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

Document Version	3.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
24.11.2011	3.2.0	AUTOSAR Administration	• Rephrased: requirement SPI002, SPI046, SPI129, SPI233, SPI163, SPI 171, SPI172, SPI289 and SPI290, block 2 in chapter 7.2.2 • Removed: requirement SPI083; SPI132, SPI284 and SPI107 removed from statement • Corrected:Dem_EventStatusType in SPI191, Spi_SyncTransmit Syn/Async changed to Synchronous, SPI_E_PARAM_POINTER in SPI371, • Reference to MCU in SPI244 and SPI342 • Added: requirement SPI140, chapter 10 - SpiCsSelection, SPI194 - SPI_JOB_QUEUED state introduced, SPI195 with error table update • Modified: SPI114 and SPI135, chapter 10 - SpiEnableCs
12.11.2010	3.1.0	AUTOSAR Administration	• Added SPI369, SPI371, SPI370 • Removed SPI190, SPI094 • Updated configuration: base on min-max value for defined parameter; SpiHwUnit belongs to SpiExternalDevice Container; updated SpiTimeClk2Cs

Document Change History

Date	Version	Changed by	Change Description
11.12.2009	3.0.0	AUTOSAR Administration	• Splitting and refinement of several requirements • Removal of redundant requirements • Introduction of new IDs to allow implementation of debugging concept • Inserted UML diagram in chapter 9 • Updating of Chapter 10 with the inclusion of 2 new container and the definition of the Chip Select configuration • Legal disclaimer revised
23.06.2008	2.2.1	AUTOSAR Administration	• Legal disclaimer revised
12.12.2007	2.2.0	AUTOSAR Administration	• Updated Chapter 10 with the inclusion of CS configuration • Document meta information extended • Small layout adaptations made
31.01.2007	2.1.0	AUTOSAR Administration	• Configuration Specification updating • General rephrasing for clarification • Syntax error • • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
28.04.2006	2.0.0	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
09.06.2005	1.0.0	AUTOSAR Administration	Initial Release

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and functional overview

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

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

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

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

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

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

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

2 Acronyms and abbreviations

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

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

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

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

3 Related documentation

3.1 Input documents

[1] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[2] General Requirements on SPAL
AUTOSAR_SRS_SPALGeneral.pdf

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

[4] Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[5] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

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

[7] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager.pdf

[8] Glossary
AUTOSAR_TR_Glossary.pdf

[9] Specification of MCU Driver
AUTOSAR_SWS_MCUDriver .pdf

[10] Specification of PORT Driver
AUTOSAR_SWS_PORTDriver

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

[12] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

[13] Specification of Standard Types,
AUTOSAR_SWS_StandardTypes.pdf

3.2 Related standards and norms

Not related.

4 Constraints and assumptions

4.1 Limitations

[SPI040]⌈The SPI Handler/Driver handles only the Master mode.⌋()

[SPI050]⌈The SPI Handler/Driver only supports full-duplex mode.⌋()

[SPI108]⌈The LEVEL 2 SPI Handler/Driver is specified for microcontrollers that have to provide, at least, two SPI busses using separated hardware units. Otherwise, using this level of functionality does not make sense.⌋()

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

[SPI239] ⌈SPI peripherals may depend on the system clock, prescaler(s) and PLL. Thus, changes of the system clock (e.g. PLL on → PLL off) may also affect the clock settings of the SPI hardware. ⌋()

[SPI244] ⌈The SPI Handler/Driver module does not take care of setting the registers which configure the clock, prescaler(s) and PLL in its init function. This has to be done by the MCU module [9].⌋()

[SPI342] ⌈Depending on microcontrollers, the SPI peripheral could share registers with other peripherals. In this typical case, the SPI Handler/Driver has a relationship with MCU module [9] for initialising and de-initialising those registers.⌋()

[SPI343] ⌈If Chip Selects are done using microcontroller pins the SPI Handler/Driver has a relationship with PORT module [10]. In this case, this specification assumes that these microcontroller pins are directly accessed by the SPI Handler/Driver module without using APIs of DIO module.
Anyhow, the SPI depends on ECU hardware design and for that reason it may depend on other modules.⌋()

5.1 File structure

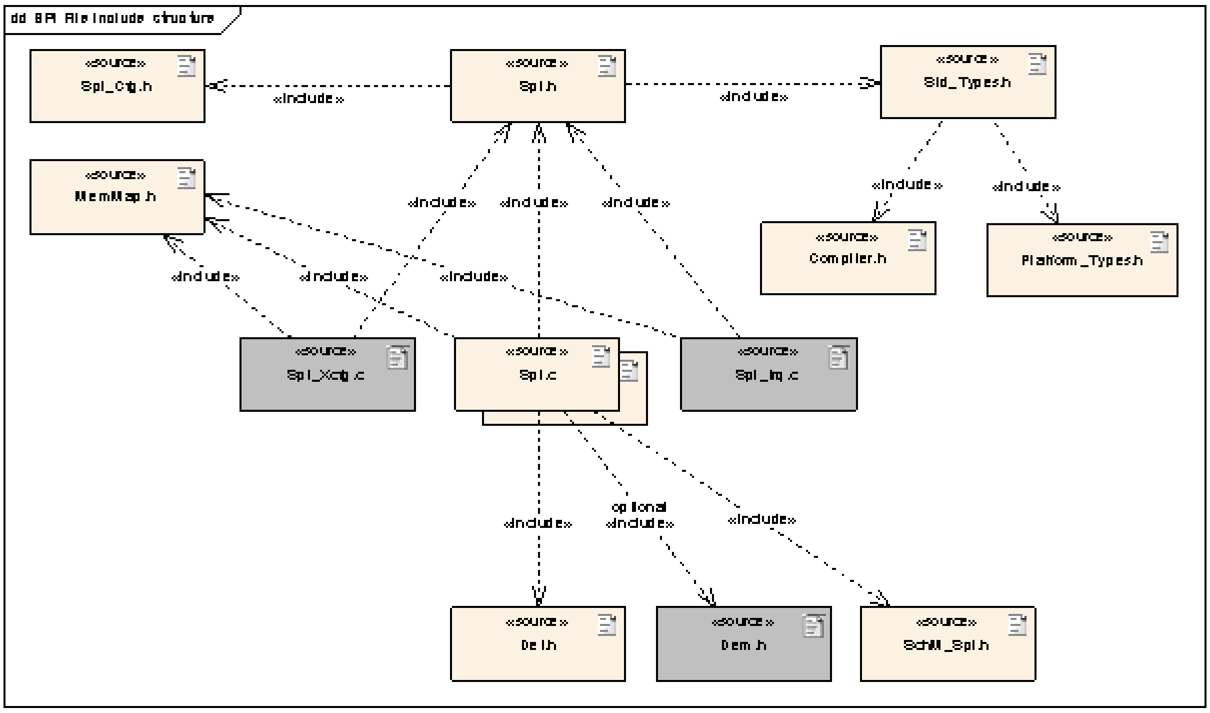
5.1.1 Code file structure

[SPI095] ⌈The code file structure shall not be defined within this specification completely.⌋(BSW00380, BSW00419, BSW158)

[SPI277] ⌈The code-file structure shall include the file named Spi_Lcfg.c – for link time and Spi_PBcfg.c – for post build time configurable parameters.⌋()

5.1.2 Header file structure

[SPI092] ⌈The SPI module shall adhere to the following include file structure: Spi.c shall include Spi.h, MemMap.h, Det.h and SchM_Spi.h.



⌋(BSW00412, BSW00415, BSW00435, BSW00436)

[SPI272] ⌈Spi.h shall include Std_Types.h.⌋()

[SPI273] ⌈Spi.h shall include Spi_Cfg.h.⌋()

[SPI274] ⌈Spi_Xcfg.c shall include Spi.h.⌋()

[SPI275] ⌈Spi_Xcfg.c shall include MemMap.h.⌋()

[SPI276] ⌈Spi_Irq.c file could exist depending upon implementation and also it could or not include Spi.h.⌋()

[SPI158] ⌈The SPI module shall optionally include the `Dem.h` file if any production error will be issued by the implemetation. By this inclusion the APIs to report errors as well as the required Event Id symbols are included.⌋(BSW00384)

[SPI159] ⌈The DEM configuration tool shall assign ECU dependent values to the Event Id symbols and publish the symbols in `Dem_IntErrId.h`.⌋(BSW00384)

The names of the Event Id symbols which are provided by XML to the DEM configuration tool are specified in this document.

6 Requirements traceability

Requirement	Satisfied by
-	SPI292
-	SPI160
-	SPI244
-	SPI164
-	SPI255
-	SPI137
-	SPI332
-	SPI080
-	SPI310
-	SPI308
-	SPI361
-	SPI349
-	SPI316
-	SPI309
-	SPI249
-	SPI154
-	SPI151
-	SPI161
-	SPI088
-	SPI307
-	SPI299
-	SPI243
-	SPI171
-	SPI293
-	SPI112
-	SPI183
-	SPI175
-	SPI236
-	SPI287
-	SPI186
-	SPI267
-	SPI256
-	SPI040
-	SPI343
-	SPI170
-	SPI275

-	SPI185
-	SPI341
-	SPI117
-	SPI143
-	SPI030
-	SPI363
-	SPI297
-	SPI037
-	SPI326
-	SPI131
-	SPI302
-	SPI188
-	SPI129
-	SPI322
-	SPI036
-	SPI368
-	SPI269
-	SPI023
-	SPI270
-	SPI140
-	SPI239
-	SPI152
-	SPI184
-	SPI176
-	SPI311
-	SPI286
-	SPI246
-	SPI328
-	SPI086
-	SPI108
-	SPI300
-	SPI150
-	SPI342
-	SPI156
-	SPI344
-	SPI364
-	SPI260
-	SPI133
-	SPI289

-	SPI278
-	SPI128
-	SPI146
-	SPI195
-	SPI354
-	SPI116
-	SPI301
-	SPI238
-	SPI313
-	SPI280
-	SPI303
-	SPI182
-	SPI028
-	SPI258
-	SPI370
-	SPI330
-	SPI145
-	SPI265
-	SPI336
-	SPI325
-	SPI350
-	SPI317
-	SPI242
-	SPI130
-	SPI254
-	SPI187
-	SPI345
-	SPI371
-	SPI366
-	SPI177
-	SPI331
-	SPI281
-	SPI272
-	SPI123
-	SPI240
-	SPI166
-	SPI320
-	SPI126
-	SPI049

-	SPI295
-	SPI355
-	SPI157
-	SPI347
-	SPI196
-	SPI334
-	SPI277
-	SPI082
-	SPI012
-	SPI179
-	SPI266
-	SPI285
-	SPI169
-	SPI359
-	SPI138
-	SPI189
-	SPI027
-	SPI358
-	SPI081
-	SPI172
-	SPI141
-	SPI290
-	SPI114
-	SPI262
-	SPI250
-	SPI264
-	SPI288
-	SPI155
-	SPI362
-	SPI136
-	SPI360
-	SPI261
-	SPI321
-	SPI351
-	SPI282
-	SPI304
-	SPI338
-	SPI273
-	SPI144

-	SPI305
-	SPI339
-	SPI233
-	SPI142
-	SPI274
-	SPI252
-	SPI356
-	SPI296
-	SPI011
-	SPI017
-	SPI268
-	SPI263
-	SPI168
-	SPI271
-	SPI365
-	SPI051
-	SPI115
-	SPI251
-	SPI335
-	SPI367
-	SPI192
-	SPI327
-	SPI346
-	SPI333
-	SPI024
-	SPI173
-	SPI191
-	SPI348
-	SPI194
-	SPI353
-	SPI324
-	SPI276
-	SPI329
-	SPI291
-	SPI245
-	SPI193
-	SPI085
-	SPI241
-	SPI237

-	SPI352
-	SPI314
-	SPI253
-	SPI178
-	SPI298
-	SPI257
-	SPI259
-	SPI337
-	SPI165
-	SPI318
-	SPI294
-	SPI167
-	SPI319
-	SPI323
-	SPI149
-	SPI181
-	SPI312
-	SPI306
-	SPI279
-	SPI180
-	SPI139
-	SPI340
-	SPI357
-	SPI315
-	SPI283
-	SPI135
-	SPI050
BSW00301	SPI999
BSW00302	SPI999
BSW00306	SPI999
BSW00307	SPI999
BSW00308	SPI999
BSW00309	SPI999
BSW00312	SPI999
BSW00323	SPI029, SPI060, SPI031, SPI032
BSW00324	SPI999
BSW00325	SPI999
BSW00326	SPI999
BSW00327	SPI004

BSW00328	SPI999
BSW00330	SPI999
BSW00331	SPI999
BSW00334	SPI999
BSW00335	SPI019, SPI061, SPI062
BSW00336	SPI022, SPI021
BSW00337	SPI098, SPI097, SPI007, SPI004
BSW00338	SPI100
BSW00339	SPI099, SPI006
BSW00341	SPI999
BSW00342	SPI999
BSW00343	SPI999
BSW00344	SPI009
BSW00347	SPI999
BSW00350	SPI005
BSW00355	SPI999
BSW00357	SPI174
BSW00359	SPI048
BSW00360	SPI048
BSW00369	SPI029, SPI006, SPI005, SPI048
BSW00375	SPI999
BSW00380	SPI095
BSW00384	SPI159, SPI158
BSW00385	SPI007, SPI004
BSW00386	SPI029, SPI005
BSW00399	SPI999
BSW004	SPI069, SPI369
BSW00400	SPI999
BSW00401	SPI999
BSW00405	SPI013, SPI008
BSW00406	SPI015, SPI046
BSW00407	SPI102, SPI101
BSW00409	SPI097
BSW00411	SPI102
BSW00412	SPI092
BSW00413	SPI999
BSW00415	SPI092
BSW00416	SPI999
BSW00417	SPI999

BSW00419	SPI095
BSW00420	SPI999
BSW00421	SPI099, SPI006
BSW00422	SPI999
BSW00423	SPI999
BSW00424	SPI999
BSW00426	SPI999
BSW00427	SPI999
BSW00428	SPI999
BSW00429	SPI999
BSW00431	SPI999
BSW00432	SPI999
BSW00433	SPI999
BSW00434	SPI999
BSW00435	SPI092
BSW00436	SPI092
BSW005	SPI999
BSW006	SPI999
BSW009	SPI999
BSW010	SPI999
BSW101	SPI013, SPI015
BSW12024	SPI008, SPI063
BSW12025	SPI009, SPI008, SPI052, SPI053, SPI063
BSW12026	SPI009
BSW12032	SPI009, SPI066
BSW12033	SPI009, SPI066
BSW12037	SPI014, SPI124, SPI127, SPI059
BSW12056	SPI009, SPI054, SPI064, SPI044
BSW12057	SPI013, SPI015
BSW12063	SPI999
BSW12064	SPI025, SPI021
BSW12067	SPI999
BSW12068	SPI999
BSW12069	SPI999
BSW12075	SPI053
BSW12077	SPI999
BSW12078	SPI999
BSW12092	SPI999
BSW12093	SPI010, SPI009, SPI034, SPI041

BSW12094	SPI009, SPI066
BSW12099	SPI016, SPI020, SPI162, SPI163
BSW12101	SPI018, SPI020, SPI162, SPI163
BSW12103	SPI020, SPI058, SPI053, SPI067, SPI162, SPI163
BSW12104	SPI025, SPI026, SPI039
BSW12108	SPI120, SPI118, SPI119, SPI057
BSW12125	SPI013, SPI009, SPI008
BSW12129	SPI999
BSW12150	SPI093, SPI009, SPI064
BSW12152	SPI016, SPI134
BSW12153	SPI018, SPI134
BSW12154	SPI134
BSW12163	SPI022, SPI021
BSW12170	SPI084, SPI042
BSW12179	SPI003, SPI009, SPI064, SPI065
BSW12180	SPI003, SPI065
BSW12181	SPI055, SPI065
BSW12197	SPI063
BSW12198	SPI077, SPI053
BSW12199	SPI003, SPI064, SPI065
BSW12200	SPI077, SPI003, SPI053, SPI065, SPI035
BSW12201	SPI077, SPI003, SPI065, SPI035
BSW12202	SPI078, SPI053
BSW12253	SPI078, SPI052
BSW12256	SPI009, SPI008, SPI034
BSW12257	SPI010, SPI009, SPI008, SPI066, SPI065, SPI063, SPI034
BSW12258	SPI003, SPI009, SPI065
BSW12259	SPI009
BSW12260	SPI093, SPI014, SPI002, SPI009, SPI059, SPI064
BSW12261	SPI003, SPI053, SPI065
BSW12262	SPI078, SPI003, SPI053, SPI065
BSW12265	SPI999
BSW12267	SPI999
BSW13400	SPI110
BSW13401	SPI109, SPI121, SPI122, SPI125, SPI111
BSW157	SPI075, SPI073, SPI026, SPI057, SPI071, SPI038, SPI039, SPI042
BSW158	SPI095
BSW161	SPI999
BSW164	SPI999

BSW168	SPI999
BSW170	SPI999
BSW172	SPI999

Document: AUTOSAR requirements on Basic Software, general

Requirement	***Satisfied by***
[BSW003] Version identification	SPI068 SPI089
[BSW004] Version check	SPI369
[BSW00300] Module naming convention	Chapter 5.1
[BSW00301] Limit imported information	Not applicable (requirement on implementation, not on specification)
[BSW00302] Limit exported information	Not applicable (requirement on implementation, not on specification)
[BSW00304] AUTOSAR integer data types	Chapters 5.1.2, 8.2, 10.2 and 10.3
[BSW00305] Self-defined data types naming convention	Chapter 8.2
[BSW00306] Avoid direct use of compiler and platform specific keywords	Not applicable (requirement on implementation, not on specification)
[BSW00307] Global variables naming convention	Not applicable (requirement on implementation, not on specification)
[BSW00308] Definition of global data	Not applicable (requirement on implementation, not on specification)
[BSW00309] Global data with read-only constraint	Not applicable (requirement on implementation, not on specification)
[BSW00310] API naming convention	Chapter 8.3
[BSW00312] Shared code shall be reentrant	Not applicable (requirement on implementation, not on specification)
[BSW00314] Separation of interrupt frames and service routines	Chapter 5.1
[BSW00318] Format of module version numbers	SPI068
[BSW00321] Enumeration of module version numbers	SPI068
[BSW00323] API parameter checking	SPI029 SPI031 SPI032 SPI060
[BSW00324] Do not use HIS I/O Library	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00325] Runtime of interrupt service routines	Not applicable (Cannot be detailed at this point of time, because this depends on module implementation.)
[BSW00326] Transition from ISRs to OS tasks	Not applicable (Cannot be detailed at this point of time, because this depends on module implementation.)
[BSW00327] Error values naming convention	SPI004
[BSW00328] Avoid duplication of code	Not applicable (requirement on implementation, not on specification)
[BSW00329] Avoidance of generic interfaces	Chapter 8

[BSW00330] Usage of macros / inline functions instead of functions	Not applicable (requirement on implementation, not on specification)
[BSW00331] Separation of error and status values	Not applicable (requirement on implementation, not on specification)
[BSW00333] Documentation of callback function context	Chapters 8.6.3.1 and 8.6.3.2
[BSW00334] Provision of XML file	Not applicable (requirement on implementation, not on specification)
[BSW00335] Status values naming convention	SPI061 SPI062 SPI019
[BSW00336] Shutdown interface	SPI021 SPI022
[BSW00337] Classification of errors	SPI004 SPI007 SPI097 SPI098
[BSW00338] Reporting of development errors	SPI100
[BSW00339] Reporting of production relevant error status	SPI006 SPI099 and Chapter 8.6.2
[BSW00341] Microcontroller compatibility documentation	Not applicable (requirement on implementation, not on specification)
[BSW00342] Usage of source code and object code	Not applicable (requirement on implementation, not on specification)
[BSW00343] Specification and configuration of time	Not applicable (requirement on implementation, not on specification)
[BSW00344] Reference to link-time configuration	SPI009 SPI091
[BSW00345] Pre-compile-time configuration	SPI056
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable (requirement on implementation, not on specification)
[BSW00348] Standard type header	Chapter 8.1
[BSW00350] Development error detection keywords	SPI005 SPI103 SPI056
[BSW00353] Platform specific type header	Chapter 8.1
[BSW00355] Do not redefine AUTOSAR integer data types	Not applicable (requirement on implementation, not on specification)
[BSW00357] Standard API return type	SPI174 Chapter 8.3
[BSW00358] Return type of init() functions	Chapter 8.3.1
[BSW00359] Return type of callback functions	SPI048
[BSW00360] Parameters of callback functions	SPI048
[BSW00361] Compiler specific language extension header	Chapter 5.1.2
[BSW00369] Do not return development error codes via API	SPI005 SPI029 SPI048 SPI006
[BSW00370] Separation of callback interface from API	Chapter 8.4
[BSW00371] Do not pass function pointers via API	Chapters 8.6.3, 10.2
[BSW00373] Main processing function naming convention	Chapter 8.5
[BSW00374] Module vendor identification	SPI068 SPI089
[BSW00375] Notification of wake-up reason	Not applicable. (Only master mode is supported. Master mode does not provide wake up events.)
[BSW00376] Return type and parameters of main processing functions	Chapter 8.5
[BSW00377] Module specific API return types	Chapters 0, 8.2.3 and 8.2.4
[BSW00378] AUTOSAR boolean type	SPI105

[BSW00379] Module identification	SPI068 SPI089
[BSW00380] Separate C-Files for configuration parameters	SPI095
[BSW00381] Separate configuration header file for pre-compile time parameters	SPI103
[BSW00383] List dependencies of configuration files	Chapter 5
[BSW00384] List dependencies to other modules	Chapter 5, SPI158 SPI159
[BSW00385] List possible error notifications	SPI004 SPI007
[BSW00386] Configuration for detecting an error	SPI005 SPI029
[BSW00387] Specify the configuration class of callback function	Chapters 8.4 and 8.6.3
[BSW00388] Introduce containers	SPI103 SPI091 SPI104 SPI105 SPI106
[BSW00389] Containers shall have names	SPI103 SPI091 SPI104 SPI105 SPI106
[BSW00390] Parameter content shall be unique within the module	SPI103 SPI091 SPI104 SPI105 SPI106 SPI068
[BSW00391] Parameter shall have unique names	SPI103 SPI091 SPI104 SPI105 SPI106 SPI068
[BSW00392] Parameters shall have a type	SPI103 SPI091 SPI104 SPI105 SPI106
[BSW00393] Parameters shall have a range	SPI103 SPI091 SPI104 SPI105 SPI106
[BSW00394] Specify the scope of the parameters	SPI103 SPI091 SPI104 SPI105 SPI106
[BSW00395] List the required parameters (per parameter)	SPI103 SPI091 SPI104 SPI105 SPI106
[BSW00396] Configuration classes	SPI056 SPI076 SPI103 SPI091 SPI104 SPI105 SPI106
[BSW00397] Pre-compile-time parameters	SPI056 SPI103
[BSW00398] Link-time parameters	SPI076 SPI091 SPI104 SPI105 SPI106
[BSW00399] Loadable Post-build time parameters	Non applicable (Cannot be detailed at this point of time, because this depends on ECU integration.)
[BSW004] Version check	SPI069
[BSW00400] Selectable Post-build time parameters	Non applicable (Cannot be detailed at this point of time, because this depends on ECU integration.)
[BSW00401] Documentation of multiple instances of configuration parameters	Not applicable (requirement on implementation, not on specification)
[BSW00402] Published information	SPI068 SPI089
[BSW00404] Reference to post build time configuration	SPI148
[BSW00405] Reference to multiple configuration sets	SPI008 SPI013 SPI076 SPI148
[BSW00406] Check module initialization	SPI015 SPI046
[BSW00407] Function to read out published parameters	SPI101 SPI102
[BSW00408] Configuration parameter naming convention	Chapter 10.2
[BSW00409] Header files for production code error IDs	SPI097
[BSW00410] Compiler switches shall have defined values	SPI103
[BSW00411] Get version info keyword	SPI102
[BSW00412] Separate H-File for configuration parameters	SPI092
[BSW00413] Accessing instances of BSW modules	Not applicable (requirement on implementation, not on specification)
[BSW00414] Parameter of init function	Chapter 8.3.1
[BSW00415] User dependent include files	SPI092

[BSW00416] Sequence of Initialization	Not applicable (this is a general software integration requirement)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable (applies only for non BSW modules)
[BSW00419] Separate C-Files for pre-compile time configuration parameters	SPI095
[BSW00420] Production relevant error event rate detection	Not applicable (applies only for DEM)
[BSW00421] Reporting of production relevant error events	SPI006 SPI099 and Chapter 8.6.2
[BSW00422] Debouncing of production relevant error status	Not applicable (applies only for DEM)
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable (EEPROM driver has no Autosar Interface)
[BSW00424] BSW main processing function task allocation	Not applicable (this is a general software integration requirement)
[BSW00425] Trigger conditions for schedulable objects	Chapter 8.5
[BSW00426] Exclusive areas in BSW modules	Not applicable (Cannot be detailed at this point of time, because this depends on module implementation.)
[BSW00427] ISR description for BSW modules	Not applicable (Cannot be detailed at this point of time, because this depends on module implementation.)
[BSW00428] Execution order dependencies of main processing functions	Not applicable (Cannot be detailed at this point of time, because this depends on module implementation.)
[BSW00429] Restricted BSW OS functionality access	Not applicable (requirement on implementation, not on specification)
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable (SPI Handler/Driver Module is not the BSW Scheduler)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable (requirement on implementation, not on specification)
[BSW00433] Calling of main processing functions	Not applicable (this is a general software integration requirement)
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (SPI Handler/Driver Module is not the BSW Scheduler)
[BSW00435] Module Header File Structure for the Basic Software Scheduler	SPI092
[BSW00436] Module Header File Structure for the Memory Mapping	SPI092
[BSW005] No hard coded horizontal interfaces within MCAL	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW006] Platform independency	Not applicable (requirement on implementation, not on specification)
[BSW007] HIS MISRA C	Not applicable (requirement on implementation, not on specification)
[BSW009] Module User Documentation	Not applicable (requirement on implementation, not on specification)

[BSW010] Memory resource documentation	Not applicable (requirement on implementation, not on specifica-tion)
[BSW101] Initialization interface	SPI013 SPI015
[BSW158] Separation of configuration from im-plementation	SPI103 SPI091 SPI089 SPI095
[BSW159] Tool-based configuration	Both static and runtime configuration parameters are located outside the source code of the mod-ule. This is the prerequisite for automatic configu-ration.
[BSW160] Human-readable configuration data	Requirement on configuration methodology and tools
[BSW161] Microcontroller abstraction	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW162] ECU layout abstraction	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW164] Implementation of interrupt service routines	Not applicable (Cannot be detailed at this point of time, because this depends on module implementation.)
[BSW167] Static configuration checking	Requirement on configuration tool
[BSW168] Diagnostic Interface of SW components	Not applicable (no use case)
[BSW170] Data for reconfiguration of AUTOSAR SW-Components	Not applicable (requirement on SW Component)
[BSW171] Configurability of optional functionality	Conflicts partly with SPAL requirement [BSW12263] Configuration after compile time.
[BSW172] Compatibility and documentation of scheduling strategy	Not applicable (requirement on implementation, not on specifica-tion)

Document: AUTOSAR requirements on Basic Software, cluster SPAL

Requirement	***Satisfied by***
[BSW12263] Object code compatible configuration concept	SPI076
[BSW12056] Configuration of notification mecha-nisms	SPI009 SPI064 SPI044 SPI054
[BSW12267] Configuration of wake-up sources	Not applicable. (Only master mode is supported. Master mode does not provide wake up events.)
[BSW12057] Driver module initialization	SPI013 SPI015
[BSW12125] Initialization of hardware resources	SPI013 SPI008 SPI009
[BSW12163] Driver module deinitialization	SPI021 SPI022
[BSW12461] Responsibility for register initialization	See chapter 5
[BSW12462] Provide settings for register initialization	Cannot be detailed at this point of time, because this depends on SPI hardware and implementa-tion.
[BSW12463] Combine and forward settings for register initialization	Cannot be detailed at this point of time (see above)
[BSW12068] MCAL initialization sequence	Not applicable (this is a general software integration requirement)
[BSW12069] Wake-up notification of ECU State Manager	Not applicable (the SPI does not cause any wake-ups)
[BSW157] Notification mechanisms of drivers and	SPI026 SPI038 SPI039 SPI042 SPI057 SPI071

handlers	SPI073 SPI075
[BSW12169] Control of operation mode	Chapter 9.2
[BSW12063] Raw value mode	Not applicable (no I/O functionality)
[BSW12075] Use of application buffers	SPI053
[BSW12129] Resetting of interrupt flags	No Applicable to the Handler API but shall be define for the Driver API.
[BSW12064] Change of operation mode during running operation	Chapter 9.2, SPI025 SPI021
[BSW12448] Behavior after development error detection	Chapters 7.5.1 and 7.5.2
[BSW12067] Setting of wake-up conditions	Not applicable (the SPI resource does not cause any wake-ups)
[BSW12077] Non-blocking implementation	Not applicable (requirement on implementation, not on specification)
[BSW12078] Runtime and memory efficiency	Not applicable (requirement on implementation, not on specification)
[BSW12092] Access to drivers	Not applicable (requirement on implementation, not on specification)
[BSW12265] Configuration data shall be kept constant	Not applicable (requirement on implementation, not on specification)
[BSW12264] Specification of configuration items	Chapter 10.2

Document: AUTOSAR requirements on Basic Software, SPI Handler/Driver

Requirement	***Satisfied by***
[BSW12093] SPI Channel support	SPI009 SPI010 SPI034 SPI041
[BSW12094] Chip select	SPI009 SPI066
[BSW12256] Support of all Controller Peripherals	SPI008 SPI009 SPI034
[BSW12257] Support of chained HW devices	SPI008 SPI063 SPI009 SPI010 SPI034 SPI065 SPI066
[BSW13400] Scalable functionality	SPI110 Chapters 7.2.1 and 7.2.4
[BSW12025] Configuration of SPI general SW and HW properties	SPI008 SPI009 SPI063 SPI052 SPI053
[BSW12179] SPI Channel linkage	SPI009 SPI003 SPI064 SPI065
[BSW12026] Assignment of SPI Channel to SPI HW Unit	SPI009
[BSW12197] Definition of data width	SPI063
[BSW13401] Statically configurable functionalities	SPI109 SPI111 SPI121 SPI122 SPI125
[BSW12258] Data shall be accessible device individually	SPI003 SPI065 SPI009
[BSW12259] Support of different timing and HW parameters	SPI009
[BSW12260] Support of different priorities of sequences	SPI009 SPI064 SPI002 SPI014 SPI059 SPI093
[BSW12180] Handling of single SPI channels	SPI003 SPI065
[BSW12181] Handling of linked SPI channels	SPI065 SPI055
[BSW12032] Chip select mode – normal mode	SPI009 SPI066
[BSW12033] Chip select mode – hold mode	SPI009 SPI066
[BSW12198] Transfer one short data sequence with variable data	SPI053 SPI077
[BSW12253] Transfer one short data sequence with constant data	SPI052 SPI078
[BSW12199] Transfer data to several devices in	SPI065 SPI003 SPI064

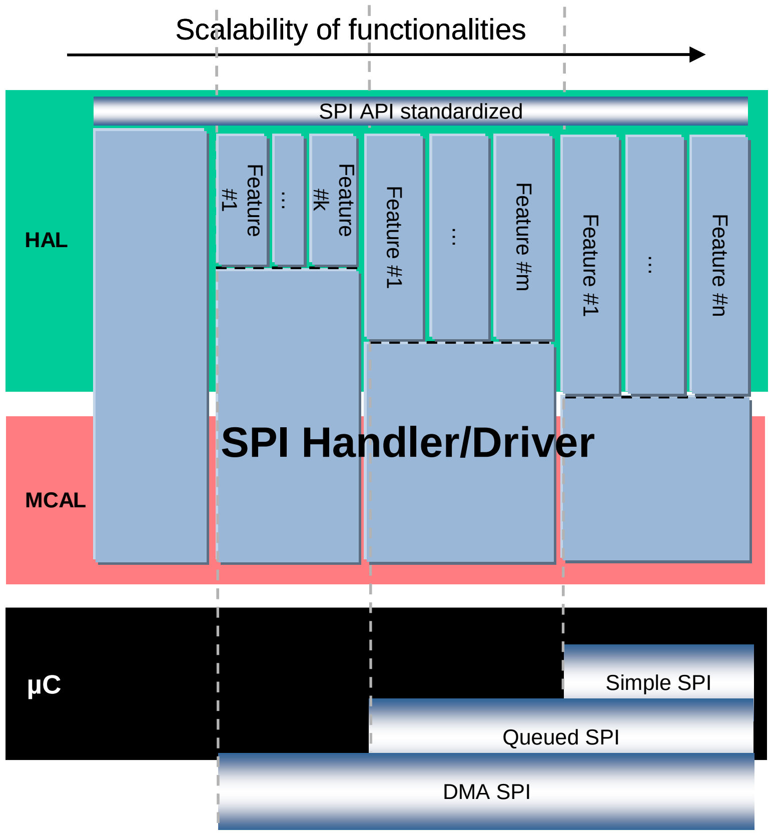
one Sequence	
[BSW12200] Read large data sequences from one slave device using dummy send data	SPI053 SPI065 SPI003 SPI035 SPI077
[BSW12261] Read large data sequences from one slave device using variable send data	SPI053 SPI065 SPI003
[BSW12201] Read large data sequences from several slave devices using dummy send data	SPI065 SPI003 SPI035 SPI077
[BSW12262] Read large data sequences from several slave devices using variable send data	SPI053 SPI065 SPI003 SPI078
[BSW12202] Support of variable data length	SPI053 SPI078
[BSW12024] Configuration of SPI HW Unit	SPI008 SPI063
[BSW12150] Configuration of SPI asynchronous SW and HW properties	SPI009 SPI064 SPI093
[BSW12108] Callback notification	Chapter 8.6.3 SPI057 SPI118 SPI119 SPI120
[BSW12099] Asynchronous Read Functionality	SPI020 SPI162 SPI163 SPI016 SPI020
[BSW12101] Asynchronous Write Functionality	SPI020 SPI162 SPI163 SPI018 SPI020
[BSW12103] Asynchronous Read-Write Functionality	SPI020 SPI053 SPI058 SPI067
[BSW12037] Job Management Strategy – Priority controlled	Chapter 7.2.3, 7.2.4 and 7.3 SPI014 SPI059 SPI124 SPI127
[BSW12104] SPI status functionality	SPI025 SPI026 SPI039
[BSW12170] Concurrent Channel access	SPI042 SPI084
[BSW12152] Synchronous Read Function	Chapter 7.2.2 SPI134 SPI016
[BSW12153] Synchronous Write Function	Chapter 7.2.2 SPI134 SPI018
[BSW12154] Synchronous Write-Read Function	Chapter 7.2.2 SPI134
[BSW12151] Job Management Strategy – Order of requests	Chapter 7.2.2

7 Functional specification

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

7.1 Overall view of functionalities and features

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



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

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

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

[SPI109] ⌈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.⌋(BSW13401)

[SPI110] ⌈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.⌋(BSW13400)

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.

[SPI041] ⌈The SPI Handler/Driver interface configuration shall be based on Channels, Jobs and Sequences as defined in this document (see chapter 2).⌋(BSW12093)

[SPI034] ⌈The SPI Handler/Driver shall support one or more Channels, Jobs and Sequences to drive all kind of SPI compatible HW devices.⌋(BSW12093, BSW12256, BSW12257)

[SPI255] ⌈Data transmissions shall be done according to Channels, Jobs and Sequences configuration parameters.⌋()

[SPI066] ⌈The Chip Select (CS) is attached to the Job definition.⌋(BSW12094, BSW12257, BSW12032, BSW12033)

[SPI263] ⌈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.⌋()

[SPI370] ⌈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 SPI212_Conf).⌋()

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.

[SPI065] ⌈A Job shall contain at least one Channel.⌋(BSW12257, BSW12179, BSW12258, BSW12180, BSW12181, BSW12199, BSW12200, BSW12261, BSW12201, BSW12262)

[SPI368] ⌈Each Channel shall have an associated index which is used for specifying the order of the Channel within the Job.⌋()

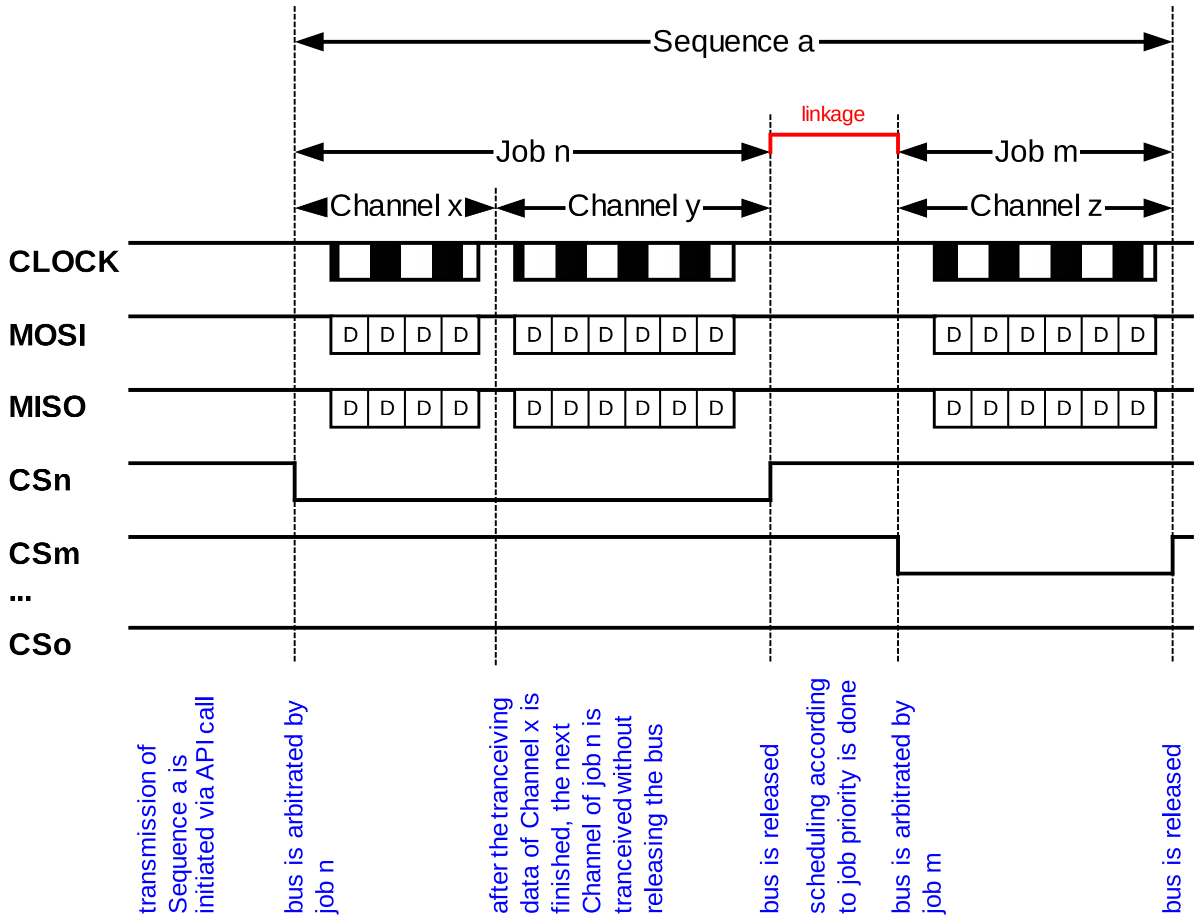
[SPI262] ⌈If a Job contains more than one Channel, all Channels contained have the same Job properties during transmission and shall be linked together statically.⌋()

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

[SPI003] ⌈A Sequence shall contain at least one Job.⌋(BSW12179, BSW12258, BSW12180, BSW12199, BSW12200, BSW12261, BSW12201, BSW12262)

[SPI236] ⌈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 [SPI028] & [SPI030]).


Sequence a
linkage
Job n
Job m
Channel x
Channel y
Channel z
CLOCK
MOSI
MISO
CSn
CSm
...
CSo
D
transmission of Sequence a is initiated via API call
bus is arbitrated by job n
after the tranceiving data of Channel x is finished, the next Channel of job n is tranceived without releasing the bus
bus is released
scheduling according to job priority is done
bus is arbitrated by job m
bus is released

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 (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`).

[SPI149] ⌈The SPI Handler/Driver shall take care of the differences between the width of channel data handled by the user and those handled by the hardware.⌋()

[SPI289] ⌈If width of channel data handled by the user and handled by the hardware is exactly the same (8 or 16 or 32 bits), the SPI Handler/Driver can send and receive data without any bit changes straightforward.⌋()

[SPI290] ⌈If width of channel data handled by the hardware is superior to data width handled by the user, means that the data transmitted through the SPI Handler/Driver shall send the lower part extended with zero. Receive the lower part, ignoring the upper part.⌋()

[SPI291] ⌈If width of channel data handled by the hardware inferior to data width handled by the user means the data transmitted through the SPI Handler/Driver shall be according to the memory alignment separate the data as two part and send and receive one by one.⌋()

This ensures that the user always gets the same interface.

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.

[SPI111] ⌈The SpiChannelBuffersAllowed parameter shall be configured with one of the 3 authorized values (0, 1 or 2) according to the described usage.⌋(BSW13401)

[SPI279] ⌈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.

[SPI052] ⌈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.⌋(BSW12025, BSW12253)

[SPI049] ⌈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.⌋()

[SPI051] ⌈The channel data to be transmitted shall be copied in 1 entry deep internal buffers by channel.⌋()

[SPI257] ⌈The SPI Handler/Driver is not able to prevent the overwriting of these "transmit" buffers by users during transmissions.⌋()

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.

[SPI053] ⌈For EB Channels the application shall provide the buffering and shall take care of the consistency of the data in the buffer during transmission.⌋(BSW12075, BSW12025, BSW12198, BSW12200, BSW12261, BSW12262, BSW12202, BSW12103)

[SPI112] ⌈The size of the Channel buffer is either fixed or variable. A maximum size for the Channel buffer shall be defined by the configuration.⌋()

[SPI280] ⌈The buffer provided by the application for the SPI Handler Driver may have a different size.⌋()

7.2.1.3 Buffering channel usage

The following table provides information about the Channel characteristics:

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

7.2.2 LEVEL 0, Simple Synchronous behaviour

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

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

[SPI160] ⌈The LEVEL 0 SPI Handler/Driver shall offer a synchronous transfer service for SPI busses.⌋()

[SPI161] ⌈ 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.⌋()

[SPI294] ⌈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:

[SPI114] ⌈The LEVEL 0 SPI Handler/Driver shall accept concurrent Spi_SyncTransmit(), if the sequences to be transmitted use different bus and parameter `SPI_SUPPORT_CONCURRENT_SYNC_TRANSMIT` is enabled. This feature shall be disabled per default. That means during a Sequence on-going transmission, all requests to transmit another Sequence shall be rejected.⌋()

[SPI115] ⌈The LEVEL 0 SPI Handler/Driver behaviour shall include the common feature: Allowed Channel Buffers, which is selected.⌋()

[SPI084] ⌈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.⌋(BSW12170)

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

7.2.3 LEVEL 1, Basic Asynchronous behavior

The intention of this functionality level is to provide a Handler/Driver with a reduced set of services to handle asynchronous transmissions only. This is often the case for ECU with functions related to SPI networks having different priorities but also for ECU using low speed external devices.

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

[SPI162] ⌈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.⌋(BSW12099, BSW12101, BSW12103)

[SPI295] ⌈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.⌋()

[SPI163] ⌈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`).⌋(BSW12099, BSW12101, BSW12103)

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.

[SPI002] ⌈Jobs have priorities assigned. Jobs linked in a Sequence shall have same or de-creasing priorities. That means the first Job shall have the equal priority or the highest priority of all Jobs within the Sequence.⌋(BSW12260)

[SPI093] ⌈Priority order of jobs shall be from the lower to the higher value defined, higher value higher priority (from 0, the lower to 3, the higher, limited to 4 priority levels see [SPI009]).⌋(BSW12260, BSW12150)

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

[SPI059] ⌈The SPI Handler/Driver scheduling method shall schedule Jobs in order to send the highest priority Job first.⌋(BSW12260, BSW12037)

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

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

[SPI116] ⌈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.⌋()

[SPI117] ⌈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).⌋()

[SPI267] ⌈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.⌋()

[SPI118] ⌈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.⌋(BSW12108)

[SPI281] ⌈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.⌋()

[SPI119] ⌈When a valid notification function pointer is configured (see [SPI071]), 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).⌋(BSW12108)

[SPI120] ⌈When a valid notification function pointer is configured (see [SPI073]), the SPI Handler/Driver shall call this notification function at the end of a Sequence transmission regardless of the result of the Sequence transmission being either `SPI_SEQ_FAILED, SPI_SEQ_OK` or `SPI_SEQ_CANCELLED` (rational: avoid deadlocks or endless loops).⌋(BSW12108)

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.

[SPI121] ⌈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.⌋(BSW13401)

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.

[SPI122] ⌈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"). ⌋(BSW13401)

[SPI123] ⌈When the SPI Handler/Driver is configured not allowing interruptible Sequences, all Sequences declared are considered as Non-Interruptible Sequences[2].⌋()

[SPI282] ⌈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.⌋()

[SPI124] ⌈According to [SPI116] and [SPI122] requirements, the SPI Handler/Driver is not allowed to suspend a Sequence transmission already started in favour of another Sequence.⌋(BSW12037)

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.

[SPI125] ⌈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").⌋(BSW13401)

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

[SPI126] ⌈**When** the SPI Handler/Driver is configured allowing interruptible Sequences, all Sequences declared shall have their dedicated parameter `SpiInterruptibleSequence` (see SPI064 & SPI106) to identify whether the Sequence can be suspended during transmission.⌋()

[SPI014] ⌈In case of a Sequence configured as Interruptible Sequence and according to [SPI125] 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 SPI002 & SPI093). 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.⌋(BSW12260, BSW12037)

[SPI127] ⌈In case of a Sequence configured as Non-Interruptible Sequence and according to requirement [SPI125], the SPI Handler/Driver is not allowed to suspend this already started Sequence transmission in favour of another Sequence.⌋(BSW12037)

[SPI080] ⌈When using Interruptible Sequences, the caller must be aware that if the multiple Sequences access the same Channels, the data for these Channels may be overwritten by the highest priority Job accessing each Channel.⌋()

7.2.5 LEVEL 2, Enhanced behaviour

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

Handling asynchronous and synchronous transmissions means that the microcontroller for which this software module is dedicated has to provide more than one SPI bus (see [SPI108]). In fact, the goal is to support SPI buses using a so-called synchronous driver and to support other SPI buses using a so-called asynchronous driver.

[SPI128] ⌈The LEVEL 2 SPI Handler/Driver shall offer a synchronous transfer service for a dedicated SPI bus and it shall also offer an asynchronous transfer service for other SPI buses.⌋()

[SPI283] ⌈In LEVEL 2 if there is no on going Sequence transmission, the SPI Handler/Driver shall be in idle state (SPI_IDLE).⌋()

[SPI129] ⌈The SPI bus dedicated for synchronous transfers is prearranged. It means, that the bus is selected for synchronous transfers. The selected bus shall be published by supplier of this software module.⌋()

This functionality level, based on a mixed usage of synchronous transmission on one prearranged SPI bus and asynchronous transmission on others, generates restrictions on configuration and usage of Sequences and Jobs.

[SPI130] ⌈The so-called synchronous Sequences shall only be composed of Jobs that are associated to the prearranged SPI bus. These Sequences shall be used with synchronous services[3] only.⌋()

[SPI131] ⌈Jobs associated with the prearranged SPI bus shall not belong to Sequences containing Jobs associated with another SPI bus. In other words, mixed Sequences (synchronous with asynchronous Jobs) shall not be allowed.⌋()

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.

[SPI155] ⌈The SPI Handler/Driver LEVEL 2 shall implement one polling mechanism mode and one interrupt mechanism mode for SPI busses handled asynchronously.⌋()

[SPI156] ⌈Both the polling mechanism and interrupt mechanism modes for SPI busses shall be selectable during execution time (see [SPI188]).⌋()

[SPI140] ⌈If `SpiHwUnitSynchronous` is set to "Synchronous" for a job, the associated bus defined by `SpiHwUnit` behave same as prearranged bus. It means that all requirements valid for prearranged bus will be valid also for the bus assigned to this job.⌋()

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

7.3 Scheduling Advices

For asynchronous levels, LEVEL 1 and LEVEL 2, the SPI Handler/Driver can call end notification functions at the end of a Job and/or Sequence transmission (see [SPI118]). In a second time, in case of interruptible Sequences (that could be suspended), if another Sequence transmit request is pending, a rescheduling is also

[3] The second part of this requirement is aim at SPI Handler/Driver users. But, it is up to the software module supplier to implement mechanisms in order to prevent potential misuses by users.

done by the SPI Handler/Driver in order to elect the next Job to transmit (see [SPI014]).

[SPI088] ⌈For asynchronous levels, LEVEL 1 and LEVEL 2, the SPI Handler/Driver can call end notification functions at the end of a Job.⌋()

[SPI268] ⌈For asynchronous levels, LEVEL 1 and LEVEL 2, the SPI Handler/Driver can call end notification functions at the end of a Sequence transmission.⌋()

[SPI269] ⌈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.⌋()

[SPI270] ⌈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.⌋()

[SPI271] ⌈In case call end notification function and rescheduling are fully done by hardware, the order could not be configured as required; the order shall be completely documented.⌋()

7.4 Error classification

[SPI004] ⌈SPI Handler/driver shall be able to detect the error SPI_E_PARAM_CHANNEL(0x0A) when API service called with wrong parameter.⌋(BSW00327, BSW00337, BSW00385)

[SPI237] ⌈SPI Handler/driver shall be able to detect the error SPI_E_PARAM_JOB(0x0B) when API service called with wrong parameter.⌋()

[SPI238] ⌈SPI Handler/driver shall be able to detect the error SPI_E_PARAM_SEQ(0x0C) when API service called with wrong parameter.⌋()

[SPI240] ⌈SPI Handler/driver shall be able to detect the error SPI_E_PARAM_LENGTH(0x0D) when API service called with wrong parameter.⌋()

[SPI241] ⌈SPI Handler/driver shall be able to detect the error SPI_E_PARAM_UNIT(0x0E) when API service called with wrong parameter.⌋()

[SPI242] ⌈SPI Handler/driver shall be able to detect the error SPI_E_UNINIT(0x1A) when API service used without module initialization.⌋()

[SPI243] ⌈SPI Handler/driver shall be able to detect the error SPI_E_SEQ_PENDING(0x2A) when services called in a wrong sequence.⌋()

[SPI245] ⌈SPI Handler/driver shall be able to detect the error SPI_E_SEQ_IN_PROCESS(0x3A) when synchronous transmission service called at wrong time.⌋()

[SPI246] ⌈SPI Handler/driver shall be able to detect the error SPI_E_ALREADY_INITIALIZED(0x4A) when API SPI_Init service called while the SPI driver has already been initialized time.⌋()

[SPI195] ⌈SPI Handler/driver shall be able to detect the error SPI_E_HARDWARE_ERROR when an hardware error occur during asynchronous transmit. Please see also SPI267.⌋()

The table below summarize the errors that the SPI Handler/Driver shall be able to detect.

Type or error	***Relevance***	***Related error code***	***Value(hex)***
API service called with wrong parameter	Development	SPI_E_PARAM_CHANNEL SPI_E_PARAM_JOB SPI_E_PARAM_SEQ SPI_E_PARAM_LENGTH SPI_E_PARAM_UNIT	0x0A 0x0B 0x0C 0x0D 0x0E
APIs called with a Null Pointer	Development	SPI_E_PARAM_POINTER	0x10
API service used without module initialization	Development	SPI_E_UNINIT	0x1A
Services called in a wrong sequence	Development	SPI_E_SEQ_PENDING	0x2A
Synchronous transmission service called at wrong time	Development	SPI_E_SEQ_IN_PROCESS	0x3A
API SPI_Init service called while the SPI driver has already been initialized	Development	SPI_E_ALREADY_INITIALIZED	0x4A
Hardware error detected during	Production	SPI_E_HARDWARE_ERROR	Assigned by DEM

asynchronous transmit			

[SPI097] ⌈Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file Dem_IntErrId.h and included via Dem.h.⌋(BSW00337, BSW00409)

[SPI007] ⌈Additional errors that are detected because of specific implementation and/or specific hardware properties shall be added to the SPI device specific implementation description.⌋(BSW00337, BSW00385)

[SPI098] ⌈Development error values are of type `uint8`.⌋(BSW00337)

7.5 Error detection

[SPI005] ⌈The detection of all development errors is configurable at pre-compile time.⌋(BSW00350, BSW00369, BSW00386)

[SPI249] ⌈The detection of all development errors is configurable as (ON/OFF) by the switch SpiDevErrorDetect.⌋()

[SPI250] ⌈The switch SpiDevErrorDetect shall activate or deactivate the detection of all development errors.⌋()

[SPI029] ⌈ If the switch `SpiDevErrorDetect` is enabled API parameter checking is also enabled. The detailed description of the detected errors can be found in chapter 7.5.1. ⌋(BSW00323, BSW00369, BSW00386)

[SPI099] ⌈The detection of production code errors cannot be switched off.⌋(BSW00339, BSW00421)

7.5.1 API parameter checking

[SPI031] ⌈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`. ⌋(BSW00323)

[SPI032] ⌈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.⌋(BSW00323)

[SPI254] ⌈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).⌋()

[SPI060] ⌈The API parameter Length of data shall have a value within the specified buffer maximum value. Related error value: SPI_E_PARAM_LENGTH.⌋(BSW00323)

[SPI258] ⌈ If the API parameter Length related service is not done and the return value shall be E_NOT_OK.⌋()

[SPI143] ⌈The API parameter HWUnit shall have a value within the specified range of values. Related error value: SPI_E_PARAM_UNIT.⌋()

[SPI288] ⌈If HWUnit related service is not done and the return value shall be SPI_UNINIT.⌋()

7.5.2 SPI state checking

[SPI046] ⌈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.⌋(BSW00406)

[SPI256] ⌈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).⌋()

[SPI233] ⌈
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.⌋()

7.6 Error notification

[SPI100] ⌈Detected development errors shall be reported to the error hook of the Development Error Tracer (DET) if the pre-processor switch `SpiDevErrorDetect` is set (see chapter 10).⌋(BSW00338)

[SPI006] ⌈Production relevant errors shall be reported to the Diagnostic Event Manager (DEM). They shall not be used as the return value of the called function.⌋(BSW00339, BSW00369, BSW00421)

7.7 Version check

[SPI069] ⌈`Spi.c` shall check if the correct version of `Spi.h` is included. This shall be done by a pre-processor check.⌋(BSW004)

[SPI369] ⌈The SPI module shall avoid the integration of incompatible files by the following pre-processor checks:
for included (external) header files,

- <MODULENAME>_AR_RELEASE_MAJOR_VERSION
- <MODULENAME>_AR_RELEASE_MINOR_VERSION

shall be verified.⌋(BSW004)

7.8 Debugging

[SPI363] ⌈Each variable that shall be accessible by AUTOSAR Debugging, shall be defined as global variable.⌋()

[SPI364] ⌈All type definitions of variables which shall be debugged, shall be accessible by the header file Spi.h.⌋()

[SPI365] ⌈The declaration of variables in the header file shall be such, that it is possible to calculate the size of the variables by C-"sizeof".⌋()

[SPI366] ⌈Variables available for debugging shall be described in the respective Basic Software Module Description⌋()

[SPI367] ⌈The states SPI_UNINIT, SPI_IDLE, SPI_BUSY shall be available for debugging.⌋()

8 API specification

8.1 Imported types

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

[SPI174] ⌈Dem_EventIdType shall be imported from Dem_Types.h.⌋(BSW00357)

[SPI296] ⌈Std_VersionInfoType shall be imported from Std_Types.h.⌋()

[SPI297] ⌈Std_ReturnType shall be imported from Std_Types.h.⌋()

Module	***Imported Type***
Dem	Dem_EventIdType
	Dem_EventStatusType
Std_Types	Std_ReturnType
	Std_VersionInfoType

8.2 Type definitions

8.2.1 Spi_ConfigType

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

[SPI344] ⌈The description of the type Spi_ConfigType is implementation specific and it shall be provided for external use.⌋()

[SPI008] ⌈The type `Spi_ConfigType` is an external data structure and shall contain the initialization data for the SPI Handler/Driver. It shall contain:
- MCU dependent properties for SPI HW units
- Definition of Channels
- Definition of Jobs
- Definition of Sequences⌋(BSW00405, BSW12125, BSW12256, BSW12257, BSW12025, BSW12024)

[SPI063] ⌈For the type `Spi_ConfigType`, the definition for each Channel shall contain:
- Buffer usage with EB/IB Channel
- Transmit data width (1 up to 32 bits)

- Number of data buffers for IB Channels (at least 1) or it is the maximum of data for EB Channels (a value of 0 makes no sense)
- Transfer start LSB or MSB
- Default transmit value⌋(BSW12257, BSW12025, BSW12197, BSW12024)

[SPI009] ⌈For the type `Spi_ConfigType`, the definition for each Job shall contain:

- Assigned SPI HW Unit
- Assigned Chip Select pin (it is possible to assign no pin)
- Chip select functionality on/off
- Chip select pin polarity high or low
- Baud rate
- Timing between clock and chip select
- Shift clock idle low or idle high
- Data shift with leading or trailing edge
- Priority (4 levels are available from 0, the lower to 3, the higher)
- Job finish end notification function
- MCU dependent properties for the Job (only if needed)
- Fixed link of Channels (at least one)⌋(BSW00344, BSW12056, BSW12125, BSW12093, BSW12094, BSW12256, BSW12257, BSW12025, BSW12179, BSW12026, BSW12259, BSW12258, BSW12260, BSW12032, BSW12033, BSW12150)

[SPI064] ⌈For the type `Spi_ConfigType`, the definition for each Sequence shall contain:

- Collection of Jobs (at least one)
- Interruptible or not interruptible after each Job
- Sequence finish end notification function⌋(BSW12056, BSW12179, BSW12260, BSW12199, BSW12150)

[SPI010] ⌈For the type `Spi_ConfigType`, the configuration will map the Jobs to the different SPI hardware units and the devices.⌋(BSW12093, BSW12257)

8.2.2 Spi_StatusType

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

[SPI061] ⌈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.⌋(BSW00335)

[SPI259] ⌈The type Spi_StatusType can be obtained calling the API service Spi_GetStatus.⌋()

[SPI260] ⌈The type Spi_StatusType can be obtained calling the API service Spi_GetHWUnitStatus.⌋()

[SPI011] ⌈After reset, the type `Spi_StatusType` shall have the default value `SPI_UNINIT` with the numeric value 0.⌋()

[SPI345] ⌈ API service Spi_GetStatus shall return SPI_UNINIT when the SPI Handler/Driver is not initialized or not usable.⌋()

[SPI346] ⌈API service Spi_GetStatus shall return SPI_IDLE when The SPI Handler/Driver is not currently transmitting any Job.⌋()
[SPI347] ⌈API service Spi_GetStatus shall return SPI_BUSY when The SPI Handler/Driver is performing a SPI Job transmit.⌋()

[SPI348] ⌈Spi_GetHWUnitStatus function shall return SPI_IDLE when The SPI Hardware microcontroller peripheral is not currently transmitting any Job,⌋()

[SPI349] ⌈Spi_GetHWUnitStatus function shall return SPI_BUSYwhen The SPI Hardware microcontroller peripheral is performing a SPI Job transmit.⌋()

8.2.3 Spi_JobResultType

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

[SPI062] ⌈The type Spi_JobResultType defines a range of specific Jobs status for SPI Handler/Driver.⌋(BSW00335)

[SPI261] ⌈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.⌋()

[SPI012] ⌈After reset, the type `Spi_JobResultType` shall have the default value `SPI_JOB_OK` with the numeric value 0.⌋()

[SPI350] ⌈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

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

[SPI351] ⌈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.⌋()

[SPI019] ⌈The type Spi_SeqResultType defines the range of specific Sequences status for SPI Handler/Driver.⌋(BSW00335)

[SPI251] ⌈The type Spi_SeqResultType defines about SPI Handler/Driver Sequence status and can be obtained calling the API service Spi_GetSequenceResult with the Sequence ID.⌋()

[SPI017] ⌈After reset, the type `Spi_SeqResultType` shall have the default value `SPI_SEQ_OK` with the numeric value 0.⌋()

[SPI352] ⌈Spi_GetSequenceResult function shall return SPI_SEQ_OK when the last transmission of the Sequence has been finished successfully.⌋()

[SPI353] ⌈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.⌋()

[SPI354] ⌈Spi_GetSequenceResult function shall return SPI_SEQ_FAILED when the last transmission of the Sequence has failed.⌋()

8.2.5 Spi_DataType

Name:	`Spi_DataType`		
Type:	`uint`		
Range:	`8..32 bit`	`--`	This is implementation specific but not all values may be valid within the type.This type shall be chosen in order to have the most efficient implementation on a specific microcontroller platform.
Description:	Type of application data buffer elements.		

[SPI355] ⌈Spi_DataType This defines the type of application data buffer elements. Type is uint8,uint16,uint32 and Range is 8 to 32 bit. it shall be provided for external use.⌋()

[SPI164] ⌈The type `Spi_DataType` refers to application data buffer elements.⌋()

8.2.6 Spi_NumberOfDataType

Name:	`Spi_NumberOfDataType`
Type:	`uint16`
Description:	Type for defining the number of data elements of the type Spi_DataType to send and / or receive by Channel

[SPI165] ⌈The type `Spi_NumberOfDataType` is used for defining the number of data elements of the type `Spi_DataType` to send and / or receive by Channel.⌋()

8.2.7 Spi_ChannelType

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

[SPI356] ⌈The type Spi_ChannelType specifies the identification (ID) for a Channel.⌋()

[SPI166] ⌈The type `Spi_ChannelType` is used for specifying the identification (ID) for a Channel.⌋()

8.2.8 Spi_JobType

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

[SPI357] ⌈The type Spi_JobType specifies the identification (ID) for a Job.⌋()

[SPI167] ⌈The type `Spi_JobType` is used for specifying the identification (ID) for a Job.⌋()

8.2.9 Spi_SequenceType

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

[SPI358] ⌈The type Spi_SequenceType specifies the identification (ID) for a sequence of jobs.⌋()

[SPI168] ⌈The type `Spi_SequenceType` is used for specifying the identification (ID) for a sequence of jobs.⌋()

8.2.10 Spi_HWUnitType

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

[SPI359] ⌈The type Spi_HWUnitType specifies the identification (ID) for a SPI Hardware microcontroller peripheral (unit).⌋()

[SPI169] ⌈The type `Spi_HWUnitType` is used for specifying the identification (ID) for a SPI Hardware microcontroller peripheral (unit).⌋()

8.2.11 Spi_AsyncModeType

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

[SPI360] ⌈The type Spi_AsyncModeType specifies the asynchronous mechanism mode for SPI buses handled asynchronously in LEVEL 2 and obtained by the API Spi_SetAsyncMode.⌋()

[SPI170] ⌈The type `Spi_AsyncModeType` is used for specifying the asynchronous mechanism mode for SPI busses handled asynchronously in LEVEL 2.⌋()

[SPI150] ⌈The type `Spi_AsyncModeType` is made available or not depending on the pre-compile time parameter: `SpiLevelDelivered`. This is only relevant for LEVEL 2.⌋()

[SPI361] ⌈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.⌋()

[SPI362] ⌈If API Spi_SetAsyncMode function is called by the parameter value SPI_INTERRUPT_MODE asynchronous mechanism is ensured by interrupt, so interrupts related to SPI buses handled asynchronously are enabled.⌋()

8.3 Function definitions

8.3.1 Spi_Init

[SPI175] ⌈void Spi_Init(const Spi_ConfigType* ConfigPtr)

Service name:	Spi_Init	

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

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ConfigPtr	Pointer to configuration set
Parameters (in-	None	

out):	
Parameters (out):	None
Return value:	None
Description:	Service for SPI initialization.

⌋()

[SPI298] ⌈The operation Spi_Init is Non Re-entrant.⌋()

[SPI299] ⌈The function Spi_Init provides the service for SPI initialization.⌋()

[SPI013] ⌈The function `Spi_Init` shall initialize all SPI relevant registers with the values of the structure referenced by the parameter `ConfigPtr.`⌋(BSW00405, BSW101, BSW12057, BSW12125)

[SPI082] ⌈The function `Spi_Init` shall define default values for required parameters of the structure referenced by the `ConfigPtr`. For example: all buffer pointers shall be initialized as a null value pointer.⌋()

[SPI015] ⌈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.`⌋(BSW00406, BSW101, BSW12057)

[SPI151] ⌈For LEVEL 2 (see chapter 7.2.5 and SPI103), the function `Spi_Init` shall set the SPI Handler/Driver asynchronous mechanism mode to `SPI_POLLING_MODE` by default. Interrupts related to SPI busses shall be disabled.⌋()

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

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

8.3.2 Spi_DeInit

[SPI176] ⌈Std_ReturnType 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 (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: de-initialisation command has been accepted E_NOT_OK: de-initialisation command has not been accepted
Description:	Service for SPI de-initialization.	

⌋()

[SPI300] ⌈The operation Std_ReturnType Spi_DeInit( ) is Non Re-entrant.⌋()

[SPI301] ⌈When the API Spi_DeInit has been accepted the return value of this function shall be E_OK.⌋()

[SPI302] ⌈When the API Spi_DeInit has not been accepted the return value of this function shall be E_NOT_OK.⌋()

[SPI303] ⌈The function Spi_DeInit provides the service for SPI de-initialization.⌋()

[SPI021] ⌈The function Spi_DeInit shall de-initialize SPI Handler/Driver.⌋(BSW00336, BSW12163, BSW12064)

[SPI252] ⌈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.⌋()

[SPI253] ⌈The function call Spi_DeInit shall be rejected if the status of SPI Handler/Driver is SPI_BUSY.⌋()

[SPI022] ⌈After the module de-initialization using the function `Spi_DeInit`, the SPI Handler/Driver shall set its state to `SPI_UNINIT.`⌋(BSW00336, BSW12163)

The SPI Handler/Driver shall have been initialized before the function `Spi_DeInit` is called, otherwise see [SPI046].

8.3.3 Spi_WriteIB

[SPI177] ⌈Std_ReturnType Spi_WriteIB(Spi_ChannelType Channel, const Spi_DataType* DataBufferPtr)

Service name:	Spi_WriteIB
Syntax:	`Std_ReturnType Spi_WriteIB(`

	`Spi_ChannelType Channel,`
	`const Spi_DataType* DataBufferPtr`
	`)`	

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

⌋()

[SPI304] ⌈The operation Spi_WriteIB is Re-entrant.⌋()

[SPI305] ⌈When the API Spi_WriteIB command has been accepted the function returns the value E_OK.⌋()

[SPI306] ⌈When the API Spi_WriteIB command has not been accepted the function returns the value E_NOT_OK.⌋()

[SPI307] ⌈The function Spi_WriteIB provides the service for writing one or more data to an IB SPI Handler/Driver Channel by the respective parameter.⌋()

[SPI018] ⌈The function `Spi_WriteIB` shall write one or more data to an IB SPI Handler/Driver Channel specified by the respective parameter.⌋(BSW12101, BSW12153)

[SPI024] ⌈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`.⌋()

[SPI023] ⌈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.⌋()

[SPI137] ⌈The function `Spi_WriteIB` shall be pre-compile time configurable by the parameter `SpiChannelBuffersAllowed.` This function is only relevant for Channels with IB.⌋()

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

The SPI Handler/Driver shall have been initialized before the function `Spi_WriteIB` is called, otherwise see [SPI046].

8.3.4 Spi_AsyncTransmit

[SPI178] ⌈Std_ReturnType Spi_AsyncTransmit(Spi_SequenceType Sequence)

Service name:	Spi_AsyncTransmit	

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

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

⌋()

[SPI308] ⌈The operation Std_ReturnType Spi_AsyncTransmit(Spi_SequenceType Sequence) is Re-entrant.⌋()

[SPI309] ⌈When the API Spi_AsyncTransmit command has been accepted the function shall return the value E_OK.⌋()

[SPI310] ⌈When the API Spi_AsyncTransmit command has not been accepted the function shall return the value E_NOT_OK.⌋()

[SPI311] ⌈The function Spi_AsyncTransmit provides service to transmit data on the SPI bus.⌋()

[SPI020] ⌈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. ⌋(BSW12099, BSW12101, BSW12103)

[SPI194] ⌈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.⌋()

[SPI157] ⌈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.⌋()

[SPI292] ⌈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.⌋()

[SPI293] ⌈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.⌋()

[SPI081] ⌈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.⌋()

[SPI266] ⌈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.⌋()

[SPI086] ⌈When the function `Spi_AsyncTransmit` is called and the requested Sequence shares Jobs with another sequence that is in the state `SPI_SEQ_PENDING`, the SPI Handler/Driver shall not take into account this new request and this function shall return the value `E_NOT_OK`. In this case and according to [SPI100], the SPI Handler/Driver shall report the `SPI_E_SEQ_PENDING` error.⌋()

[SPI035] ⌈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 will be transmitted. (See also [SPI028])⌋(BSW12200, BSW12201)

[SPI036] ⌈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 [SPI030])⌋()

[SPI055] ⌈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.⌋(BSW12181)

[SPI057] ⌈ At the end of a sequence transmission initiated by the function `Spi_AsyncTransmit` and if configured, the SPI Handler/Driver shall invoke the sequence notification call-back function after the last Job end notification if this one is also configured.⌋(BSW157, BSW12108)

[SPI133] ⌈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.⌋()

[SPI173] ⌈The SPI Handler/Driver's environment shall call the function `Spi_AsyncTransmit` after a function call of `Spi_SetupEB` for EB Channels or a function call of `Spi_WriteIB` for IB Channels but before the function call `Spi_ReadIB`.⌋()

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

The SPI Handler/Driver shall have been initialized before the function `Spi_AsyncTransmit` is called otherwise see [SPI046].

8.3.5 Spi_ReadIB

[SPI179] ⌈Std_ReturnType Spi_ReadIB(Spi_ChannelType Channel, Spi_DataType* DataBufferPointer)

Service name:	Spi_ReadIB	

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

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Channel ID.
Parameters (in-out):	None	
Parameters (out):	DataBufferPointer	Pointer to destination data buffer in RAM
Return value:	Std_ReturnType	E_OK: read command has been accepted E_NOT_OK: read command has not been accepted
Description:	Service for reading synchronously one or more data from an IB SPI Handler/Driver Channel specified by parameter.	

⌋()

[SPI312] ⌈The operation Spi_ReadIB is Re-entrant.⌋()

[SPI313] ⌈The function Spi_ReadIB return values E_OK: read command has been accepted.⌋()

[SPI314] ⌈The function Spi_ReadIB return values E_NOT_OK: read command has not been accepted.⌋()

[SPI315] ⌈The function Spi_ReadIB provides the service for reading synchronously one or more data from an IB SPI Handler/Driver Channel specified by parameter.⌋()

[SPI016] ⌈The function `Spi_ReadIB` shall read synchronously one or more data from an IB SPI Handler/Driver Channel specified by the respective parameter.⌋(BSW12099, BSW12152)

[SPI027] ⌈The SPI Handler/Driver's environment shall call the function `Spi_ReadIB` after a Transmit method call to have relevant data within IB Channel.⌋()

[SPI138] ⌈The function `Spi_ReadIB` is pre-compile time configurable by the parameter `SpiChannelBuffersAllowed.` This function is only relevant for Channels with IB.⌋()

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

The SPI Handler/Driver shall have been initialized before the function `Spi_ReadIB` is called otherwise see [SPI046].

8.3.6 Spi_SetupEB

[SPI180] ⌈Std_ReturnType Spi_SetupEB(Spi_ChannelType Channel, const Spi_DataType* SrcDataBufferPtr, Spi_DataType* DesDataBufferPtr, Spi_NumberOfDataType Length)

Service name:	Spi_SetupEB

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

	`Spi_NumberOfDataType Length`
	`)`	

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Channel ID.
	SrcDataBufferPtr	Pointer to source data buffer.
	DesDataBufferPtr	Pointer to destination data buffer in RAM.
	Length	Length (in bytes) of the data to be transmitted from Srcda-taBufferPtr and/or received from DesDataBufferPtr Min.: 1 Max.: Max of data specified at configuration for this channel
Parameters (in-out):	None	
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.	

⌋()

[SPI316] ⌈The operation Spi_SetupEB is Re-entrant.⌋()

[SPI317] ⌈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.⌋()

[SPI318] ⌈The function Spi_SetupEB provides the service to setup the buffers and the length of data for the EB SPI Handler/Driver Channel specified.⌋()

[SPI058] ⌈The function `Spi_SetupEB` shall set up the buffers and the length of data for the specific EB SPI Handler/Driver Channel.⌋(BSW12103)

[SPI067] ⌈The function `Spi_SetupEB` shall update the buffer pointers and length attributes of the specified Channel with the provided values.⌋(BSW12103)

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

[SPI028] ⌈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 [SPI035])⌋()

[SPI030] ⌈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 [SPI036])⌋()

[SPI037] ⌈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.⌋()

[SPI139] ⌈The function `Spi_SetupEB` is pre-compile time configurable by the parameter `SpiChannelBuffersAllowed.` This function is only relevant for Channels with EB.⌋()

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

The SPI Handler/Driver shall have been initialized before the function `Spi_SetupEB` is called otherwise see [SPI046].

8.3.7 Spi_GetStatus

[SPI181] ⌈Spi_StatusType 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 (in-out):	None	
Parameters (out):	None	
Return value:	Spi_StatusType	Spi_StatusType
Description:	Service returns the SPI Handler/Driver software module status.	

⌋()

[SPI319] ⌈The operation Spi_GetStatus is Re-entrant.⌋()

[SPI320] ⌈The function Spi_GetStatus returns the SPI Handler/Driver software module status.⌋()

[SPI025] ⌈The function `Spi_GetStatus` shall return the SPI Handler/Driver software module status.⌋(BSW12064, BSW12104)

8.3.8 Spi_GetJobResult

[SPI182] ⌈Spi_JobResultType Spi_GetJobResult(Spi_JobType Job)

Service name:	Spi_GetJobResult	

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

Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Job	Job ID. An invalid job ID will return an undefined result.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Spi_JobResultType	Spi_JobResultType

Description:	This service returns the last transmission result of the specified Job.

⌋()

[SPI321] ⌈The operation Spi_GetJobResult is Re-entrant.⌋()

[SPI322] ⌈The function Spi_GetJobResult service returns the last transmission result of the specified Job.⌋()

[SPI026] ⌈The function `Spi_GetJobResult` shall return the last transmission result of the specified Job. ⌋(BSW157, BSW12104)

[SPI038] ⌈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`).⌋(BSW157)

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

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

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

8.3.9 Spi_GetSequenceResult

[SPI183] ⌈Spi_SeqResultType Spi_GetSequenceResult(Spi_SequenceType Sequence)

Service name:	Spi_GetSequenceResult	

Syntax:	`Spi_SeqResultType Spi_GetSequenceResult(`
	`    Spi_SequenceType Sequence`
	`)`	

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

⌋()

[SPI323] ⌈The operation Spi_GetSequenceResult is Re-entrant.⌋()

[SPI324] ⌈The function Spi_GetSequenceResult shall return the last transmission result of the specified Sequence.⌋()

[SPI039] ⌈The function `Spi_GetSequenceResult` shall return the last transmission result of the specified Sequence. ⌋(BSW157, BSW12104)

[SPI042] ⌈The SPI Handler/Driver's environment shall call the function `Spi_GetSequenceResult` to inquire whether the full Sequence transmission has succeeded (`SPI_SEQ_OK`) or failed (`SPI_SEQ_FAILED`).⌋(BSW157, BSW12170)

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

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

8.3.10 Spi_GetVersionInfo

[SPI184] ⌈void Spi_GetVersionInfo(Std_VersionInfoType* versioninfo)

Service name:	Spi_GetVersionInfo	
Syntax:	`void Spi_GetVersionInfo( Std_VersionInfoType* versioninfo )`	
Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	This service returns the version information of this module.	

⌋()

[SPI325] ⌈The operation Spi_GetVersionInfo is Non Re-entrant.⌋()

[SPI326] ⌈The function Spi_GetVersionInfo service returns the version information of this module.⌋()

[SPI101] ⌈The function `Spi_GetVersionInfo` shall return the version information of this module according to the definition of `Std_VersionInfoType [13]` ⌋(BSW00407)

[SPI196] ⌈If source code for caller and callee of `Spi_GetVersionInfo` is available, the SPI Handler/Driver should realize `Spi_GetVersionInfo` as a macro, defined in the module's header file.⌋()

[SPI102] ⌈The function Spi_GetVersionInfo is pre compile time configurable by the configuration parameter SpiVersionInfoApi.⌋(BSW00407, BSW00411)

[SPI278] ⌈The function Spi_GetVersionInfo is pre compile time configurable On/Off by the configuration parameter SpiVersionInfoApi.⌋()

[SPI371] ⌈If Det is enabled, the parameter versioninfo shall be checked for being NULL. The error SPI_E_PARAM_POINTER shall be reported in case the value is a NULL pointer.⌋()

8.3.11 Spi_SyncTransmit

[SPI185] ⌈Std_ReturnType Spi_SyncTransmit(Spi_SequenceType Sequence)

Service name:	Spi_SyncTransmit	

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

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

⌋()

[SPI327] ⌈The operation Spi_SyncTransmit is Re-entrant.⌋()

[SPI328] ⌈Return value of the function Spi_SyncTransmit is E_OK: when Transmission command has been accepted.⌋()

[SPI329] ⌈Return value of the function Spi_SyncTransmit is E_NOT_OK: When Transmission command has not been accepted.⌋()

[SPI330] ⌈The function Spi_SyncTransmit provides the service to transmit data on the SPI bus.⌋()

[SPI134] ⌈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.⌋(BSW12152, BSW12153, BSW12154)

[SPI285] ⌈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.⌋()

[SPI286] ⌈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.⌋()

[SPI135] ⌈When the function `Spi_SyncTransmit` is called while a sequence is on transmission and `SPI_SUPPORT_CONCURRENT_SYNC_TRANSMIT` is disabled or another sequence is on transmition 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 [SPI114]). In this case and according to [SPI100], the SPI Handler/Driver shall report the `SPI_E_SEQ_IN_PROCESS` error.⌋()

[SPI136] ⌈The function `Spi_SyncTransmit` is pre-compile time selectable by the configuration parameter `SpiLevelDelivered`. This function is only relevant for LEVEL 0 and LEVEL 2.⌋()

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

8.3.12 Spi_GetHWUnitStatus

[SPI186] ⌈Spi_StatusType Spi_GetHWUnitStatus(Spi_HWUnitType HWUnit)

Service name:	Spi_GetHWUnitStatus

Syntax:	`Spi_StatusType Spi_GetHWUnitStatus(`
	`    Spi_HWUnitType HWUnit`
	`)`

Service ID[hex]:	0x0b
Sync/Async:	Synchronous

Reentrancy:	Reentrant	
Parameters (in):	HWUnit	SPI Hardware microcontroller peripheral (unit) ID.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Spi_StatusType	Spi_StatusType
Description:	This service returns the status of the specified SPI Hardware microcontroller peripheral.	

⌋()

[SPI331] ⌈The operation Spi_GetHWUnitStatus is Re-entrant.⌋()

[SPI332] ⌈The function Spi_GetHWUnitStatus service returns the status of the specified SPI Hardware microcontroller peripheral.⌋()

[SPI141] ⌈The function Spi_GetHWUnitStatus shall return the status of the specified SPI Hardware microcontroller peripheral.⌋()

[SPI287] ⌈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.⌋()

[SPI142] ⌈The function `Spi_GetHWUnitStatus` is pre-compile time configurable On / Off by the configuration parameter `SpiHwStatusApi`.⌋()

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

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

8.3.13 Spi_Cancel

[SPI187] ⌈void Spi_Cancel(Spi_SequenceType Sequence)

Service name:	Spi_Cancel	

Syntax:	`void Spi_Cancel(`
	`    Spi_SequenceType Sequence`
	`)`	

Service ID[hex]:	0x0c	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	Sequence	Sequence ID.
Parameters (in-out):	None	

Parameters (out):	None
Return value:	None
Description:	Service cancels the specified on-going sequence transmission.

⌋()

[SPI333] ⌈The operation Spi_Cancel is Re-entrant.⌋()

[SPI334] ⌈The function Spi_Cancel service cancels the specified on-going sequence transmission.⌋()

[SPI144] ⌈The function `Spi_Cancel` shall cancel the specified on-going sequence transmission without cancelling any Job transmission and set the sequence result to `SPI_SEQ_CANCELLED`.⌋()

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

[SPI145] ⌈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.⌋()

[SPI146] ⌈The function `Spi_Cancel` is pre-compile time configurable On / Off by the configuration parameter `SpiCancelApi`.⌋()

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

8.3.14 Spi_SetAsyncMode

[SPI188] ⌈Std_ReturnType Spi_SetAsyncMode(Spi_AsyncModeType Mode)

Service name:	Spi_SetAsyncMode	

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

Service ID[hex]:	0x0d	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Mode	New mode required.
Parameters (in-out):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Setting command has been done

		E_NOT_OK: setting command has not been accepted
Description:	Service to set the asynchronous mechanism mode for SPI busses handled asynchronously.	

⌋()

[SPI335] ⌈The operation Spi_SetAsyncMode is Non Re-entrant.⌋()

[SPI336] ⌈Return value of the function Spi_SetAsyncMode is E_OK: Setting command has been done.⌋()

[SPI337] ⌈Return value of the function Spi_SetAsyncMode is E_NOT_OK: setting command has not been accepted.⌋()

[SPI338] ⌈The function Spi_SetAsyncMode service to set the asynchronous mechanism mode for SPI buses handled asynchronously.⌋()

[SPI152] ⌈The function `Spi_SetAsyncMode` according to the given parameter shall set the asynchronous mechanism mode for SPI channels configured to behave asynchronously.⌋()

[SPI171] ⌈If the function `Spi_SetAsyncMode` is called while the SPI Handler/Driver status is `SPI_BUSY` and an asynchronous transmition 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`.⌋()

[SPI172] ⌈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`.⌋()

[SPI154] ⌈The function `Spi_SetAsyncMode` is pre-compile time selectable by the configuration parameter `SpiLevelDelivered`. This function is only relevant for LEVEL 2.⌋()

8.4 Callback notifications

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

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

8.5 Scheduled functions

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

The SPI Handler/Driver module requires a scheduled function for the management of the asynchronous mode managed with polling (see SPI361). The specified functions below exemplify how to implement them if they are needed.

8.5.1 Spi_MainFunction_Handling

[SPI189] ⌈void Spi_MainFunction_Handling (void)

Service name:	Spi_MainFunction_Handling

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

Service ID[hex]:	0x10
Timing:	FIXED_CYCLIC
Description:	--

⌋()

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

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

The SPI Handler/Driver module does not define any interface which is required to fulfill its core functionality.

8.6.2 Optional Interfaces

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

[SPI191] ⌈void Dem_ReportErrorStatus(Dem_EventIdType EventId, Dem_EventStatusType EventStatus)⌋()

[SPI339] ⌈void Det_ReportError(uint16 ModuleId, uint8 InstanceId, uint8 ApiId, uint8 ErrorId)

API function	***Description***
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by BSW modules). The interface has an asynchronous behavior, because the processing of the event is done within the Dem main function.
Det_ReportError	Service to report development errors.

⌋()

8.6.3 Configurable interfaces

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

[SPI075] ⌈The SPI Handler/Driver shall use the callback routines Spi_JobEndNotification to inform other software modules about certain states or state changes.⌋(BSW157)

[SPI264] ⌈The SPI Handler/Driver shall use the callback routines Spi_SeqEndNotification to inform other software modules about certain states or state changes.⌋()

[SPI265] ⌈For implement the call back function other modules are required to provide the routines in the expected manner.⌋()

he callback notifications `Spi_JobEndNotification` and `Spi_SeqEndNotification` as function pointers defined within the initialization data structure (`Spi_ConfigType`).⌋(BSW12056)

The callback notifications `Spi_JobEndNotification` and `Spi_SeqEndNotification` shall have no parameters and no return value.⌋(BSW00359, BSW00360, BSW00369)

[SPI054] ⌈If a callback notification is configured as null pointer, no callback shall be executed.⌋(BSW12056)

[SPI085] ⌈It is allowed to use the following API calls within the SPI callback notifications:

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

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

8.6.3.1 Spi_JobEndNotification

[SPI192] ⌈void (*Spi_JobEndNotification)( )

Service name:	(*Spi_JobEndNotification)

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

Sync/Async:	Synchronous
Reentrancy:	Reentrant
Parameters (in):	None
Parameters (in-out):	None
Parameters (out):	None
Return value:	None
Description:	Callback routine provided by the user for each Job to notify the caller that a job has been finished.

⌋()

[SPI340] ⌈The operation SpiJobEndNotification is Re-entrant.⌋()

[SPI071] ⌈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.⌋(BSW157)

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

8.6.3.2 Spi_SeqEndNotification

[SPI193] ⌈void (*Spi_SeqEndNotification)( )

Service name:	(*Spi_SeqEndNotification)

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

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

⌋()

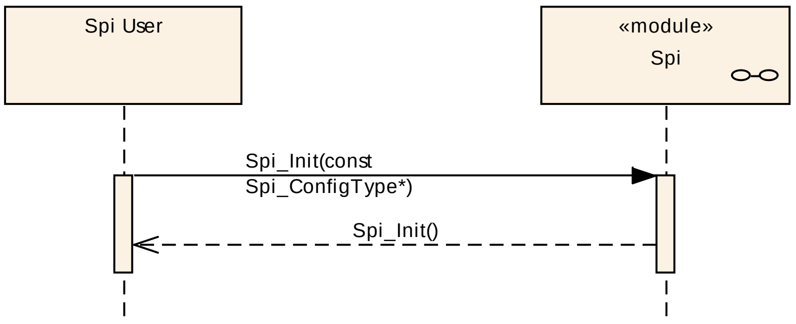
[SPI341] ⌈The operation SpiJobEndNotification is Re-entrant.⌋()

[SPI073] ⌈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.⌋(BSW157)

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

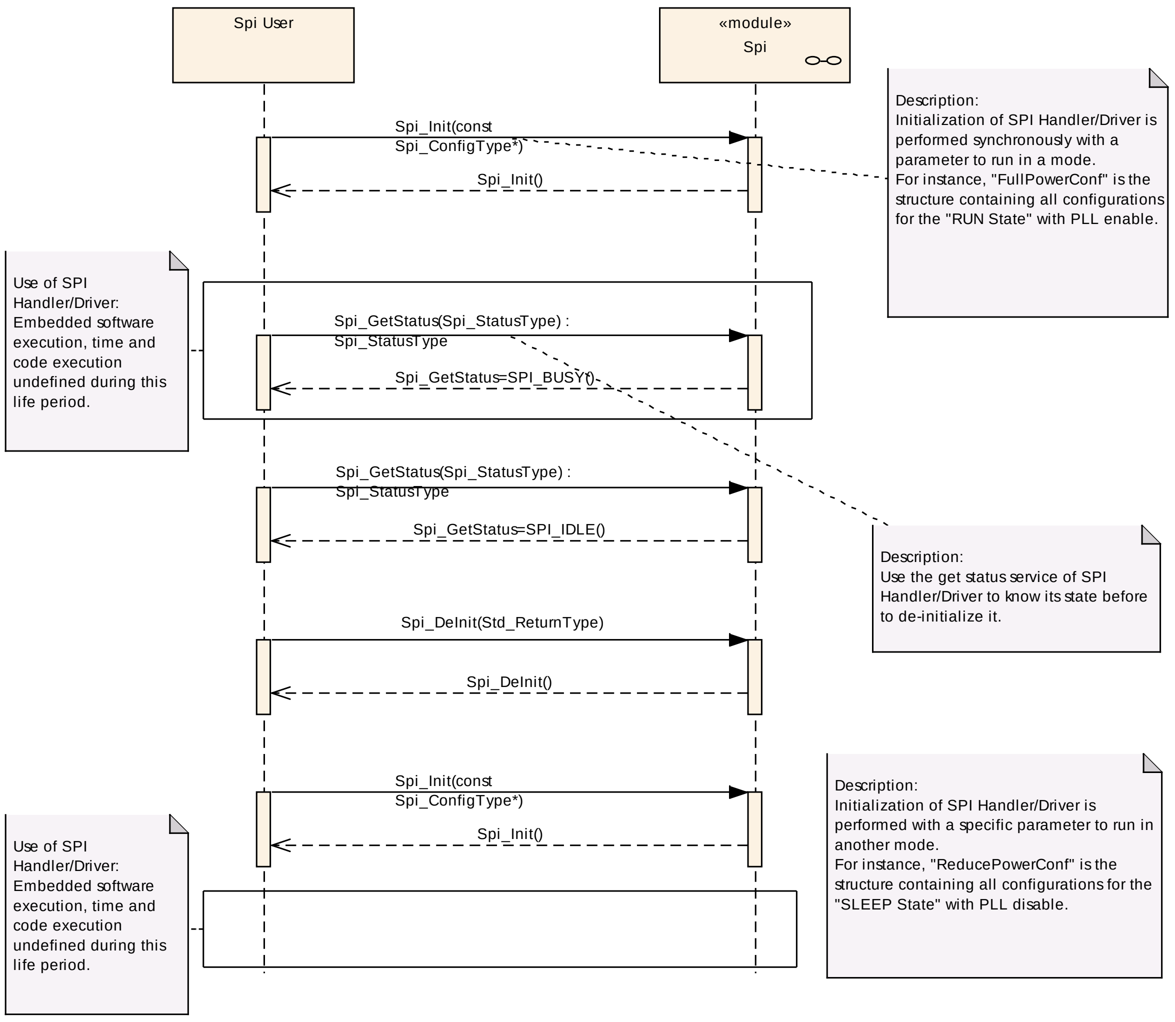
9 Sequence diagrams

9.1 Initialization



9.2 Modes transitions

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


Spi User
«module»
Spi
Spi_Init(const Spi_ConfigType*)
Spi_Init()
Description:
Initialization of SPI Handler/Driver is performed synchronously with a parameter to run in a mode.
For instance, "FullPowerConf" is the structure containing all configurations for the "RUN State" with PLL enable.
Use of SPI Handler/Driver: Embedded software execution, time and code execution undefined during this life period.
Spi_GetStatus(Spi_StatusType) : Spi_StatusType
Spi_GetStatus=SPI_BUSY()
Spi_GetStatus(Spi_StatusType) : Spi_StatusType
Spi_GetStatus=SPI_IDLE()
Description:
Use the get status service of SPI Handler/Driver to know its state before to de-initialize it.
Spi_DeInit(Std_ReturnType)
Spi_DeInit()
Spi_Init(const Spi_ConfigType*)
Spi_Init()
Description:
Initialization of SPI Handler/Driver is performed with a specific parameter to run in another mode.
For instance, "ReducePowerConf" is the structure containing all configurations for the "SLEEP State" with PLL disable.
Use of SPI Handler/Driver: Embedded software execution, time and code execution undefined during this life period.

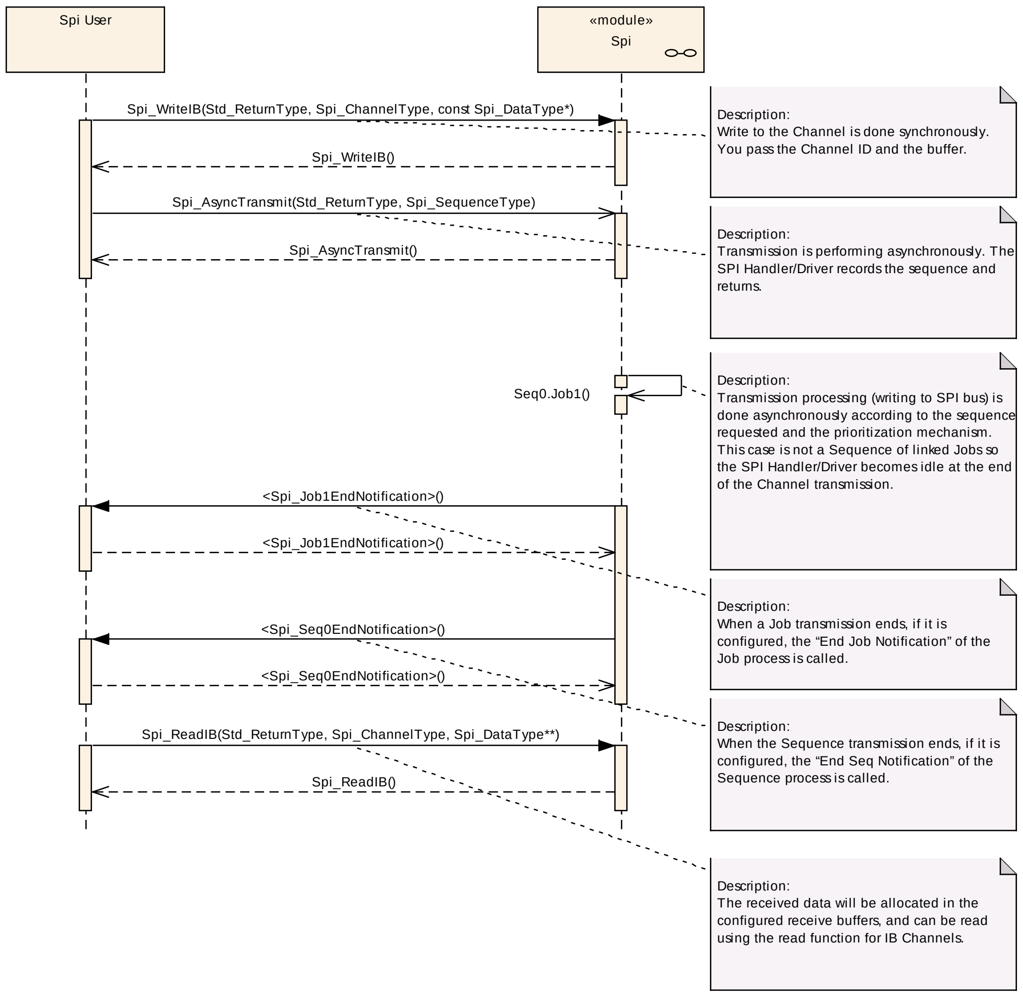
9.3 Write/AsyncTransmit/Read (IB)

9.3.1 One Channel, one Job then one Sequence

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

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

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

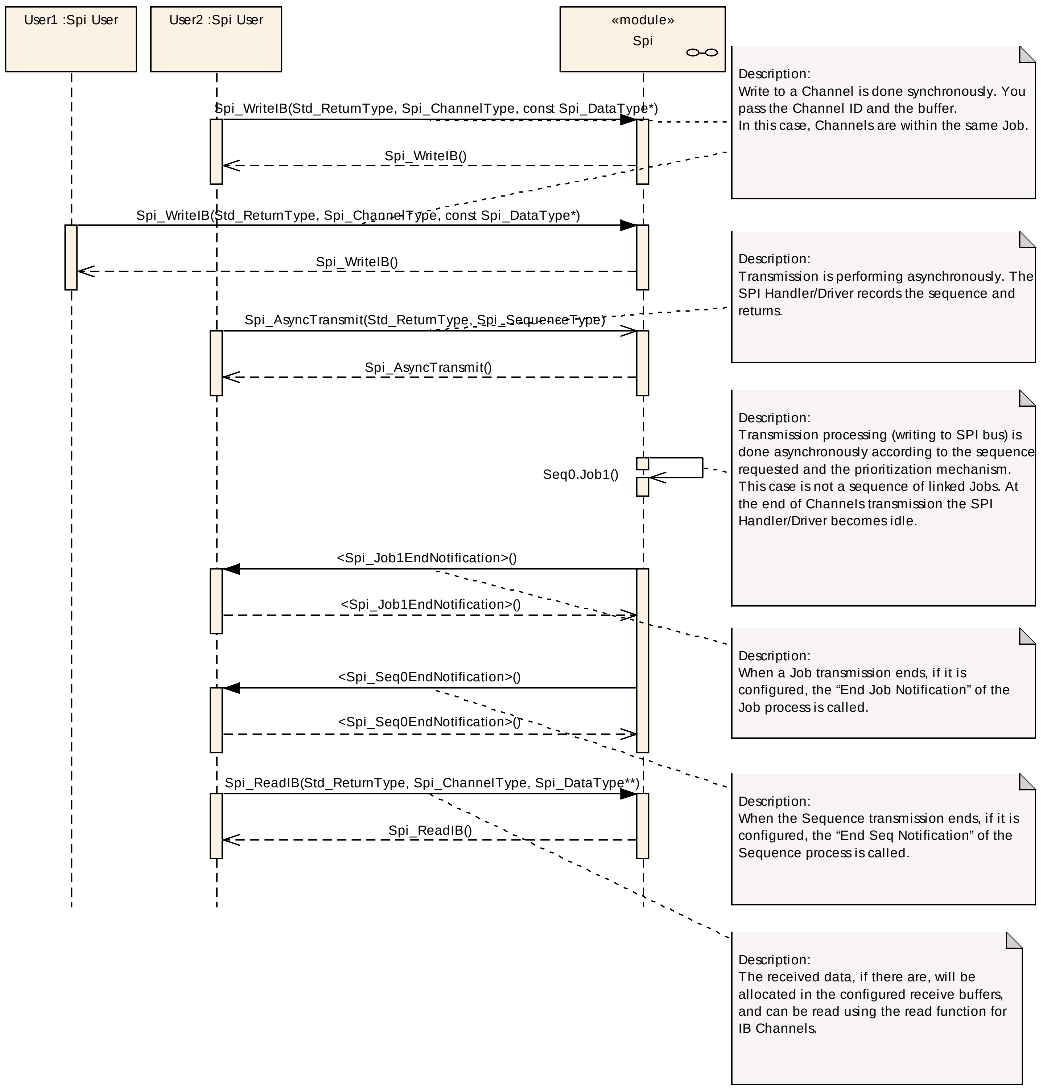

Spi User
«module»
Spi
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1()
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()
Spi_ReadIB(Std_ReturnType,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

9.3.2 Many Channels, one Job then one Sequence

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

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

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


User1 :Spi User
User2 :Spi User
«module»
Spi
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1()
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()
Spi_ReadIB(Std_ReturnType, Spi_ChannelType, Spi_DataType**)
Spi_ReadIB()

Description:
Write to a Channel is done synchronously. You pass the Channel ID and the buffer.
In this case, Channels are within the same Job.

Description:
Transmission is performing asynchronously. The SPI Handler/Driver records the sequence and returns.

Description:
Transmission processing (writing to SPI bus) is done asynchronously according to the sequence requested and the prioritization mechanism. This case is not a sequence of linked Jobs. At the end of Channels transmission the SPI Handler/Driver becomes idle.

Description:
When a Job transmission ends, if it is configured, the "End Job Notification" of the Job process is called.

Description:
When the Sequence transmission ends, if it is configured, the "End Seq Notification" of the Sequence process is called.

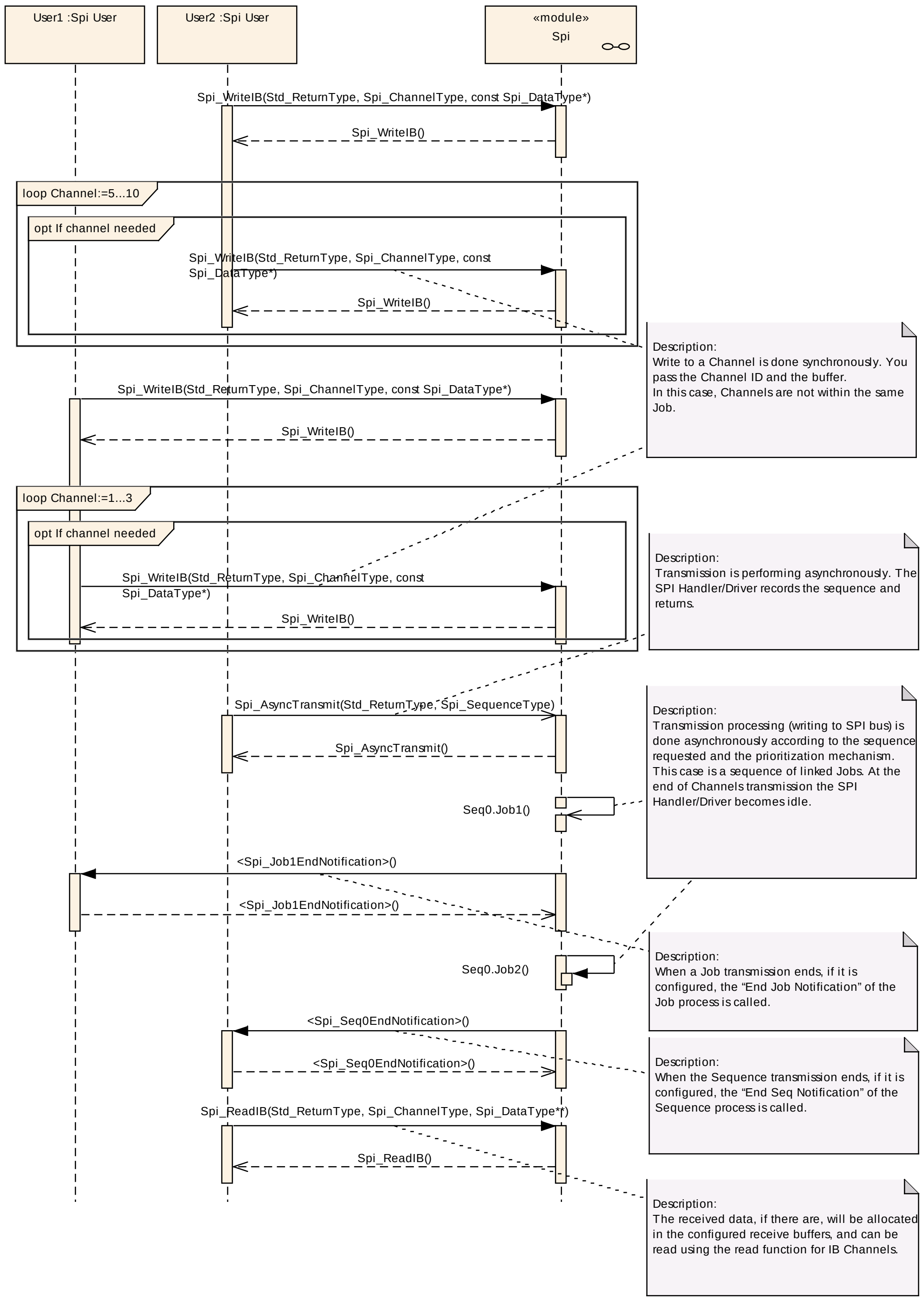
Description:
The received data, if there are, will be allocated in the configured receive buffers, and can be read using the read function for IB Channels.

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_DataType*)
Spi_WriteIB()
loop Channel:=5...10
opt If channel needed
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
loop Channel:=1...3
opt If channel needed
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1()
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
Seq0.Job2()
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()
Spi_ReadIB(Std_ReturnType, Spi_ChannelType, Spi_DataType*)
Spi_ReadIB()

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.

Description:
Transmission is performing asynchronously. The SPI Handler/Driver records the sequence and returns.

Description:
Transmission processing (writing to SPI bus) is done asynchronously according to the sequence requested and the prioritization mechanism.
This case is a sequence of linked Jobs. At the end of Channels transmission the SPI Handler/Driver becomes idle.

Description:
When a Job transmission ends, if it is configured, the "End Job Notification" of the Job process is called.

Description:
When the Sequence transmission ends, if it is configured, the "End Seq Notification" of the Sequence process is called.

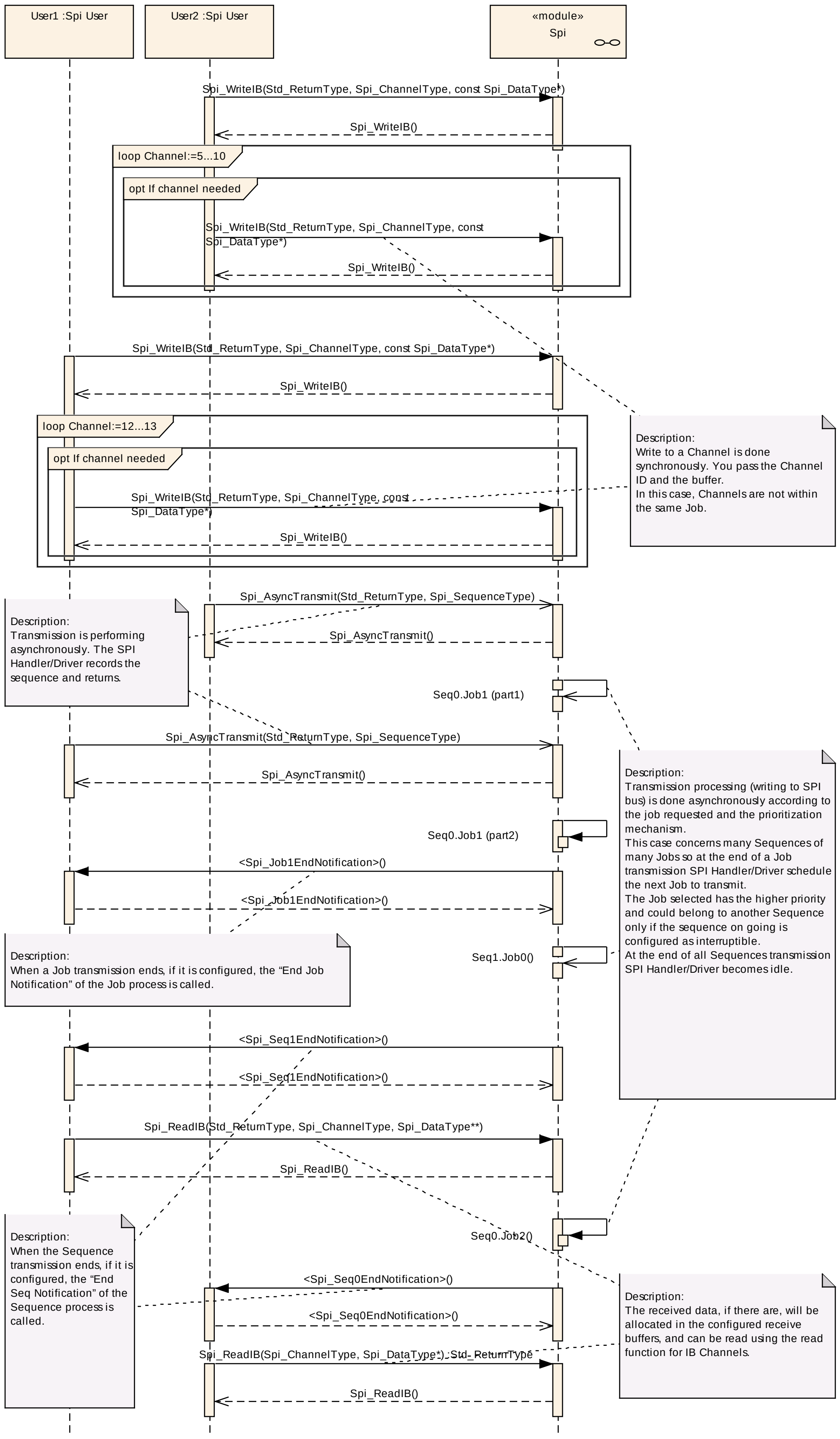
Description:
The received data, if there are, will be allocated in the configured receive buffers, and can be read using the read function for IB Channels.

9.3.4 Many Channels, many Jobs and many Sequences

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

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

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


User1 :Spi User
User2 :Spi User
«module»
Spi
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
loop Channel:=5...10
opt If channel needed
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
loop Channel:=12...13
opt If channel needed
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1 (part1)
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1 (part2)
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
Seq1.Job0()
<Spi_Seq1EndNotification>()
<Spi_Seq1EndNotification>()
Spi_ReadIB(Std_ReturnType, Spi_ChannelType, Spi_DataType**)
Spi_ReadIB()
Seq0.Job2()
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()
Spi_ReadIB(Spi_ChannelType, Spi_DataType*) :Std_ReturnType
Spi_ReadIB()

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.

Description:
Transmission is performing asynchronously. The SPI Handler/Driver records the sequence and returns.

Description:
Transmission processing (writing to SPI bus) is done asynchronously according to the 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.

Description:
When a Job transmission ends, if it is configured, the "End Job Notification" of the Job process is called.

Description:
When the Sequence transmission ends, if it is configured, the "End Seq Notification" of the Sequence process is called.

Description:
The received data, if there are, will be allocated in the configured receive buffers, and can be read using the read function for IB Channels.

9.4 Setup/AsyncTransmit (EB)

9.4.1 Variable Number of Data / Constant Number of Data

[SPI077] ⌈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.⌋(BSW12198, BSW12200, BSW12201)

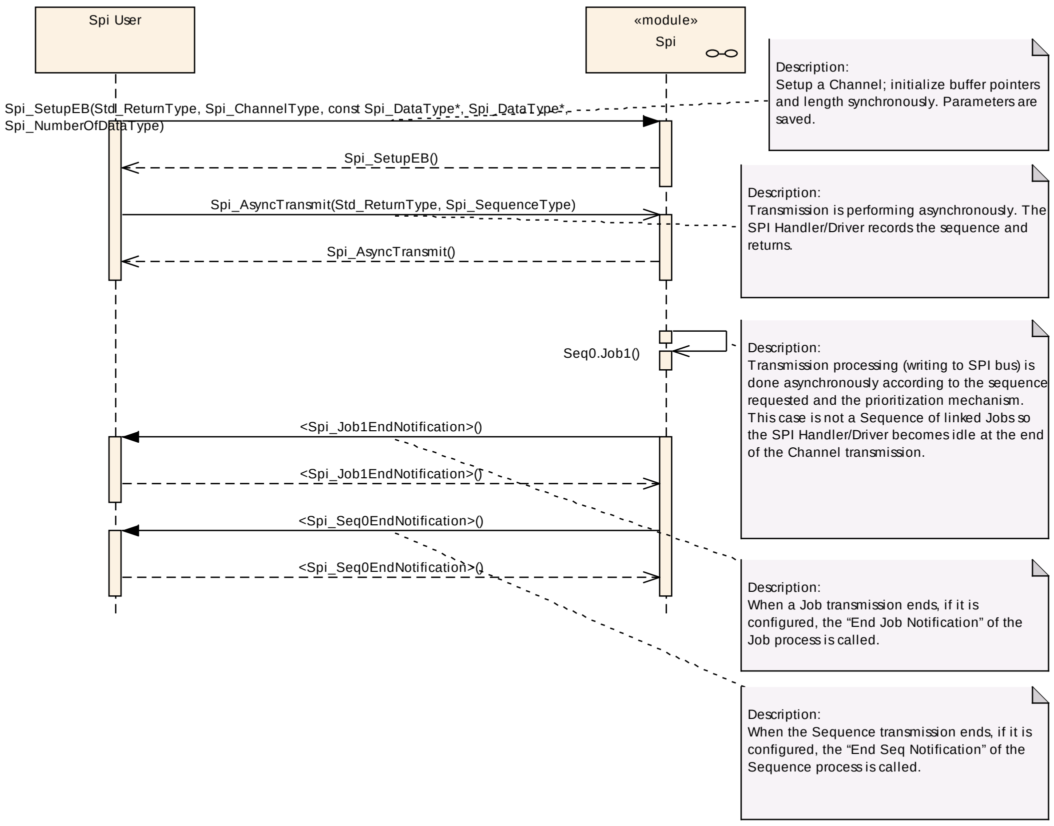
[SPI078] ⌈To transmit a constant number of data, it is only mandatory to call the `Spi_SetupEB` function to store parameters within SPI Handler/Driver before the first `Spi_AsyncTransmit` function call.⌋(BSW12253, BSW12262, BSW12202)

9.4.2 One Channel, one Job then one Sequence

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

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

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

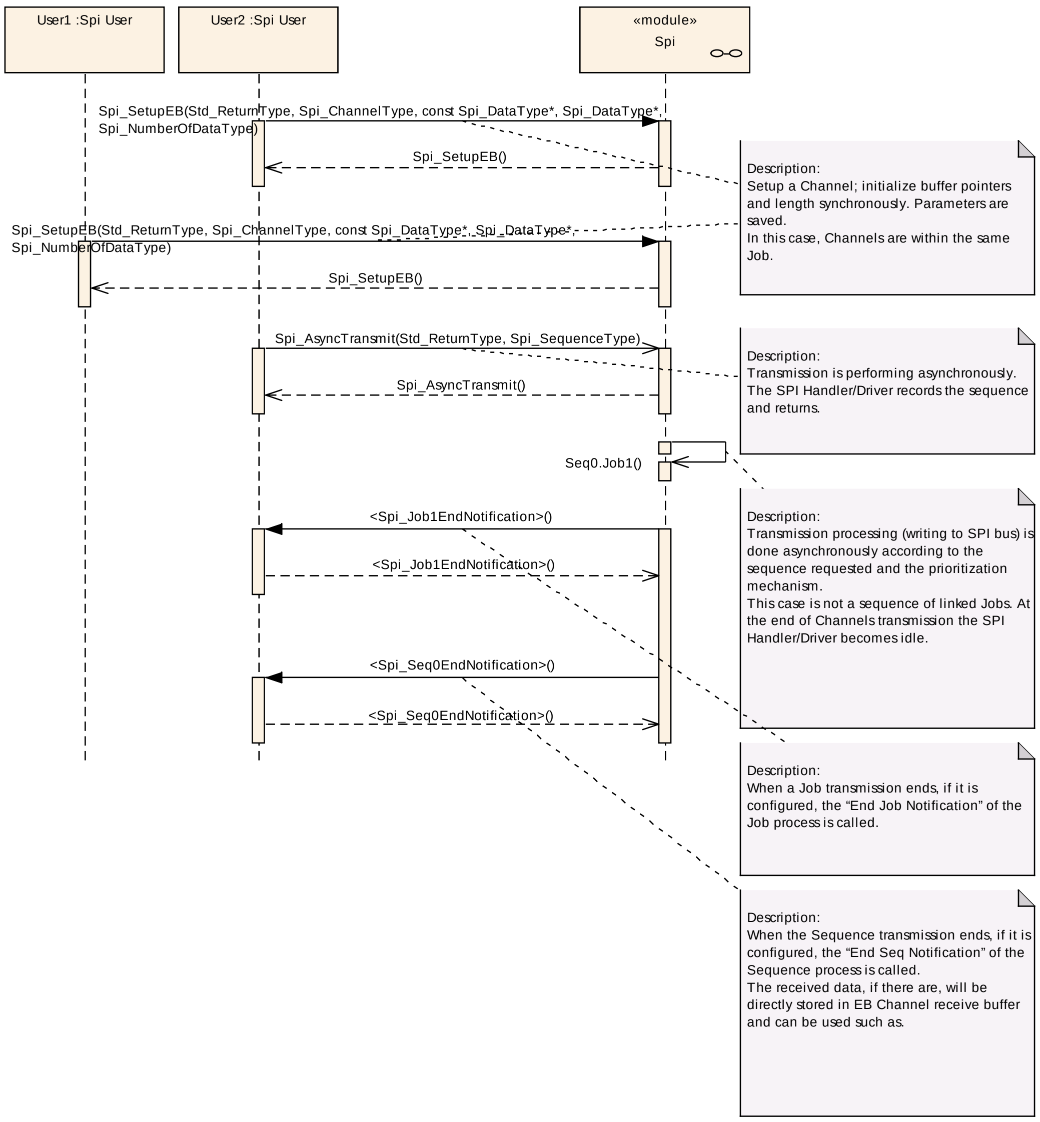

Spi User
«module»
Spi
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*, Spi_DataType*, Spi_NumberOfDataType)
Spi_SetupEB()
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1()
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
<Spi_Seq0EndNotification>()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

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


User1 :Spi User
User2 :Spi User
«module»
Spi
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*, Spi_DataType*, Spi_NumberOfDataType)
Spi_SetupEB()
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*, Spi_DataType*, Spi_NumberOfDataType)
Spi_SetupEB()
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1()
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()

Description:
Setup a Channel; initialize buffer pointers and length synchronously. Parameters are saved.
In this case, Channels are within the same Job.

Description:
Transmission is performing asynchronously. The SPI Handler/Driver records the sequence and returns.

Description:
Transmission processing (writing to SPI bus) is done asynchronously according to the sequence requested and the prioritization mechanism.
This case is not a sequence of linked Jobs. At the end of Channels transmission the SPI Handler/Driver becomes idle.

Description:
When a Job transmission ends, if it is configured, the "End Job Notification" of the Job process is called.

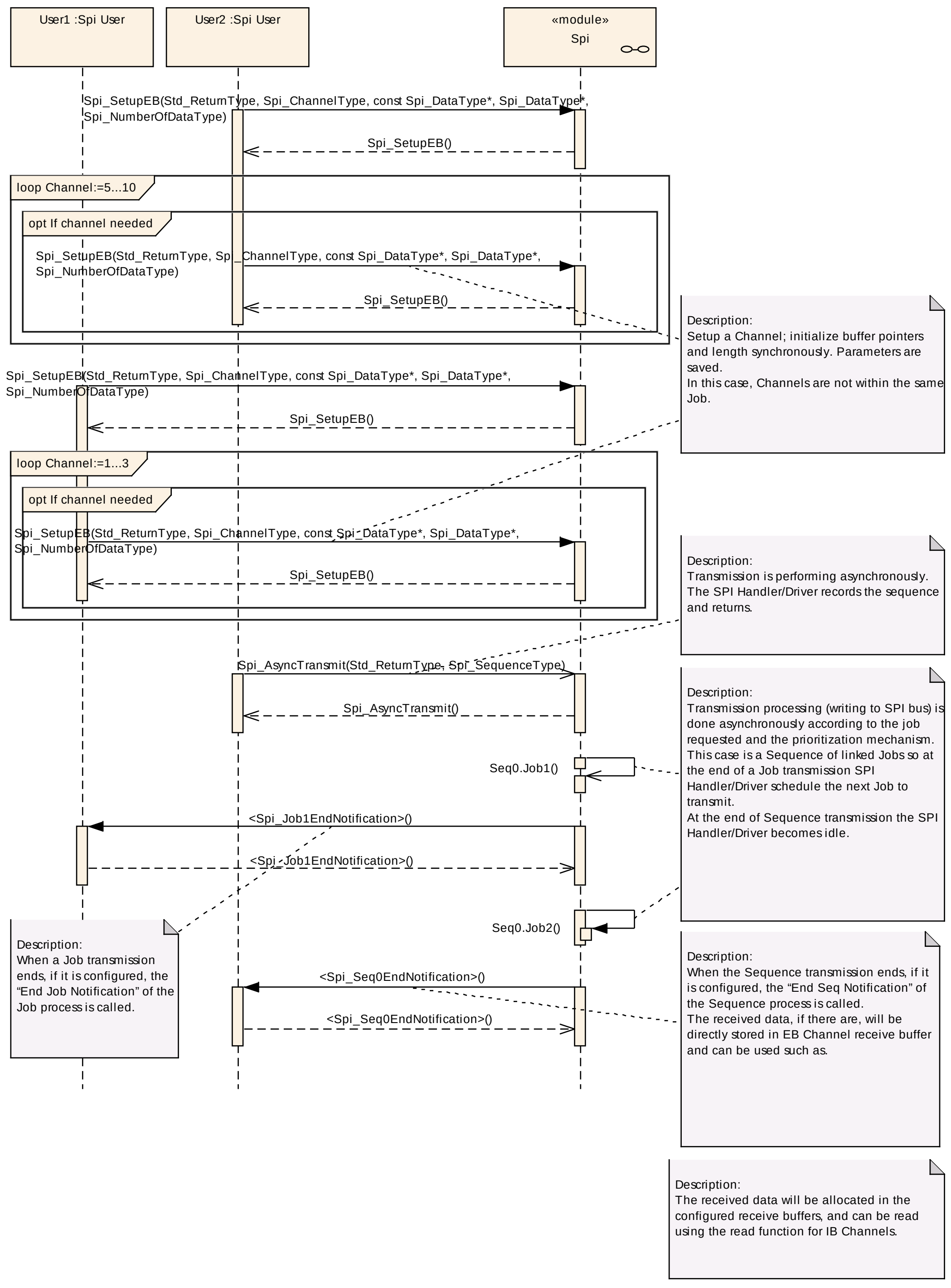
Description:
When the Sequence transmission ends, if it is configured, the "End Seq Notification" of the Sequence process is called.
The received data, if there are, will be directly stored in EB Channel receive buffer and can be used such as.

9.4.4 Many Channels, many Jobs and one Sequence

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

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

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

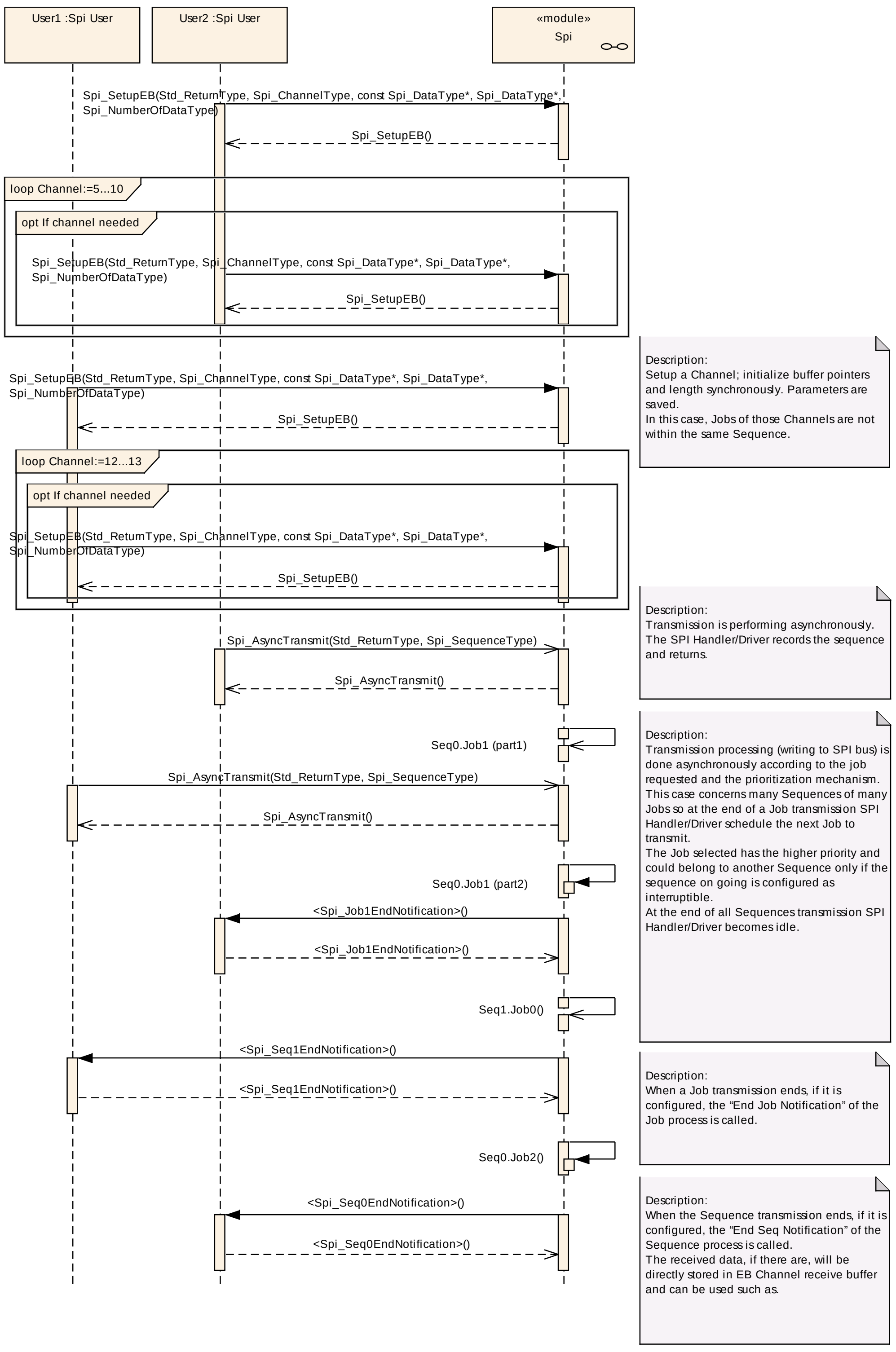

User1 :Spi User
User2 :Spi User
«module»
Spi
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*, Spi_DataType*, Spi_NumberOfDataType)
Spi_SetupEB()
loop Channel:=5...10
opt If channel needed
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*, Spi_DataType*, Spi_NumberOfDataType)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

9.4.5 Many Channels, many Jobs and many Sequences

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

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

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


User1 :Spi User
User2 :Spi User
«module»
Spi
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*, Spi_DataType*, Spi_NumberOfDataType)
Spi_SetupEB()
loop Channel:=5...10
opt If channel needed
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*, Spi_DataType*, Spi_NumberOfDataType)
Spi_SetupEB()
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*, Spi_DataType*, Spi_NumberOfDataType)
Spi_SetupEB()
loop Channel:=12...13
opt If channel needed
Spi_SetupEB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*, Spi_DataType*, Spi_NumberOfDataType)
Spi_SetupEB()
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1 (part1)
Spi_AsyncTransmit(Std_ReturnType, Spi_SequenceType)
Spi_AsyncTransmit()
Seq0.Job1 (part2)
<Spi_Job1EndNotification>()
<Spi_Job1EndNotification>()
Seq1.Job0()
<Spi_Seq1EndNotification>()
<Spi_Seq1EndNotification>()
Seq0.Job2()
<Spi_Seq0EndNotification>()
<Spi_Seq0EndNotification>()

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.

Description:
Transmission is performing asynchronously. The SPI Handler/Driver records the sequence and returns.

Description:
Transmission processing (writing to SPI bus) is done asynchronously according to the 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.

Description:
When a Job transmission ends, if it is configured, the "End Job Notification" of the Job process is called.

Description:
When the Sequence transmission ends, if it is configured, the "End Seq Notification" of the Sequence process is called.
The received data, if there are, will be directly stored in EB Channel receive buffer and can be used such as.

9.5 Mixed Jobs Transmission

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

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

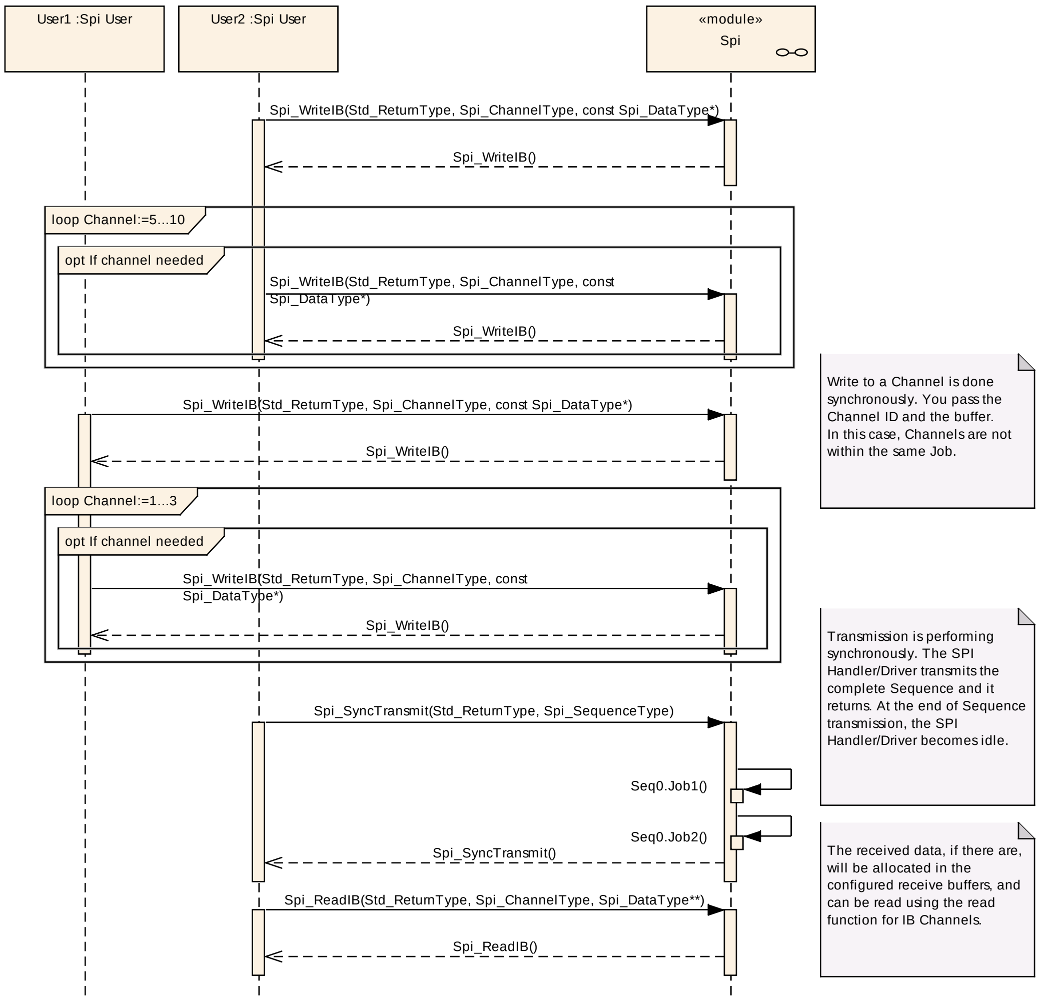
9.6 LEVEL 0 SyncTransmit diagrams

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

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

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

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

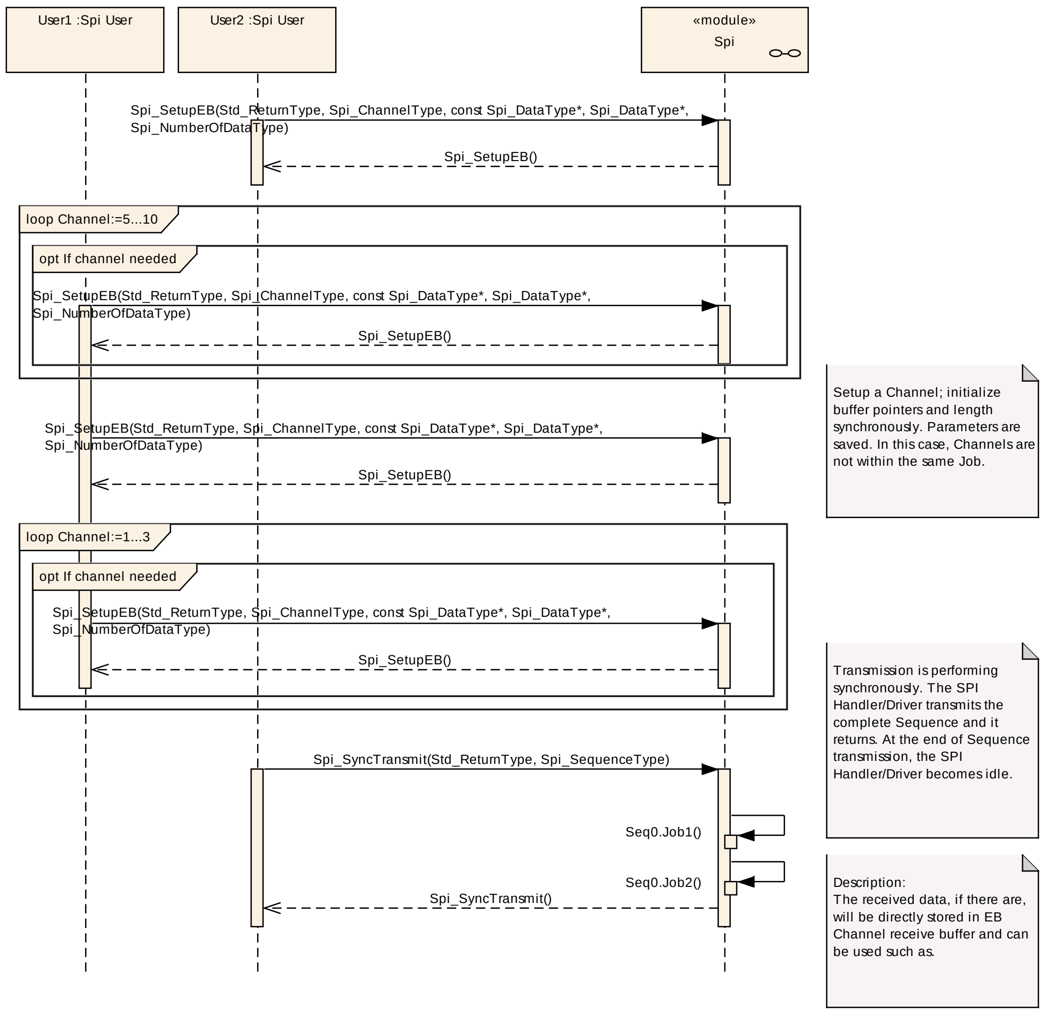

User1 :Spi User
User2 :Spi User
«module» Spi
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
loop Channel:=5...10
opt If channel needed
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
loop Channel:=1...3
opt If channel needed
Spi_WriteIB(Std_ReturnType, Spi_ChannelType, const Spi_DataType*)
Spi_WriteIB()
Spi_SyncTransmit(Std_ReturnType, Spi_SequenceType)
Seq0.Job1()
Seq0.Job2()
Spi_SyncTransmit()
Spi_ReadIB(Std_ReturnType, Spi_ChannelType, Spi_DataType**)
Spi_ReadIB()
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.
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.
The received data, if there are, will be allocated in the configured receive buffers, and can be read using the read function for IB Channels.

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

10 Configuration specification

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:
- AUTOSAR Layered Software Architecture [1]
- AUTOSAR ECU Configuration Specification [5]
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Containers

Containers structure the set of configuration parameters. This means:
- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.1.3 Specification template for configuration parameters

The following tables consist of three sections:
- the general section
- the configuration parameter section
- the section of included/referenced containers

Pre-compile time	-	specifies whether the configuration parameter shall be of configuration class *Pre-compile time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Pre-compile time*.
--	The configuration parameter shall never be of configuration class *Pre-compile time*.

Link time	-	specifies whether the configuration parameter shall be of configuration class *Link time* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Link time*.
--	The configuration parameter shall never be of configuration class *Link time*.

Post Build	-	specifies whether the configuration parameter shall be of configuration class *Post Build* or not

Label	***Description***
x	The configuration parameter shall be of configuration class *Post Build* and no specific implementation is required.
L	*Loadable* - the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* - the configuration parameter shall be of configuration class *Post Build* and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter shall never be of configuration class *Post Build*.

10.2 Containers and configuration parameters

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

10.2.1 Variants

[SPI056] ⌈VARIANT-PRE-COMPILE: Only parameters with "Pre-compile time" configuration are allowed in this variant.⌋(BSW00345, BSW00350, BSW00396, BSW00397)

[SPI076] ⌈VARIANT-LINK-TIME: Only parameters with "Pre-compile time" and "Link time" are allowed in this variant.⌋(BSW00396, BSW00398, BSW00405, BSW12263)

[SPI148] ⌈VARIANT-POST-BUILD: Parameters with "Pre-compile time", "Link time" and "Post-build time" are allowed in this variant.⌋(BSW00404, BSW00405)

[SPI234] ⌈The initialization function of this module shall always have a pointer as a parameter, even though for Variant PC no configuration set shall be given. Instead a NULL pointer shall be passed to the initialization function.⌋()

[SPI235] ⌈If not applicable, the SPI Handler/Driver module's environment shall pass a NULL pointer to the function Spi_Init.⌋()

10.2.2 Spi

SWS Item	**SPI103_Conf :**
Module Name	*Spi*
Module Description	Configuration of the Spi (Serial Peripheral Interface) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
SpiDriver	1	Configuration of one instance (if multiplicity is 1, it is the sole configuration) of an SPI driver.
SpiGeneral	1	General configuration settings for SPI-Handler
SpiPublishedInformation	1	Container holding all SPI specific published information parameters

10.2.3 SpiGeneral

SWS Item	**SPI225_Conf :**
Container Name	SpiGeneral
Description	General configuration settings for SPI-Handler
Configuration Parameters	

SWS Item	**SPI226_Conf :**		
Name	SpiCancelApi {SPI_CANCEL_API}		
Description	Switches the Spi_Cancel function ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**SPI227_Conf :**		
Name	SpiChannelBuffersAllowed {SPI_CHANNEL_BUFFERS_ALLOWED}		
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	--		
ConfigurationClass	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**SPI228_Conf :**		
Name	SpiDevErrorDetect {SPI_DEV_ERROR_DETECT}		
Description	Switches the Development Error Detection and Notification ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**SPI229_Conf :**		
Name	SpiHwStatusApi {SPI_HW_STATUS_API}		
Description	Switches the Spi_GetHWUnitStatus function ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	**Pre-compile time**	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**SPI230_Conf :**		
Name	SpiInterruptibleSeqAllowed {SPI_INTERRUPTIBLE_SEQ_ALLOWED}		
Description	Switches the Interruptible Sequences handling functionality ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	**Pre-compile time**	X	All Variants

	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module dependency: This parameter depends on SPI_LEVEL_DELIVERED value. It is only used for SPI_LEVEL_DELIVERED configured to 1 or 2.		

SWS Item	**SPI231_Conf :**		
Name	SpiLevelDelivered {SPI_LEVEL_DELIVERED}		
Description	Selects the SPI Handler/Driver level of scalable functional-ity that is available and delivered.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 2		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**SPI237_Conf :**		
Name	SpiSupportConcurrentSyncTransmit {SPI_SUPPORT_CONCURRENT_SYNC_TRANSMIT}		
Description	Specifies whether concurrent Spi_SyncTransmit() calls for different se-quences shall be configurable.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**SPI232_Conf :**		
Name	SpiVersionInfoApi {SPI_VERSION_INFO_API}		
Description	Switches the Spi_GetVersionInfo function ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

No Included Containers

10.2.4 SpiSequence

SWS Item	**SPI106_Conf :**
Container Name	SpiSequence{SpiSequenceConfiguration}
Description	All data needed to configure one SPI-sequence
Configuration Parameters	

SWS Item	**SPI222_Conf :**
Name	SpiInterruptibleSequence {SPI_INTERRUPTIBLE_SEQUENCE}
Description	This parameter allows or not this Sequence to be suspended by an-

	other one.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module dependency: This SPI_INTERRUPTIBLE_SEQ_ALLOWED parame-ter as to be configured as ON.		

SWS Item	**SPI223_Conf :**		
Name	SpiSeqEndNotification {SPI_SEQ_END_NOTIFICATION}		
Description	This parameter is a reference to a notification function.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**SPI224_Conf :**		
Name	SpiSequenceId {SPI_SEQUENCE_NAME}		
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**SPI221_Conf :**		
Name	SpiJobAssignment {SPI_JOB_LINKING}		
Description	A sequence references several jobs, which are exe-cuted during a communication sequence		
Multiplicity	1..*		
Type	Reference to [SpiJob]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers

10.2.5 SpiChannel

SWS Item	**SPI104_Conf :**
Container Name	SpiChannel{SpiChannelConfiguration}

Description	All data needed to configure one SPI-channel
Configuration Parameters	

SWS Item	**SPI200_Conf :**		
Name	SpiChannelId {SPI_CHANNEL_NAME}		
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**SPI201_Conf :**		
Name	SpiChannelType {SPI_CHANNEL_TYPE}		
Description	Buffer usage with EB/IB channel.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	EB	External Buffer	
	IB	Internal Buffer	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU dependency: SPI_CHANNEL_BUFFERS_ALLOWED		

SWS Item	**SPI202_Conf :**		
Name	SpiDataWidth {SPI_DATA_WIDTH}		
Description	This parameter is the width of a transmitted data unit.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 32		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**SPI203_Conf :**		
Name	SpiDefaultData {SPI_DEFAULT_DATA}		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**SPI204_Conf :**		
Name	SpiEbMaxLength {SPI_EB_MAX_LENGTH}		
Description	This parameter contains the maximum size (in bytes) of data buffers in case of EB Channels and only.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module 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.		

SWS Item	**SPI205_Conf :**		
Name	SpiIbNBuffers {SPI_IB_N_BUFFERS}		
Description	This parameter contains the maximum number of data buffers in case of IB Channels and only.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module dependency: The SPI_CHANNEL_TYPE parameter has to be configured as IB for this Channel. The SPI_CHANNEL_BUFFERS_ALLOWED parameter has to be configured as 0 or 2.		

SWS Item	**SPI206_Conf :**		
Name	SpiTransferStart {SPI_TRANSFER_START}		
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	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

No Included Containers

10.2.6 SpiChannelList

SWS Item	**SPI233_Conf :**
Container Name	SpiChannelList{SpiChannelList}
Description	References to SPI channels and their order within the Job.

Configuration Parameters

SWS Item	**SPI234_Conf :**		
Name	SpiChannelIndex		
Description	This parameter specifies the order of Channels within the Job.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**SPI215_Conf :**		
Name	SpiChannelAssignment {SPI_CHANNEL_LINKING}		
Description	A job reference to a SPI channel.		
Multiplicity	1		
Type	Reference to [SpiChannel]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

No Included Containers

10.2.7 SpiJob

SWS Item	**SPI105_Conf :**
Container Name	SpiJob{SpiJobConfiguration}
Description	All data needed to configure one SPI-Job, amongst others the connection between the internal SPI unit and the special settings for an external device is done.
Configuration Parameters	

SWS Item	**SPI238_Conf :**		
Name	SpiHwUnitSynchronous {SPI_HW_UNIT_SYNCHRONOUS}		
Description	If SpiHwUnitSynchronous is set to "SYNCHRONOUS", the SpiJob uses its containing SpiDriver in a synchronous manner. If it is set to "ASYNCHRONOUS", it uses the driver in an asynchronous way. If the parameter is not set, the SpiChannel uses the driver also in an asynchronous way.		
Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	ASYNCHRONOUS	--	
	SYNCHRONOUS	--	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**SPI218_Conf :**
Name	SpiJobEndNotification {SPI_JOB_END_NOTIFICATION}

Description	This parameter is a reference to a notification function.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**SPI219_Conf :**		
Name	SpiJobId {SPI_JOB_NAME}		
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**SPI220_Conf :**		
Name	SpiJobPriority {SPI_JOB_PRIORITY}		
Description	Priority set accordingly to SPI093: 0, lowest, 3, highest priority		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 3		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**SPI216_Conf :**		
Name	SpiDeviceAssignment		
Description	Reference to the external device used by this job		
Multiplicity	1		
Type	Reference to [SpiExternalDevice]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

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

10.2.8 SpiExternalDevice

SWS Item	**SPI207_Conf :**
Container Name	SpiExternalDevice
Description	The communication settings of an external device. Closely linked to SpiJob.
Configuration Parameters	

SWS Item	**SPI208_Conf :**		
Name	SpiBaudrate {SPI_BAUDRATE}		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**SPI209_Conf :**		
Name	SpiCsIdentifier {SPI_CS_IDENTIFIER}		
Description	This parameter is the symbolic name to identify the Chip Select (CS) allocated to this Job.		
Multiplicity	1		
Type	EcucStringParamDef (Symbolic Name generated for this parameter)		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**SPI210_Conf :**		
Name	SpiCsPolarity {SPI_CS_POLARITY}		
Description	This parameter defines the active polarity of Chip Select.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	HIGH	--	
	LOW	--	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**SPI239_Conf :**
Name	SpiCsSelection {SPI_CS_SELECTION}
Description	When the Chip select handling is enabled (see SpiEnableCs), then this parameter specifies if the chip select is handled automatically by Peripheral HW engine or via general purpose IO by Spi driver.

Multiplicity	0..1		
Type	EcucEnumerationParamDef		
Range	CS_VIA_GPIO	chip select handled via gpio by Spi driver.	
	CS_VIA_PERIPHERAL_ENGINE	chip select is handled automatically by Peripheral HW engine. (default)	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module dependency: SpiEnableCs		

SWS Item	**SPI211_Conf :**		
Name	SpiDataShiftEdge {SPI_DATA_SHIFT_EDGE}		
Description	This parameter defines the SPI data shift edge.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	LEADING	--	
	TRAILING	--	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**SPI212_Conf :**		
Name	SpiEnableCs {SPI_ENABLE_CS}		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**SPI217_Conf :**		
Name	SpiHwUnit {SPI_HW_UNIT}		
Description	This parameter is the symbolic name to identify the HW SPI Hardware microcontroller peripheral allocated to this Job.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CSIB0	--	
	CSIB1	--	
	CSIB2	--	
	CSIB3	--	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE

	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**SPI213_Conf :**		
Name	SpiShiftClockIdleLevel {SPI_SHIFT_CLOCK_IDLE_LEVEL}		
Description	This parameter defines the SPI shift clock idle level.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	HIGH	--	
	LOW	--	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

SWS Item	**SPI214_Conf :**		
Name	SpiTimeClk2Cs {SPI_TIME_CLK2CS}		
Description	Timing between clock and chip select (in seconds) - This parameter allows to use a range of values from 0 up to 0.0001 seconds. The real configuration-value used in software BSW-SPI is calculated out of this by the generator-tools		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. 1E-4`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: module		

No Included Containers

10.2.9 SpiDriver

SWS Item	**SPI091_Conf :**
Container Name	SpiDriver{SpiDriverConfiguration} [Multi Config Container]
Description	Configuration of one instance (if multiplicity is 1, it is the sole configuration) of an SPI driver.
Configuration Parameters	

SWS Item	**SPI197_Conf :**		
Name	SpiMaxChannel {SPI_MAX_CHANNEL}		
Description	This parameter contains the number of Channels configured. It will be gathered by tools during the configuration stage.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**SPI198_Conf :**		
Name	SpiMaxJob {SPI_MAX_JOB}		
Description	Total number of Jobs configured.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	*Pre-compile time*	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**SPI199_Conf :**		
Name	SpiMaxSequence {SPI_MAX_SEQUENCE}		
Description	Total number of Sequences configured.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	*Pre-compile time*	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Included Containers		
Container Name	*Multiplicity*	*Scope / Dependency*
SpiChannel	1..*	All data needed to configure one SPI-channel
SpiDemEventParameter-Refs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references.
SpiExternalDevice	1..*	The communication settings of an external device. Closely linked to SpiJob.
SpiJob	1..*	All data needed to configure one SPI-Job, amongst others the connection between the internal SPI unit and the special settings for an external device is done.
SpiSequence	1..*	All data needed to configure one SPI-sequence

10.2.10 SpiPublishedInformation

SWS Item	**SPI235_Conf :**
Container Name	SpiPublishedInformation
Description	Container holding all SPI specific published information parameters
Configuration Parameters	

SWS Item	**SPI236_Conf :**		
Name	SpiMaxHwUnit		
Description	Number of different SPI hardware microcontroller peripherals (units/busses) available and handled by this SPI Handler/Driver module.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default value	--		
ConfigurationClass	*Published Information*	X	All Variants

Scope / Dependency	

No Included Containers

10.3 Published information

[SPI089] ⌈The following description specifies information that is published in the module's header file `Spi.h` or in the module's description file. Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.⌋(BSW003, BSW00374, BSW00379, BSW0402, BSW158)

[SPI068] ⌈The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [3]shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [12].⌋(BSW003, BSW00318, BSW00321, BSW00374, BSW00379, BSW00390, BSW00391, BSW0402)

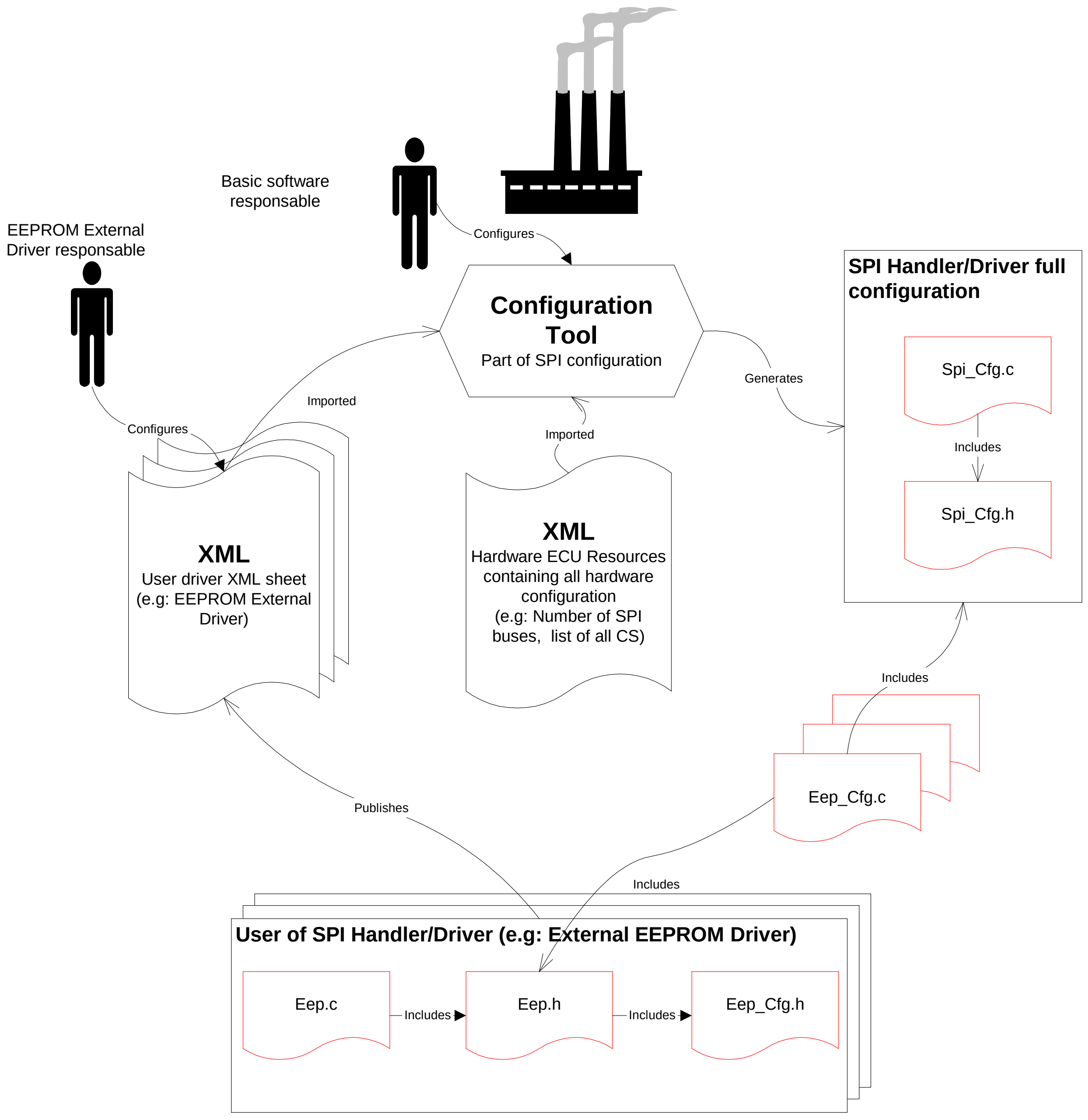
Additional module-specific published parameters are listed below if applicable."

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.


Basic software responsable
EEPROM External Driver responsable
Configures
Configures
Configuration Tool
Part of SPI configuration
Imported
Imported
Generates
SPI Handler/Driver full configuration
Spi_Cfg.c
Includes
Spi_Cfg.h
XML
User driver XML sheet (e.g: EEPROM External Driver)
XML
Hardware ECU Resources containing all hardware configuration (e.g: Number of SPI buses, list of all CS)
Includes
Eep_Cfg.c
Publishes
Includes
User of SPI Handler/Driver (e.g: External EEPROM Driver)
Eep.c
Includes
Eep.h
Includes
Eep_Cfg.h

The steps on the diagrams are:

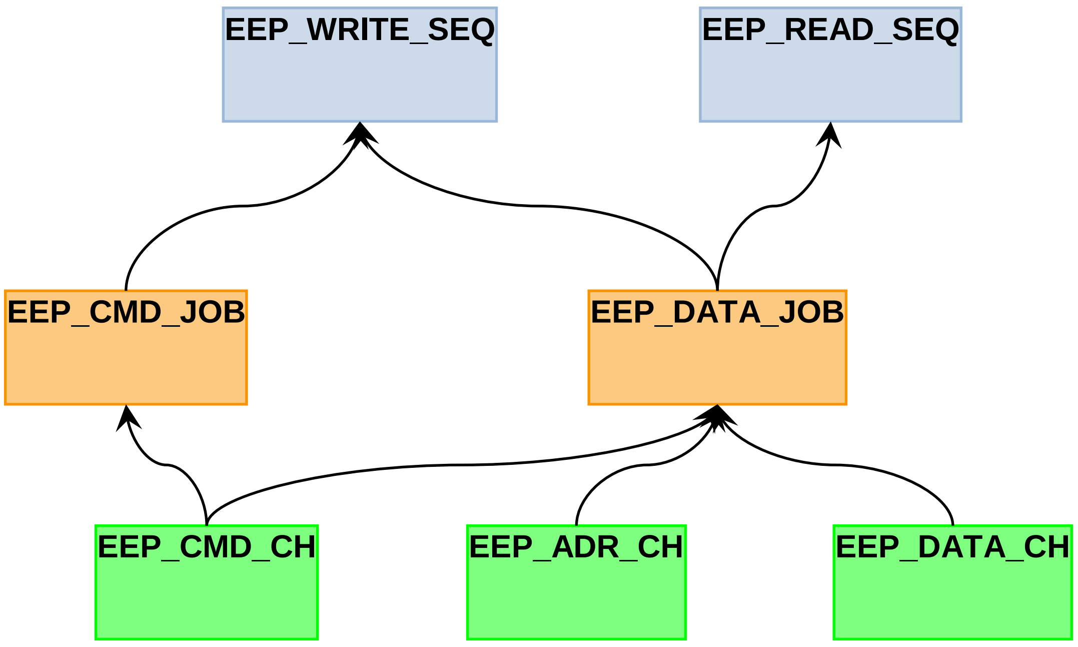
1. The user (External EEPROM Driver) of SPI Handler/Driver edits a XML configuration file. This XML configuration file is the same used by the user to generate its own configuration.
2. For each ECU, a XML HW configuration document contains information which should be used in order to configure some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

11 Not applicable requirements

[SPI999]「These requirements are not applicable to this specification.」(BSW00301, BSW00302, BSW00306, BSW00307, BSW00308, BSW00309, BSW00312, BSW00324, BSW00325, BSW00326, BSW00328, BSW00330, BSW00331, BSW00334, BSW00341, BSW00342, BSW00343, BSW00347, BSW00355, BSW00375, BSW00399, BSW00400, BSW00401, BSW00413, BSW00416, BSW00417, BSW00420, BSW00422, BSW00423, BSW00424, BSW00426, BSW00427, BSW00428, BSW00429, BSW00431, BSW00432, BSW00433, BSW00434, BSW005, BSW006, BSW009, BSW010, BSW161, BSW164, BSW168, BSW170, BSW172, BSW12267, BSW12068, BSW12069, BSW12063, BSW12129, BSW12067, BSW12077, BSW12078, BSW12092, BSW12265)

12 Appendix

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



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

13 Changes to Release 1

13.1 Deleted SWS Items

SWS Item	***Rationale***
SPI090	Redundant with the new version of SPI089

13.2 Replaced SWS Items

SWS Item of Release 1	***replaced by SWS Item***	***Rationale***
SPI056	SPI056, SPI103	To split the old requirement into two requirements to fit to the new SWS template with containers and variants.
SPI053	SPI053, SPI112	To split the old requirement into two requirements to improve the testability. Description for the maximum size of External Buffers.

13.3 Changed SWS Items

SWS Item	***Rationale***
SPI089	To take in account the new template sentence to describe requirement.
SPI029	To take in account the new template location and sentence to describe requirement.
SPI092	Clarify the structure of includes files as described in new template.
SPI076	To take in account the new SWS template with variants.
SPI091	To take in account the new SWS template with containers definitions.
SPI001	To take in account the scalabilty with Levels of Functionalities concept.
SPI014	Improvement for interruptible sequences behavior.
SPI021	Changes
SPI052	Changes
SPI031, SPI032, SPI060, SPI046	Changes to fulfill BSW12448
SPI103	After creation, add of new parameters for pre-compile time configuration
SPI044	Changed to fulfill a requirement concerning object code delivery
SPI085	To add new interfaces
SPI020	Delete the Job result setting from this service.
SPI094	Fulfill the SWS template

13.4 Added SWS Items

SWS Item	***Rationale***
SPI094	Additional requirement to identify the table of published parameters and creation of new parameters.
SPI095	New item to fullfil the required code file structure.
SPI096	New item to describe the relationship with the Dem module.
SPI097	New item to describe Dem Ids allocation rules.
SPI098	Clarify development errors C type.
SPI099	New requirement to the production errors detection.
SPI100	Clarify development errors reporting.
SPI101	New item for Spi_GetVersionInfo service description.
SPI102	New item for Spi_GetVersionInfo configuration rules.

SPI104	Creation of SpiChannel container with all its parameters.
SPI105	Creation of SpiJobConfiguration container with all its parameters.
SPI106	Creation of SpiSequence with all its parameters.
SPI108	Restriction to LEVEL 2 usage at microcontrollers with more than 1 SPI bus.
SPI109	The level is selected at pre-compile time.
SPI110	Define the parameter to configure level of functionality.
SPI111	Define the parameter to configure buffers usage IB / EB / Both.
SPI113	Global requirement for the LEVEL 0 synchronous behavior.
SPI114	Multiple sequences transmission restriction for synchronous level.
SPI115	Requirement to include buffers usage in LEVEL 0.
SPI116	Multiple sequences transmission acceptance rule for asynchronous level.
SPI117	Requirement to include buffers usage and interruptible sequences in LEV-EL 1.
SPI118	Requirement for End Notification Function.
SPI119	Additional requirement for Job end notification.
SPI120	Additional requirement for Sequence end notification.
SPI121	Define the parmeter to configure interruptible sequences.
SPI122	Description of behavior in case of interruptible sequences disabled.
SPI123	Additional requirement in case of interruptible sequences disabled.
SPI124	Additional requirement in case of interruptible sequences disabled.
SPI125	Description of behavior in case of interruptible sequences enabled.
SPI126	Additional requirement in case of interruptible sequences enabled.
SPI127	Additional requirement in case of interruptible sequences enabled.
SPI128	Global requirement for the LEVEL 2 synchronous and asynchronous be-havior.
SPI129	Description for the prearrange SPI bus for synchronous transmissions..
SPI130	Description of a so-called synchronous sequence.
SPI131	Restrictions to Jobs linkage within a Sequence.
SPI133	`Spi_AsyncTransmit()` configuration dependance.
SPI134	`Spi_SyncTransmit()` main behavior requirement.
SPI135	`Spi_SyncTransmit()` re-entrance behavior requirement.
SPI136	`Spi_SyncTransmit()` configuration dependance.
SPI137	`Spi_WriteIB()` configuration dependance.
SPI138	`Spi_ReadIB()` configuration dependance.
SPI139	`Spi_SetupEB()` configuration dependance.
SPI141	Creation of API interface `Spi_GetHWUnitStatus` to get the status of a specified SPI Hardware microcontroller peripheral (unit)
SPI142	`Spi_GetHWUnitStatus()` configuration dependance.
SPI143	Creation in order to fulfill BSW12448
SPI144	Creation of API interface `Spi_Cancel` to stop a specified Sequence transmission.
SPI145	Additional requirement for end sequence notification in case of cancelling.
SPI146	`Spi_Cancel()` configuration dependance.
SPI147	Additional requirement for checking API parameter and what should be done in case of error
SPI148	Creation of a dedicated variant for post build-time parameters.
SPI149	Global requirement concerning data width handled by HW and data type given by users.
SPI150	Creation of API type `Spi_AsyncModeType` configurable at pre-compile time
SPI151	Additional requirement to the service in order to cover the polling or inter-rupt handling at initialisation for LEVEL 2.
SPI152	Creation of API interface `Spi_SetAsyncMode` to set the asynchronous mechanism mode.
SPI153	Additional requirement in case of setting mode while SPI Handler/Driver is busy.
SPI154	`Spi_SetAsyncMode()` configuration dependance.

SPI155	Requirement to include both polling and interrupt asynchronous mechanisms in LEVEL 2.
SPI156	Additional requirement to have selectable modes during execution time.
SPI157	Additional requirement for asynchronous transmissions of Jobs and specially for setting their results.

14 Changes during SWS Improvements by Technical Office

14.1 Deleted SWS Items

SWS Item	***Rationale***
SPI079	Not a requirement but an example (sequence diagram)
SPI147	Redundant to SPI032

14.2 Replaced SWS Items

SWS Item of Release 1	***replaced by SWS Item***	***Rationale***
SPI096	SPI158, SPI159	Splitted because original requirement was on different objects.
SPI113	SPI160, SPI161	Splitted because original requirement was on different issues.
SPI001	SPI162, SPI163	Splitted because original requirement was on different issues.
SPI153	SPI171, SPI172	Splitted because for better distinction between modes
SPI070	SPI174	Replaced by UML Model linking of imported types

14.3 Changed SWS Items

Many requirements have been changed to improve understandability without changing the technical contents.

14.4 Added SWS Items

SWS Item	***Rationale***
SPI164	Definition of Spi_DataType
SPI165	Definition of Spi_NumberOfDataType
SPI166	Definition of Spi_ChannelType
SPI167	Definition of Spi_JobType
SPI168	Definition of Spi_SequenceType
SPI169	Definition of Spi_HWUnitType
SPI170	Definition of Spi_AsyncModeType
SPI173	Requirement had no ID
SPI175	UML Model linking of Spi_Init
SPI176	UML Model linking of Spi_DeInit
SPI177	UML Model linking of Spi_WriteIB
SPI178	UML Model linking of Spi_AsyncTransmit
SPI179	UML Model linking of Spi_ReadIB
SPI180	UML Model linking of Spi_SetupEB
SPI181	UML Model linking of Spi_GetStatus
SPI182	UML Model linking of Spi_GetJobResult
SPI183	UML Model linking of Spi_GetSequenceResult
SPI184	UML Model linking of Spi_GetVersionInfo

SPI185	UML Model linking of Spi_SyncTransmit
SPI186	UML Model linking of Spi_GetHWUnitStatus
SPI187	UML Model linking of Spi_Cancel
SPI188	UML Model linking of Spi_SetAsyncMode
SPI189	UML Model linking of Spi_MainFunction_Handling
SPI190	UML Model linking of Spi_MainFunction_Driving
SPI191	UML Model linking of the optional interfaces
SPI192	UML Model linking of Spi_JobEndNotification
SPI193	UML Model linking of Spi_SeqEndNotification
SPI196	Hint Spi_GetVersionInfo
SPI233	Requirement added for inclusion of development error SPI_E_ALREADY_INITIALIZED.
SPI234	sentence agreed within SPAL,
SPI235	Extension of SPI198

15 Changes to Release 3

15.1 Deleted SWS Items

SWS Item	***Rationale***
SPI190	SPI doesn't need to use a FIXED_CYCLIC scheduled function
SPI094	Already covered SPI068

15.2 Splitted SWS Items

SWS Item of Release 1	***Splitted in SWS Item***	***Rationale***
SPI003	SPI003, SPI236	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI004	SPI004, SPI237, SPI238, SPI240, SPI241, SPI242, SPI243, SPI245, SPI246	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI005	SPI005, SPI249, SPI250,	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI019	SPI019, SPI251	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI021	SPI021, SPI252, SPI253	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI032	SPI032, SPI254	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI034	SPI034, SPI255	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI046	SPI046, SPI256	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI051	SPI051, SPI257	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI060	SPI060, SPI258	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI061	SPI061, SPI259, SPI260	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI062	SPI062, SPI261	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI065	SPI065, SPI262	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI066	SPI066, SPI263	SWS items that need to be split up into more than one requirement to yield atomic require-

		ments.
SPI075	SPI075, SPI264, SPI265	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI081	SPI081, SPI266	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI083	SPI083, SPI267	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI088	SPI088, SPI268, SPI269, SPI270, SPI271	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI092	SPI092, SPI272, SPI273, SPI274, SPI275, SPI276	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI095	SPI095, SPI277	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI102	SPI102, SPI278	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI111	SPI111, SPI279	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI112	SPI112, SPI280	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI118	SPI118, SPI281	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI123	SPI123, SPI282	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI128	SPI128, SPI283	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI134	SPI134, SPI285, SP286	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI141	SPI141, SPI287	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI143	SPI143, SPI288	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI149	SPI149, SPI289, SPI290, SPI291	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI157	SPI157, SPI292, SPI293	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI161	SPI161, SPI294	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI162	SPI162, SPI295	SWS items that need to be split up into more than one requirement to yield atomic requirements.
SPI174	SPI174, SPI296,	SWS items that need to be split up into more

	SP297	than one requirement to yield atomic require-ments.
SPI175	SPI175, SPI298, SPI299	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI176	SPI176, SPI300, SPI301, SPI302, SPI303	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI177	SPI177, SPI304, SPI305, SPI306, SPI307	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI178	SPI178, SPI308, SPI309, SPI310, SPI311	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI179	SPI179, SPI312, SPI313, SPI314, SPI315	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI180	SPI180, SPI316, SPI317, SPI318	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI181	SPI181, SPI319, SPI320	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI182	SPI182, SPI321, SPI322	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI183	SPI183, SPI323, SPI324	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI184	SPI184, SPI325, SPI326	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI185	SPI185, SPI327, SPI328, SPI329, SPI330	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI186	SPI186, SPI331, SPI332	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI187	SPI187, SPI333, SPI334	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI188	SPI188, SPI335, SPI336, SPI337, SPI338	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI191	SPI191, SPI339	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI192	SPI192, SPI340	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
SPI193	SPI193, SPI341	SWS items that need to be split up into more than one requirement to yield atomic require-ments.
		SWS items that need to be split up into more than one requirement to yield atomic require-ments.

15.3 Changed SWS Items

Many requirements have been changed to improve understandability without changing the technical contents.

SPI089	Change with revising “Published information”.
SPI068	Rework of Published Information

15.4 Added SWS Items

SWS Item	***Rationale***
SPI239	Requirement already contained in the specification but without specific ID
SPI244	Requirement already contained in the specification but without specific ID
SPI342	Requirement already contained in the specification but without specific ID
SPI343	Requirement already contained in the specification but without specific ID
SPI344	Requirement already contained in the specification but without specific ID
SPI345	Requirement already contained in the specification but without specific ID
SPI346	Requirement already contained in the specification but without specific ID
SPI347	Requirement already contained in the specification but without specific ID
SPI348	Requirement already contained in the specification but without specific ID
SPI349	Requirement already contained in the specification but without specific ID
SPI350	Requirement already contained in the specification but without specific ID
SPI351	Requirement already contained in the specification but without specific ID
SPI352	Requirement already contained in the specification but without specific ID
SPI353	Requirement already contained in the specification but without specific ID
SPI354	Requirement already contained in the specification but without specific ID
SPI355	Requirement already contained in the specification but without specific ID
SPI356	Requirement already contained in the specification but without specific ID
SPI357	Requirement already contained in the specification but without specific ID
SPI358	Requirement already contained in the specification but without specific ID
SPI359	Requirement already contained in the specification but without specific ID
SPI360	Requirement already contained in the specification but without specific ID
SPI361	Requirement already contained in the specification but without specific ID
SPI362	Requirement already contained in the specification but without specific ID
SPI363	Requirement to implement debugging concept
SPI364	Requirement to implement debugging concept
SPI365	Requirement to implement debugging concept
SPI366	Requirement to implement debugging concept
SPI367	Requirement to implement debugging concept
SPI368	Added the decsription of the Channel Index related to the SpiChannelIndex parameter inside the SpiChannelList container
SPI369	Satisfaction of BSW004
SPI370	Chip Select handling
SPI371	Added reporting of passed parameter is a NULL pointer