		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00216 :**		
Name	SpiDeviceAssignment		
Parent Container	SpiJob		
Description	Reference to the external device used by this job		
Multiplicity	1		
Type	Reference to [SpiExternalDevice]		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
SpiChannelList	1..*	References to SPI channels and their order within the Job.

10.2.8 SpiExternalDevice

SWS Item	**ECUC_Spi_00207 :**
Container Name	SpiExternalDevice
Description	The communication settings of an external device. Closely linked to Spi-Job.
Configuration Parameters	

SWS Item	**ECUC_Spi_00208 :**		
Name	SpiBaudrate		
Parent Container	SpiExternalDevice		
Description	This parameter is the communication baudrate - This parameter allows using a range of values, from the point of view of configuration tools, from Hz up to MHz.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00209 :**
Name	SpiCsIdentifier

Parent Container	SpiExternalDevice		
Description	This parameter is the symbolic name to identify the Chip Select (CS) allocated to this Job.		
Multiplicity	1		
Type	EcucStringParamDef (Symbolic Name generated for this parameter)		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
Post-Build Variant Value	false		
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00210 :**		
Name	SpiCsPolarity		
Parent Container	SpiExternalDevice		
Description	This parameter defines the active polarity of Chip Select.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	HIGH	--	
	LOW	--	
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00239 :**		
Name	SpiCsSelection		
Parent Container	SpiExternalDevice		
Description	When the Chip select handling is enabled (see SpiEnableCs), then this parameter specifies if the chip select is handled automatically by Peripheral HW engine or via general purpose IO by Spi driver.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	CS_VIA_GPIO		chip select handled via gpio by Spi driver.
	CS_VIA_PERIPHERAL_ENGINE		chip select is handled automatically by Peripheral HW engine.
Default value	CS_VIA_PERIPHERAL_ENGINE		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: SpiEnableCs		

SWS Item	**ECUC_Spi_00211 :**		
Name	SpiDataShiftEdge		
Parent Container	SpiExternalDevice		
Description	This parameter defines the SPI data shift edge.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	LEADING	--	
	TRAILING	--	
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00212 :**		
Name	SpiEnableCs		
Parent Container	SpiExternalDevice		
Description	This parameter enables or not the Chip Select handling functions. If this parameter is enabled then parameter SpiCsSelection further details the type of chip selection.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00217 :**		
Name	SpiHwUnit		
Parent Container	SpiExternalDevice		
Description	This parameter is the symbolic name to identify the HW SPI Hardware microcontroller peripheral allocated to this Job.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CSIB0	--	
	CSIB1	--	
	CSIB2	--	
	CSIB3	--	
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00213 :**
Name	SpiShiftClockIdleLevel
Parent Container	SpiExternalDevice
Description	This parameter defines the SPI shift clock idle level.
Multiplicity	1

Type	EcucEnumerationParamDef		
Range	HIGH	--	
	LOW	--	
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00214 :**		
Name	SpiTimeClk2Cs		
Parent Container	SpiExternalDevice		
Description	Timing between clock and chip select (in seconds) - This parameter allows to use a range of values from 0 up to 0.0001 seconds. The real configuration-value used in software BSW-SPI is calculated out of this by the generator-tools		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. 1E-4]		
Default value	--		
Post-Build Variant Value	true		
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00246 :**		
Name	SpiDeviceEcucPartitionRef		
Parent Container	SpiExternalDevice		
Description	Maps an SPI external device to zero or multiple ECUC partitions to limit the access to this external device. The ECUC partitions referenced are a subset of the ECUC partitions where the SPI driver is mapped to. **Tags:** atp.Status=draft		
Multiplicity	0..*		
Type	Reference to [EcucPartition]		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Value Configuration Class	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

[SWS_Spi_CONSTR_00002] DRAFT [The ECUC partitions referenced by SpiDeviceEcucPartitionRef shall be a subset of the ECUC partitions referenced by SpiEcucPartitionRef.] ()

10.2.9 SpiDriver

SWS Item	**ECUC_Spi_00091 :**		
Container Name	SpiDriver		
Description	This container contains the configuration parameters and sub containers of the AUTOSAR Spi module.		
Configuration Parameters			

SWS Item	**ECUC_Spi_00197 :**		
Name	SpiMaxChannel		
Parent Container	SpiDriver		
Description	This parameter contains the number of Channels configured. It will be gathered by tools during the configuration stage.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Multi-plicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00198 :**		
Name	SpiMaxJob		
Parent Container	SpiDriver		
Description	Total number of Jobs configured.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
Post-Build Variant Multi-plicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

SWS Item	**ECUC_Spi_00199 :**		
Name	SpiMaxSequence		
Parent Container	SpiDriver		
Description	Total number of Sequences configured.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
Post-Build Variant Multi-	true		

plicity			
Post-Build Variant Value	true		
Multiplicity Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
SpiChannel	1..*	All data needed to configure one SPI-channel
SpiExternalDevice	1..*	The communication settings of an external device. Closely linked to SpiJob.
SpiJob	1..*	All data needed to configure one SPI-Job, amongst others the connection between the internal SPI unit and the special settings for an external device is done.
SpiSequence	1..*	All data needed to configure one SPI-sequence

10.2.10 SpiPublishedInformation

SWS Item	**ECUC_Spi_00235 :**		
Container Name	SpiPublishedInformation		
Description	Container holding all SPI specific published information parameters		
Configuration Parameters			

SWS Item	**ECUC_Spi_00236 :**		
Name	SpiMaxHwUnit		
Parent Container	SpiPublishedInformation		
Description	Number of different SPI hardware microcontroller peripherals (units/busses) available and handled by this SPI Handler/Driver module.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default value	--		
Post-Build Variant Value	false		
Value Configuration Class	***Published Information***	X	All Variants
Scope / Dependency	scope: local		

No Included Containers

10.3 Published information

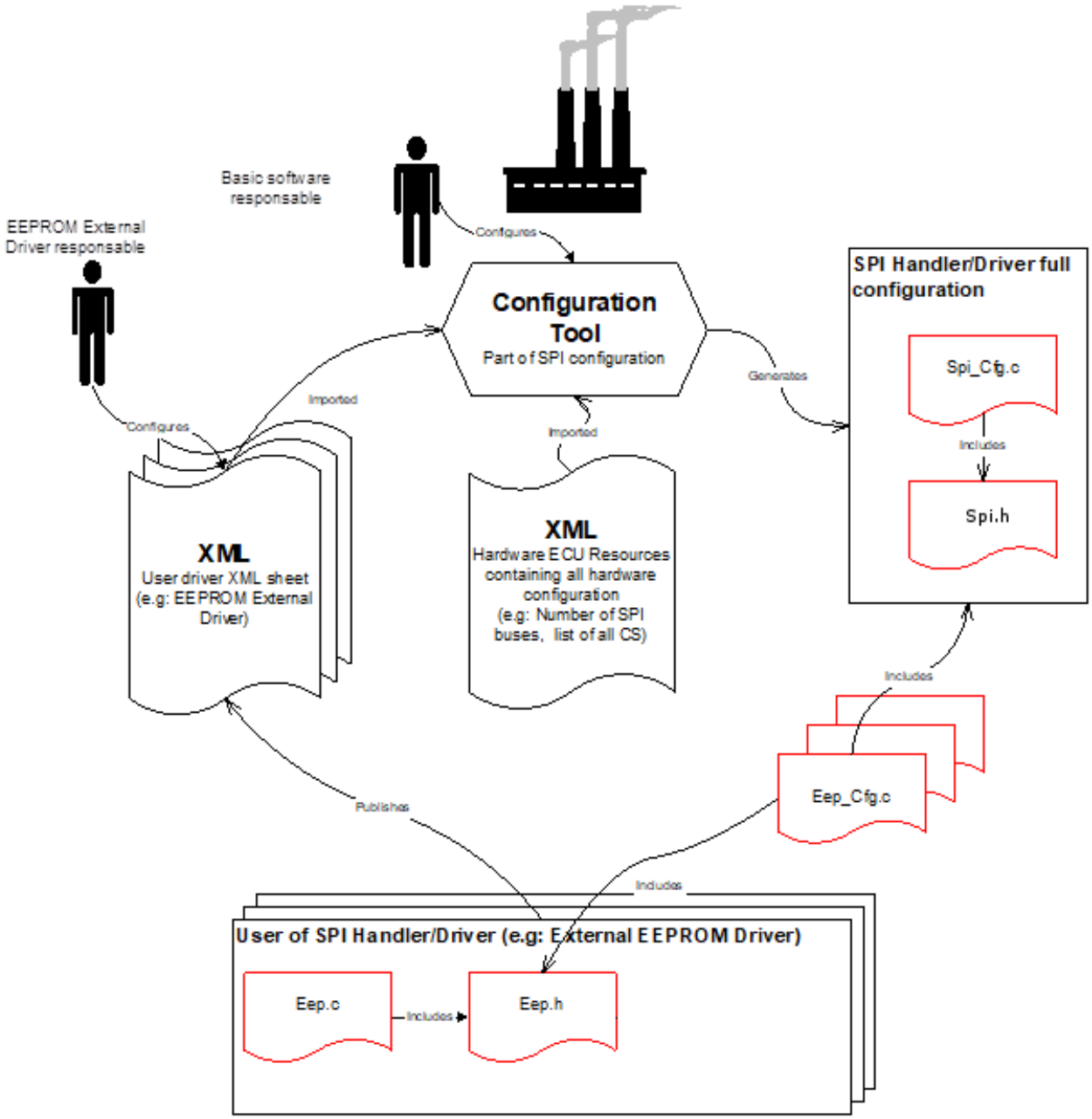
For details refer to the chapter 10.3 “Published Information” in *SWS_BSWGeneral*

10.4 Configuration concept

There is a relationship between the SPI Handler/Driver module and the modules that use it. This relationship is resolved during the configuration stage and the result of it influences the proper API and behaviour between those modules.

The user needs to provide to the SPI Handler/Driver part of the configuration to adapt it to its necessities. The SPI Handler/Driver shall take this configuration and provide the needed tools to the user.

The picture shows the information flow during the configuration of the SPI Handler/Driver. It is shown only for one user, using an External EEPROM Driver as example, but this situation is common to all users of the SPI Handler/Driver. To highlight the situation where more users are affected, several overlapping documents are drawn.



The steps on the diagrams are:

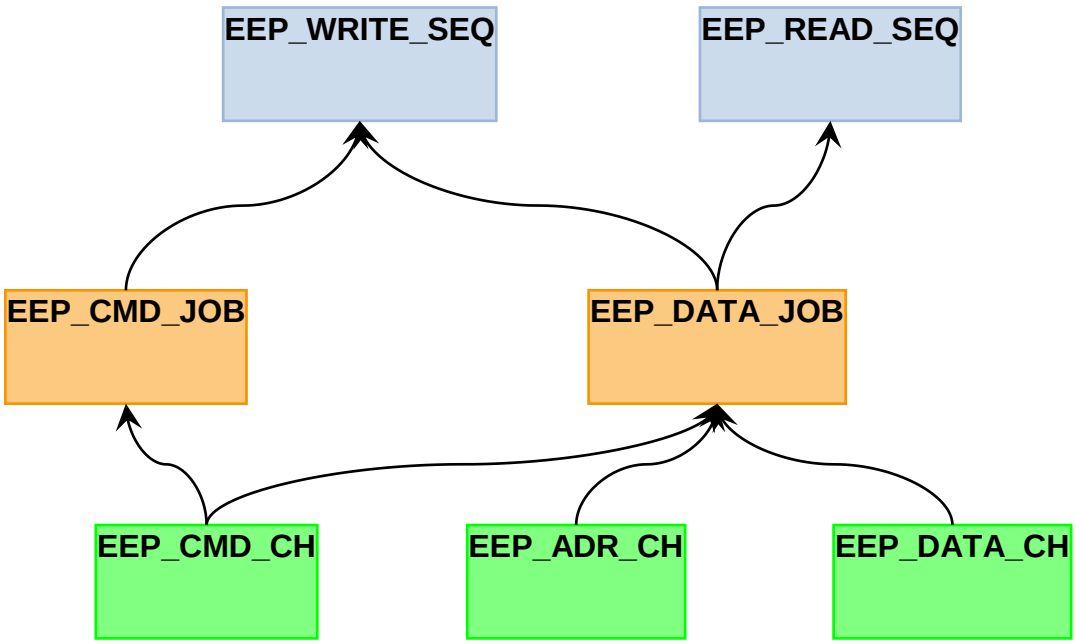
1. The user (External EEPROM Driver) of SPI Handler/Driver edits a XML configuration file. This XML configuration file is the same used by the user to generate its own configuration.
2. For each ECU, a XML HW configuration document contains information which should be used in order to configure some parameters.
3. The “SPI generation tool”. The Generation tool (here is reflected only the part that generates code to SPI usage) shall generate the handles to export and the instance of the configuration sets. In this step the software integrator will provide missing information.
4. SPI instance configuration file. As a result of the generation all the symbolic handlers needed by the user are included in the configuration header file of the SPI Handler/Driver.
5. User gets the symbolic name of handlers. User imports the handle generated to make use of them as requested by its XML configuration file.

11 Not applicable requirements

[SWS_Spi_00999] ⌈ These requirements are not applicable to this specification. ⌋ (SRS_BSW_00301, SRS_BSW_00302, SRS_BSW_00306, SRS_BSW_00307, SRS_BSW_00308, SRS_BSW_00309, SRS_BSW_00312, SRS_BSW_00325, SRS_BSW_00328, SRS_BSW_00330, SRS_BSW_00331, SRS_BSW_00334, SRS_BSW_00341, SRS_BSW_00342, SRS_BSW_00343, SRS_BSW_00347, SRS_BSW_00375, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00401, SRS_BSW_00413, SRS_BSW_00416, SRS_BSW_00417, SRS_BSW_00422, SRS_BSW_00423, SRS_BSW_00424, SRS_BSW_00426, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00005, SRS_BSW_00006, SRS_BSW_00009, SRS_BSW_00010, SRS_BSW_00161, SRS_BSW_00164, SRS_BSW_00168, SRS_BSW_00170, SRS_BSW_00172, SRS_SPAL_12267, SRS_SPAL_12068, SRS_SPAL_12069, SRS_SPAL_12063, SRS_SPAL_12129, SRS_SPAL_12067, SRS_SPAL_12077, SRS_SPAL_12078, SRS_SPAL_12092, SRS_SPAL_12265)

12 Appendix

The table shown on the next page is just an example to help future users (and/or developers) that have to configure software modules to use the SPI Handler/Driver. This table is independent of the `Spi_ConfigType` structure but contains all elements and aggregations like Channels, Jobs and Sequences.



External EEPROM Write/Read Configuration for SPI Handler/Driver								
Sequences			**Jobs**			**Channels**		
Symbolic Name	**ID**	**Attributes**	**Symbolic Name**	**ID**	**Attributes**	**Symbolic Name**	**ID**	**Attributes**
EEP_WRITE_SEQ	0	2 (Number of Jobs), {EEP_CMD_JOB, EEP_DATA_JOB} (List of Jobs), Not Interruptible, EEP_vidEndOfWriteSeq	EEP_CMD_JOB	0	SPI_BUS_0, CS_EEPROM, CS_ON, CS_LOW, CLK_2MHz, 1 (time in µs), Polarity 180, Falling Edge, 3, EEP_vidEndOfStartWrJob, 1 (Number of Channels) {EEP_CMD_CH} (List of Channels)	EEP_CMD_CH	0	EB, 8 bits, 1 data to TxD, MSB First, Default value is 0x00
EEP_READ_SEQ	1	1 (Number of Jobs), {EEP_DATA_JOB} (List of Jobs), Not Interruptible, EEP_vidEndOfReadSeq	EEP_DATA_JOB	1	SPI_BUS_0, CS_EEPROM, CS_ON, CS_LOW, CLK_2MHz, 1 (time in µs), Polarity 180, Falling Edge, 2, NULL, 3 (Number of Channels) {EEP_CMD_CH, EEP_ADR_CH, EEP_DATA_CH} (List of Channels)	EEP_ADR_CH	1	EB, 16 bits, 1 data to TxD, MSB First, Default value is 0x0000
						EEP_DATA_CH	2	EB, 8 bits, 32 data to TxD, MSB First, Default value is 0x00