Document Title	Specification of SPI Handler/Driver
Document Category	Software specification (SWS)
Document UID	38
Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R26-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2026-11-26	R26-11	AUTOSAR Release Management	• Removed GetVersionInfo API • Aligned the API chapter structure with the updated SWS template • Changed the extended production errors into production errors
2025-11-27	R25-11	AUTOSAR Release Management	• No content changes
2024-11-27	R24-11	AUTOSAR Release Management	• [SRS_BSW_00334] removed from [SWS_Spi_NA_00999]
2023-11-23	R23-11	AUTOSAR Release Management	• [SWS_Spi_00389] moved to mandatory interfaces • Editorial changes
2022-11-24	R22-11	AUTOSAR Release Management	• Reworked or rephrased requirements: [SWS_Spi_NA_00999], [SWS_Spi_00126], [SWS_Spi_00151], [ECUC_Spi_00220], [SWS_Spi_00377], [SWS_Spi_00389], [SWS_Spi_00150], [ECUC_Spi_00214] • Editorial changes

△

2021-11-25	R21-11	AUTOSAR Release Management	• Chapter 10 diagrams updated • New configuration parameter: [ECUC_Spi_00249] • Reworked or rephrased requirements: [SWS_Spi_00128], [SWS_Spi_00382], [SWS_Spi_00360], [SWS_Spi_00170], [SWS_Spi_00150], [SWS_Spi_00329], [SWS_Spi_00154], [ECUC_Spi_00208], [ECUC_Spi_00214], [ECUC_Spi_00202], [ECUC_Spi_00204], [ECUC_Spi_00205], [ECUC_Spi_00234], [ECUC_Spi_00242], [ECUC_Spi_00197], [ECUC_Spi_00198], [ECUC_Spi_00199], [ECUC_Spi_00236] • Removed requirements: [SWS_Spi_00108], [SWS_Spi_00155], [SWS_Spi_00152], [SWS_Spi_00271], [SWS_Spi_00008], [SWS_Spi_00009], [SWS_Spi_00010], [SWS_Spi_00063], [SWS_Spi_00064], [SWS_Spi_00344] • Editorial changes, errors descriptions updated, SpiDataWidth up to 64 bits
2020-11-30	R20-11	AUTOSAR Release Management	• Error sections refactored • New configuration parameters: [SWS_Spi_00247], [SWS_Spi_00248] • Removed requirements: [SWS_Spi_00008], [SWS_Spi_00009], [SWS_Spi_00010], [SWS_Spi_00063] and [SWS_Spi_00064] • Chapter 8.2: enumeration types have the values specified
2019-11-28	R19-11	AUTOSAR Release Management	• [SWS_Spi_00082] removed • Changed Document Status from Final to published
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 pre-arranged 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] • Improved the traceability within 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
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 going 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] Length is in data elements instead of bytes • MemMap header file rename • Added Subchapter 3.x due to SWS General Rollout
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 SPI 083; SPI 132, SPI 284 and SPI 107 removde 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 SPI 190, SPI 094 • Updated configuration: base on min-max value for defined parameter; SpiHwUnit belongs to SpiExternalDevice Container; updated SpiTimeClk2Cs
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 inclusions 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 12

2 Acronyms and Abbreviations 13

3 Related documentation 14
3.1 Input documents & related standards and norms . . . 14
3.2 Related specification . . . 14

4 Constraints and assumptions 15
4.1 Limitations . . . 15
4.2 Applicability to car domains . . . 15

5 Dependencies to other modules 16

6 Requirements Tracing 17

7 Functional specification 20
7.1 Overall view of functionalities and features . . . 20
7.2 General behaviour . . . 21
7.2.1 Common configurable feature: Allowed Channel Buffers . . . 24
7.2.1.1 Behaviour of IB channels . . . 25
7.2.1.2 Behaviour of EB channels . . . 25
7.2.1.3 Buffering channel usage . . . 26
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
7.2.4.1 Behavior of Non-Interruptible Sequences . . . 30
7.2.4.2 Behavior of Mixed Sequences . . . 30
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 Production Errors . . . 33
7.4.3.1 SPI_E_HARDWARE_ERROR . . . 33
7.5 Security Events . . . 34

8 API specification 35
8.1 Imported types . . . 35
8.2 Type definitions . . . 35
8.2.1 Spi_ConfigType . . . 35
8.2.2 Spi_StatusType . . . 35
8.2.3 Spi_JobResultType . . . 36
8.2.4 Spi_SeqResultType . . . 37
8.2.5 Spi_DataBufferType . . . 38

8.2.6 Spi_NumberOfDataType . . . 39
8.2.7 Spi_ChannelType . . . 39
8.2.8 Spi_JobType . . . 39
8.2.9 Spi_SequenceType . . . 40
8.2.10 Spi_HWUnitType . . . 40
8.2.11 Spi_AsyncModeType . . . 41
8.3 Function definitions . . . 41
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 . . . 49
8.3.7 Spi_GetStatus . . . 50
8.3.8 Spi_GetJobResult . . . 51
8.3.9 Spi_GetSequenceResult . . . 52
8.3.10 Spi_SyncTransmit . . . 53
8.3.11 Spi_GetHWUnitStatus . . . 54
8.3.12 Spi_Cancel . . . 55
8.3.13 Spi_SetAsyncMode . . . 56
8.3.14 Callback notifications . . . 56
8.4 Scheduled functions . . . 57
8.4.1 Spi_MainFunction_Handling . . . 57
8.5 Expected interfaces . . . 57
8.5.1 Mandatory interfaces . . . 57
8.5.2 Optional interfaces . . . 58
8.5.3 Callout interfaces . . . 58
8.5.4 Configurable interfaces . . . 58
8.5.4.1 Spi_JobEndNotification . . . 59
8.5.4.2 Spi_SeqEndNotification . . . 60
8.6 Error detection . . . 61
8.6.1 API parameter checking . . . 61
8.6.2 SPI state checking . . . 62
8.6.3 SPI runtime checking . . . 62
9 Sequence diagrams 63
9.1 Initialization . . . 63
9.2 Modes transitions . . . 63
9.3 Write/AsyncTransmit/Read (IB) . . . 64
9.3.1 One Channel, one Job then one Sequence . . . 64
9.3.2 Many Channels, one Job then one Sequence . . . 65
9.3.3 Many Channels, many Jobs and one Sequence . . . 66
9.3.4 Many Channels, many Jobs and many Sequences . . . 68
9.4 Setup/AsyncTransmit (EB) . . . 70

9.4.1 Variable Number of Data / Constant Number of Data . . . 70
9.4.2 One Channel, one Job then one Sequence . . . 70
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 . . . 80
10.2.3 SpiGeneral . . . 80
10.2.4 SpiSequence . . . 85
10.2.5 SpiChannel . . . 87
10.2.6 SpiChannelList . . . 91
10.2.7 SpiJob . . . 93
10.2.8 SpiExternalDevice . . . 95
10.2.9 SpiDriver . . . 102
10.2.10 SpiPublishedInformation . . . 104
10.3 Published Information . . . 104
10.4 Configuration concept . . . 104
A Not applicable requirements 107
B Appendix 108
C Change history of AUTOSAR traceable items 110
C.1 Traceable item history of this document according to AUTOSAR Release R26-11 . . . 110
C.1.1 Added Specification Items in R26-11 . . . 110
C.1.2 Changed Specification Items in R26-11 . . . 110
C.1.3 Deleted Specification Items in R26-11 . . . 110
C.1.4 Added Constraints in R26-11 . . . 110
C.1.5 Changed Constraints in R26-11 . . . 110
C.1.6 Deleted Constraints in R26-11 . . . 111
C.2 Traceable item history of this document according to AUTOSAR Release R25-11 . . . 111
C.2.1 Added Specification Items in R25-11 . . . 111
C.2.2 Changed Specification Items in R25-11 . . . 111
C.2.3 Deleted Specification Items in R25-11 . . . 111

C.2.4 Added Constraints in R25-11 111
C.2.5 Changed Constraints in R25-11 111
C.2.6 Deleted Constraints in R25-11 111

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 SPI Handler/Driver can transmit data frames in asynchronous or synchronous way according to the API function called and 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

The glossary below includes acronyms and abbreviations relevant to the SPI Handler/-Driver module that are not included in the [1, AUTOSAR 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

Table 2.1: Acronyms used in the scope of this Document

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).

Table 2.2: Abbreviations used in the scope of this Document

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 (one chip select = one external device). A Job is considered atomic and therefore cannot be interrupted by another Job. A Job has an assigned priority. Depending on the configuration, the CS may be kept asserted for the whole job (so for all the Channels) or released for each data frame at SPI bus level.
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.
Data frame	A data frame is the physical frame of bits on the SPI bus in relation with SpiDataWidth.

Table 2.3: Definitions used in the scope of this Document

3 Related documentation

3.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[3] Specification of MCU Driver
AUTOSAR_CP_SWS_MCUDriver

[4] Specification of Port Driver
AUTOSAR_CP_SWS_PortDriver

[5] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[6] General Requirements on SPAL
AUTOSAR_CP_RS_SPALGeneral

[7] Requirements on SPI Handler/Driver
AUTOSAR_CP_RS_SPIHandlerDriver

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [2, 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.⌋

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 [3].⌋

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 [3] 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 [4]. 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 Tracing

The following tables reference the requirements specified in [5], [6], [7] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[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_00323]	All AUTOSAR Basic Software Modules shall check passed API parameters for validity	[SWS_Spi_00031] [SWS_Spi_00032] [SWS_Spi_00060]
[SRS_BSW_00327]	Error values naming convention	[SWS_Spi_00004]
[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_00359]	Callback Function Return Types for AUTOSAR BSW	[SWS_Spi_00048]
[SRS_BSW_00360]	AUTOSAR Basic Software Modules callback functions are allowed to have parameters	[SWS_Spi_00048]
[SRS_BSW_00369]	All AUTOSAR Basic Software Modules shall not return specific development error codes via the API	[SWS_Spi_00048]
[SRS_BSW_00385]	List possible error notifications	[SWS_Spi_00004]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_Spi_00013]
[SRS_BSW_00406]	API handling in uninitialized state	[SWS_Spi_00015] [SWS_Spi_00046] [SWS_Spi_00373]
[SRS_BSW_00458]	Classification of production errors	[SWS_Spi_00383]
[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_00044] [SWS_Spi_00054]
[SRS_SPAL_12057]	All driver modules shall implement an interface for initialization	[SWS_Spi_00013] [SWS_Spi_00015]
[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_12075]	All drivers with random streaming capabilities shall use application buffers	[SWS_Spi_00053]
[SRS_SPAL_12125]	All driver modules shall only initialize the configured resources	[SWS_Spi_00013]
[SRS_SPAL_12163]	All driver modules shall implement an interface for de-initialization	[SWS_Spi_00021] [SWS_Spi_00022]



△

Requirement	Description	Satisfied by
[SRS_Spi_12025]	The SPI Handler/Driver shall allow the static configuration of all software and hardware properties related to SPI	[SWS_Spi_00052] [SWS_Spi_00053]
[SRS_Spi_12032]	For an SPI channel assigned to an SPI HW Unit the chip select mode "normal" shall be available	[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_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_00034] [SWS_Spi_00041] [SWS_Spi_00135]
[SRS_Spi_12094]	The SPI Handler/Driver shall handle the chip select	[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_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_00065]
[SRS_Spi_12180]	The SPI Driver shall access the SPI bus only for the channel	[SWS_Spi_00003] [SWS_Spi_00065]
[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]

▽

△

Requirement	Description	Satisfied by
[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_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_00034]
[SRS_Spi_12257]	The SPI Handler/Driver shall support the communication to daisy chained HW devices	[SWS_Spi_00034] [SWS_Spi_00065] [SWS_Spi_00066]
[SRS_Spi_12258]	Data shall be accessible from each device individually	[SWS_Spi_00003] [SWS_Spi_00065]
[SRS_Spi_12260]	Different priorities of sequences shall be supported	[SWS_Spi_00002] [SWS_Spi_00014] [SWS_Spi_00059] [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 shall be statically configurable	[SWS_Spi_00109] [SWS_Spi_00111] [SWS_Spi_00121] [SWS_Spi_00122] [SWS_Spi_00125]

Table 6.1: Requirements Tracing

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.

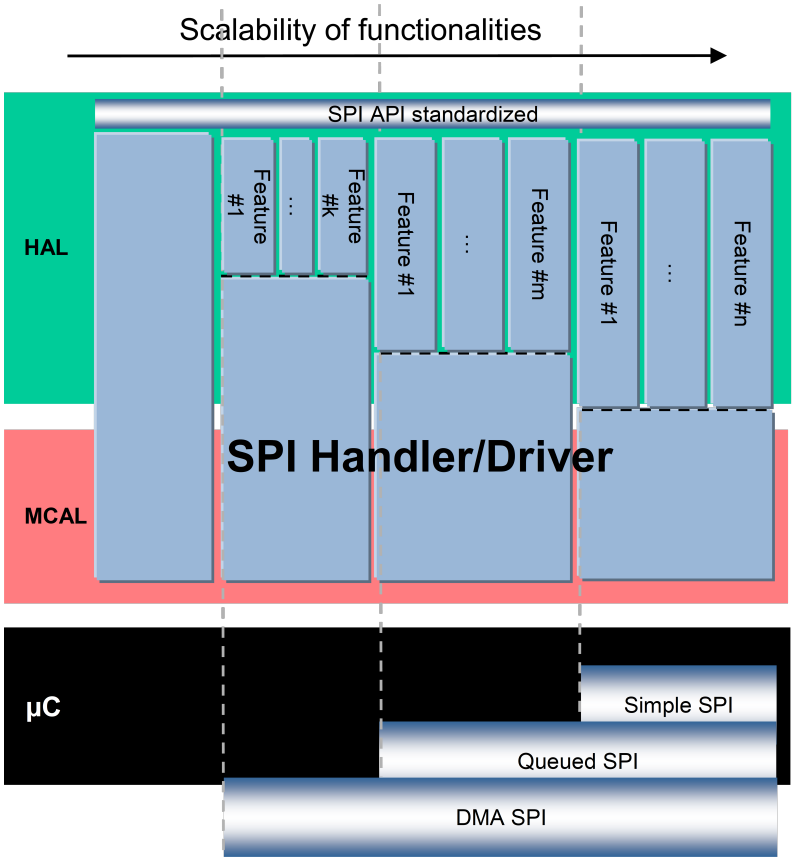


Figure 7.1

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 (using polling mechanism) and 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 (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 "Simple Synchronous" level.

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

Even if notification functions are specified for jobs and/or sequences used in synchronous transmission, these are not called in case of **LEVEL0**.

[SWS_Spi_00109]

Upstream requirements: SRS_Spi_13401

⌈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.⌋

[SWS_Spi_00110]

Upstream requirements: SRS_Spi_13400

⌈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.⌋

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]

Upstream requirements: SRS_Spi_12093

⌈The SPI Handler/Driver interface configuration shall be based on Channels, Jobs and Sequences as defined in this document.⌋

[SWS_Spi_00034]

Upstream requirements: SRS_Spi_12093, SRS_Spi_12256, SRS_Spi_12257

⌈The SPI Handler/Driver shall support one or more Channels, Jobs and Sequences to drive all kind of SPI compatible HW devices.⌋

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

[SWS_Spi_00066]

Upstream requirements: SRS_Spi_12094, SRS_Spi_12257, SRS_Spi_12032, SRS_Spi_12033

⌈The Chip Select (CS) is attached to the Job definition.⌋

[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]).⌋

It is up to the implementation to decide whether the behavior of the chip select configured into `SpiCsBehavior` is applicable when `SpiCsSelection` = `CS_VIA_GPIO`.

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]

Upstream requirements: 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

⌈A Job shall contain at least one Channel.⌋

[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]

Upstream requirements: SRS_Spi_12179, SRS_Spi_12258, SRS_Spi_12180, SRS_Spi_12199, SRS_Spi_12200, SRS_Spi_12261, SRS_Spi_12201, SRS_Spi_12262

⌈A Sequence shall contain at least one Job.⌋

[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]).

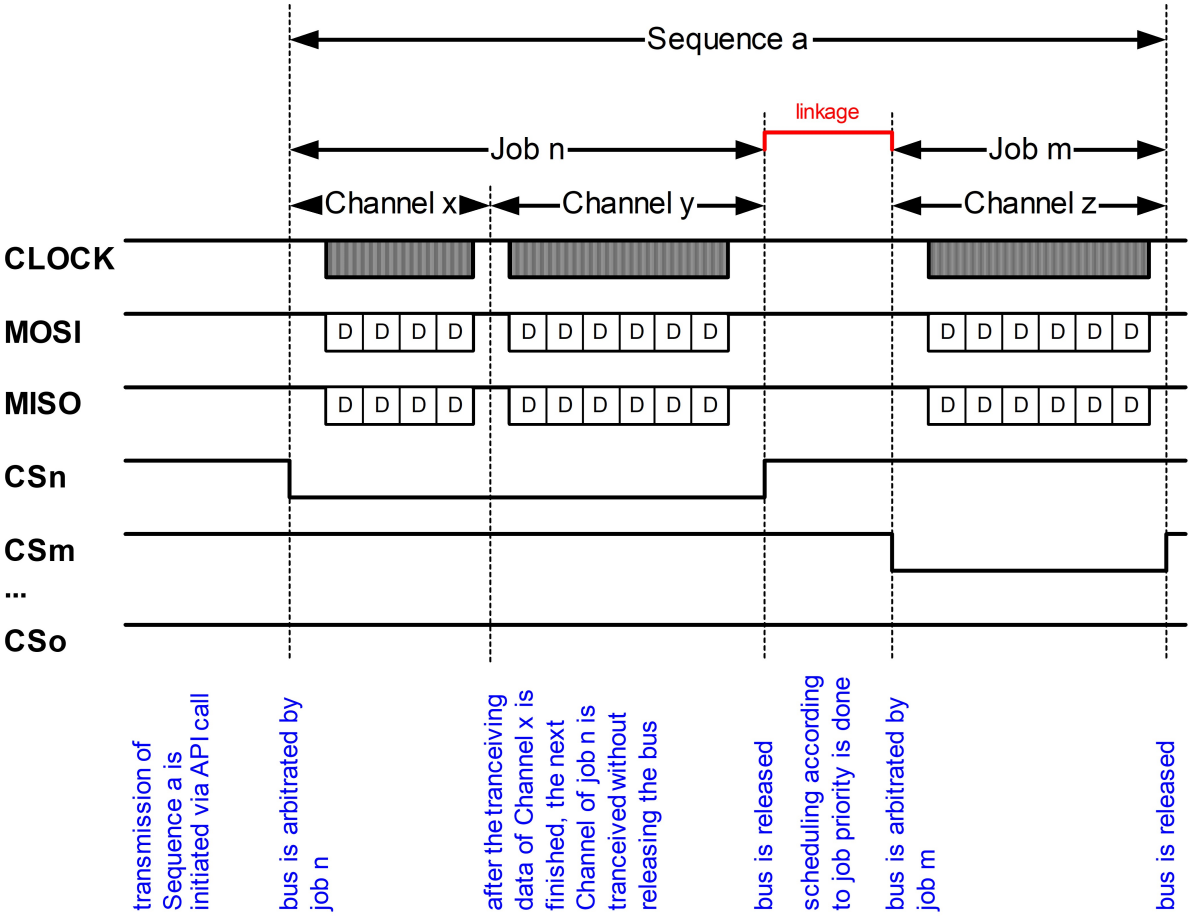


Figure 7.2

Note: the figure above corresponds to a configuration with `SpiCsBehavior`=`CS_KEEP_ASSERTED`.

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]

Upstream requirements: SRS_Spi_13401

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

[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]

Upstream requirements: SRS_Spi_12025, SRS_Spi_12253

⌈For the IB Channels, the Handler/Driver shall provide the buffering but it is not able to take care of the consistency of the data in the buffer during transmission. The size of the Channel buffer is fixed.⌋

[SWS_Spi_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]

Upstream requirements: SRS_SPAL_12075, SRS_Spi_12025, SRS_Spi_12198, SRS_Spi_12200, SRS_Spi_12261, SRS_Spi_12262, SRS_Spi_12202, SRS_Spi_12103

⌈For EB Channels the application shall provide the buffering and shall take care of the consistency of the data in the buffer during transmission.⌋

[SWS_Spi_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.

Note:
For each channel, the user configures the number of IB buffers (at least 1) if IB is selected for the current channel, or the maximum of data for EB buffers if EB is selected for the current channel.

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

SpiSupportConcurrentSyncTransmit 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]

Upstream requirements: SRS_Spi_12170

⌈If different Jobs (and consequently also Sequences) have common Channels, the SPI Handler/Driver' environment shall ensure that read and/or write functions are not called during transmission.⌋

[SWS_Spi_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.

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_00156] ⌈Both the polling mechanism and interrupt mechanism modes for SPI busses shall be selectable during execution time (see [SWS_Spi_00188]).⌋

[SWS_Spi_00162]

Upstream requirements: SRS_Spi_12099, SRS_Spi_12101, SRS_Spi_12103

⌈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.⌋

[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]

Upstream requirements: SRS_Spi_12099, SRS_Spi_12101, SRS_Spi_12103

⌈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).⌋

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]

Upstream requirements: SRS_Spi_12260

⌈Jobs have priorities assigned. Jobs linked in a Sequence shall have same or decreasing priorities. That means the first Job shall have the equal priority or the highest priority of all Jobs within the Sequence.⌋

[SWS_Spi_00093]

Upstream requirements: SRS_Spi_12260, SRS_Spi_12150

⌈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.⌋

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

[SWS_Spi_00059]

Upstream requirements: SRS_Spi_12260, SRS_Spi_12037

⌈The SPI Handler/Driver scheduling method shall schedule Jobs in order to send the highest priority Job first.⌋

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:

[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]

Upstream requirements: SRS_Spi_12108

⌈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.⌋

[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]

Upstream requirements: SRS_Spi_12108

⌈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).⌋

[SWS_Spi_00120]

Upstream requirements: SRS_Spi_12108

⌈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_CANCELED (rational: avoid deadlocks or endless loops).⌋

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]

Upstream requirements: SRS_Spi_13401

⌈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.⌋

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]

Upstream requirements: SRS_Spi_13401

⌈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").⌋

[SWS_Spi_00123] ⌈When the SPI Handler/Driver is configured not allowing interruptible Sequences, all Sequences declared are considered as Non-Interruptible Sequences[1].⌋

[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]

Upstream requirements: SRS_Spi_12037

⌈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.⌋

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.

[1] 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_00125]

Upstream requirements: SRS_Spi_13401

⌈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").⌋

[SWS_Spi_00126] ⌈When the SPI Handler/Driver is configured allowing interruptible Sequences, all Sequences declared shall have their dedicated parameter `SpiInterruptibleSequence` (see [ECUC_Spi_00106]) to identify whether the Sequence can be suspended during transmission.⌋

[SWS_Spi_00014]

Upstream requirements: SRS_Spi_12260, SRS_Spi_12037

⌈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 rescheduling in order to elect the next Job to transmit.⌋

[SWS_Spi_00127]

Upstream requirements: SRS_Spi_12037

⌈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.⌋

[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.

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.

The requirements from LEVEL 0 apply to synchronous behaviour.

The requirements from LEVEL 1 apply to asynchronous behaviour.

[SWS_Spi_00128] ⌈The LEVEL 2 SPI Handler/Driver shall offer both an asynchronous transfer service and a synchronous transfer service for 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).⌋

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.⌋

7.4 Error Classification

Chapter [2, General Specification of Basic Software Modules] 7.2 *"Error Handling"* describes the error handling of the Basic Software in detail. Above all, it constitutes a classification scheme consisting of following error types which may occur in BSW modules.

Based on this foundation, the following section specifies particular errors arranged in the respective subsections below.

7.4.1 Development Errors

[SWS_Spi_91001] Definition of development errors in module Spi ⌈

Type of error	*Related error code*	*Error value*
API service called with wrong channel	SPI_E_PARAM_CHANNEL	0x0A
API service called with wrong job	SPI_E_PARAM_JOB	0x0B
API service called with wrong sequence	SPI_E_PARAM_SEQ	0x0C
API service called with wrong lenghth for EB	SPI_E_PARAM_LENGTH	0x0D
API service called with wrong hardware unit	SPI_E_PARAM_UNIT	0x0E
APIs called with an unexpected value for the 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 been already initialized	SPI_E_ALREADY_INITIALIZED	0x4A

⌋

7.4.2 Runtime Errors

[SWS_Spi_91002] Definition of runtime errors in module Spi ⌈

Type of error	*Related error code*	*Error value*
API Spi_AsyncTransmit service called in a wrong order	SPI_E_SEQ_PENDING	0x2A
API Spi_SyncTransmit service called at wrong time	SPI_E_SEQ_IN_PROCESS	0x3A

⌋

7.4.3 Production Errors

7.4.3.1 SPI_E_HARDWARE_ERROR

[SWS_Spi_00383] A hardware error occurred during asynchronous or synchronous SPI transmit.

Upstream requirements: SRS_BSW_00458

⌈

Diagnostic Event (Error Name)	SPI_E_HARDWARE_ERROR
Description	This Production Error shall be issued when any error bit inside the SPI hardware transmit status register is raised.
Failed condition	The SPI transmit status register information shall be reported to DEM as Dem_Set EventStatus (SPI_E_HARDWARE_ERROR, DEM_EVENT_STATUS_FAILED) when any error bit inside the SPI transmit status register is set.(SWS_Spi_00385).

▽

△

Passed condition	The SPI transmit status register information shall be reported to DEM as Dem_Set EventStatus (SPI_E_HARDWARE_ERROR, DEM_EVENT_STATUS_PASSED) when no error bit inside the SPI transmit status register is set.(SWS_Spi_00386).

⌋

[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 Security Events

The module does not report security events.

8 API specification

8.1 Imported types

In this chapter all types included from the following files are listed.

[SWS_Spi_91003] Definition of imported datatypes of module Spi ⌈

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

⌋

8.2 Type definitions

8.2.1 Spi_ConfigType

[SWS_Spi_00372] Definition of datatype Spi_ConfigType ⌈

Name	Spi_ConfigType	
Kind	Structure	
Elements	Implementation Specific	
	Type	–
	Comment	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	

⌋

8.2.2 Spi_StatusType

[SWS_Spi_00373] Definition of datatype Spi_StatusType

Upstream requirements: SRS_BSW_00406, SRS_BSW_00335

⌈

Name	Spi_StatusType		
Kind	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		

⌋

[SWS_Spi_00061]

Upstream requirements: SRS_BSW_00335

⌈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.⌋

[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_BUSY`when The SPI Hardware microcontroller peripheral is performing a SPI Job transmit.⌋

8.2.3 Spi_JobResultType

[SWS_Spi_00374] Definition of datatype Spi_JobResultType ⌈

Name	Spi_JobResultType		
Kind	Enumeration		
Range	SPI_JOB_OK	0x00	The last transmission of the Job has been finished successfully.

▽

△

	SPI_JOB_PENDING	0x01	The SPI Handler/Driver is performing a SPI Job. The meaning of this status is equal to SPI_BUSY.
	SPI_JOB_FAILED	0x02	The last transmission of the Job has failed.
	SPI_JOB_QUEUED	0x03	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]

Upstream requirements: SRS_BSW_00335

⌈The type `Spi_JobResultType` defines a range of specific Jobs status for SPI Handler/Driver.⌋

[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] Definition of datatype Spi_SeqResultType ⌈

Name	Spi_SeqResultType		
Kind	Enumeration		
Range	SPI_SEQ_OK	0x00	The last transmission of the Sequence has been finished successfully.
	SPI_SEQ_PENDING	0x01	The SPI Handler/Driver is performing a SPI Sequence. The meaning of this status is equal to SPI_BUSY.
	SPI_SEQ_FAILED	0x02	The last transmission of the Sequence has failed.
	SPI_SEQ_CANCELED	0x03	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]

Upstream requirements: SRS_BSW_00335

⌈The type Spi_SeqResultType defines the range of specific Sequences status for SPI Handler/Driver.⌋

[SWS_Spi_00251] ⌈The type Spi_SeqResultType defines about SPI Handler/-Driver Sequence status and can be obtained calling the API service Spi_Get-SequenceResult 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] Definition of datatype Spi_DataBufferType ⌈

Name	Spi_DataBufferType
Kind	Type
Derived from	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] Definition of datatype Spi_NumberOfDataType ⌈

Name	Spi_NumberOfDataType
Kind	Type
Derived from	uint16
Description	Type for defining the number of data elements 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] Definition of datatype Spi_ChannelType ⌈

Name	Spi_ChannelType
Kind	Type
Derived from	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] Definition of datatype Spi_JobType ⌈

Name	Spi_JobType
Kind	Type
Derived from	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] Definition of datatype Spi_SequenceType ⌈

Name	Spi_SequenceType
Kind	Type
Derived from	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] Definition of datatype Spi_HWUnitType ⌈

Name	Spi_HWUnitType
Kind	Type
Derived from	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] Definition of datatype Spi_AsyncModeType ⌈

Name	Spi_AsyncModeType		
Kind	Enumeration		
Range	SPI_POLLING_MODE	0x00	The asynchronous mechanism is ensured by polling, so interrupts related to SPI busses handled asynchronously are disabled.
	SPI_INTERRUPT_MODE	0x01	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.		
Available via	Spi.h		

⌋

[SWS_Spi_00360] ⌈The asynchronous mechanism is selected 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.⌋

[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 1 and 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

This chapter contains all APIs provided by the module.

8.3.1 Spi_Init

[SWS_Spi_00175] Definition of API function Spi_Init ⌈

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 (inout)	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]

Upstream requirements: SRS_BSW_00405, SRS_BSW_00101, SRS_SPAL_12057, SRS_SPAL_12125

⌈The function `Spi_Init` shall initialize all SPI relevant registers with the values of the structure referenced by the parameter `ConfigPtr`.⌋

[SWS_Spi_00015]

Upstream requirements: SRS_BSW_00406, SRS_BSW_00101, SRS_SPAL_12057

⌈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`.⌋

[SWS_Spi_00151] ⌈For LEVEL 2, 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 8.6.1.

8.3.2 Spi_DeInit

[SWS_Spi_00176] Definition of API function Spi_DeInit ⌈

Service Name	Spi_DeInit	
Syntax	`Std_ReturnType Spi_DeInit ( void )`	
Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	None	
Parameters (inout)	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]

Upstream requirements: SRS_BSW_00336, SRS_SPAL_12163, SRS_SPAL_12064

⌈The function `Spi_DeInit` shall de-initialize SPI Handler/Driver.⌋

[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]

Upstream requirements: SRS_BSW_00336, SRS_SPAL_12163

⌈After the module de-initialization using the function `Spi_DeInit`, the SPI Handler/-Driver shall set its state to `SPI_UNINIT`.⌋

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] Definition of API function Spi_WriteIB ⌈

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 (inout)	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]

Upstream requirements: SRS_Spi_12101, SRS_Spi_12153

⌈The function Spi_WriteIB shall write one or more data to an IB SPI Handler/Driver Channel specified by the respective parameter.⌋

[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 8.6.1.

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] Definition of API function Spi_AsyncTransmit ⌈

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 (inout)	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]

Upstream requirements: SRS_Spi_12099, SRS_Spi_12101, SRS_Spi_12103

⌈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.⌋

[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 re-quest 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]

Upstream requirements: SRS_Spi_12200, SRS_Spi_12201

⌈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].)⌋

[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]

Upstream requirements: SRS_Spi_12181

⌈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.⌋

[SWS_Spi_00057]

Upstream requirements: SRS_SPAL_00157, SRS_Spi_12108

⌈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 callback function after the last Job end notification if this one is also configured.⌋

[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 8.6.1.

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] Definition of API function Spi_ReadIB ⌈

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 (inout)	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]

Upstream requirements: SRS_Spi_12099, SRS_Spi_12152

⌈The function `Spi_ReadIB` shall read synchronously one or more data from an IB SPI Handler/Driver Channel specified by the respective parameter.⌋

[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 8.6.1.

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] Definition of API function Spi_SetupEB ⌈

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 (inout)	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]

Upstream requirements: SRS_Spi_12103

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

[SWS_Spi_00067]

Upstream requirements: SRS_Spi_12103

⌈The function `Spi_SetupEB` shall update the buffer pointers and length attributes of the specified Channel with the provided values.⌋

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 8.6.1.

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] Definition of API function Spi_GetStatus ⌈

Service Name	Spi_GetStatus	
Syntax	`Spi_StatusType Spi_GetStatus ( void )`	
Service ID [hex]	0x06	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	None	
Parameters (inout)	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]

Upstream requirements: SRS_SPAL_12064, SRS_Spi_12104

⌈The function `Spi_GetStatus` shall return the SPI Handler/Driver software module status.⌋

8.3.8 Spi_GetJobResult

[SWS_Spi_00182] Definition of API function Spi_GetJobResult ⌈

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 (inout)	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]

Upstream requirements: SRS_SPAL_00157, SRS_Spi_12104

⌈The function `Spi_GetJobResult` shall return the last transmission result of the specified Job.⌋

[SWS_Spi_00038]

Upstream requirements: SRS_SPAL_00157

⌈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`).⌋

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 8.6.1.

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] Definition of API function Spi_GetSequenceResult ⌈

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 (inout)	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]

Upstream requirements: SRS_SPAL_00157, SRS_Spi_12104

⌈The function Spi_GetSequenceResult shall return the last transmission result of the specified Sequence.⌋

[SWS_Spi_00042]

Upstream requirements: SRS_SPAL_00157, SRS_Spi_12170

⌈The SPI Handler/Driver's environment shall call the function Spi_GetSequence-Result to inquire whether the full Sequence transmission has succeeded (SPI_SEQ_OK) or failed (SPI_SEQ_FAILED).⌋

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_Get-SequenceResult is called, the return value is undefined.

Parameters of the function Spi_GetSequenceResult shall be checked as it is explained in section 8.6.1.

8.3.10 Spi_SyncTransmit

[SWS_Spi_00185] Definition of API function Spi_SyncTransmit ⌈

Service Name	Spi_SyncTransmit	
Syntax	`Std_ReturnType Spi_SyncTransmit ( Spi_SequenceType Sequence )`	
Service ID [hex]	0x0a	
Sync/Async	Synchronous	
Reentrancy	Reentrant	
Parameters (in)	Sequence	Sequence ID.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	`E_OK`: Transmission has been successful `E_NOT_OK`: Transmission failed
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] ⌈The function `Spi_SyncTransmit` returns `E_OK` if the transmission request has been successful.⌋

[SWS_Spi_00329] ⌈The function `Spi_SyncTransmit` returns `E_NOT_OK` if the transmission request failed.⌋

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

[SWS_Spi_00134]

Upstream requirements: SRS_Spi_12152, SRS_Spi_12153, SRS_Spi_12154

⌈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`.⌋

[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]

Upstream requirements: SRS_Spi_12093

⌈When the function `Spi_SyncTransmit` is called while a sequence is on transmission and `SpiSupportConcurrentSyncTransmit` 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].⌋

[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 8.6.1.

8.3.11 Spi_GetHWUnitStatus

[SWS_Spi_00186] Definition of API function Spi_GetHWUnitStatus ⌈

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 (inout)	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 8.6.1.

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

8.3.12 Spi_Cancel

[SWS_Spi_00187] Definition of API function Spi_Cancel ⌈

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 (inout)	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_CANCELED.⌋

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!

8.3.13 Spi_SetAsyncMode

[SWS_Spi_00188] Definition of API function Spi_SetAsyncMode ⌈

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 (inout)	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_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 1 and 2.⌋

8.3.14 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.

8.4 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.

8.4.1 Spi_MainFunction_Handling

[SWS_Spi_00189] Definition of scheduled function Spi_MainFunction_Handling ⌈

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.

8.5 Expected interfaces

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

8.5.1 Mandatory interfaces

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

[SWS_Spi_00389] Definition of mandatory interfaces required by module Spi ⌈

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.

⌋

8.5.2 Optional interfaces

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

[SWS_Spi_00339] Definition of optional interfaces requested by module Spi ⌈

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. This API will be available only if ({Dem/Dem ConfigSet/DemEventParameter/DemEvent ReportingType} == STANDARD_REPORTING)
Det_ReportError	Det.h	Service to report development errors.

⌋

8.5.3 Callout interfaces

No callout interfaces.

8.5.4 Configurable interfaces

In this section, all interfaces are listed where the target function could be configured. The target function is usually a callback function. The names of this kind of interfaces are not fixed because they are configurable.

[SWS_Spi_00075]

Upstream requirements: SRS_SPAL_00157

⌈The SPI Handler/Driver shall use the callback routines `Spi_JobEndNotification` to inform other software modules about certain states or state changes.⌋

[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]

Upstream requirements: SRS_SPAL_12056

⌈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`).⌋

[SWS_Spi_00048]

Upstream requirements: SRS_BSW_00359, SRS_BSW_00360, SRS_BSW_00369

⌈The callback notifications `Spi_JobEndNotification` and `Spi_SeqEndNotification` shall have no parameters and no return value.⌋

[SWS_Spi_00054]

Upstream requirements: SRS_SPAL_12056

⌈If a callback notification is configured as null pointer, no callback shall be executed.⌋

[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.⌋

8.5.4.1 Spi_JobEndNotification

[SWS_Spi_00192] Definition of configurable interface (*Spi_JobEndNotification) ⌈

Service Name	(*Spi_JobEndNotification)
Syntax	`void (*Spi_JobEndNotification) ( void )`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (inout)	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]

Upstream requirements: SRS_SPAL_00157

⌈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.⌋

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

8.5.4.2 Spi_SeqEndNotification

[SWS_Spi_00193] Definition of configurable interface (*Spi_SeqEndNotification) ⌈

Service Name	(*Spi_SeqEndNotification)
Syntax	`void (*Spi_SeqEndNotification) ( void )`
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (inout)	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]

Upstream requirements: SRS_SPAL_00157

⌈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.⌋

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

8.6 Error detection

8.6.1 API parameter checking

[SWS_Spi_00004]

Upstream requirements: SRS_BSW_00327, SRS_BSW_00337, SRS_BSW_00385

⌈SPI Handler/driver shall be able to detect the error SPI_E_PARAM_CHANNEL when API service called with wrong parameter.⌋

[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_00031]

Upstream requirements: SRS_BSW_00323

⌈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.⌋

[SWS_Spi_00032]

Upstream requirements: SRS_BSW_00323

⌈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.⌋

[SWS_Spi_00060]

Upstream requirements: SRS_BSW_00323

⌈The API parameter Length of data shall have a value within the specified buffer maximum value. Related error value: SPI_E_PARAM_LENGTH.⌋

[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`.⌋

8.6.2 SPI state checking

[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_00046]

Upstream requirements: SRS_BSW_00406

⌈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.⌋

[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_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.⌋

8.6.3 SPI runtime checking

[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_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].⌋

[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_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`).⌋

9 Sequence diagrams

9.1 Initialization

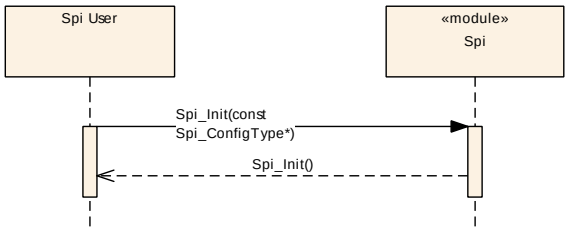


Figure 9.1

9.2 Modes transitions

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

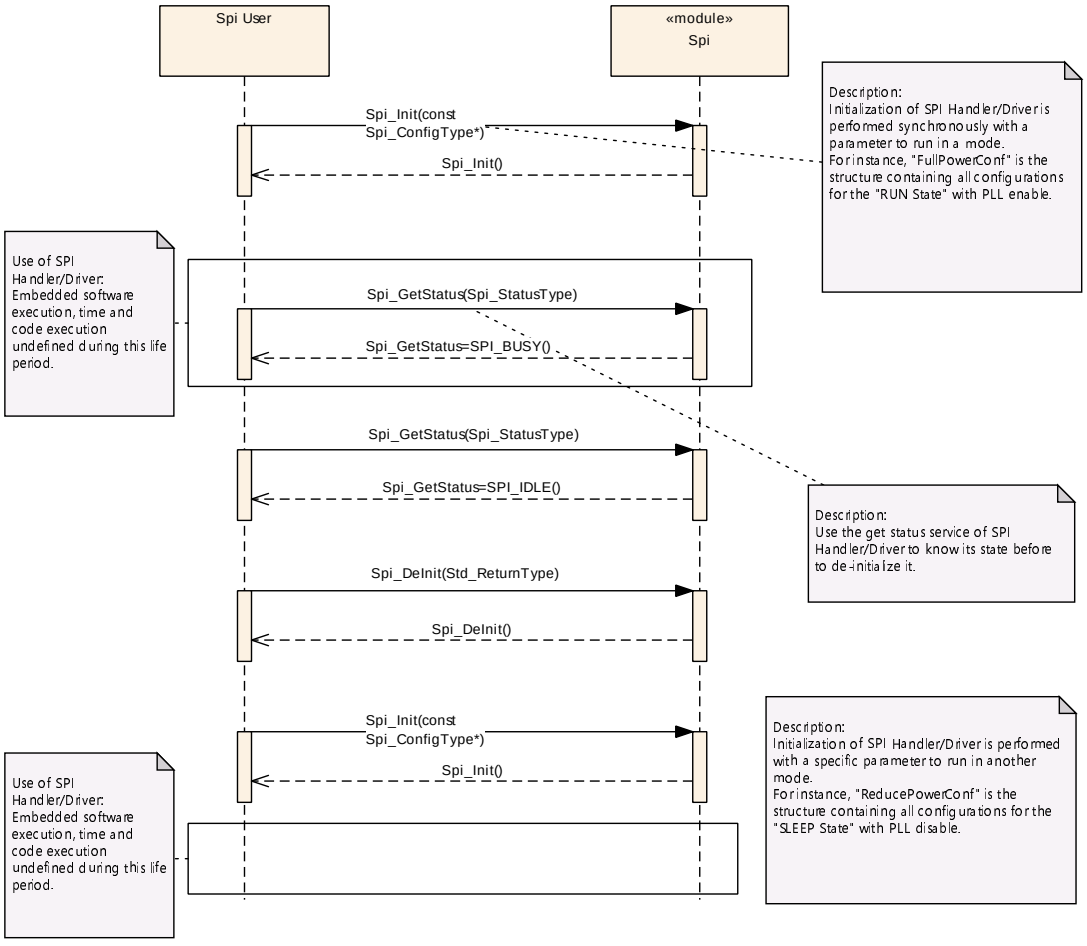


Figure 9.2

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

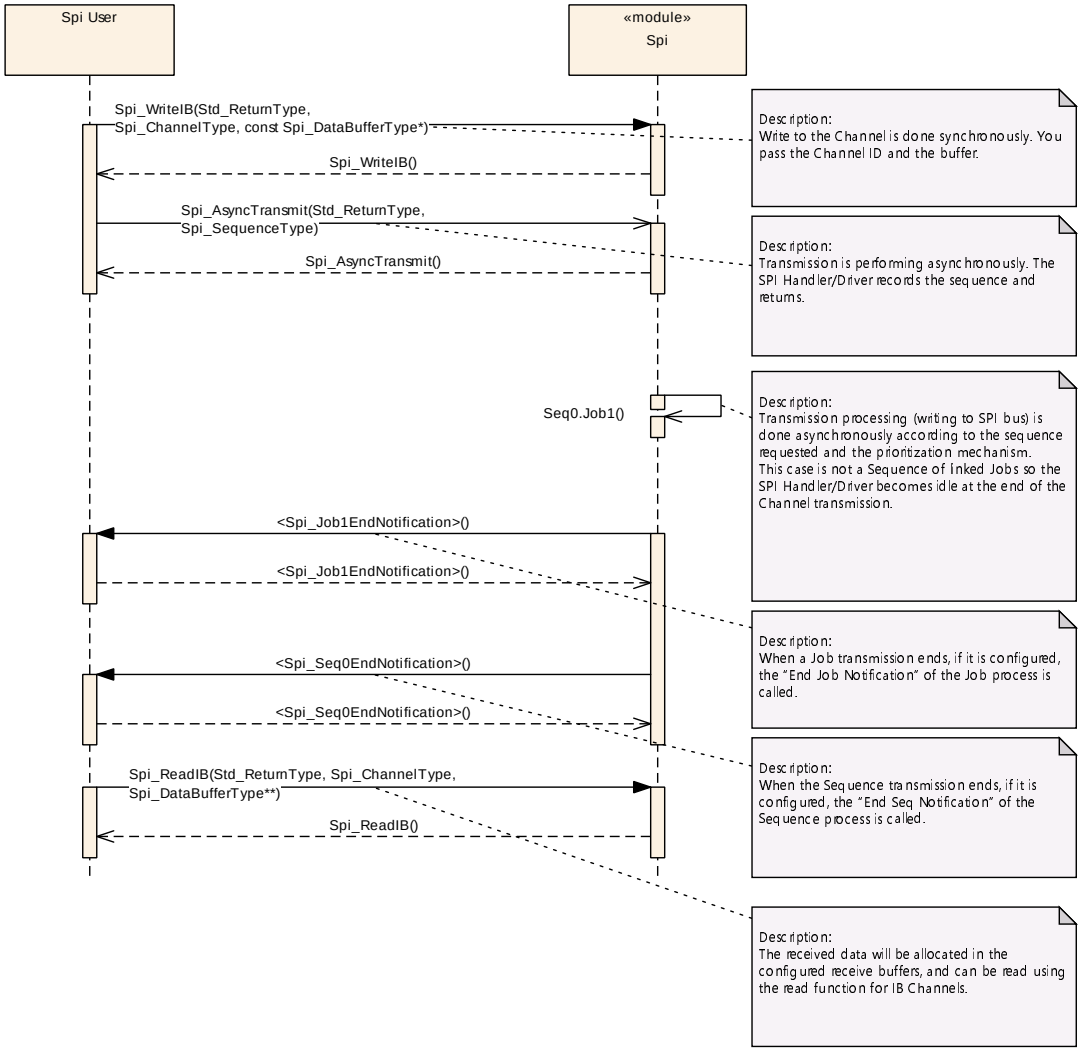


Figure 9.3

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

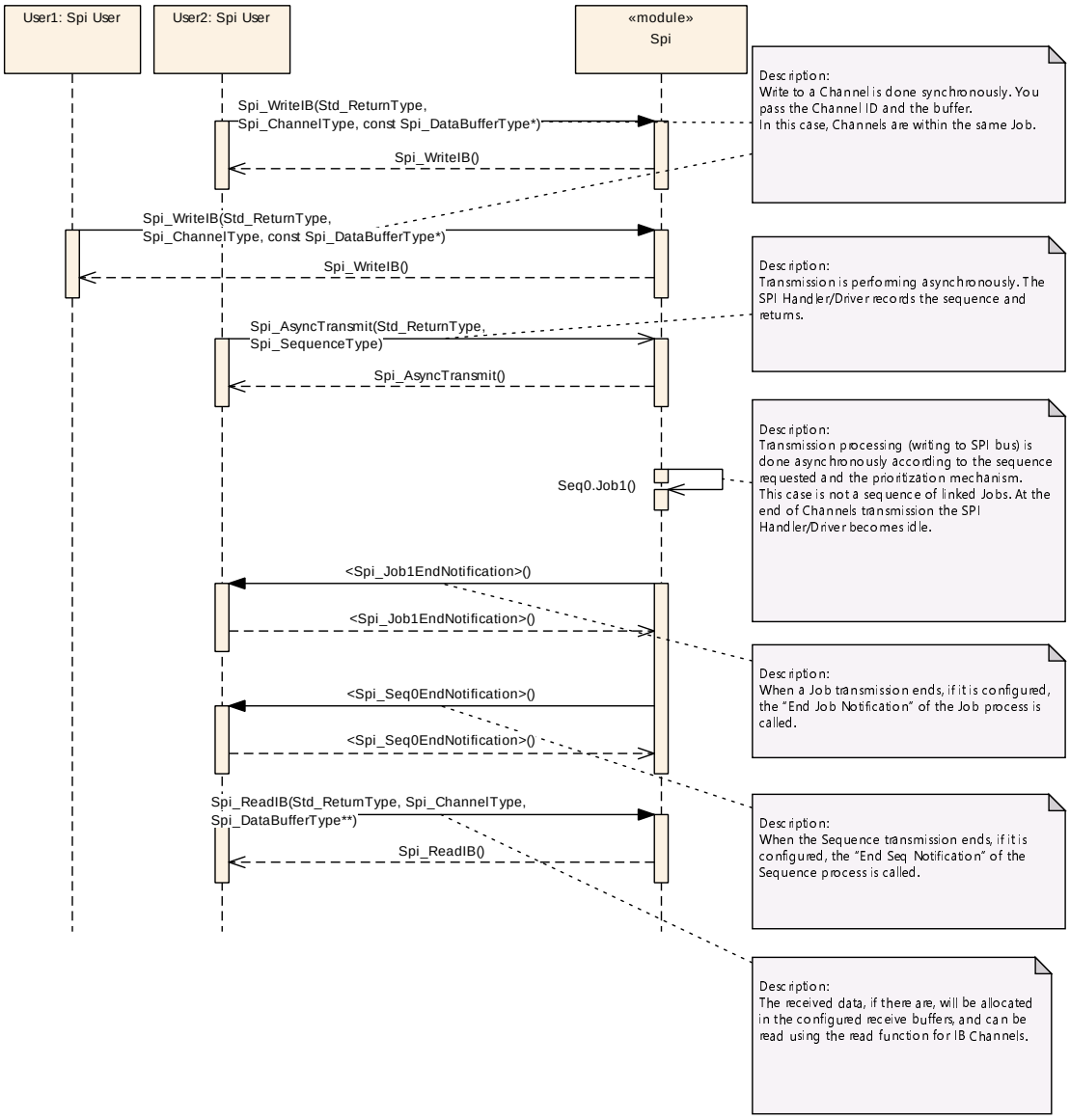


Figure 9.4

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

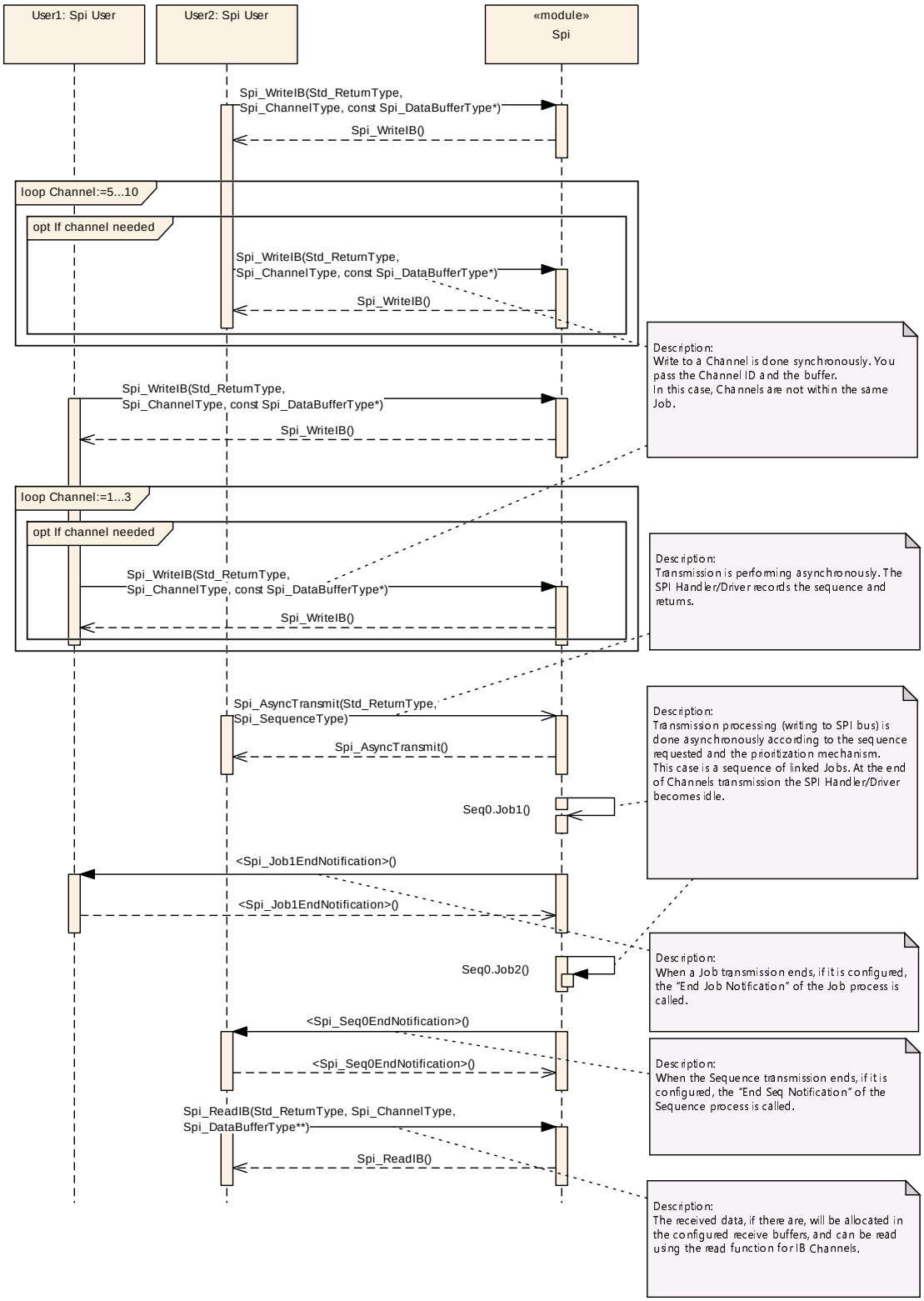

User1: Spi User
User2: Spi User
«module»
Spi
Spi_WriteIB(Std_ReturnType,
Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
loop Channel:=5...10
opt If channel needed
Spi_WriteIB(Std_ReturnType,
Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
Description:
Write to a Channel is done synchronously. 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.

Figure 9.5

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

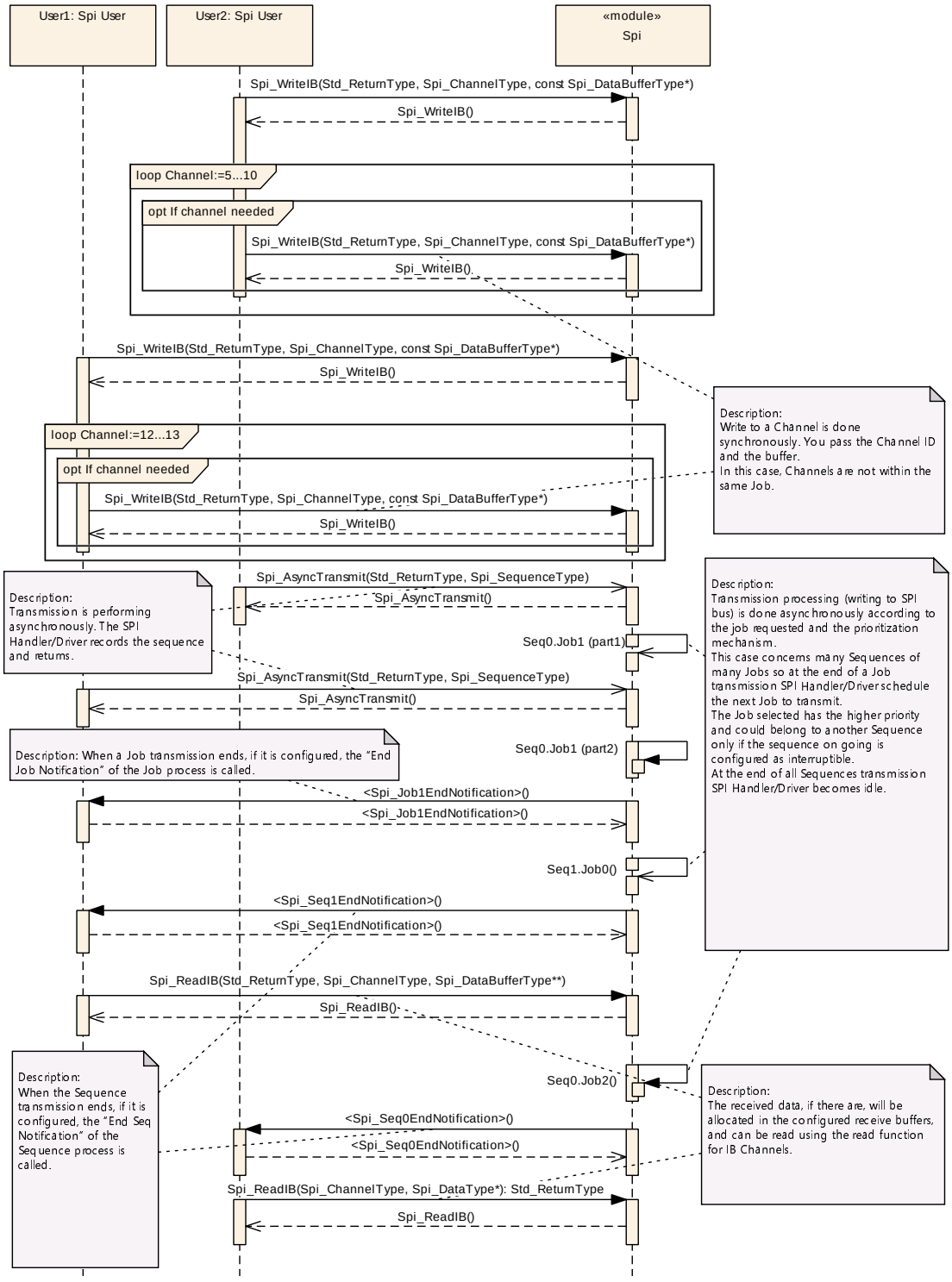

User1: Spi User
User2: Spi User
«module»
Spi
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
loop Channel:=5...10
opt If channel needed
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataBufferType*)
Spi_WriteIB()
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.
loop Channel:=12...13
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 (part1)
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_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Description: When a Job transmission ends, if it is configured, the "End Job Notification" of the Job process is called.
Seq0.Job1 (part2)
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
Seq1.Job0()
<Spi_Seq1EndNotification>()
<Spi_Seq1EndNotification>()
Spi_ReadIB(Std_ReturnType, Spi_ChannelType, Spi_DataBufferType**)
Spi_ReadIB()
Description:
When the Sequence transmission ends, if it is configured, the "End Seq Notification" of the Sequence process is called.
Seq0.Job2()
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.
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()
Spi_ReadIB(Spi_ChannelType, Spi_DataType*): Std_ReturnType
Spi_ReadIB()

Figure 9.6

9.4 Setup/AsyncTransmit (EB)

9.4.1 Variable Number of Data / Constant Number of Data

[SWS_Spi_00077]

Upstream requirements: SRS_Spi_12198, SRS_Spi_12200, SRS_Spi_12201

⌈To transmit a variable number of data, it is mandatory to call the `Spi_SetupEB` function to store new parameters within SPI Handler/Driver before each `Spi_AsyncTransmit` function call.⌋

[SWS_Spi_00078]

Upstream requirements: SRS_Spi_12253, SRS_Spi_12262, SRS_Spi_12202

⌈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.⌋

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

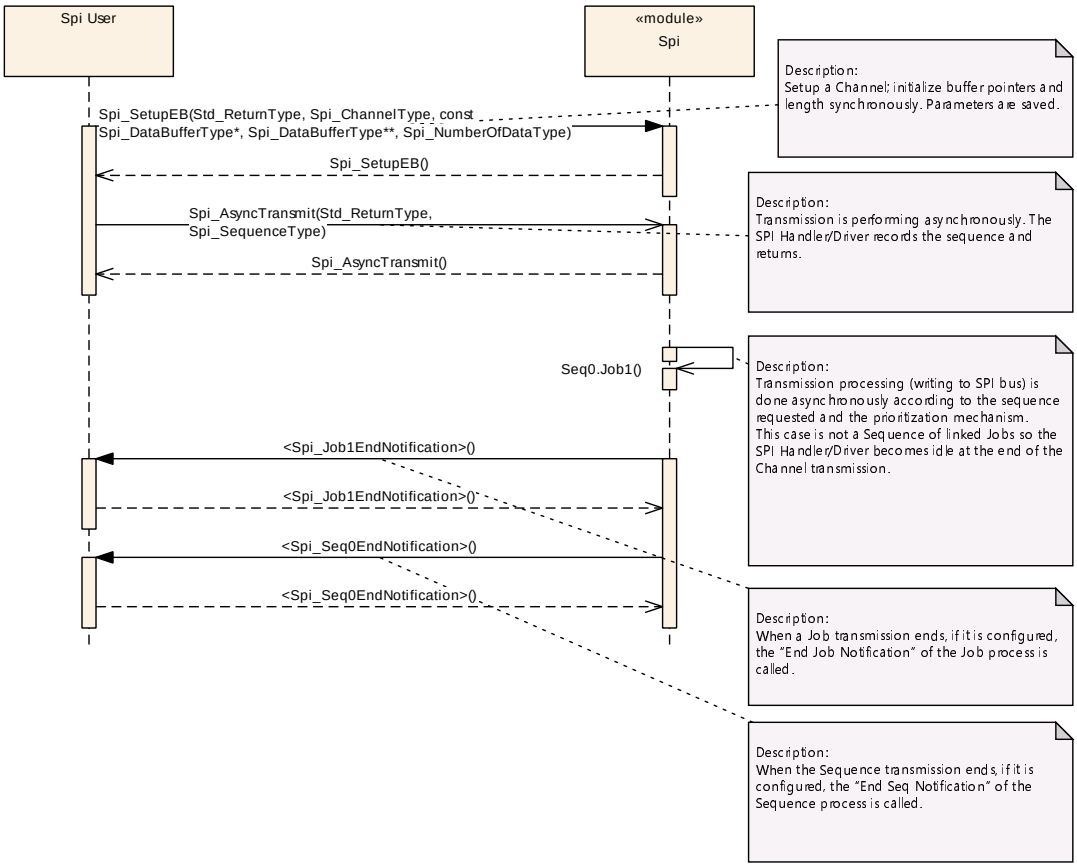


Figure 9.7

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

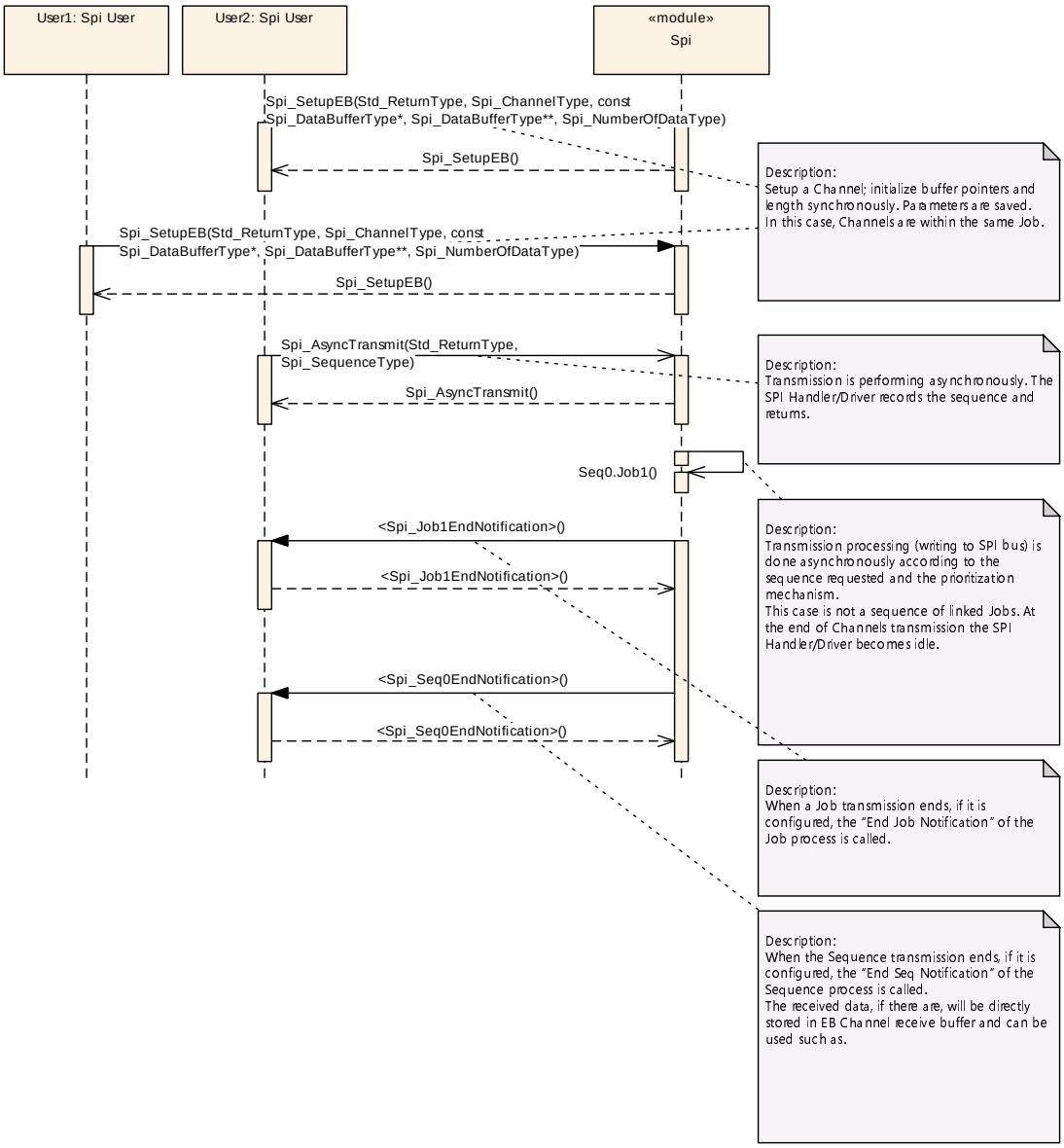


Figure 9.8

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

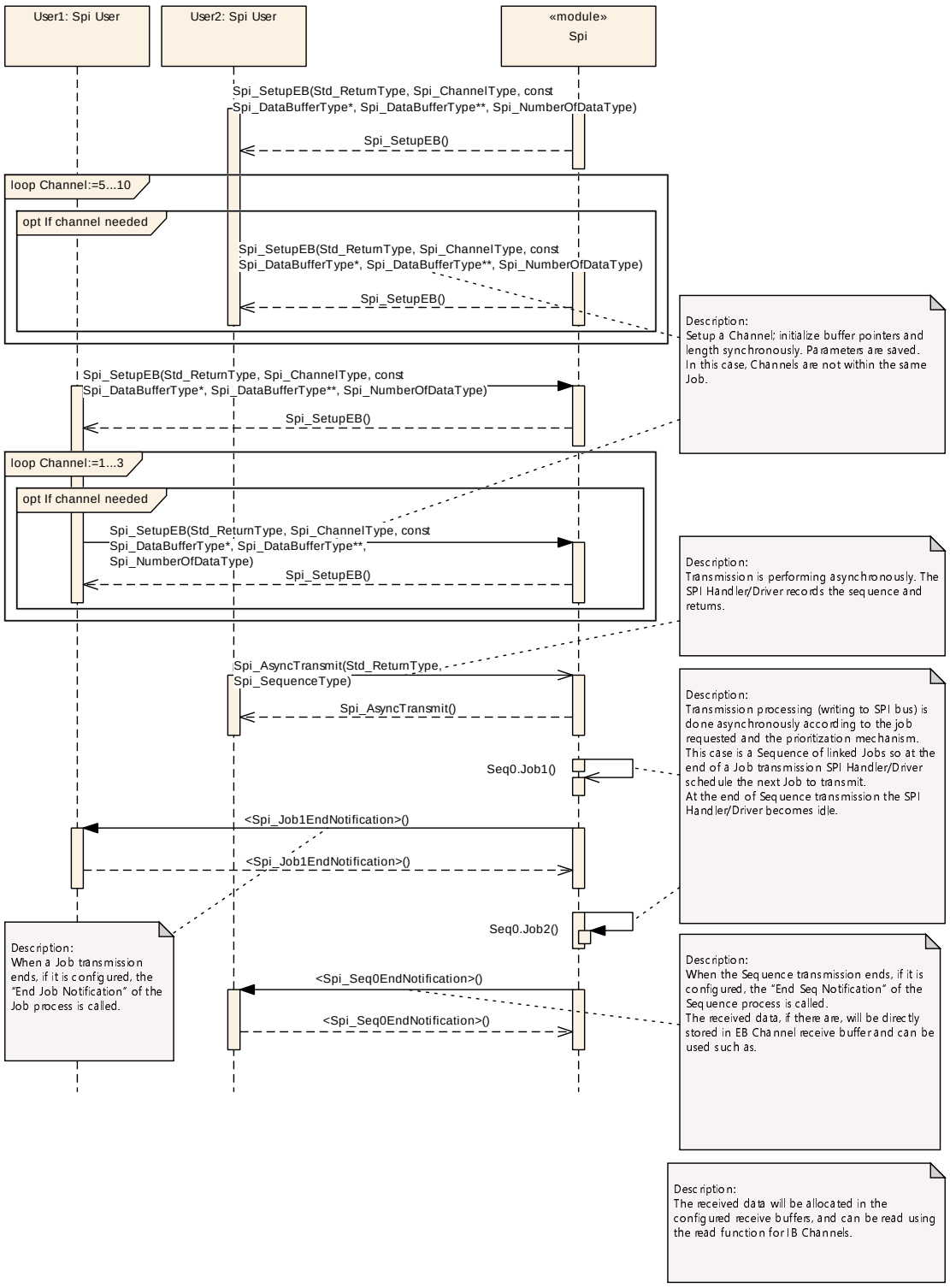


Figure 9.9

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

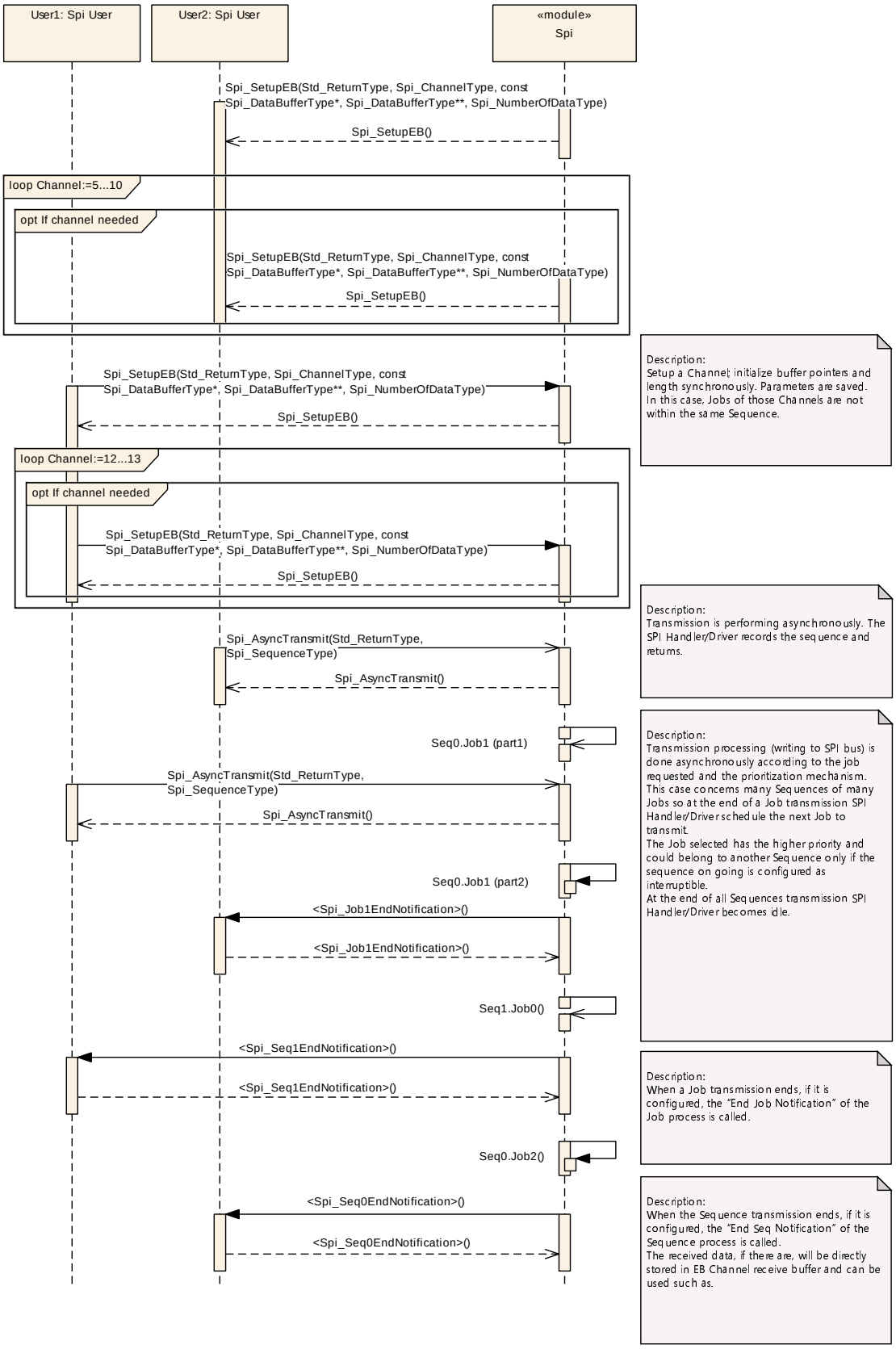

User1: Spi User
User2: Spi User
«module»
Spi
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const
Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
loop Channel:=5...10
opt If channel needed
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const
Spi_DataBufferType*, Spi_DataBufferType**, 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.

Figure 9.10

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

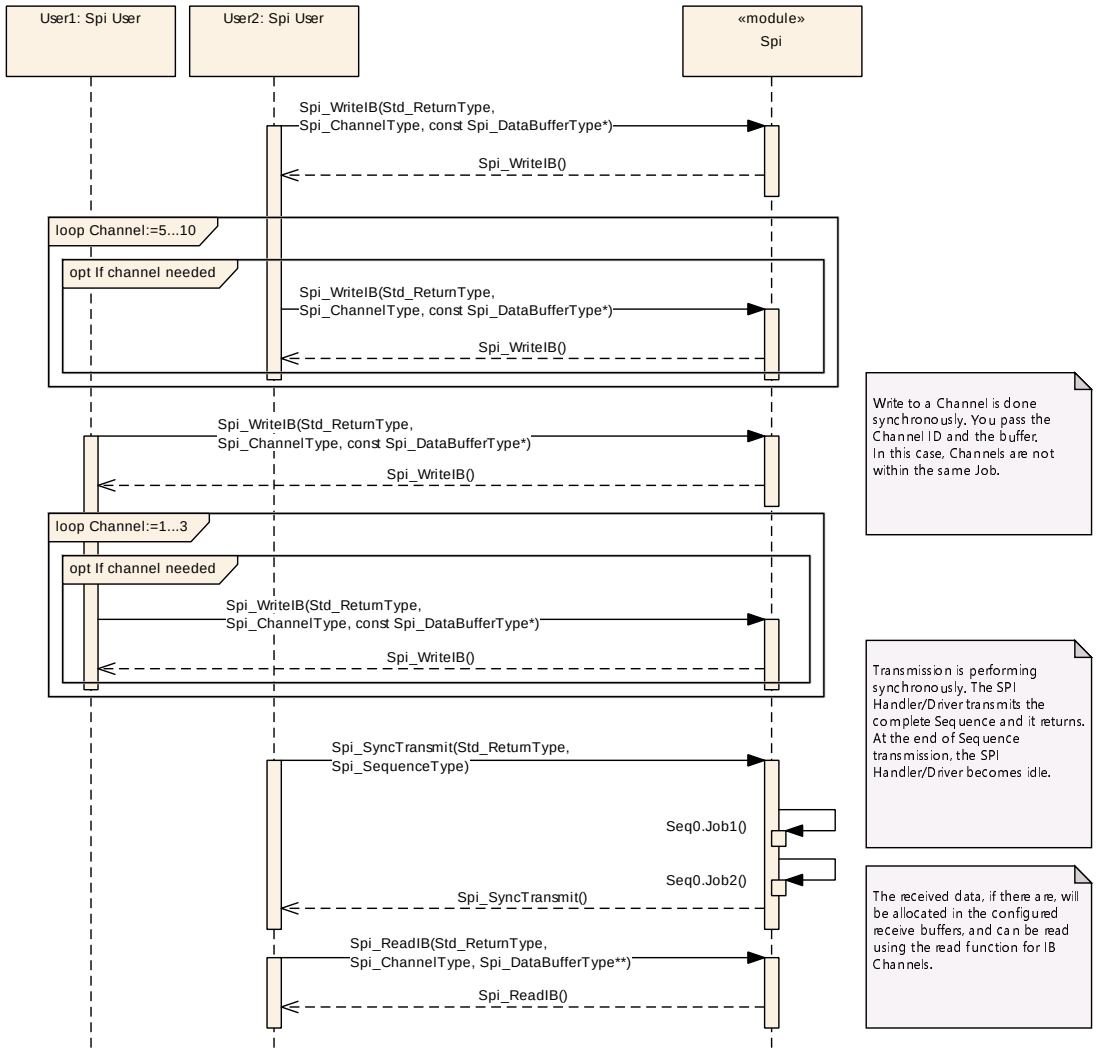


Figure 9.11

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

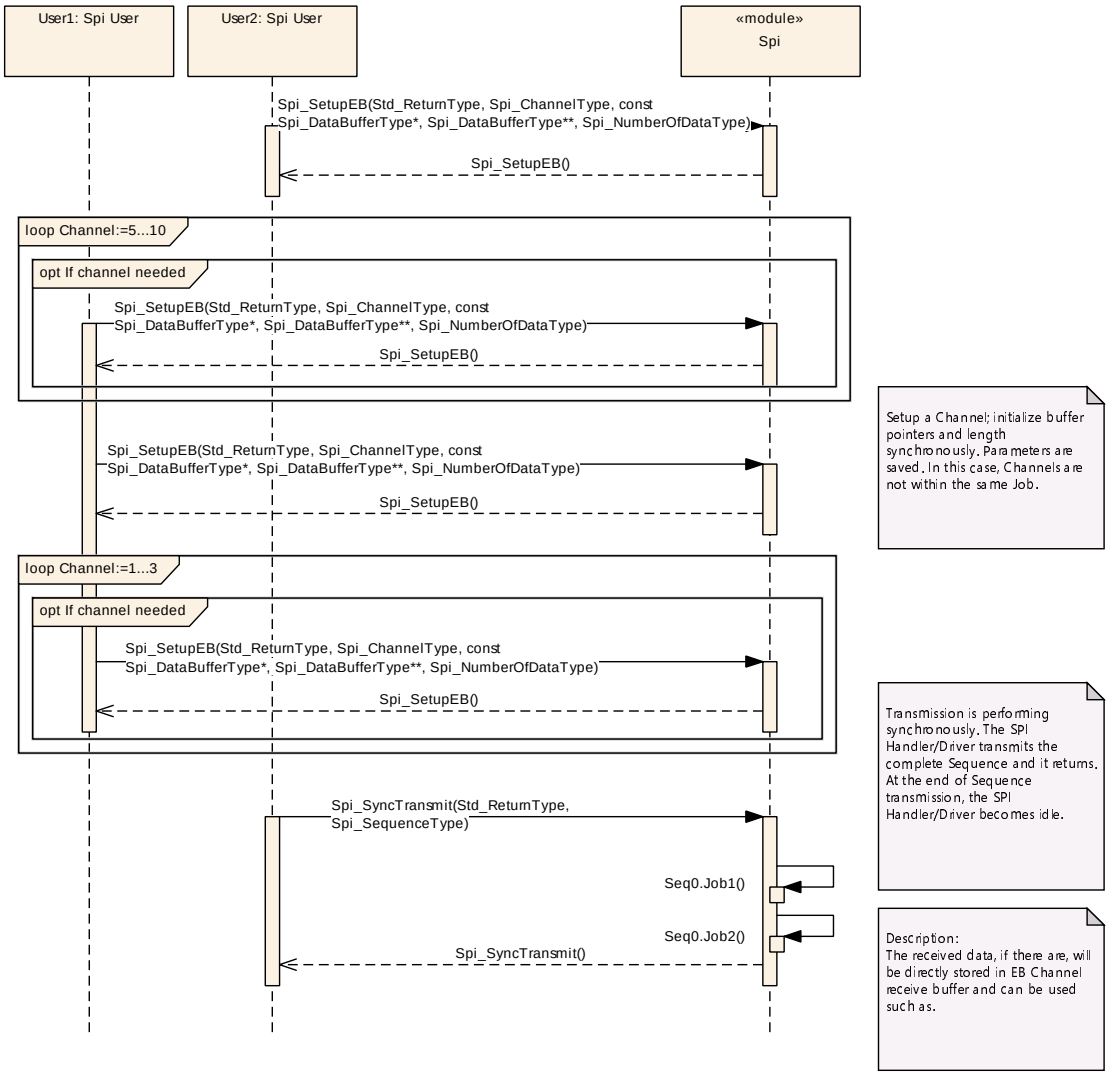

User1: Spi User
User2: Spi User
«module»
Spi
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const
Spi_DataBufferType*, Spi_DataBufferType**, Spi_NumberOfDataType)
Spi_SetupEB()
loop Channel:=5...10
opt If channel needed
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const
Spi_DataBufferType*, Spi_DataBufferType**,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

Figure 9.12

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module SPI Handler/Driver.

Chapter 10.3 specifies published information of the module SPI Handler/Driver.

10.1 How to read this chapter

For details refer to [2] Chapter 10.1 *"Introduction to configuration specification"*.

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters describe Chapter 7 and Chapter 8.

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

10.2.1 Spi

[ECUC_Spi_00103] Definition of EcucModuleDef Spi ⌈

Module Name	Spi
Description	Configuration of the Spi (Serial Peripheral Interface) module.
Post-Build Selectable Support	true
Post-Build Loadable Support	true

Included Containers		
Container Name	**Multiplicity**	**Description**
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

[ECUC_Spi_00240] Definition of EcucParamConfContainerDef SpiDemEventParameterRefs ⌈

Container Name	SpiDemEventParameterRefs
Parent Container	Spi
Upstream mapping defined	–
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The Event Id 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.
Multiplicity	0..1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SPI_E_HARDWARE_ERROR	0..1	[ECUC_Spi_00241]

No Included Containers

⌋

[ECUC_Spi_00241] Definition of EcucReferenceDef SPI_E_HARDWARE_ERROR ⌈

Parameter Name	SPI_E_HARDWARE_ERROR
Parent Container	SpiDemEventParameterRefs
Upstream mapping defined	–
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
Pre-Compile only	true

⌋

10.2.3 SpiGeneral

[ECUC_Spi_00225] Definition of EcucParamConfContainerDef SpiGeneral ⌈

Container Name	SpiGeneral
Parent Container	Spi
Upstream mapping defined	–
Description	General configuration settings for SPI-Handler
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	Multiplicity	ECUC ID
SpiCancelApi	1	[ECUC_Spi_00226]
SpiChannelBuffersAllowed	1	[ECUC_Spi_00227]
SpiDevErrorDetect	1	[ECUC_Spi_00228]
SpiHwStatusApi	1	[ECUC_Spi_00229]
SpiInterruptibleSeqAllowed	1	[ECUC_Spi_00230]
SpiLevelDelivered	1	[ECUC_Spi_00231]
SpiMainFunctionPeriod	0..1	[ECUC_Spi_00242]
SpiSupportConcurrentSyncTransmit	1	[ECUC_Spi_00237]
SpiEcucPartitionRef	0..*	[ECUC_Spi_00244]
SpiKernelEcucPartitionRef	0..1	[ECUC_Spi_00245]

No Included Containers

⌋

[ECUC_Spi_00226] Definition of EcucBooleanParamDef SpiCancelApi ⌈

Parameter Name	SpiCancelApi
Parent Container	SpiGeneral
Upstream mapping defined	–
Description	Switches the Spi_Cancel function ON or OFF.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Spi_00227] Definition of EcucIntegerParamDef SpiChannelBuffersAllowed ⌈

Parameter Name	SpiChannelBuffersAllowed
Parent Container	SpiGeneral
Upstream mapping defined	–
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	–	
Pre-Compile only	true	

⌋

[ECUC_Spi_00228] Definition of EcucBooleanParamDef SpiDevErrorDetect ⌈

Parameter Name	SpiDevErrorDetect
Parent Container	SpiGeneral
Upstream mapping defined	–
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
Pre-Compile only	true

⌋

[ECUC_Spi_00229] Definition of EcucBooleanParamDef SpiHwStatusApi ⌈

Parameter Name	SpiHwStatusApi
Parent Container	SpiGeneral
Upstream mapping defined	–
Description	Switches the Spi_GetHWUnitStatus function ON or OFF.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Spi_00230] Definition of EcucBooleanParamDef SpiInterruptibleSeqAllowed ⌈

Parameter Name	SpiInterruptibleSeqAllowed
Parent Container	SpiGeneral
Upstream mapping defined	–
Description	Switches the Interruptible Sequences handling functionality ON or OFF. Dependency: This parameter depends on SPI_LEVEL_DELIVERED value. It is only used for SPI_LEVEL_DELIVERED configured to 1 or 2.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Spi_00231] Definition of EcucIntegerParamDef SpiLevelDelivered ⌈

Parameter Name	SpiLevelDelivered	
Parent Container	SpiGeneral	
Upstream mapping defined	–	
Description	Selects the SPI Handler/Driver level of scalable functionality that is available and delivered.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 2	
Default value	–	
Pre-Compile only	true	

⌋

[ECUC_Spi_00242] Definition of EcucFloatParamDef SpiMainFunctionPeriod ⌈

Parameter Name	SpiMainFunctionPeriod	
Parent Container	SpiGeneral	
Upstream mapping defined	–	
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	
Pre-Compile only	true	

⌋

[ECUC_Spi_00237] Definition of EcucBooleanParamDef SpiSupportConcurrentSyncTransmit ⌈

Parameter Name	SpiSupportConcurrentSyncTransmit
Parent Container	SpiGeneral
Upstream mapping defined	–
Description	Specifies whether concurrent Spi_SyncTransmit() calls for different sequences shall be configurable.
Multiplicity	1
Type	EcucBooleanParamDef
Default value	–
Pre-Compile only	true

⌋

[ECUC_Spi_00244] Definition of EcucReferenceDef SpiEcucPartitionRef ⌈

Parameter Name	SpiEcucPartitionRef
Parent Container	SpiGeneral
Upstream mapping defined	–

▽

△

Description	Maps the SPI driver to zero or multiple ECUC partitions to make the driver API available in the according partition.	
Multiplicity	0..*	
Type	Reference to EcucPartition	
Post-Build Multiplicity	-	Selectable
Post-Build Value	-	Selectable

⌋

[ECUC_Spi_00245] Definition of EcucReferenceDef SpiKernelEcucPartitionRef ⌈

Parameter Name	SpiKernelEcucPartitionRef	
Parent Container	SpiGeneral	
Upstream mapping defined	–	
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.	
Multiplicity	0..1	
Type	Reference to EcucPartition	
Post-Build Multiplicity	-	Selectable
Post-Build Value	-	Selectable

⌋

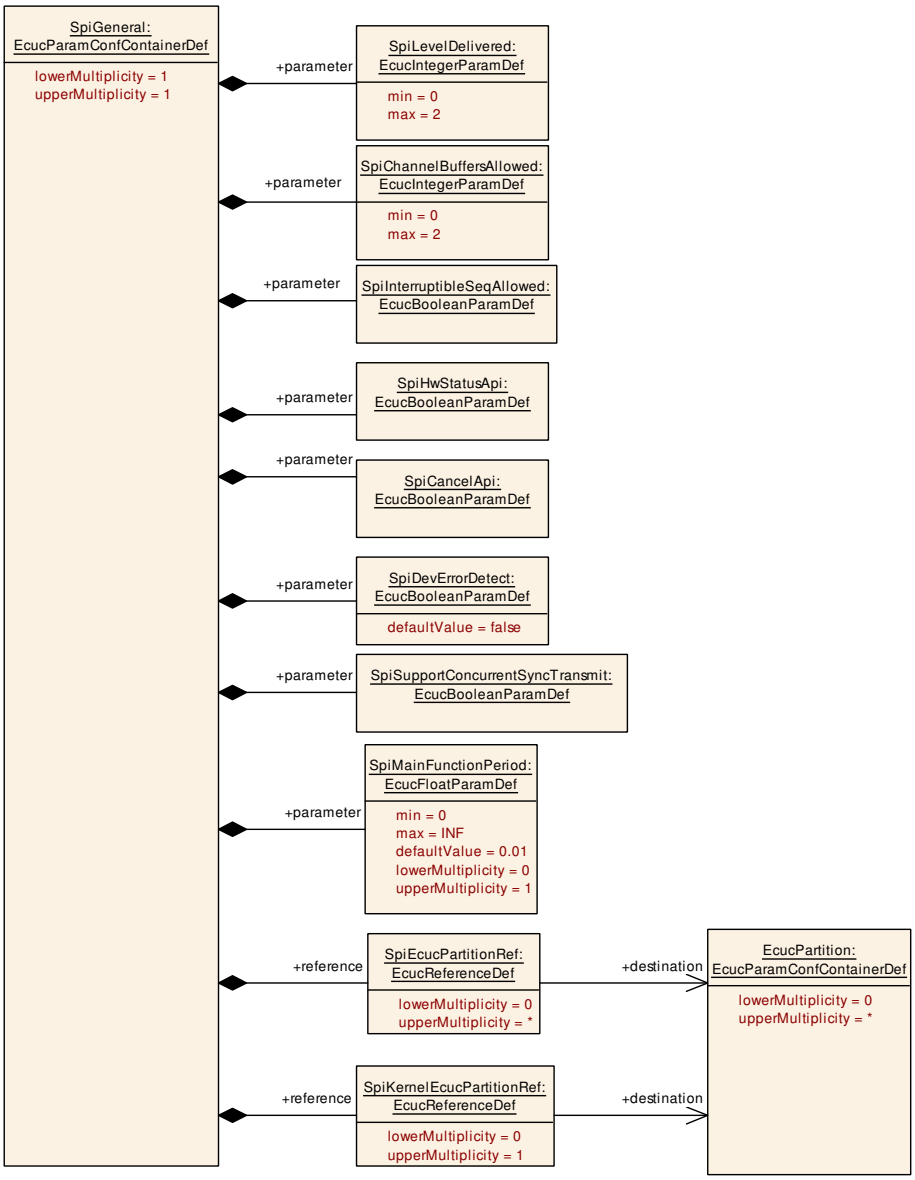


Figure 10.1

[SWS_Spi_CONSTR_00001] ⌈The ECUC partitions referenced by SpiKernelEcuc-PartitionRef shall be a subset of the ECUC partitions referenced by SpiEcuc-PartitionRef.⌋

[SWS_Spi_CONSTR_00003] ⌈If SpiEcucPartitionRef references one or more ECUC partitions, SpiKernelEcucPartitionRef shall have a multiplicity of one and reference one of these ECUC partitions as well.⌋

10.2.4 SpiSequence

[ECUC_Spi_00106] Definition of EcucParamConfContainerDef SpiSequence ⌈

Container Name	SpiSequence
Parent Container	SpiDriver
Upstream mapping defined	–
Description	All data needed to configure one SPI-sequence
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SpiInterruptibleSequence	1	[ECUC_Spi_00222]
SpiSeqEndNotification	0..1	[ECUC_Spi_00223]
SpiSequenceId	1	[ECUC_Spi_00224]
SpiJobAssignment	1..*	[ECUC_Spi_00221]

No Included Containers

⌋

[ECUC_Spi_00222] Definition of EcucBooleanParamDef SpiInterruptibleSequence ⌈

Parameter Name	SpiInterruptibleSequence	
Parent Container	SpiSequence	
Upstream mapping defined	–	
Description	This parameter allows or not this Sequence to be suspended by another one. Dependency: This SPI_INTERRUPTIBLE_SEQ_ALLOWED parameter as to be configured as ON.	
Multiplicity	1	
Type	EcucBooleanParamDef	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00223] Definition of EcucFunctionNameDef SpiSeqEndNotification ⌈

Parameter Name	SpiSeqEndNotification	
Parent Container	SpiSequence	
Upstream mapping defined	–	
Description	This parameter is a reference to a notification function.	
Multiplicity	0..1	
Type	EcucFunctionNameDef	
Default value	–	
Regular Expression	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00224] Definition of EcucIntegerParamDef SpiSequenceId ⌈

Parameter Name	SpiSequenceId	
Parent Container	SpiSequence	
Upstream mapping defined	–	
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	–	
Pre-Compile only	true	

⌋

[ECUC_Spi_00221] Definition of EcucReferenceDef SpiJobAssignment ⌈

Parameter Name	SpiJobAssignment	
Parent Container	SpiSequence	
Upstream mapping defined	–	
Description	A sequence references several jobs, which are executed during a communication sequence	
Multiplicity	1..*	
Type	Reference to SpiJob	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

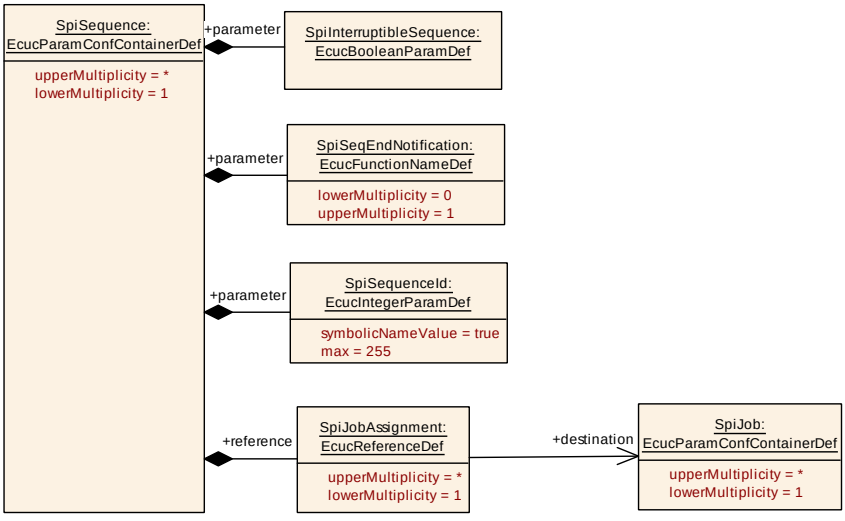


Figure 10.2

10.2.5 SpiChannel

[ECUC_Spi_00104] Definition of EcucParamConfContainerDef SpiChannel ⌈

Container Name	SpiChannel
Parent Container	SpiDriver
Upstream mapping defined	–
Description	All data needed to configure one SPI-channel
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SpiChannelId	1	[ECUC_Spi_00200]
SpiChannelType	1	[ECUC_Spi_00201]
SpiDataWidth	1	[ECUC_Spi_00202]
SpiDefaultData	0..1	[ECUC_Spi_00203]
SpiEbMaxLength	0..1	[ECUC_Spi_00204]
SpiIbNBuffers	0..1	[ECUC_Spi_00205]
SpiTransferStart	1	[ECUC_Spi_00206]

No Included Containers

⌋

[ECUC_Spi_00200] Definition of EcucIntegerParamDef SpiChannelId ⌈

Parameter Name	SpiChannelId	
Parent Container	SpiChannel	
Upstream mapping defined	–	
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	–	
Pre-Compile only	true	

⌋

[ECUC_Spi_00201] Definition of EcucEnumerationParamDef SpiChannelType ⌈

Parameter Name	SpiChannelType	
Parent Container	SpiChannel	
Upstream mapping defined	–	
Description	Buffer usage with EB/IB channel. Dependency: SPI_CHANNEL_BUFFERS_ALLOWED	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	EB	External Buffer
	IB	Internal Buffer

▽

△

Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00202] Definition of EcucIntegerParamDef SpiDataWidth ⌈

Parameter Name	SpiDataWidth	
Parent Container	SpiChannel	
Upstream mapping defined	–	
Description	This parameter is the width of a transmitted data unit.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	1 .. 64	
Default value	32	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00203] Definition of EcucIntegerParamDef SpiDefaultData ⌈

Parameter Name	SpiDefaultData	
Parent Container	SpiChannel	
Upstream mapping defined	–	
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 Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00204] Definition of EcucIntegerParamDef SpiEbMaxLength ⌈

Parameter Name	SpiEbMaxLength	
Parent Container	SpiChannel	
Upstream mapping defined	–	
Description	This parameter contains the maximum size (number of data elements) of data buffers in case of EB Channels and only. Dependency: The SPI_CHANNEL_TYPE parameter has to be configured as EB for this Channel. The SPI_CHANNEL_BUFFERS_ALLOWED parameter has to be configured as 1 or 2.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	1 .. 1048576	
Default value	1024	
Post-Build Multiplicity	-	Selectable

▽

△

Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00205] Definition of EcucIntegerParamDef SpiIbNBuffers ⌈

Parameter Name	SpiIbNBuffers	
Parent Container	SpiChannel	
Upstream mapping defined	–	
Description	This parameter contains the maximum number of data buffers in case of IB Channels and only. 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.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	1 .. 65535	
Default value	1	
Post-Build Multiplicity	-	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00206] Definition of EcucEnumerationParamDef SpiTransferStart ⌈

Parameter Name	SpiTransferStart	
Parent Container	SpiChannel	
Upstream mapping defined	–	
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 Value	Loadable	Selectable

⌋

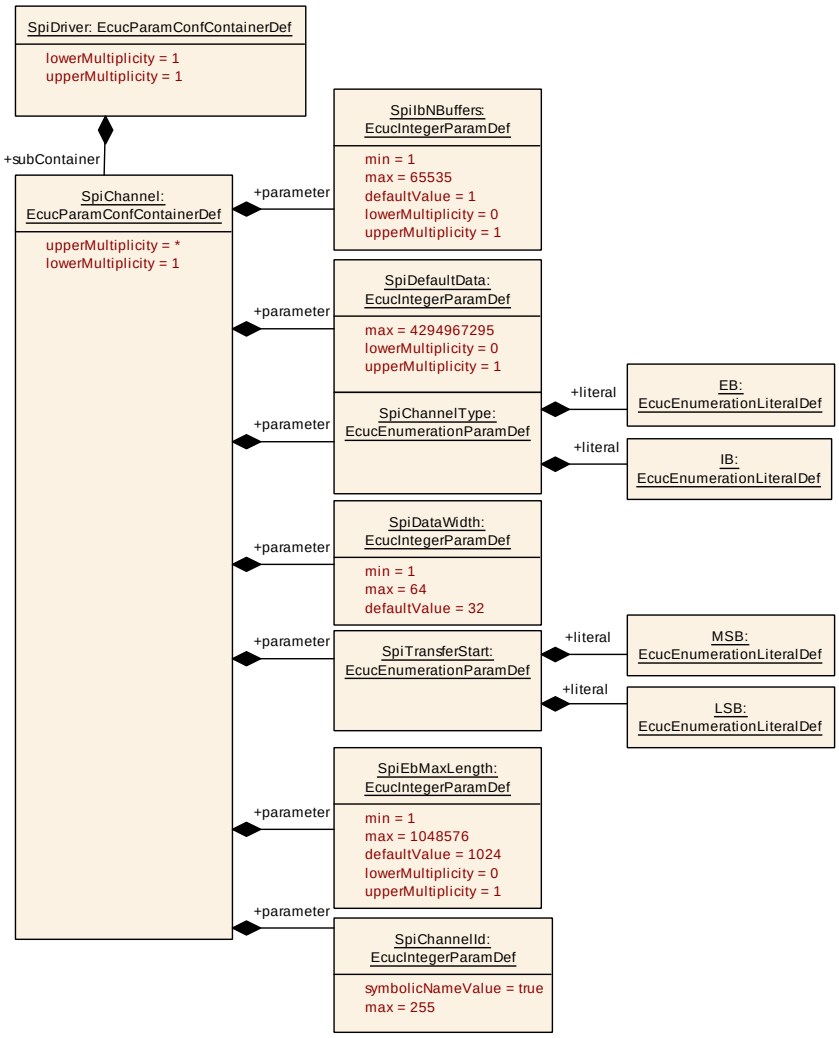


Figure 10.3

10.2.6 SpiChannelList

[ECUC_Spi_00233] Definition of EcucParamConfContainerDef SpiChannelList ⌈

Container Name	SpiChannelList
Parent Container	SpiJob
Upstream mapping defined	–
Description	References to SPI channels and their order within the Job.
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SpiChannelIndex	1	[ECUC_Spi_00234]
SpiChannelAssignment	1	[ECUC_Spi_00215]

No Included Containers

⌋

[ECUC_Spi_00234] Definition of EcucIntegerParamDef SpiChannelIndex ⌈

Parameter Name	SpiChannelIndex	
Parent Container	SpiChannelList	
Upstream mapping defined	–	
Description	This parameter specifies the order of Channels within the Job.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	0	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00215] Definition of EcucReferenceDef SpiChannelAssignment ⌈

Parameter Name	SpiChannelAssignment	
Parent Container	SpiChannelList	
Upstream mapping defined	–	
Description	A job reference to a SPI channel.	
Multiplicity	1	
Type	Reference to SpiChannel	
Post-Build Value	Loadable	Selectable

⌋

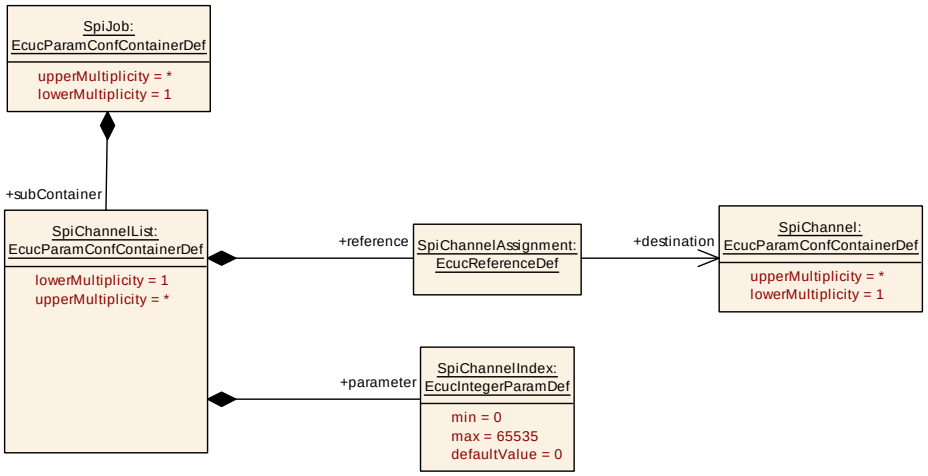


Figure 10.4

10.2.7 SpiJob

[ECUC_Spi_00105] Definition of EcucParamConfContainerDef SpiJob ⌈

Container Name	SpiJob
Parent Container	SpiDriver
Upstream mapping defined	–
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.
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SpiJobEndNotification	0..1	[ECUC_Spi_00218]
SpiJobId	1	[ECUC_Spi_00219]
SpiJobPriority	1	[ECUC_Spi_00220]
SpiDeviceAssignment	1	[ECUC_Spi_00216]

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

⌋

[ECUC_Spi_00218] Definition of EcucFunctionNameDef SpiJobEndNotification ⌈

Parameter Name	SpiJobEndNotification	
Parent Container	SpiJob	
Upstream mapping defined	–	
Description	This parameter is a reference to a notification function.	
Multiplicity	0..1	
Type	EcucFunctionNameDef	
Default value	–	
Regular Expression	–	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00219] Definition of EcucIntegerParamDef SpiJobId ⌈

Parameter Name	SpiJobId
Parent Container	SpiJob
Upstream mapping defined	–
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	–	
Pre-Compile only	true	

⌋

[ECUC_Spi_00220] Definition of EcucIntegerParamDef SpiJobPriority ⌈

Parameter Name	SpiJobPriority	
Parent Container	SpiJob	
Upstream mapping defined	–	
Description	Priority: 0, lowest, 3, highest (see SWS_Spi_00093)	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 3	
Default value	–	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00216] Definition of EcucReferenceDef SpiDeviceAssignment ⌈

Parameter Name	SpiDeviceAssignment
Parent Container	SpiJob
Upstream mapping defined	–
Description	Reference to the external device used by this job
Multiplicity	1
Type	Reference to SpiExternalDevice
Pre-Compile only	true

⌋

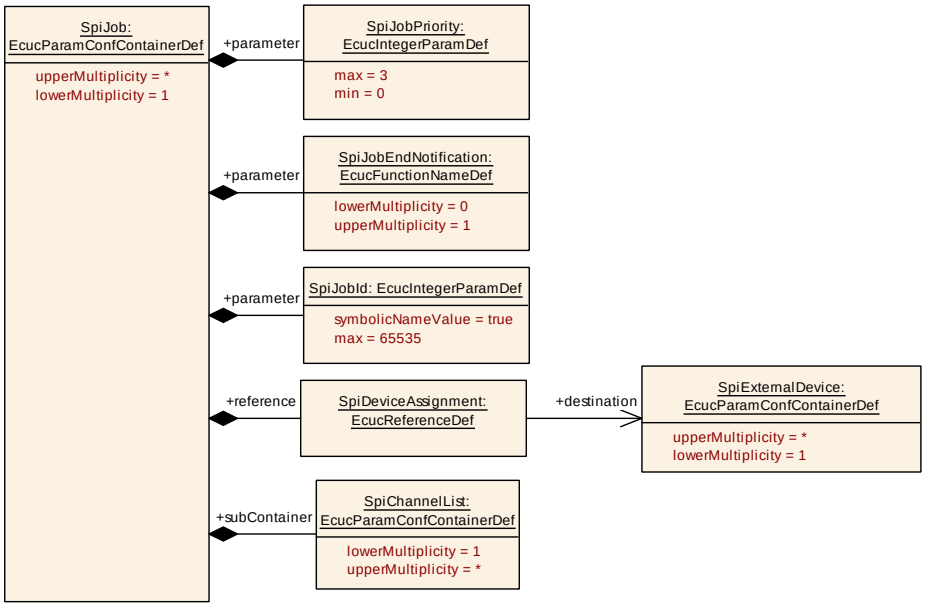


Figure 10.5

10.2.8 SpiExternalDevice

[ECUC_Spi_00207] Definition of EcucParamConfContainerDef SpiExternalDevice ⌈

Container Name	SpiExternalDevice
Parent Container	SpiDriver
Upstream mapping defined	–
Description	The communication settings of an external device. Closely linked to SpiJob.
Multiplicity	1..*
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SpiBaudrate	1	[ECUC_Spi_00208]
SpiCsBehavior	1	[ECUC_Spi_00249]
SpiCsIdentifier	1	[ECUC_Spi_00209]
SpiCsPolarity	1	[ECUC_Spi_00210]
SpiCsSelection	0..1	[ECUC_Spi_00239]
SpiDataShiftEdge	1	[ECUC_Spi_00211]
SpiEnableCs	1	[ECUC_Spi_00212]
SpiHwUnit	1	[ECUC_Spi_00217]
SpiShiftClockIdleLevel	1	[ECUC_Spi_00213]
SpiTimeClk2Cs	1	[ECUC_Spi_00214]
SpiTimeCs2Clk	1	[ECUC_Spi_00247]

▽

△

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SpiTimeCs2Cs	1	[ECUC_Spi_00248]
SpiDeviceEcucPartitionRef	0..*	[ECUC_Spi_00246]

No Included Containers

⌋

[ECUC_Spi_00208] Definition of EcucFloatParamDef SpiBaudrate ⌈

Parameter Name	SpiBaudrate	
Parent Container	SpiExternalDevice	
Upstream mapping defined	–	
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	1000000	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00249] Definition of EcucEnumerationParamDef SpiCsBehavior ⌈

Parameter Name	SpiCsBehavior	
Parent Container	SpiExternalDevice	
Upstream mapping defined	–	
Description	This parameter is used to define the chip select behavior. Either the CS is toggled for each data frame (bit frame on the SPI bus in relation with SpiDataWidth) inside the channel(s) composing the job or the CS is kept asserted for the whole job.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CS_KEEP_ASSERTED	The chip select is kept asserted for the whole job
	CS_TOGGLE	The chip select is released after each data frame completion
Default value	CS_KEEP_ASSERTED	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00209] Definition of EcucStringParamDef SpiCsIdentifier ⌈

Parameter Name	SpiCsIdentifier
Parent Container	SpiExternalDevice
Upstream mapping defined	–
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	–
Regular Expression	–
Pre-Compile only	true

⌋

[ECUC_Spi_00210] Definition of EcucEnumerationParamDef SpiCsPolarity ⌈

Parameter Name	SpiCsPolarity	
Parent Container	SpiExternalDevice	
Upstream mapping defined	–	
Description	This parameter defines the active polarity of Chip Select.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	HIGH	Chip select polarity high
	LOW	Chip select polarity low
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00239] Definition of EcucEnumerationParamDef SpiCsSelection ⌈

Parameter Name	SpiCsSelection	
Parent Container	SpiExternalDevice	
Upstream mapping defined	–	
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. Dependency: SpiEnableCs	
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 Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00211] Definition of EcucEnumerationParamDef SpiDataShiftEdge ⌈

Parameter Name	SpiDataShiftEdge
Parent Container	SpiExternalDevice
Upstream mapping defined	–
Description	This parameter defines the SPI data shift edge.
Multiplicity	1

▽

△

Type	EcucEnumerationParamDef	
Range	LEADING	Shift out data at the leading edge of the clock signal
	TRAILING	Shift out data at the trailing edge of the clock signal
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00212] Definition of EcucBooleanParamDef SpiEnableCs ⌈

Parameter Name	SpiEnableCs	
Parent Container	SpiExternalDevice	
Upstream mapping defined	–	
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 Value	Loadable	Selectable

⌋

[ECUC_Spi_00217] Definition of EcucEnumerationParamDef SpiHwUnit ⌈

Parameter Name	SpiHwUnit	
Parent Container	SpiExternalDevice	
Upstream mapping defined	–	
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	Hardware unit 0
	CSIB1	Hardware unit 1
	CSIB2	Hardware unit 2
	CSIB3	Hardware unit 3
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00213] Definition of EcucEnumerationParamDef SpiShiftClockIdle Level ⌈

Parameter Name	SpiShiftClockIdleLevel
Parent Container	SpiExternalDevice
Upstream mapping defined	–
Description	This parameter defines the SPI shift clock idle level.
Multiplicity	1
Type	EcucEnumerationParamDef

▽

△

Range	HIGH	Shift clock idle level high
	LOW	Shift clock idle level low
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00214] Definition of EcucFloatParamDef SpiTimeClk2Cs ⌈

Parameter Name	SpiTimeClk2Cs	
Parent Container	SpiExternalDevice	
Upstream mapping defined	–	
Description	Timing between clock and chip select assertion (in seconds) - This parameter allows to use a range of values from 10 ns up to 0.01 seconds. The real configuration-value used in software BSW-SPI is calculated out of this by the generator-tools.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[1E-8 .. 0.01]	
Default value	1E-6	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00247] Definition of EcucFloatParamDef SpiTimeCs2Clk ⌈

Parameter Name	SpiTimeCs2Clk	
Parent Container	SpiExternalDevice	
Upstream mapping defined	–	
Description	Timing between chip select assertion and clock (in seconds) - This parameter allows to use a range of values from 10ns up to 0.01 seconds. The real configuration-value used in software BSW-SPI is calculated out of this by the generator-tools.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[1E-8 .. 0.01]	
Default value	1E-6	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00248] Definition of EcucFloatParamDef SpiTimeCs2Cs ⌈

Parameter Name	SpiTimeCs2Cs	
Parent Container	SpiExternalDevice	
Upstream mapping defined	–	
Description	Timing between the negation of the chip select at the end of frame and the assertion of the chip select at the beginning of the next frame (in seconds) - This parameter allows to use a range of values from 10ns up to 0.01 seconds. The real configuration-value used in software BSW-SPI is calculated out of this by the generator-tools.	
Multiplicity	1	
Type	EcucFloatParamDef	
Range	[1E-8 .. 0.01]	

▽

△

Default value	1E-6	
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00246] Definition of EcucReferenceDef SpiDeviceEcucPartitionRef ⌈

Parameter Name	SpiDeviceEcucPartitionRef	
Parent Container	SpiExternalDevice	
Upstream mapping defined	–	
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.	
Multiplicity	0..*	
Type	Reference to EcucPartition	
Post-Build Multiplicity	-	Selectable
Post-Build Value	-	Selectable

⌋

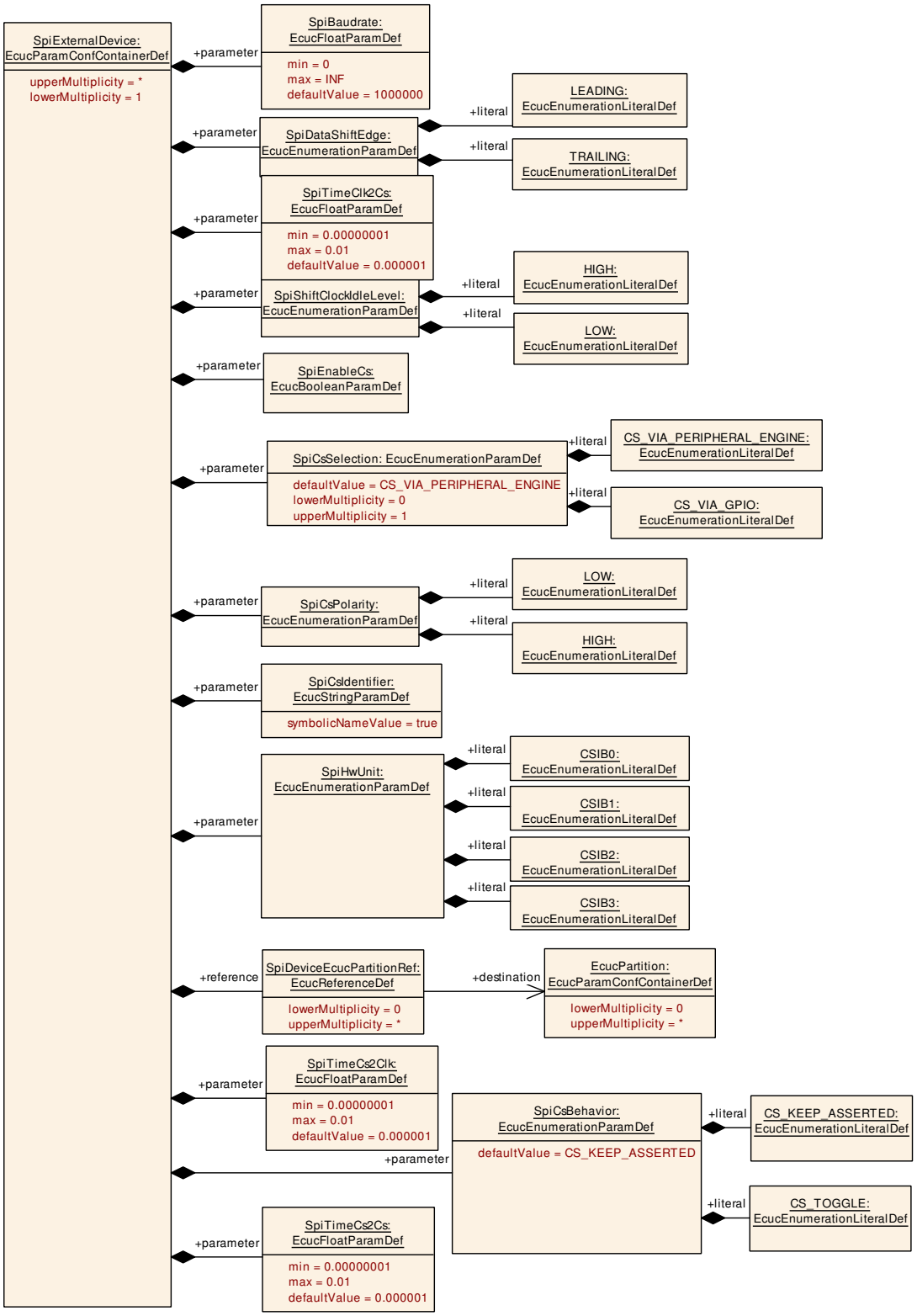


Figure 10.6

[SWS_Spi_CONSTR_00002] ⌈The ECUC partitions referenced by `SpiDeviceEcucPartitionRef` shall be a subset of the ECUC partitions referenced by `SpiEcucPartitionRef`.⌋

[SWS_Spi_CONSTR_00004] ⌈If `SpiEcucPartitionRef` references one or more ECUC partitions, `SpiDeviceEcucPartitionRef` shall have a multiplicity of greater than zero and reference one or several of these ECUC partitions as well.⌋

10.2.9 SpiDriver

[ECUC_Spi_00091] Definition of EcucParamConfContainerDef SpiDriver ⌈

Container Name	SpiDriver
Parent Container	Spi
Upstream mapping defined	–
Description	This container contains the configuration parameters and sub containers of the AUTOSAR Spi module.
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SpiMaxChannel	0..1	[ECUC_Spi_00197]
SpiMaxJob	0..1	[ECUC_Spi_00198]
SpiMaxSequence	0..1	[ECUC_Spi_00199]

Included Containers		
Container Name	**Multiplicity**	**Description**
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

⌋

[ECUC_Spi_00197] Definition of EcucIntegerParamDef SpiMaxChannel ⌈

Parameter Name	SpiMaxChannel	
Parent Container	SpiDriver	
Upstream mapping defined	–	
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 .. 65535	
Default value	0	
Post-Build Multiplicity	Loadable	Selectable

▽

△

Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00198] Definition of EcucIntegerParamDef SpiMaxJob ⌈

Parameter Name	SpiMaxJob	
Parent Container	SpiDriver	
Upstream mapping defined	–	
Description	Total number of Jobs configured.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	0	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

[ECUC_Spi_00199] Definition of EcucIntegerParamDef SpiMaxSequence ⌈

Parameter Name	SpiMaxSequence	
Parent Container	SpiDriver	
Upstream mapping defined	–	
Description	Total number of Sequences configured.	
Multiplicity	0..1	
Type	EcucIntegerParamDef	
Range	0 .. 65535	
Default value	0	
Post-Build Multiplicity	Loadable	Selectable
Post-Build Value	Loadable	Selectable

⌋

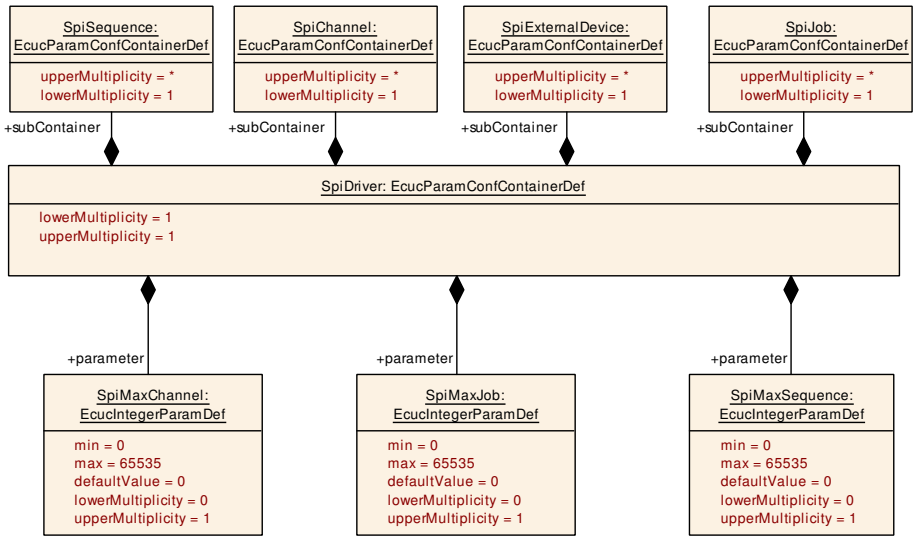


Figure 10.7

10.2.10 SpiPublishedInformation

[ECUC_Spi_00235] Definition of EcucParamConfContainerDef SpiPublishedInformation ⌈

Container Name	SpiPublishedInformation
Parent Container	Spi
Upstream mapping defined	–
Description	Container holding all SPI specific published information parameters
Multiplicity	1
Pre-Compile only	true
Configuration Parameters	

Included Parameters		
Parameter Name	**Multiplicity**	**ECUC ID**
SpiMaxHwUnit	1	[ECUC_Spi_00236]

No Included Containers

⌋

[ECUC_Spi_00236] Definition of EcucIntegerParamDef SpiMaxHwUnit ⌈

Parameter Name	SpiMaxHwUnit	
Parent Container	SpiPublishedInformation	
Upstream mapping defined	–	
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 .. 65535	
Default value	0	
Pre-Compile only	true	

⌋

10.3 Published Information

For details refer to [2] Chapter 10.3 *"Published Information"*.

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.

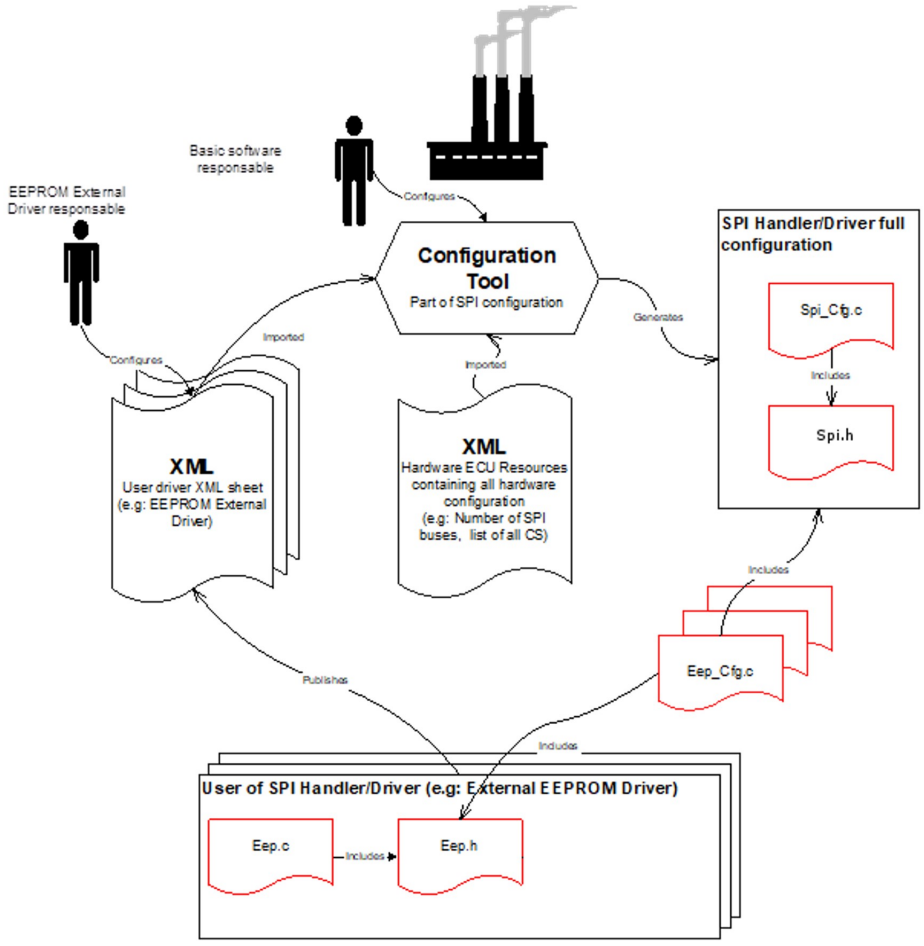


Figure 10.8

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.

A Not applicable requirements

[SWS_Spi_NA_00999]

Upstream requirements: SRS_BSW_00375, SRS_BSW_00399, SRS_BSW_00400, 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_00168, SRS_BSW_00170, 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

⌈These requirements are not applicable to this specification.⌋

B 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.

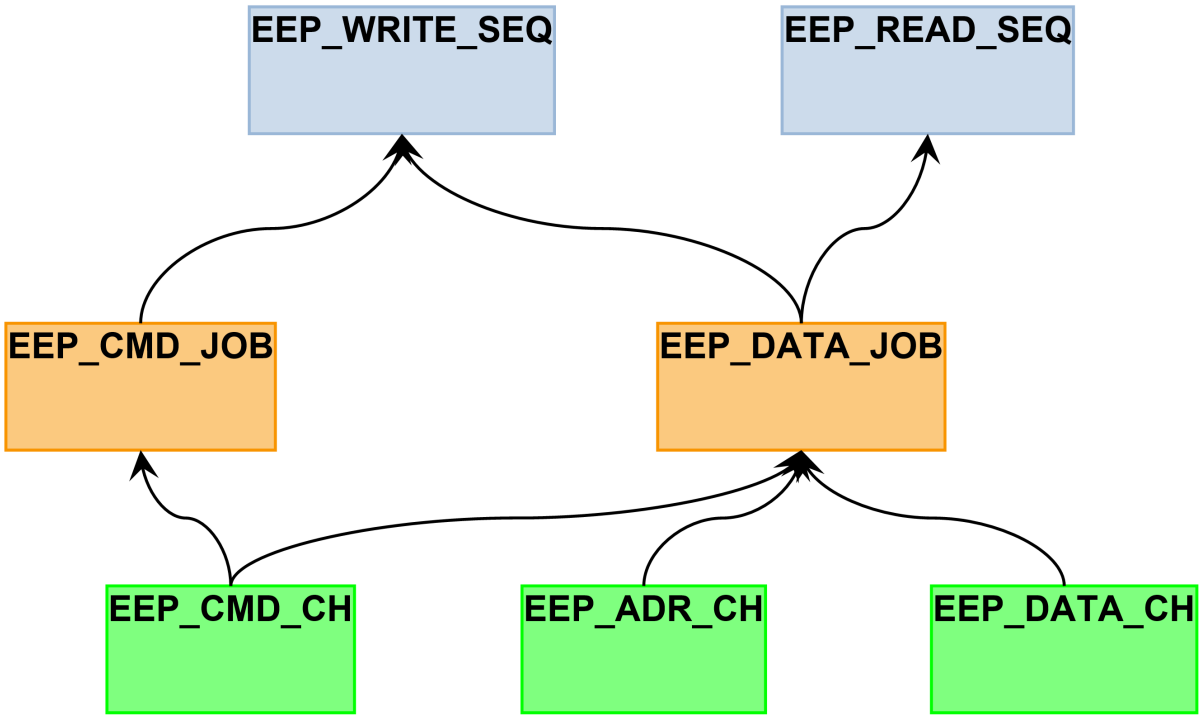


Figure B.1

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_vidEnd-OfWriteSeq	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_vidEndOfS-tartWrJob, 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_vidEndOf-ReadSeq	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

C Change history of AUTOSAR traceable items

Please note that the lists in this chapter also include traceable items that have been removed from the specification in a later version. These items do not appear as hyperlinks in the document.

C.1 Traceable item history of this document according to AUTOSAR Release R26-11

C.1.1 Added Specification Items in R26-11

none

C.1.2 Changed Specification Items in R26-11

Number	Heading
[ECUC_Spi_00225]	Definition of EcucParamConfContainerDef SpiGeneral
[SWS_Spi_00383]	A hardware error occurred during asynchronous or synchronous SPI transmit.
[SWS_Spi_91003]	Definition of imported datatypes of module Spi

Table C.1: Changed Specification Items in R26-11

C.1.3 Deleted Specification Items in R26-11

Number	Heading
[ECUC_Spi_00232]	Definition of EcucBooleanParamDef SpiVersionInfoApi
[SWS_Spi_00184]	Definition of API function Spi_GetVersionInfo
[SWS_Spi_00371]	

Table C.2: Deleted Specification Items in R26-11

C.1.4 Added Constraints in R26-11

none

C.1.5 Changed Constraints in R26-11

none

C.1.6 Deleted Constraints in R26-11

none

C.2 Traceable item history of this document according to AUTOSAR Release R25-11

C.2.1 Added Specification Items in R25-11

none

C.2.2 Changed Specification Items in R25-11

none

C.2.3 Deleted Specification Items in R25-11

none

C.2.4 Added Constraints in R25-11

none

C.2.5 Changed Constraints in R25-11

none

C.2.6 Deleted Constraints in R25-11

none