Document Title	Specification of SPI Handler/Driver
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Version	2.1.0
Document Status	Draft
Part of Release	2.1
Revision	0014

Document Change History

Date	Version	Changed by	Change Description
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 12
09.06.2005	1.0.0	AUTOSAR Administration	Initial Release

Release Notes

Errata and known deficiencies

All modifications planned in the scope of Release 2.1 for the incorporation into this document are completed. The document, however, has not yet undergone the necessary finalization.

Disclaimer

Any use of these specifications requires membership within the AUTOSAR Development Partnership or an agreement with the AUTOSAR Development Partnership. The AUTOSAR Development Partnership will not be liable for any use of these specifications.

Following the completion of the development of the AUTOSAR specifications commercial exploitation licenses will be made available to end users by way of written License Agreement only.

No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Copyright © 2004-2006 AUTOSAR Development Partnership. All rights reserved.

Advice to users of AUTOSAR Specification Documents:

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.
SPI107: 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 should 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 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 could 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 accesses 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
https://svn.autosar.org/repos/10Releases/
AUTOSAR_LayeredSoftwareArchitecture.pdf

[2] General Requirements on SPAL
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_SPAL_General.pdf

[3] General Requirements on Basic Software Modules
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_General.pdf

[4] Specification of Development Error Tracer
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[5] Specification of ECU Configuration
https://svn.autosar.org/repos/10Releases/
AUTOSAR_ECU_Configuration.pdf

[6] Requirements on SPI Handler/Driver
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_SPI_HandlerDriver.pdf

[7] Specification of Diagnostics Event Manager
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_DEM.pdf

[8] Glossary
https://svn.autosar.org/repos/10Releases/
AUTOSAR_Glossary.pdf

[9] Specification of MCU Driver
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_MCU_Driver .pdf

[10] Specification of PORT Driver
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_PORT_Driver

3.2 Related standards and norms

Not related.

4 Constraints and assumptions

4.1 Limitations

SPI040: SPI Handler/Driver handles only the Master mode.

SPI050: 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 makes no sense.

4.2 Applicability to car domains

No restrictions.

5 Dependencies to other modules

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. Module SPI Handler/Driver do 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 0.

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 0 for initialising and de-initialising those registers.

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. At this point it shall be pointed out that the code-file structure shall include the following file named:

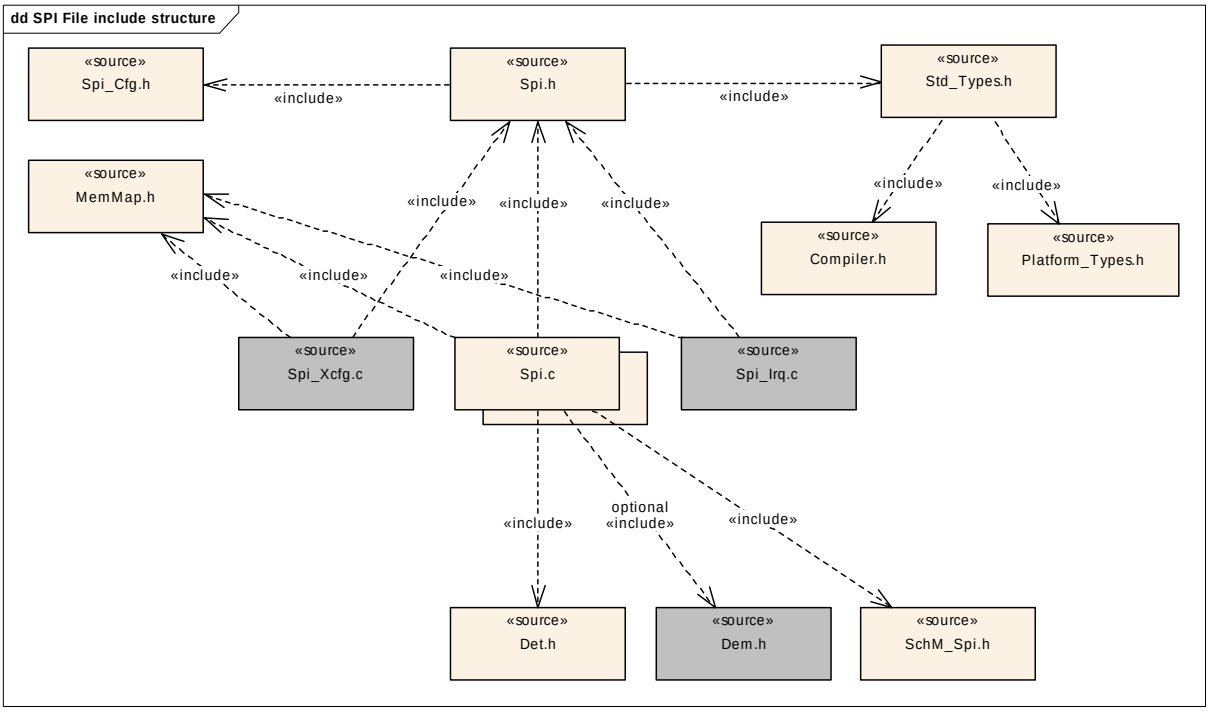
- Spi_Lcfg.c – for link time and for post-build configurable parameters and
- Spi_PBcfg.c – for post build time configurable parameters.

These files shall contain all link time and post-build time configurable parameters.

This file shall contain all link time and post-build time configurable parameters.

5.1.2 Header file structure

SPI092: The include file structure shall be as follows:



- `Spi.c` shall include `Spi.h`
- `Spi_Xcfg.c` shall include `Spi.h`
- `Spi.h` shall include `Spi_Cfg.h`
- `Spi_Irq.c` this file could exist depending of implementation and also it could or not include `Spi.h`

SPI096: The module shall include the Dem.h file. By this inclusion the APIs to report errors as well as the required Event Id symbols are included. This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in Dem_IntErrId.h.

6 Requirements traceability

Document: AUTOSAR requirements on Basic Software, general

Requirement	***Satisfied by***
[BSW003] Version identification	SPI068 SPI089 SPI094
[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	SPI094
[BSW00321] Enumeration of module version numbers	SPI094
[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
[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	SPI070 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, 10.2.5 and 10.2.6
[BSW00373] Main processing function naming convention	Chapter 8.5
[BSW00374] Module vendor identification	SPI068 SPI089 SPI094
[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 8.2.2, 8.2.3 and 8.2.4
[BSW00378] AUTOSAR boolean type	SPI105
[BSW00379] Module identification	SPI068 SPI089 SPI094
[BSW00380] Separate C-Files for configuration parameters	SPI095
[BSW00381] Separate configuration header file for pre-compile time parameters	SPI103
[BSW00383] List dependencies of configuration	Chapter 5

files	
[BSW00384] List dependencies to other modules	Chapter 5, SPI096
[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 SPI094
[BSW00391] Parameter shall have unique names	SPI103 SPI091 SPI104 SPI105 SPI106 SPI094
[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 SPI094
[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 specification)
[BSW101] Initialization interface	SPI013 SPI015
[BSW158] Separation of configuration from implementation	SPI095 SPI103 SPI091 SPI089
[BSW159] Tool-based configuration	Both static and runtime configuration parameters are located outside the source code of the

	module. This is the prerequisite for automatic configuration.
[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 specification)

Document: AUTOSAR requirements on Basic Software, cluster SPAL

Requirement	***Satisfied by***
[BSW12263] Object code compatible configuration concept	SPI076
[BSW12056] Configuration of notification mechanisms	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 implementation.
[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 handlers	SPI026 SPI038 SPI039 SPI042 SPI057 SPI071 SPI073 SPI075
[BSW12169] Control of operation mode	SPI079
[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	SPI079 SPI025 SPI021

[BSW12448] Behavior after development error detection	Chapters 8.7 and 8.8
[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	SPI107 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 one Sequence	SPI065 SPI003 SPI064
[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

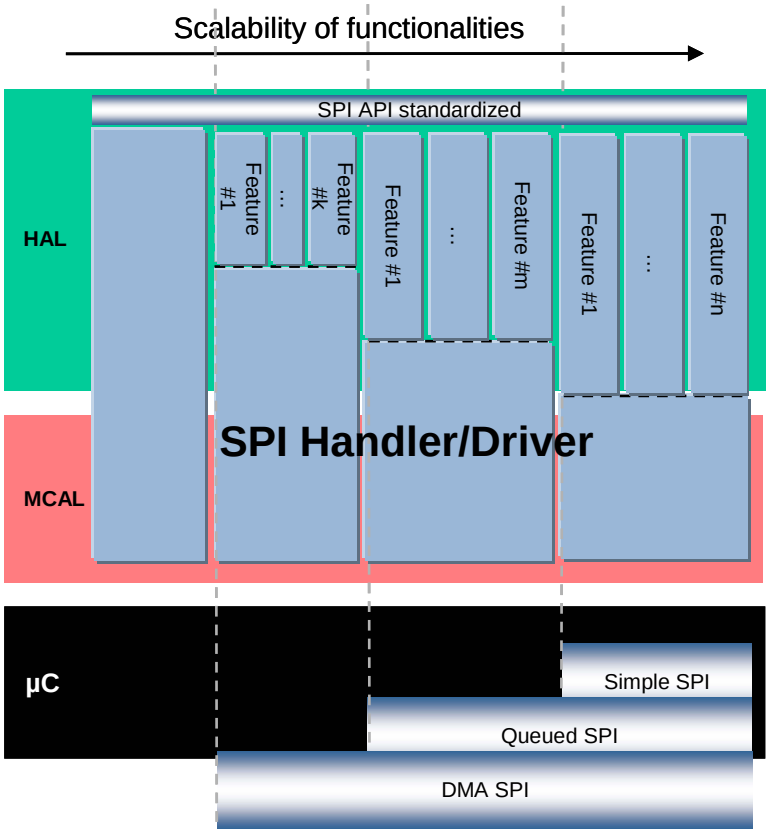
[BSW12262] Read large data sequences from several slave devices using variable send data	SPI053 SPI065 SPI003 SPI078
[BSW12202] Support of variable data length	SPI053 SPI078
[BSW12024] Configuration of SPI HW Unit	SPI008 SPI063
[BSW12150] Configuration of SPI asynchronous SW and HW properties	SPI009 SPI064 SPI093
[BSW12108] Callback notification	Chapter 8.6.3 SPI057 SPI118 SPI119 SPI120
[BSW12099] Asynchronous Read Functionality	SPI020 SPI001 SPI016 SPI020
[BSW12101] Asynchronous Write Functionality	SPI020 SPI001 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. This chapter intention is to summarize how the scalability of this monolithic SPI Handler/Driver allows getting a simple software module that fits poor 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 level of scalable functionality shall be always statically configurable, that means configured at pre-compile time, to allow the best source code optimisation.

SPI110: In order to select the SPI Handler/Driver level of scalable functionality, the `SPI_LEVEL_DELIVERED` parameter shall be configured with one of the 3 authorized values according to the described levels: 0, 1 or 2.

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 is based on Channels, Jobs and Sequences as defined in this document (see chapter 2).

SPI034: It shall support one or more Channels, Jobs and Sequences to drive all kind of SPI compatible HW devices. Data transmissions shall be done according to Channels, Jobs and Sequences configuration parameters.

SPI066: The Chip Select (CS) is attached to the Job definition. Chip Select shall be handled during Job transmission and shall be released at the end of it[1]. This Chip Select handling shall be done according to the Job configuration parameters.

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.

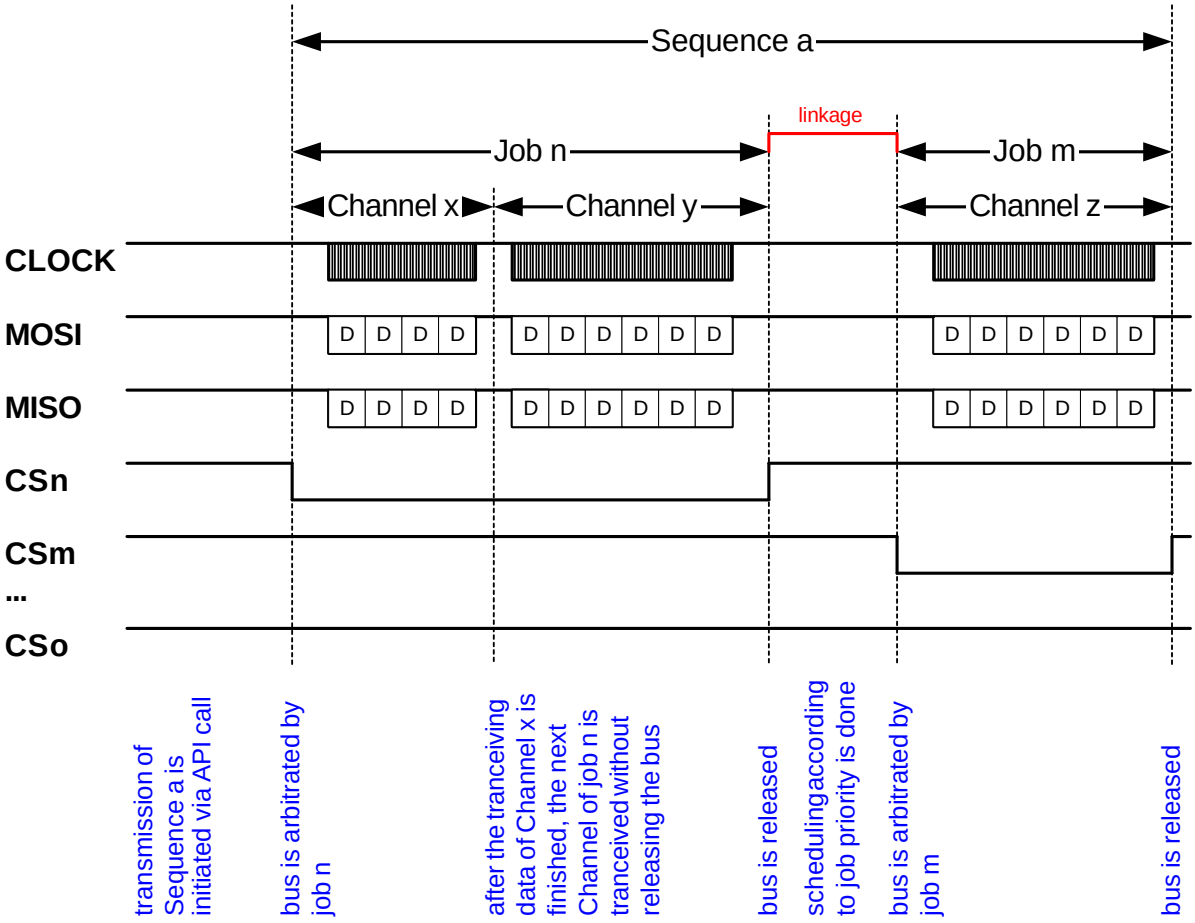
SPI065: A Job shall contain at least one Channel. If it contains more than one, all Channels contained will have the same Job properties during transmission and shall be linked together statically.

[1] The software implementation to handle CS depends on several parameters such as microcontroller capabilities and/or ECU hardware design. For this reason, the specification does not specify how to do it but only how to configure a CS reference to a Job.

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. If it contains more than one, all Jobs contained will 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 intended that it is allowed to pass NULL pointer as source and destination pointers to generate a dummy transmission (See also [SPI028] & [SPI030]).



Channels 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 `SPI_DATA_WIDTH`).

SPI149: In order to always get the same user interface, the SPI Handler/Driver shall handle these differences using the following proposed rules:

- Spi_DataType 8/16/32 bits, data width 8/16/32 bits. Straightforward send and receive.
- Spi_DataType superior to data width. Send the lower part, ignore the upper part. Receive the lower part, extend with zero.

- Spi_DataType inferior to data width. According to the memory alignement use prior both rules.

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 amount of data from/to a dedicated memory location, all levels have an optional feature about location of Channel Buffers.

Hence two main kinds of channels buffering could 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: In order to select which Channel Buffers the SPI Handler/Driver manages, the `SPI_CHANNEL_BUFFERS_ALLOWED` parameter shall be configured with one of the 3 authorized values according to the described usage: 0, 1 or 2.

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.

SPI049: The channel data received shall be stored in 1 entry deep internal buffers by channel. The SPI Handler/Driver will 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. The SPI Handler/Driver is not able to prevent the overwriting of these “transmit” buffers by users during transmissions, see [SPI084].

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.

SPI112: The size of the Channel buffer is either fixed or variable. A maximum size for the Channel buffer is fixed by configuration, but the buffer really provided by the application 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 user calling the transmission service is blocked until the transmission is not terminated.

SPI113: The LEVEL 0 SPI Handler/Driver shall offer a synchronous transfer service for SPI busses. When there is no on-going Sequence transmission, the SPI Handler/Driver shall be in idle state (`SPI_IDLE`).

This monolithic SPI Handler/Driver is able to handle one to n SPI busses 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.

SPI114: Hence, the LEVEL 0 SPI Handler/Driver shall not allow transmitting more than one Sequence at the same time. 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 includes the common feature: Allowed Channel Buffers, which is selected.

SPI084: Different Jobs, consequently also Sequences, can have in common Channels. But, the use of those common Channels is restricted. Read and Write functions can not guarantee the data integrity while Channel data is being transmitted. User must ensure that Read and/or Write functions are not called during transmission.

7.2.3 LEVEL 1, Basic Asynchronous behaviour

The intention of this functionality level is to provide a Handler/Driver with a reduced set of services to handle only asynchronous transmissions. 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 during the transmission is on-going. Furthermore, the user could be notified at the end of transmission[2].

SPI001: The LEVEL 1 SPI Handler/Driver shall offer an asynchronous transfer service for SPI busses. When there is no on-going Sequence transmission, the SPI Handler/Driver shall be in idle state (`SPI_IDLE`).

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

SPI002: Jobs have priorities assigned. Jobs linked in a Sequence shall have decreasing priorities. That means the first Job shall have the highest priority of all Jobs within the Sequence.

SPI093: Priority order of jobs shall be from the lower to the higher value defined (limited to 4 priority levels see [SPI009]).

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.

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

[2] 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.

Sequences. Consequently, Jobs, on different SPI buses, could belong to the same Sequence.

SPI116: Hence, 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 includes the common feature: Allowed Channel Buffers, which is selected, and the configured asynchronous feature: Interruptible Sequence (see next chapter).

SPI083: On hardware error detection the current Sequence is stopped and an error is reported to the error hook of the DET or to the DEM as configured[3]. The state of Job will be `SPI_JOB_FAILED` and the state of Sequence will be `SPI_SEQ_FAILED`.

SPI118: Jobs and Sequences could have a specific end notification function configured. This one shall be called at the end of the Job/Sequence transmission by the SPI Handler/Driver (see [SPI071] & [SPI073]).

SPI119: Whatever the result of a Job transmission, either `SPI_JOB_FAILED` or `SPI_JOB_OK` and a valid notification function pointer is configured (see [SPI071]), this notification function shall be called (rational: avoid deadlocks or endless loops) at the end of Job transmission.

SPI120: Whatever the result of a Sequence transmission, either `SPI_SEQ_FAILED`, `SPI_SEQ_OK` or `SPI_SEQ_CANCELLED` and a valid notification function pointer is configured (see [SPI073]), this notification function shall be called (rational: avoid deadlocks or endless loops) at the end of Sequence transmission.

7.2.4 Asynchronous configurable feature: Interruptible Sequences

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

Hence two main kinds of sequences could 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: In order to select which kind of Sequences the SPI Handler/Driver manages, the `SPI_INTERRUPTIBLE_SEQ_ALLOWED` parameter shall be configured (`ON` / `OFF`).

[3] Implementation and hardware capabilities related errors are specified in this document, Production errors could be defined later during the software design stage.

7.2.4.1 Behaviour of Non Interruptible Sequences

The intention of 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: In case of the `SPI_INTERRUPTIBLE_SEQ_ALLOWED` parameter is switch off, that means configured with “OFF” value, Interruptible Sequences are not allowed within levels 1 and 2 of SPI Handler/Driver.

SPI123: In this configuration, all Sequences declared are considered as Non Interruptible Sequences. That means, their dedicated parameter `SPI_INTERRUPTIBLE_SEQUENCE` (see SPI064 & SPI106) could be omitted or the `FALSE` value should be used as default[4].

SPI124: According to [SPI116] and [SPI122] requirements, the SPI Handler/Driver is not allowed suspending a Sequence transmission already started in aid of another Sequence.

7.2.4.2 Behaviour of Mixed Sequences

The intention of 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: In case of the `SPI_INTERRUPTIBLE_SEQ_ALLOWED` parameter is switch on, that means configured with “ON” value, Interruptible Sequences are allowed within levels 1 and 2 of SPI Handler/Driver.

SPI126: In this configuration, all Sequences declared shall have their dedicated parameter `SPI_INTERRUPTIBLE_SEQUENCE` (see SPI064 & SPI106) to identify if the Sequence could be or not suspendable during transmission.

SPI014: In case of a Sequence configured as Interruptible Sequence and according to [SPI125] requirement, the SPI Handler/Driver is allowed suspending this already started Sequence transmission in aid of another Sequence with a higher priority Job (see SPI002 & SPI093). That means, at the end of a Job (that belongs to the interruptible sequence) transmission with another Sequence transmit request pending, the SPI Handler/Driver shall perform a rescheduling in order to elect the next Job to transmit.

SPI127: In case of a Sequence configured as Non Interruptible Sequence and according to [SPI125] requirement, the SPI Handler/Driver is not allowed suspending this already started Sequence transmission in aid of another Sequence.

[4] 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.

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 others 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 others SPI busses. When 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. This information shall be published by the supplier of this software module (see [SPI094]).

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 by Jobs that are associated to the prearranged SPI bus (see [SPI094]). These Sequences have to be used only with synchronous services[5].

SPI131: Jobs associated to the prearranged SPI bus (see [SPI094]) shall not belong to Sequences containing Jobs associated to other SPI bus. In other words, mixed Sequences (synchronous with asynchronous Jobs) are not 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 implemented one polling mechanism mode and one interrupt mechanism mode for SPI busses handled asynchronously.
SPI156: Both mechanisms are selectable during execution time (see 8.2.11 and 8.2.11).

[5] 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.

SPI132: Requirements for LEVEL 0 and for LEVEL 1 apply to this level respectively to synchronous behaviour and to asynchronous behaviour.

7.3 Scheduling Advices

For asynchronous levels, LEVEL 1 and LEVEL 2, the SPI Handler/Driver could 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 done by the SPI Handler/Driver in order to elect the next Job to transmit (see [SPI014]).

SPI088: In case of these two actions are fully done by software; the order between them shall be first scheduling and after the call of end notification function. Otherwise, if they are done by hardware and the order could not be configured as required, the order shall be completely documented.

7.4 Error classification

SPI004: Errors in the table below shall be detectable by the SPI Handler/Driver depending on its build version (development/production mode):

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.

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`

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`

SPI007: Additional errors that are detected because of specific implementation and/or specific hardware properties shall be added in the SPI device specific implementation description. The classification and enumeration shall be compatible to the errors listed above [SPI004].

SPI098: Development error values are of type `uint8`.

7.5 Error detection

SPI005: The detection of all development errors is configurable (On / Off) at pre-compile time. The switch SPI_DEV_ERROR_DETECT (see chapter 10) shall activate or deactivate the detection of all development errors.

SPI029: If the switch SPI_DEV_ERROR_DETECT is enabled API parameter checking is also enabled. The detailed description of the detected errors can be found §8.7.

SPI099: The detection of production code errors cannot be switched off.

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 SPI_DEV_ERROR_DETECT is set (see chapter 10).

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.

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.

8 API specification

8.1 Imported types

8.1.1 Standard types

SPI070: Types `Std_VersionInfoType` and `Std_ReturnType` (defined in `Std_Types.h`) shall be used.

8.2 Type definitions

8.2.1 Spi_ConfigType

Type:	`Structure`
Range:	`Implementation Specific` The contents of the initialization data structure are SPI specific.
Description:	SPI008: This type of the external data structure 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 SPI063: 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 SPI009: 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) SPI064: 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 SPI010: The configuration will map the Jobs to the different SPI hardware units and the devices.

8.2.2 Spi_StatusType

Type:	`enumeration`	
Range:	`SPI_UNINIT`	The SPI Handler/Driver is not initialized or not usable. SPI011: This shall be the default value after reset. This status shall have the value 0.
	`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:	SPI061: This type defines a range of specific status for SPI Handler/Driver. It informs about the SPI Handler/Driver status and can be obtained calling the API service `Spi_GetStatus` or the configurable `Spi_GetHWUnitStatus`.	

8.2.3 Spi_JobResultType

Type:	`enumeration`	
Range:	`SPI_JOB_OK`	The last transmission of the Job has been finished successfully. SPI012: This shall be the default value after reset. This status shall have the value 0.
	`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.
Description:	SPI062: This type defines a range of specific Jobs status for SPI Handler/Driver. It informs about a SPI Handler/Driver Job status and can be obtained calling the API service `Spi_GetJobResult` with the Job ID.	

8.2.4 Spi_SeqResultType

Type:	`enumeration`	
Range:	`SPI_SEQ_OK`	The last transmission of the Sequence has been finished successfully. SPI017: This shall be the default value after reset. This status shall have the value 0.
	`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_CANCELLED`	The last transmission of the Sequence has been cancelled by user.
Description:	SPI019: This type defines a range of specific Sequences status for SPI Handler/Driver. It informs about a SPI Handler/Driver Sequence status and can be obtained calling the API service `Spi_GetSequenceResult` with the Sequence ID.	

8.2.5 Spi_DataType

Type:	`uint8…uint32`	
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.	

8.2.6 Spi_NumberOfDataType

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

8.2.7 Spi_ChannelType

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

8.2.8 Spi_JobType

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

8.2.9 Spi_SequenceType

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

8.2.10 Spi_HWUnitType

Type:	`Uint8`
Range:	
Description:	Specifies the identification (ID) for a SPI Hardware microcontroller peripheral (unit).

8.2.11 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. SPI150: This type is available or not according to the pre compile time parameter: SPI_LEVEL_DELIVERED. This is only relevant for LEVEL 2.	

8.3 Function definitions

8.3.1 Spi_Init

Service name:	Spi_Init	
Syntax:	void Spi_Init (const Spi_ConfigType *ConfigPtr)	
Service ID [hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non re-entrant	
Parameters (in):	ConfigPtr	Pointer to configuration set
Parameters (out):	None	--
Return value:	None	
Description:	SPI013: Service for SPI initialization. The Initialization function shall initialize all SPI relevant registers with the values of the structure referenced by the parameter ConfigPtr. SPI082: This Init function shall define defaults values for needed parameters of the structure referenced by the ConfigPtr. For example: all buffer pointers shall be initialized as a null value pointer. SPI015: After the end of the module initialization, the SPI Handler/Driver state shall be set to SPI_IDLE and set the Sequences result shall be set to SPI_SEQ_OK and the jobs result shall be set to SPI_JOB_OK. SPI151: For LEVEL 2, (see chapter 7.2.5 and SPI103) this function sets the SPI Handler/Driver asynchronous mechanism mode to SPI_POLLING_MODE by default. Interrupts related to SPI busses shall be disabled. Parameters shall be checked as it is explained in section API parameter checking	
Caveats:	None	
Configuration:	None	

8.3.2 Spi_DeInit

Service name:	Spi_DeInit	
Syntax:	`Std_ReturnType Spi_DeInit ( void )`	
Service ID [hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Non re-entrant	
Parameters (in):	None	--
Parameters (out):	None	--
Return value:	`E_OK`: de-initialisation command has been accepted `E_NOT_OK`: de-initialisation command has not been accepted	
Description:	SPI021: Service for SPI de-initialization. In case of a `SPI_BUSY` state, command shall be rejected otherwise the De-Initialization function shall put all already initialized microcontroller SPI peripherals in the same state such as Power On Reset. SPI022: After the end of module de-initialization, the SPI Handler/Driver state shall be set to `SPI_UNINIT`.	
Caveats:	The SPI Handler/Driver shall have been initialized before this service is called otherwise see [SPI046].	
Configuration:	None	

8.3.3 Spi_WriteIB

Service name:	`Spi_WriteIB`	
Syntax:	`Std_ReturnType Spi_WriteIB ( Spi_ChannelType Channel, const Spi_DataType *DataBufferPtr )`	
Service ID [hex]:	0x02	
Sync/Async:	Synchronous	
Reentrancy:	Re-entrant	
Parameters (in):	`Channel`	Channel ID.
	`DataBufferPtr`	Pointer to source data buffer. SPI023: 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 (out):	None	--
Return value:	`E_OK`: write command has been accepted `E_NOT_OK`: write command has not been accepted	
Description:	SPI018: Service for writing one or more data to an IB SPI Handler/Driver Channel specified by parameter. SPI024: This service shall take over the given parameters, and save the pointed data to the internal buffer defined with `Spi_Init()`. Parameters shall be checked as it is explained in section API parameter checking	
Caveats:	The SPI Handler/Driver shall have been initialized before this service is called otherwise see [SPI046].	

Configuration:	SPI137: This function is pre compile time configurable by the parameter: `SPI_CHANNEL_BUFFERS_ALLOWED`. This service is only relevant for Channels with IB.

8.3.4 Spi_AsyncTransmit

Service name:	`Spi_AsyncTransmit`	
Syntax:	`Std_ReturnType Spi_AsyncTransmit ( Spi_SequenceType Sequence )`	
Service ID [hex]:	0x03	
Sync/Async:	Asynchronous	
Reentrancy:	Re-entrant	
Parameters (in):	`Sequence`	Sequence ID.
Parameters (out):	None	--
Return value:	`E_OK`: Transmission command has been accepted `E_NOT_OK`: Transmission command has not been accepted	
Description:	SPI020: Service to transmit data on the SPI bus. This service shall take over the given parameter, initiate a transmission, and set the SPI Handler/Driver status to `SPI_BUSY`, set the sequence result to `SPI_SEQ_PENDING` and return. SPI157: Jobs' results are handled by the SPI Handler/Driver when transmission Jobs is started (result set to `SPI_JOB_PENDING`) and/or ended (set either to `SPI_JOB_OK` or `SPI_JOB_FAILED`). SPI081:If the requested Sequence is already in the state `SPI_SEQ_PENDING` the SPI Handler/Driver does not take in account this new request and this service returns value `E_NOT_OK`. According to [SPI100], the SPI Handler/Driver shall report the `SPI_E_SEQ_PENDING` error. SPI086: If the requested Sequence shares Jobs with another sequence that is in the state `SPI_SEQ_PENDING`, the SPI Handler/Driver does not take in account this new request and this service returns value `E_NOT_OK`. According to [SPI100], the SPI Handler/Driver shall report the `SPI_E_SEQ_PENDING` error. SPI035: With EB, when 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]) SPI036: With EB, when destination data pointer has been provided as null using the `Spi_SetupEB` method the receiving data shall be ignored. (See also [SPI030]) SPI055: When a transmission is requested for a Sequence with linked Jobs, it will transmit from the first Job up to the last Job in the sequence. SPI057: At the end of a sequence transmission, if configured, the sequence notification call-back function shall be executed after the last Job end notification if this one is also configured. Parameters shall be checked as it is explained in section API parameter checking	
Caveats:	The SPI shall have been initialized before this service is called otherwise see [SPI046].	

	This method shall be called after a `Spi_SetupEB` method for EB Channels or `Spi_WriteIB` method for IB Channels but before the `Spi_ReadIB` method.
Configuration:	SPI133: This function is pre compile time selectable by the configuration parameter: `SPI_LEVEL_DELIVERED`. This function is only relevant for LEVEL 1 and LEVEL 2.

8.3.5 Spi_ReadIB

Service name:	`Spi_ReadIB`	
Syntax:	`Std_ReturnType Spi_ReadIB ( Spi_ChannelType Channel, Spi_DataType *DataBufferPtr )`	
Service ID [hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Re-entrant	
Parameters (in):	`Channel`	Channel ID.
Parameters (out):	DataBufferPtr	Pointer to destination data buffer in RAM
Return value:	`E_OK`: read command has been accepted `E_NOT_OK`: read command has not been accepted	
Description:	SPI016: Service for reading synchronously one or more data from an IB SPI Handler/Driver Channel specified by parameter. Parameters shall be checked as it is explained in section API parameter checking	
Caveats:	The SPI Handler/Driver shall have been initialized before this service is called otherwise see [SPI046]. SPI027: This method shall be called after one Transmit method call to have relevant data within IB Channel.	
Configuration:	SPI138: This function is pre compile time configurable by the parameter: `SPI_CHANNEL_BUFFERS_ALLOWED`. This service is only relevant for Channels with IB.	

8.3.6 Spi_SetupEB

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:	Re-entrant	
Parameters (in):	`Channel`	Channel ID.
	`SrcDataBufferPtr`	Pointer to source data buffer.

	`Length`	Length of the data to be transmitted from SrcdataBufferPtr and/or received from DesDataBufferPtr Min.: 1 Max.: Max of data specified at configuration for this channel
Parameters (out):	DesDataBufferPtr	Pointer to destination data buffer in RAM.
Return value:	`E_OK`: Setup command has been accepted `E_NOT_OK`: Setup command has not been accepted	
Description:	SPI058: Service to setup the buffers and the length of data for the EB SPI Handler/Driver Channel specified. SPI067: This service shall update the buffer pointers and lengh attributes of the specified Channel with the provided values. As these attributes are persistent, they will be used for all succeeding calls to one Transmit method (for the specified Channel). SPI028: If `SrcDataBufferPtr` is a null pointer the default transmit value configured for the channel will be transmitted when a Transmit method is requested. (See also [SPI035]) SPI030: If `DesDataBufferPtr` is a null pointer the received data will be ignored when a Transmit method is requested.(See also [SPI036]) Parameters shall be checked as it is explained in section API parameter checking	
Caveats:	The SPI Handler/Driver shall have been initialized before this service is called otherwise see [SPI046]. SPI037: This method shall be called one time for all Channels with EB declared before to call a Transmit method for them.	
Configuration:	SPI139: This function is pre compile time configurable by the parameter: `SPI_CHANNEL_BUFFERS_ALLOWED`. This service is only relevant for Channels with EB.	

8.3.7 Spi_GetStatus

Service name:	`Spi_GetStatus`	
Syntax:	`Spi_StatusType Spi_GetStatus ( void )`	
Service ID [hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Re-entrant	
Parameters (in):	None	--
Parameters (out):	None	--
Return value:	`Spi_StatusType`	
Description:	SPI025: This service shall return the SPI Handler/Driver software module status.	
Caveats:	None	
Configuration:	None	

8.3.8 Spi_GetJobResult

Service name:	`Spi_GetJobResult`	
Syntax:	`Spi_JobResultType Spi_GetJobResult ( Spi_JobType Job )`	
Service ID [hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Re-entrant	
Parameters (in):	Job	Job ID. An invalid job ID will return an undefined result.
Parameters (out):	None	--
Return value:	`Spi_JobResultType`	
Description:	SPI026: This service shall return the last transmission result of the specified Job. NOTE: Every new transmit job that has been accepted by the SPI Handler/Driver overwrites the previous job result with `SPI_JOB_PENDING`. SPI038: The user shall call `Spi_GetJobResult()` method to know if the Job transmission succeeded (`SPI_JOB_OK`) or failed (`SPI_JOB_FAILED`) Parameters shall be checked as it is explained in section API parameter checking	
Caveats:	If SPI Handler/Driver has not been initialized before this service is called, the return value is undefined.	
Configuration:	None	

8.3.9 Spi_GetSequenceResult

Service name:	`Spi_GetSequenceResult`	
Syntax:	`Spi_SeqResultType Spi_GetSequenceResult ( Spi_SequenceType Sequence )`	
Service ID *[hex]*:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Re-entrant	
Parameters (in):	`Sequence`	Sequence ID. An invalid sequence ID will return an undefined result.
Parameters (out):	None	--
Return value:	`Spi_SeqResultType`	
Description:	SPI039: This service shall return the last transmission result of the specified Sequence. NOTE: Every new transmit sequence that has been accepted by the SPI Handler/Driver overwrites the previous sequence result with `SPI_SEQ_PENDING`. SPI042: The user shall call `Spi_GetSequenceResult()` method to know if the fully Sequence transmission succeeded (`SPI_SEQ_OK`) or failed (`SPI_SEQ_FAILED`) Parameters shall be checked as it is explained in section API parameter checking	
Caveats:	If SPI Handler/Driver has not been initialized before this service is called, the return value is undefined.	

Configuration:	None

8.3.10 Spi_GetVersionInfo

Service name:	`Spi_GetVersionInfo`	
Syntax:	`void Spi_GetVersionInfo ( Std_VersionInfoType *VersionInfoPtr )`	
Service ID [hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Non re-entrant	
Parameters (in):	None	--
Parameters (out):	`VersionInfoPtr`	Pointer to where to store the version information of this module.
Return value:	None	--
Description:	SPI101: This service returns the version information of this module. The version information includes: - Module Id - Vendor Id - Vendor specific version numbers (BSW00407). Hint: If source code for caller and called of this function is available this function should be realized as a macro. The macro should be defined in the modules header file.	
Caveats:	None	
Configuration:	SPI102: This function is pre compile time configurable `On/Off` by the configuration parameter: `SPI_VERSION_INFO_API`	

8.3.11 Spi_SyncTransmit

Service name:	`Spi_SyncTransmit`	
Syntax:	`Std_ReturnType Spi_SyncTransmit ( Spi_SequenceType Sequence )`	
Service ID [hex]:	0x0A	
Sync/Async:	Synchronous	
Reentrancy:	Re-entrant	
Parameters (in):	`Sequence`	Sequence ID.
Parameters (out):	None	--
Return value:	`E_OK`: Transmission command has been accepted `E_NOT_OK`: Transmission command has not been accepted	
Description:	SPI134: Service to transmit data on the SPI bus. This service shall take over the given parameter, set the SPI Handler/Driver status to `SPI_BUSY`, set the sequence result to `SPI_SEQ_PENDING`, set the first job result to `SPI_JOB_PENDING` and perform the transmission. SPI135: If a request is done while a Sequence is on transmission, the SPI Handler/Driver does not take in account this new request and this service returns value `E_NOT_OK` (see [SPI114]). According to [SPI100], the SPI Handler/Driver	

	shall report the SPI_E_SEQ_IN_PROCESS error. Parameters shall be checked as it is explained in section API parameter checking
Caveats:	None
Configuration:	SPI136: This function is pre compile time selectable by the configuration parameter: SPI_LEVEL_DELIVERED. This function is only relevant for LEVEL 0 and LEVEL 2.

8.3.12 Spi_GetHWUnitStatus

Service name:	Spi_GetHWUnitStatus	
Syntax:	Spi_StatusType Spi_GetHWUnitStatus (Spi_HWUnitType HWUnit)	
Service ID [hex]:	0x0B	
Sync/Async:	Synchronous	
Reentrancy:	Re-entrant	
Parameters (in):	HWUnit	SPI Hardware microcontroller peripheral (unit) ID.
Parameters (out):	None	--
Return value:	Spi_StatusType	
Description:	SPI141: This service shall return the status of the specified SPI Hardware microcontroller peripheral. The user shall call Spi_GetHWUnitStatus() method to know if the specified SPI Hardware microcontroller peripheral is SPI_IDLE or SPI_BUSY. Parameters shall be checked as it is explained in section API parameter checking	
Caveats:	If SPI Handler/Driver has not been initialized before this service is called, the return value is undefined.	
Configuration:	SPI142: This function is pre compile time configurable On / Off by the configuration parameter: SPI_HW_STATUS_API.	

8.3.13 Spi_Cancel

Service name:	Spi_Cancel	
Syntax:	void Spi_Cancel (Spi_SequenceType Sequence)	
Service ID [hex]:	0x0C	
Sync/Async:	Asynchronous	
Reentrancy:	Re-entrant	
Parameters (in):	Sequence	Sequence ID. SPI147: It shall be within the specified range of values. Related error value: SPI_E_PARAM_SEQ. Otherwise, the service is not done.
Parameters (out):	None	--
Return value:	None	
Description:	SPI144: This service shall cancel the specified on-going sequence transmission	

	without cancelling any Job transmission and the SPI Handler/Driver will 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 actually cancelled, if configured, the sequence notification call-back function shall be executed instead of starting a potential next job belonging to it.
Caveats:	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 user to be aware to what it is doing!
Configuration:	SPI146: This function is pre compile time configurable On / Off by the configuration parameter: SPI_CANCEL_API.

8.3.14 Spi_SetAsyncMode

Service name:	Spi_SetAsyncMode	
Syntax:	Std_ReturnType Spi_SetAsyncMode (Spi_AsyncModeType Mode)	
Service ID [hex]:	0x0D	
Sync/Async:	Synchronous	
Reentrancy:	Non re-entrant	
Parameters (in):	Mode	New mode required.
Parameters (out):	None	--
Return value:	E_OK: Setting command has been done E_NOT_OK: setting command has not been accepted	
Description:	SPI152: Service to set the asynchronous mechanism mode for SPI busses handled asynchronously. This service shall set the mode following the given parameter. SPI153: When in asynchronous mode if a request is done while the SPI Handler/Driver status is not in SPI_IDLE, it does not take in account this new request, the mode is kept as it is and this service returns value E_NOT_OK. When in synchronous mode the service is accepted even if the status is SPI_BUSY.	
Caveats:	None	
Configuration:	SPI154: This function is pre compile time selectable by the configuration parameter: SPI_LEVEL_DELIVERED. This function is only relevant for LEVEL 2.	

8.4 Callback notifications

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

The SPI Handler/Driver module does not require any schedulable function. The specified functions below exemplify how to implement them if they are needed.

These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non re-entrant.

8.5.1 Spi_MainFunction_Handling

Service name:	Spi_MainFunction_Handling
Service ID [hex]:	0x10
Description:	<Set of local software requirements including ID that define the operation of this API call.>
Timing:	<fixed cyclic / variable cyclic / on pre condition>
Pre condition:	<List of assumptions about the environment in which the API call must operate.>
Configuration:	<Description of statically configurable attributes that affect this API call. For instance cycle time(s) in case of fixed cyclic timing. Reference to chapter 10.>

8.5.2 Spi_MainFunction_Driving

Service name:	Spi_MainFunction_Driving
Service ID [hex]:	0x11
Description:	<Set of local software requirements including ID that define the operation of this API call.>
Timing:	<fixed cyclic / variable cyclic / on pre condition>
Pre condition:	This routine might be called on interrupt level, depending on the level of functionality delivered.
Configuration:	<Description of statically configurable attributes that affect this API call. For instance cycle time(s) in case of fixed cyclic timing. Reference to chapter 10.>

8.6 Expected Interfaces

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.

API function	*Module*	*Description*	*Configuration parameter (description see chapter 10)*
`Det_ReportError`	Det	Development error notification	`SPI_DEV_ERROR_DETECT`
`Dem_ReportErrorStatus`	Dem	Production error status	Implementation Specific

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 following callback routines are used by the SPI Handler/Driver to inform other software modules about certain states or state changes. These other modules are required to provide the routines in the expected manner.

The callback notifications shall be as function pointers within the initialization data structure (`Spi_ConfigType`).

The callback notifications shall have no parameters and no return value.

SPI054: If a callback notification is configured as null pointer, no callback shall be executed.

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

SPI071: At the end of a Job transmission if the corresponding `SPI_JOB_END_NOTIFICATION` is configured as non null pointer, it shall be called.

Name:	`Spi_JobEndNotification`

Syntax:	`void (* Spi_JobEndNotification) (void)`	
Reentrancy:	`Re-entrant`	
Parameters (in):	`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.	
Caveats:	This routine might be called on interrupt level, depending on the calling function.	
Configuration:	--	

8.6.3.2 Spi_SeqEndNotification

SPI073: At the end of a Sequence transmission if the corresponding `SPI_SEQ_END_NOTIFICATION` is configured as non null pointer, it shall be called.

Name:	`Spi_SeqEndNotification`	
Syntax:	`void (* Spi_SeqEndNotification) (void)`	
Reentrancy:	`Re-entrant`	
Parameters (in):	`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.	
Caveats:	This routine might be called on interrupt level, depending on the calling function.	
Configuration:	--	

8.7 API parameter checking

SPI031: `Channel` shall be 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`.

SPI032: `Sequence` and `Job` shall be within the specified range of values. Related errors values: `SPI_E_PARAM_SEQ` or `SPI_E_PARAM_JOB`. Otherwise, the 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: `Length` of data shall be within the specified buffer maximum values (see SPI063). Related error value: `SPI_E_PARAM_LENGTH`. Otherwise, the service is not done and the return value shall be `E_NOT_OK`.

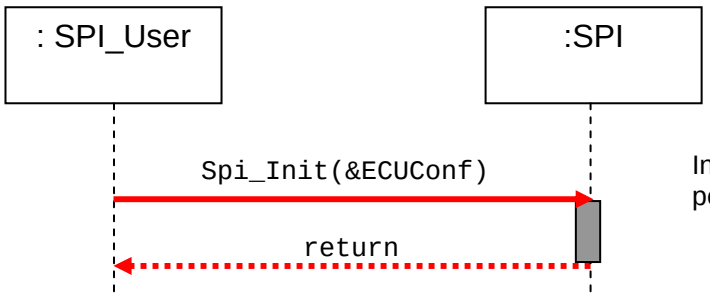
SPI143: `HWUnit` shall be within the specified range of values. Related error value: `SPI_E_PARAM_UNIT`. Otherwise, the service is not done and the return value shall be `SPI_UNINIT`.

8.8 SPI state checking

SPI046: If any other API function is called before initialization, an error should be reported to the DET according to the configuration (see chapter 7.5). Related error detection: `SPI_E_UNINIT`. Functions are not done and, depending on functions either the return value shall be `E_NOT_OK` or a failed result (`SPI_JOB_FAILED` or `SPI_SEQ_FAILED`).

9 Sequence diagrams

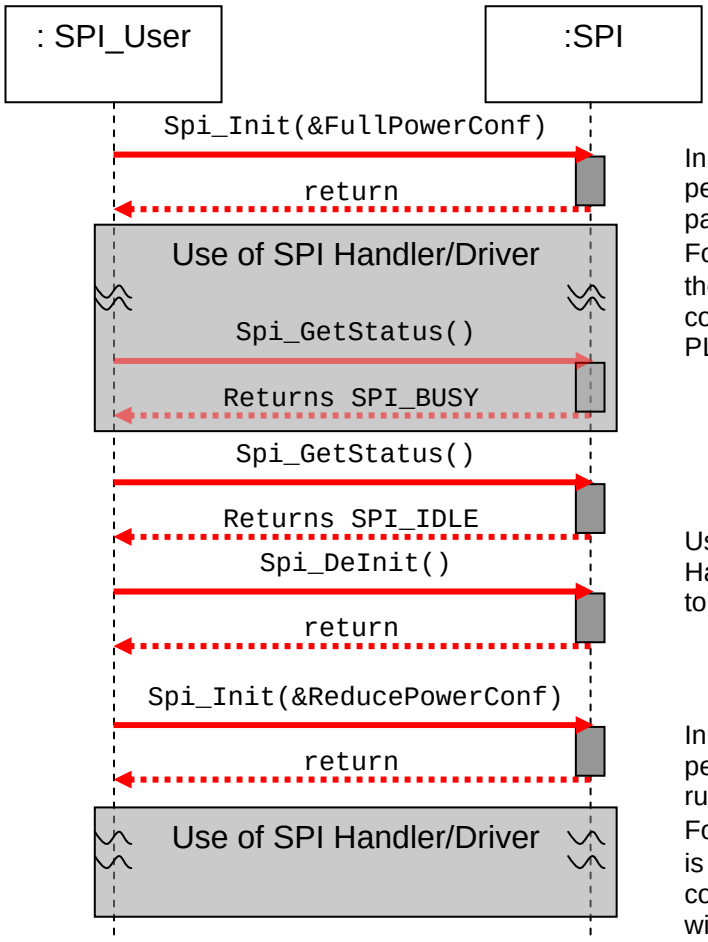
9.1 Initialization



Initialization of SPI Handler/Driver is performed synchronously.

9.2 Modes transitions

SPI079: The following sequence diagram shows an example of an Init / DeInit calls for a running mode transition.



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 the get status service of SPI Handler/Driver to know its state before to de-initialize it.

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.

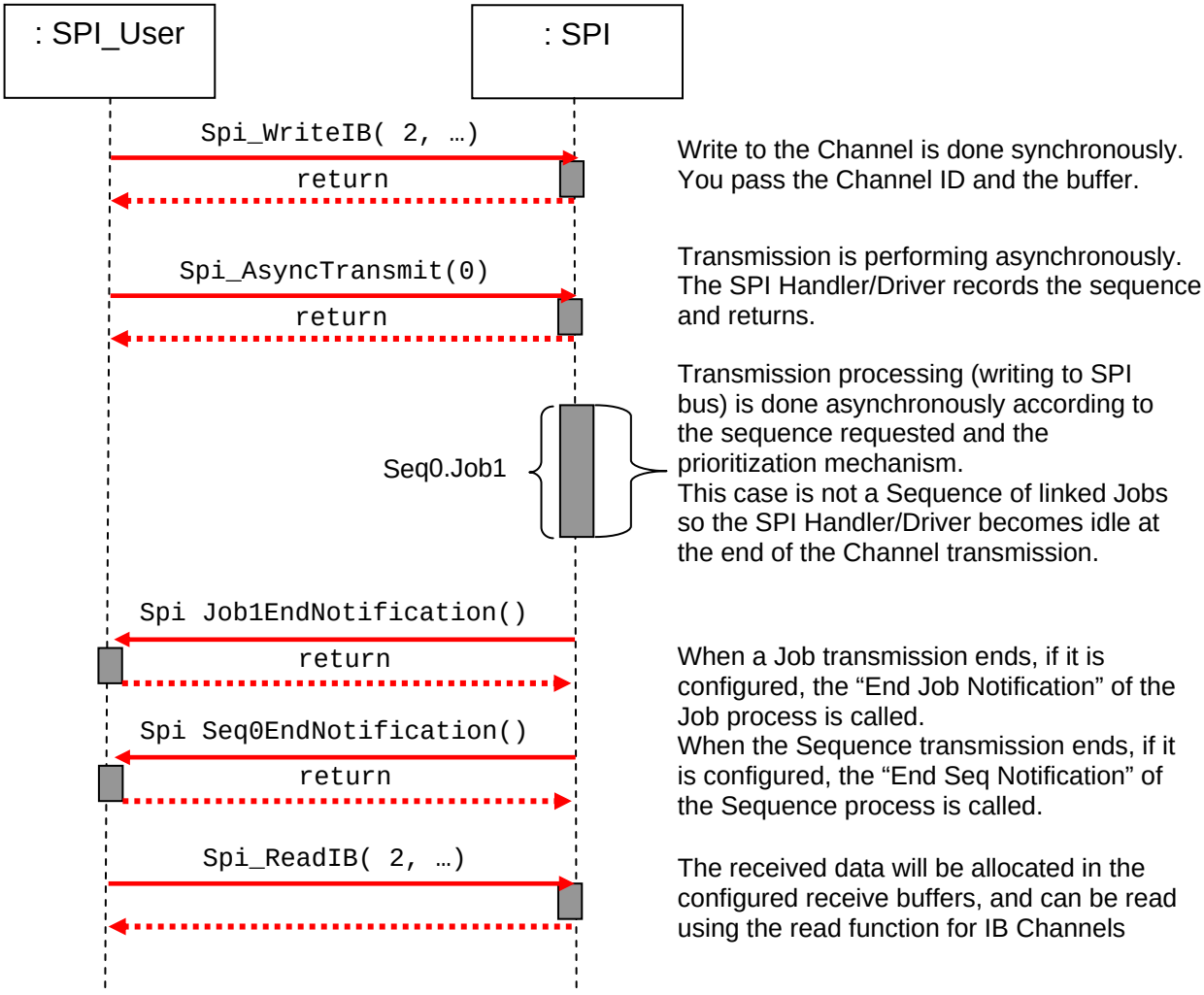
9.3 Write/AsyncTransmit/Read (IB)

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

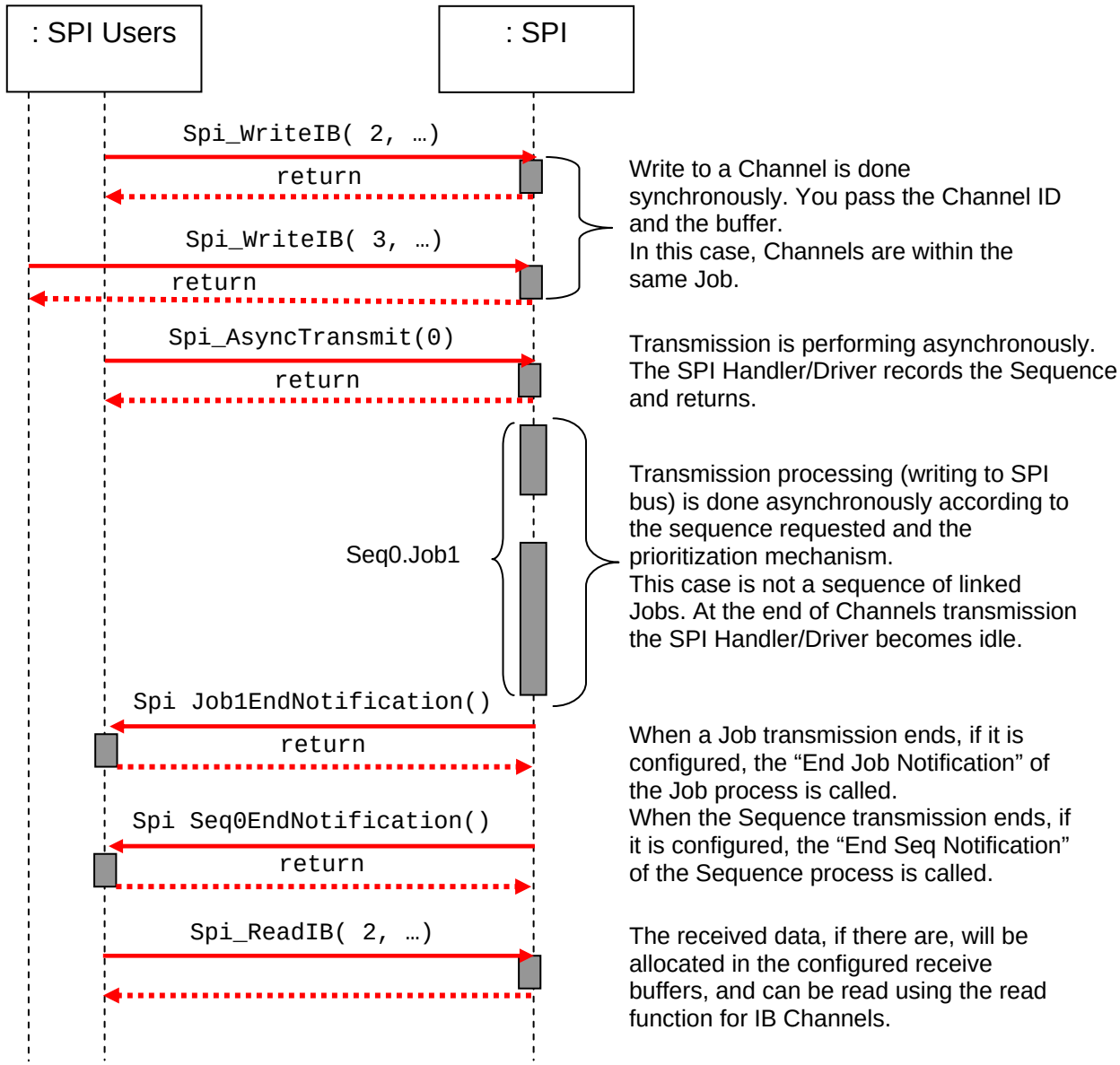


9.3.2 Many Channels, one Job then one Sequence

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

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

Sequence	*Job*	*Channel*
ID0	ID1	ID2
		ID3

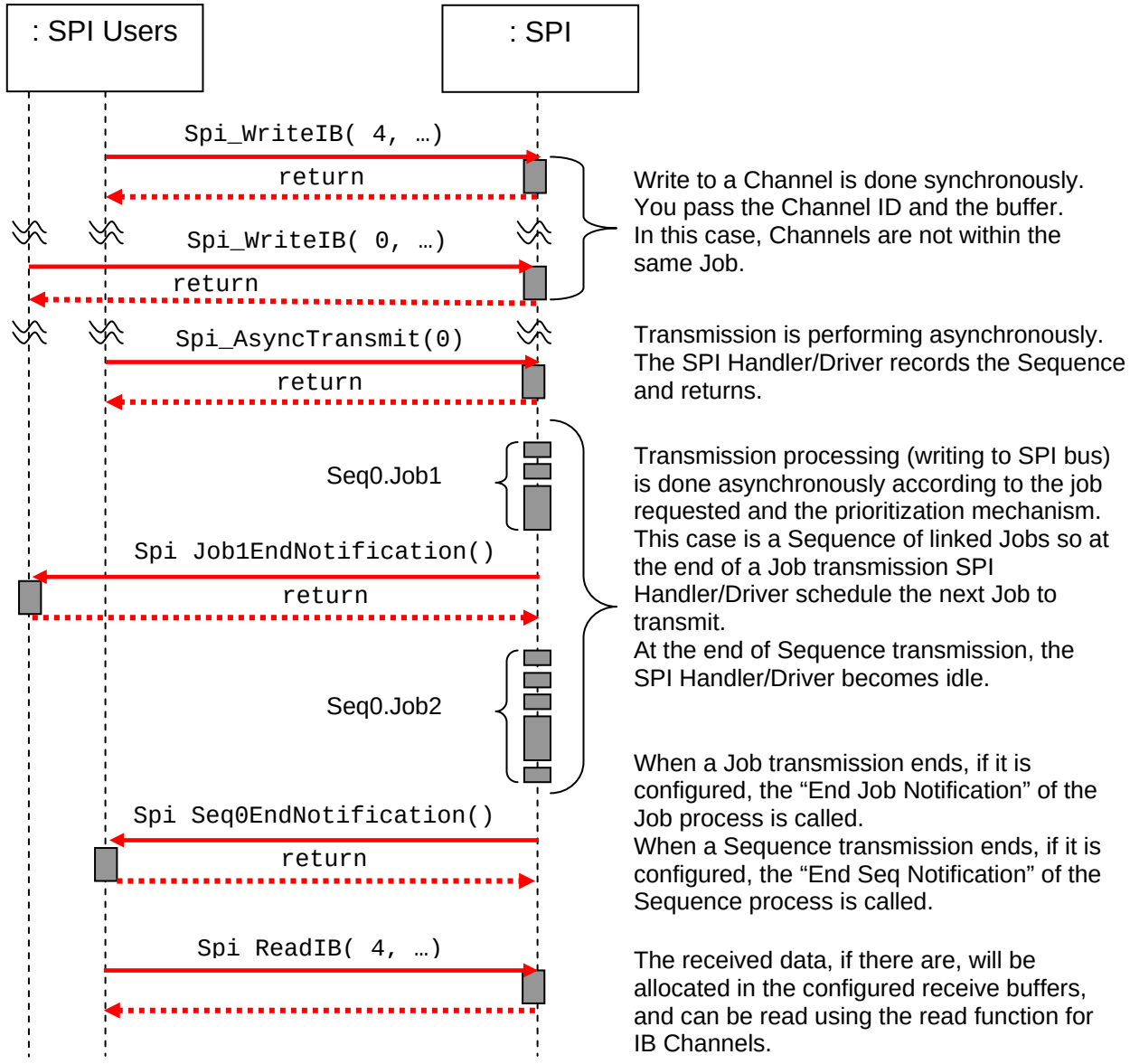


9.3.3 Many Channels, many Jobs and one Sequence

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

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

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

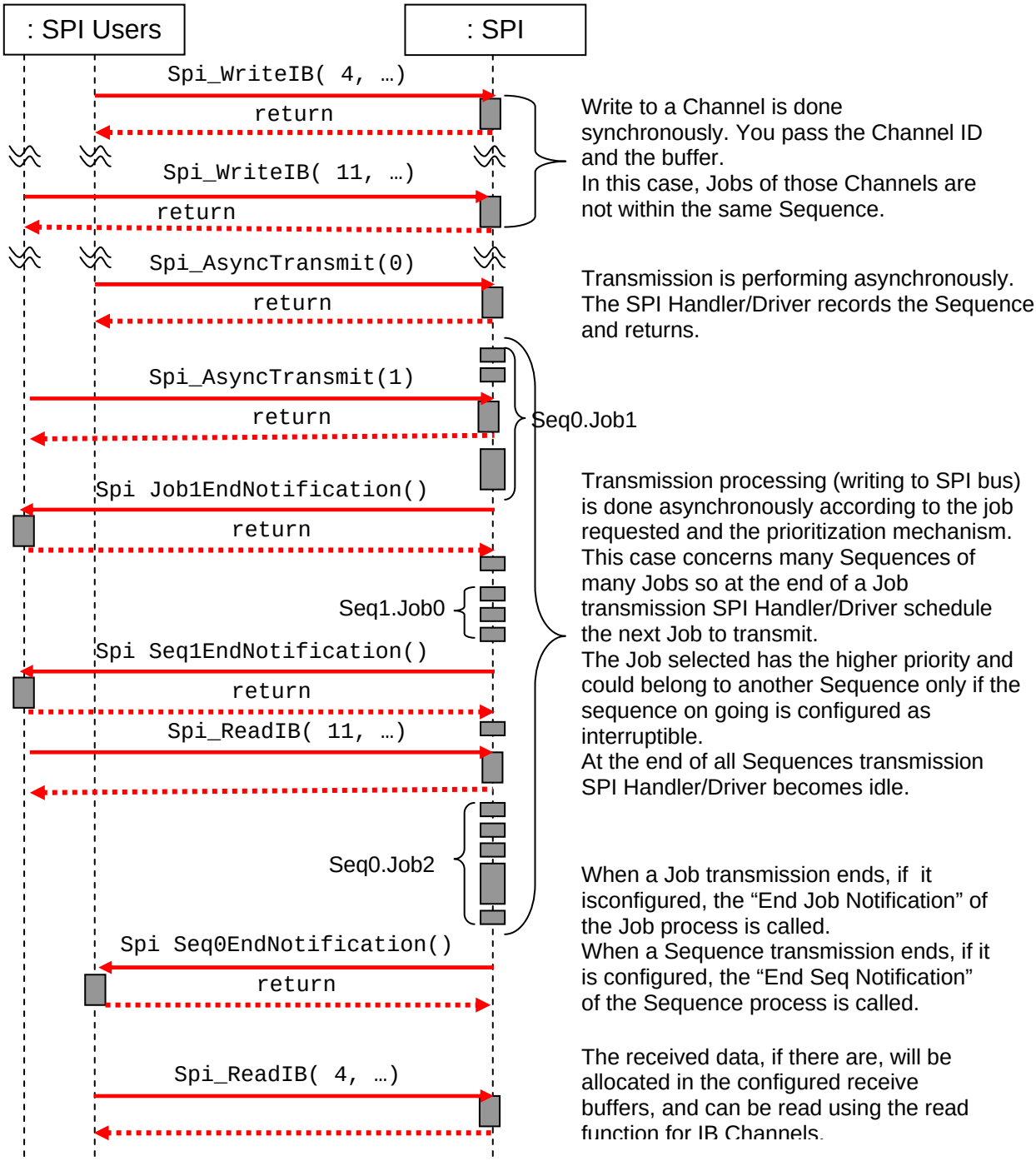


9.3.4 Many Channels, many Jobs and many Sequences

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


: SPI Users
: SPI
Spi_WriteIB(4, …)
return
Spi_WriteIB(11, …)
return
Spi_AsyncTransmit(0)
return
Spi_AsyncTransmit(1)
return
Seq0.Job1
Spi Job1EndNotification()
return
Seq1.Job0
Spi Seq1EndNotification()
return
Spi_ReadIB(11, …)
Seq0.Job2
Spi Seq0EndNotification()
return
Spi_ReadIB(4, …)
Write to a Channel is done synchronously. You pass the Channel ID and the buffer.
In this case, Jobs of those Channels are not within the same Sequence.
Transmission is performing asynchronously. The SPI Handler/Driver records the Sequence and returns.
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.
When a Job transmission ends, if it isconfigured, the “End Job Notification” of the Job process is called.
When a 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 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 `Spi_SetupEB` function to store new parameters within SPI Handler/Driver before each `Spi_AsyncTransmit` function call.

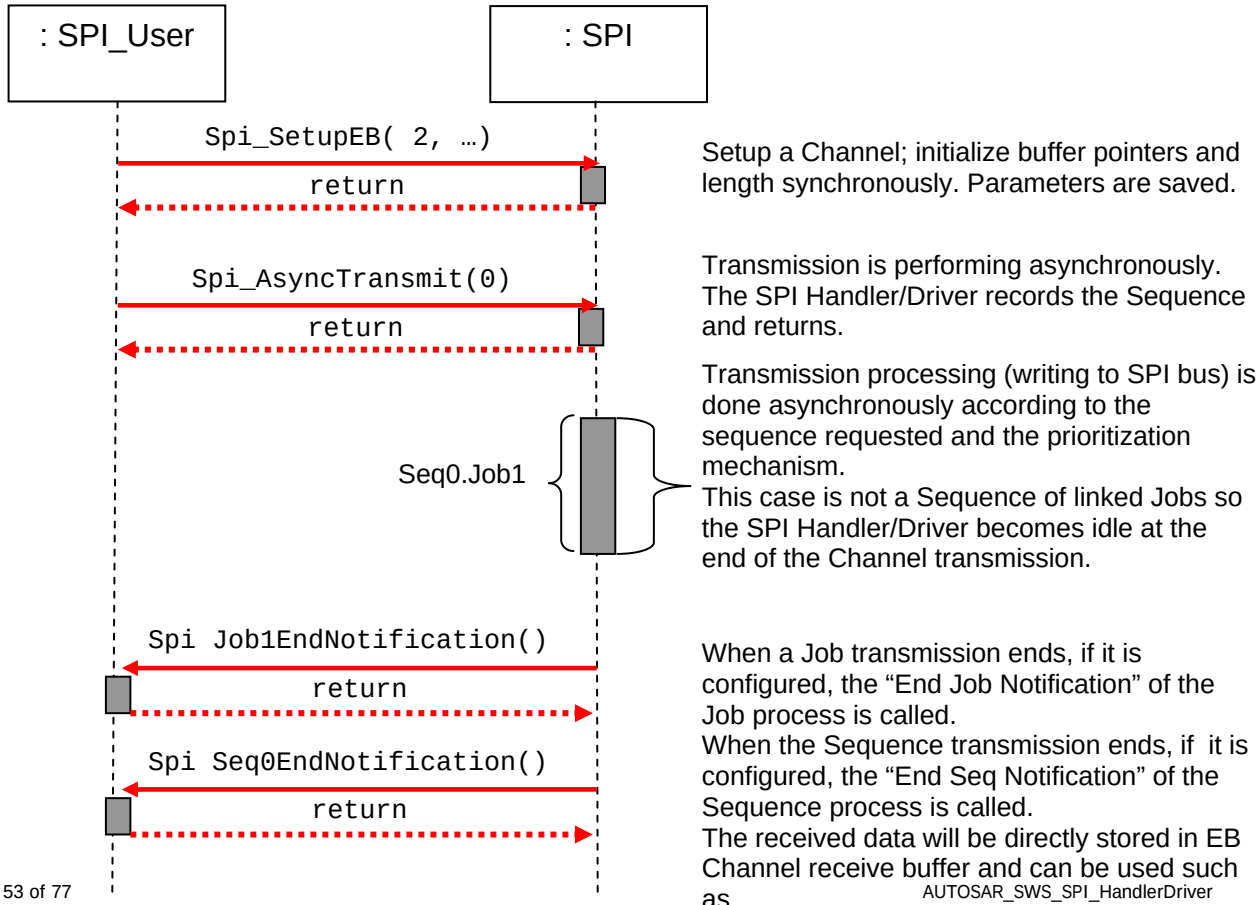
SPI078: To transmit a constant number of data, it is only mandatory to call `Spi_SetupEB` function to store parameters within SPI Handler/Driver before the first `Spi_AsyncTransmit` function call.

9.4.2 One Channel, one Job then one Sequence

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

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

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

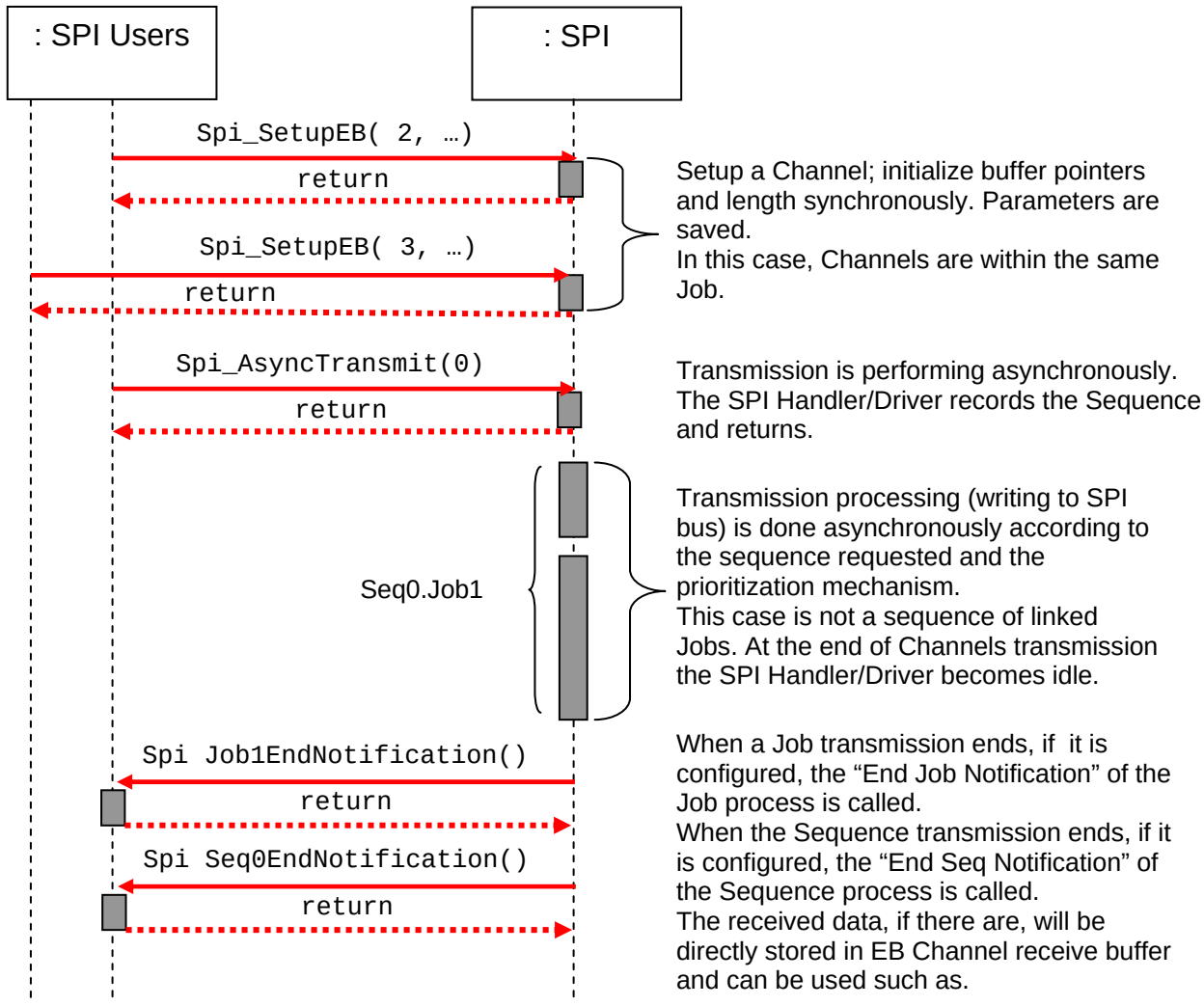


9.4.3 Many Channels, one Job then one Sequence

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

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

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

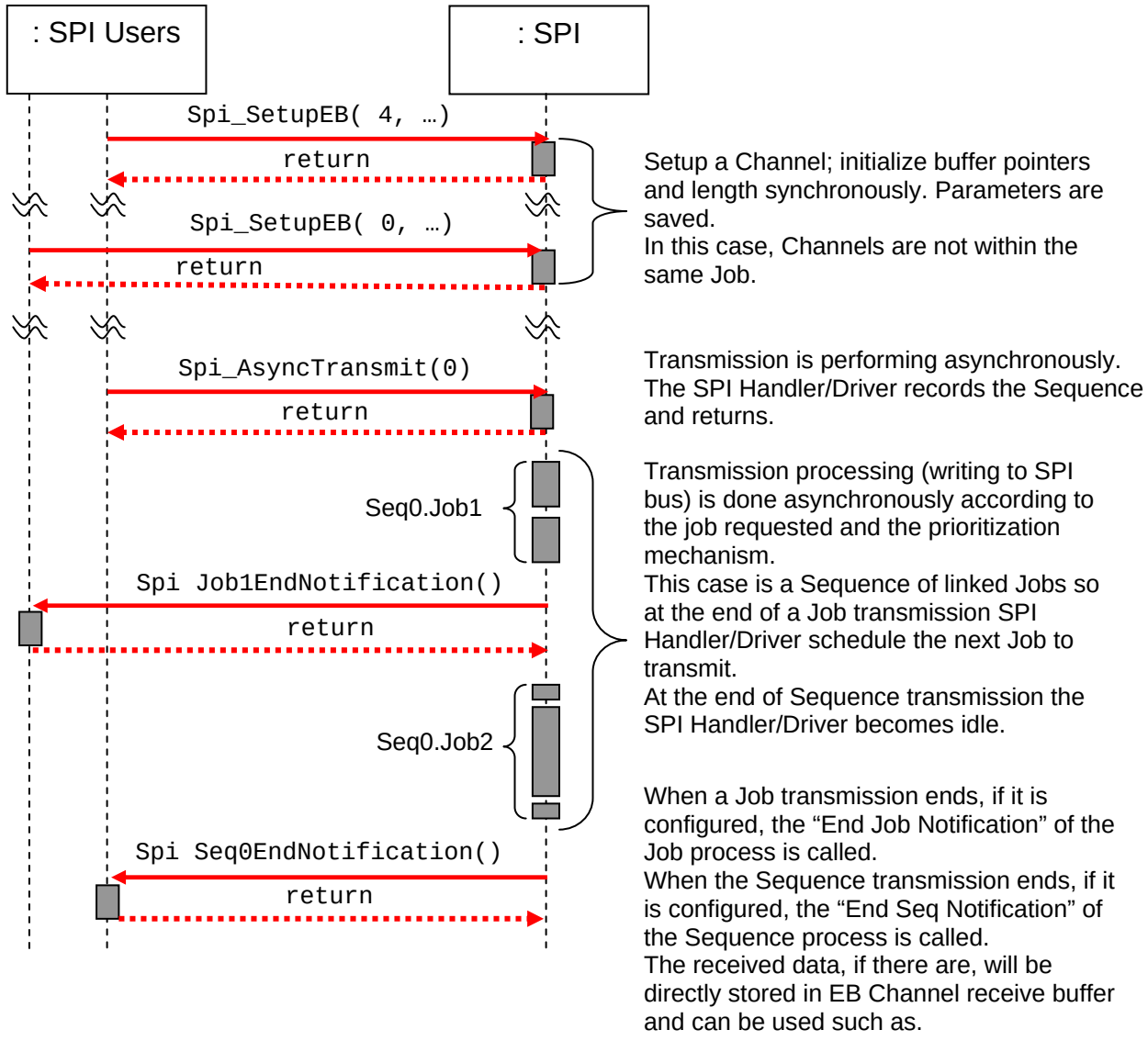


9.4.4 Many Channels, many Jobs and one Sequence

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

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

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

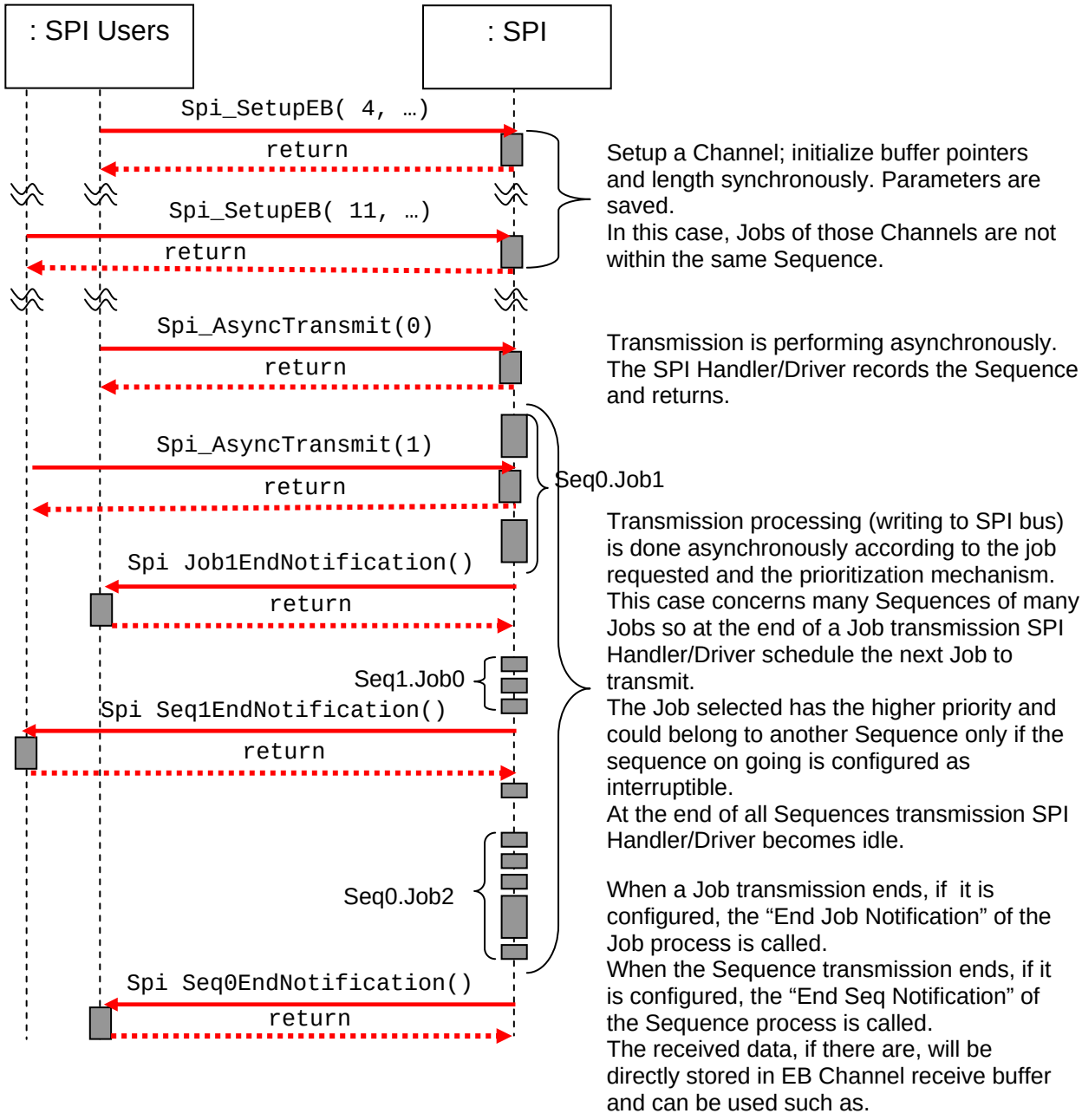


9.4.5 Many Channels, many Jobs and many Sequences

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



9.5 Mixed Jobs Transmission

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

The user knows which Channels are in use, then accordingly to the types of these Channels the appropriated methods shall be called.

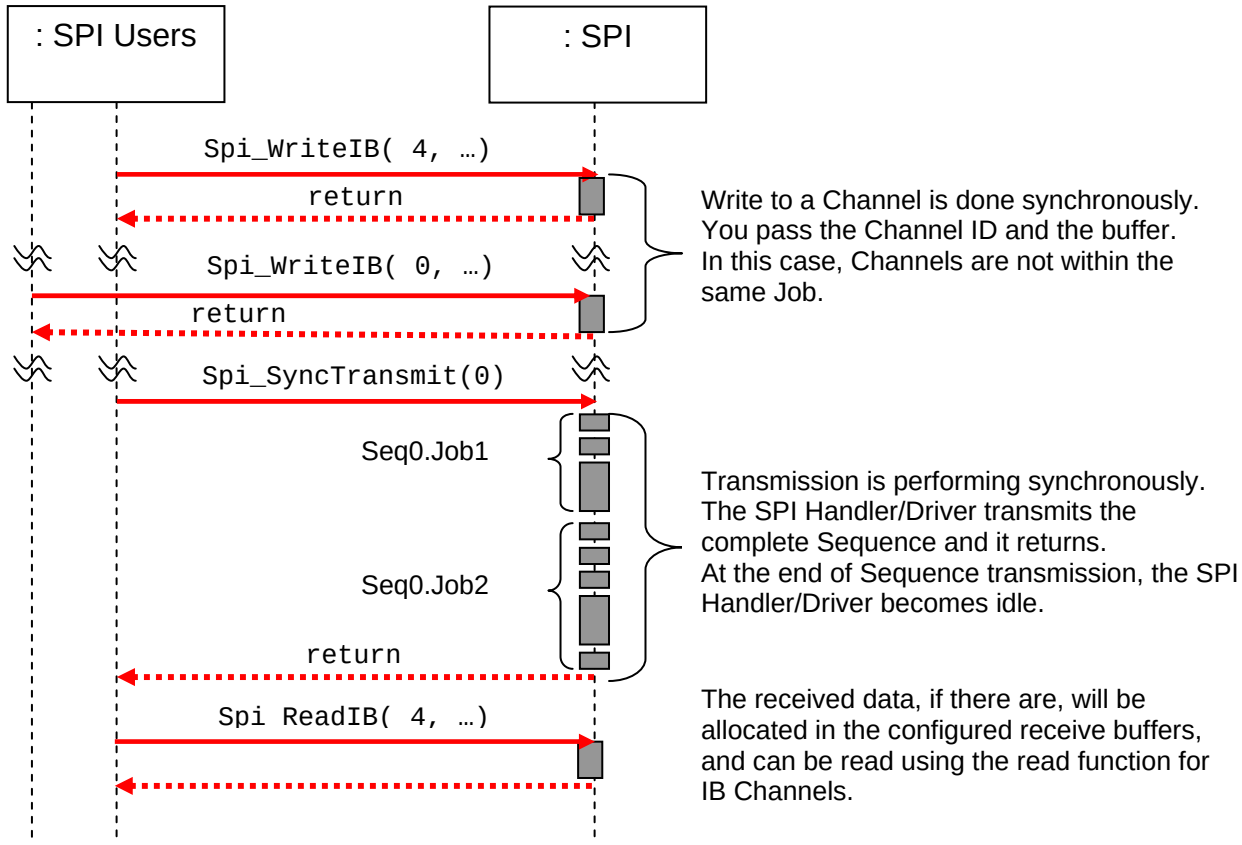
9.6 LEVEL 0 SyncTransmit diagrams

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

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

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

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



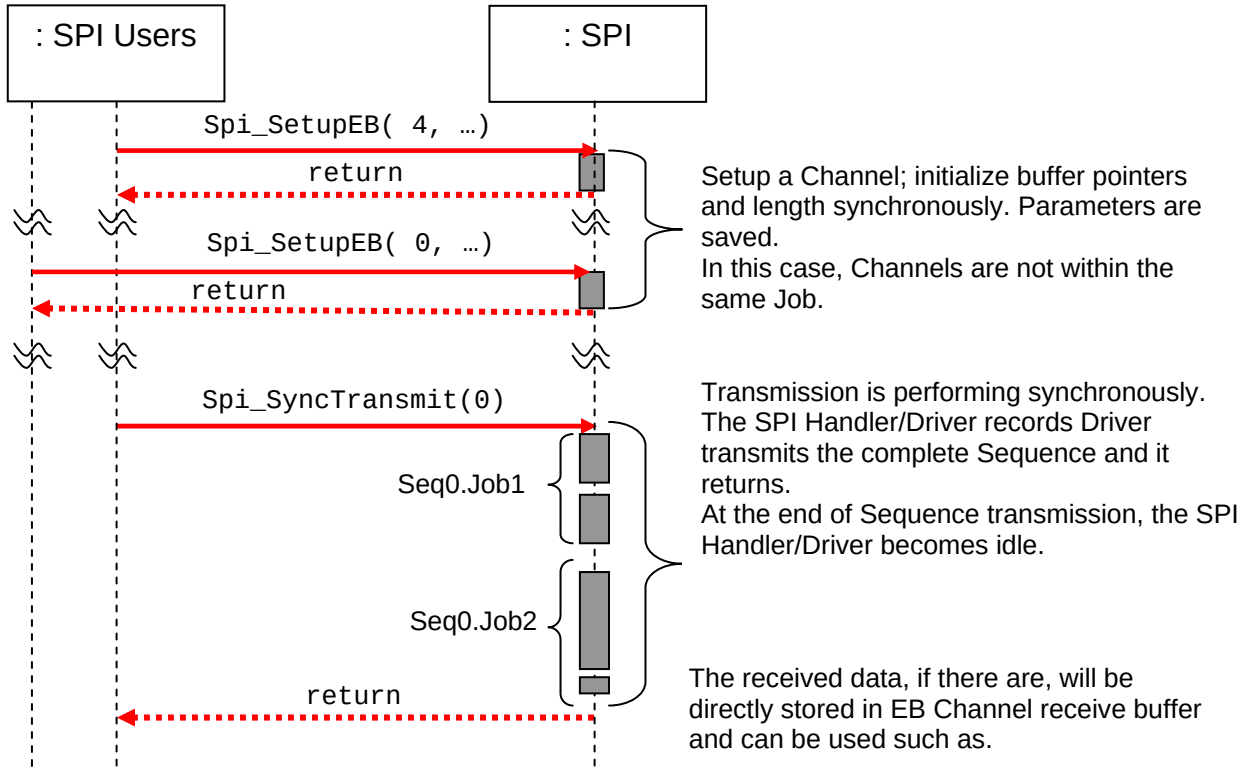
9.6.2 Setup/SyncTransmit (EB): Many Channels, many Jobs and one Sequence

The following sequence diagram shows an example of `Spi_SetupEB` / `Spi_SyncTransmit` calls for a Sequence transmission of linked Jobs. Write or

Read accesses are “User Dependant” and could be skipped when Job is just reading or writing respectively.

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

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



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 describe Chapters 7 and Chapters 8. Further hardware / implementation specific parameters can be added if necessary.

10.2.1 Variants

SPI056: Variant PC: This variant is limited to pre-compile-configuration parameters only. The intention of this variant is to optimize the parameters configuration for a source code delivery.

SPI076: Variant LT: This variant allows a mix of pre-compile time-, link time-configuration parameters. The intention of this variant is to optimize the parameters configuration for an object code delivery.

SPI148: Variant PB: This variant allows a mix of pre-compile time-, post build-time configuration parameters. The intention of this variant is to optimize the parameters configuration for a re-loadable binary.

10.2.2 SpiGeneralConfiguration

SWS Item	SPI103:
Container Name	SpiGeneralConfiguration
Description	This container contains the general configuration (parameters) of the SPI Handler/Driver software module.
Configuration Parameters	

Name	SPI_DEV_ERROR_DETECT		
Description	Switches the Development Error Detection and Notification ON or OFF.		
Type	#define		
Unit	--		
Range	ON	enabled	
	OFF	disabled	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	none		

Name	SPI_VERSION_INFO_API		
Description	Switches the Spi_GetVersionInfo function ON or OFF. (see chapter 8.3.10)		
Type	#define		
Unit	--		
Range	ON	enabled	
	OFF	disabled	
	Pre-compile	x	all Variants
	Link time	--	--

	Post Build	--	--
Scope	module		
Dependency	none		

Name	SPI_LEVEL_DELIVERED		
Description	Selects the SPI Handler/Driver level of scalable functionality that is available and delivered. (see chapter 7.1)		
Type	#define		
Unit	--		
Range	0	The LEVEL 0 Simple Synchronous SPI Handler/Driver functionalities are selected.	
	1	The LEVEL 1 Basic Asynchronous SPI Handler/Driver functionalities are selected.	
	2	The LEVEL 2 Enhanced SPI Handler/Driver functionalities are selected.	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	none		

Name	SPI_CHANNEL_BUFFERS_ALLOWED		
Description	Selects the SPI Handler/Driver Channel Buffers usage allowed and delivered. (see chapter 7.2.1)		
Type	#define		
Unit	--		
Range	0	The Usage 0, only Internal Buffers are selected in Handler/Driver.	
	1	The Usage 1, only External Buffers are selected in Handler/Driver.	
	2	The Usage 2, both Internal and External Buffers are selected in Handler/Driver.	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	none		

Name	SPI_INTERRUPTIBLE_SEQ_ALLOWED		
Description	Switches the Interruptible Sequences handling functionality ON or OFF. (see chapter 7.2.4)		
Type	#define		
Unit	--		
Range	ON	enabled	
	OFF	disabled	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	This parameter depends on SPI_LEVEL_DELIVERED value. It is only used for SPI_LEVEL_DELIVERED configured to 1 or 2.		

Name	SPI_HW_STATUS_API	
Description	Switches the Spi_GetHWUnitStatus function ON or OFF. (see chapters 8.2.10 and 8.3.12)	
Type	#define	
Unit	--	
Range	ON	enabled

	OFF	disabled	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	none		

Name	SPI_CANCEL_API		
Description	Switches the Spi_Cancel function ON or OFF. (see chapter 8.3.13)		
Type	#define		
Unit	--		
Range	ON	enabled	
	OFF	disabled	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	None		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
SpiDriverConfiguration	1..*	ECU	The multiplicity of this container depends on ECU low consumption mode.

10.2.3 SpiDriverConfiguration

SWS Item	SPI091:
Container Name	SpiDriverConfiguration
Description	This container contains the Handler/Driver configuration (parameters) depending on external devices connected, their usage and communication protocols. The organization of this data structure could contain dependencies to the microcontroller so this is left up to the implementer and its location is left up to the configuration.
Configuration Parameters	

Name	SPI_MAX_CHANNEL		
Description	This parameter contains the number of Channels configured. It will be gathered by tools during the configuration stage.		
Type	Spi_ChannelType		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

Name	SPI_MAX_JOB

Description	This parameter contains the number of Jobs configured. It will be gathered by tools during the configuration stage.		
Type	Spi_JobType		
Unit	--		
Range	--	---	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

Name	SPI_MAX_SEQUENCE		
Description	This parameter contains the number of Sequences configured. It will be gathered by tools during the configuration stage.		
Type	Spi_SequenceType		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

Included Containers			
Container Name	***Multiplicity***	***Scope***	***Dependency***
SpiChannelConfiguration	1..SPI_MAX_CHANNEL	ECU	
SpiJobConfiguration	1..SPI_MAX_JOB	ECU	
SpiSequenceConfiguration	1..SPI_MAX_SEQUENCE	ECU	

10.2.4 SpiChannelConfiguration

SWS Item	SPI104:
Container Name	SpiChannelConfiguration
Description	This container contains the configuration (parameters) to describe a Channel as defined above in this document.
Configuration Parameters	

Name	SPI_CHANNEL_NAME		
Description	This parameter is the symbolic name to be used on the API. This symbolic name allows accessing Channel data.		
Type	Spi_ChannelType		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	none		

Name	SPI_CHANNEL_TYPE
Description	This parameter is the buffer type used by this Channel.

Type	--		
Unit	--		
Range	IB	SPI Handler/Driver Internal Buffer	
	EB	External Buffer	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	ECU		
Dependency	SPI_CHANNEL_BUFFERS_ALLOWED		

Name	SPI_DATA_WIDTH		
Description	This parameter is the width of a transmitted data unit.		
Type	--		
Unit	bits		
Range	1..32		
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

Name	SPI_IB_N_BUFFERS		
Description	This parameter contains the maximum number of data buffers in case of IB Channels and only.		
Type	Spi_NumberOfDataType		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
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.		

Name	SPI_EB_MAX_LENGTH		
Description	This parameter contains the maximum size of data buffers in case of EB Channels and only.		
Type	Spi_NumberOfDataType		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
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.		

Name	SPI_TRANSFER_START	
Description	This parameter defines the first starting bit for transmission.	
Type	--	
Unit	--	
Range	LSB	Transmission starts with the Least Significant Bit first
	MSB	Transmission starts with the Most

		Significant Bit first	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

Name	SPI_DEFAULT_DATA		
Description	This parameter is the default value to transmit.		
Type	--		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

10.2.5 SpiJobConfiguration

SWS Item	SPI105:
Container Name	SpiJobConfiguration
Description	This container contains the configuration (parameters) to describe a Job as defined above in this document.
Configuration Parameters	

Name	SPI_JOB_NAME		
Description	This parameter is the symbolic name to be used on the API. This symbolic name allows accessing Job information.		
Type	Spi_JobType		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	none		

Name	SPI_HW_UNIT		
Description	This parameter is the symbolic name to identify the HW SPI Hardware microcontroller peripheral allocated to this Job.		
Type	Spi_HWUnitType		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

Name	SPI_ENABLE_CS
Description	This parameter enables or not the Chip Select handling functions.
Type	--
Unit	boolean

Range	TRUE	Enable Chip Select handling function	
	FALSE	Disable Chip Select handling function	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

Name	SPI_CS_POLARITY		
Description	This parameter defines the active polarity of Chip Select.		
Type	--		
Unit	--		
Range	HIGH	Active polarity Chip Select is a high voltage level	
	LOW	Active polarity Chip Select is a low voltage level	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

Name	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.		
Type	Hardware dependent / Implementation dependent		
Unit	Hz		
Range	--	Range of values configurable by tools.	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

Name	SPI_TIME_CLK2CS		
Description	This parameter is the minimum time between clock and Chip Select – This parameter allows using a range of values, from the point of view of configuration tools, from 0 up to 100µs.		
Type	Hardware dependent / Implementation dependent		
Unit	µs		
Range	0..100	Range of values configurable by tools.	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

Name	SPI_SHIFT_CLOCK_IDLE_LEVEL		
Description	This parameter defines the SPI shift clock idle level.		
Type	--		
Unit	--		
Range	HIGH	Shift clock idle level is a high voltage level	
	LOW	Shift clock idle level is a low voltage level	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		

Dependency	none

Name	SPI_DATA_SHIFT_EDGE		
Description	This parameter defines the SPI data shift edge.		
Type	--		
Unit	--		
Range	LEADING	First data shift edge is a leading edge	
	TRAILING	First data shift edge is a trailing edge	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

Name	SPI_JOB_PRIORITY		
Description	This parameter defines the Job priority.		
Type	--		
Unit	--		
Range	0..3	Range of configurable values	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	none		

Name	SPI_JOB_END_NOTIFICATION		
Description	This parameter is a reference to a notification function.		
Type	Function pointer		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	ECU		
Dependency	This parameter depends on SPI_LEVEL_DELIVERED value.It is only used for SPI_LEVEL_DELIVERED configured to 1 or 2		

Name	SPI_CHANNEL_LINKING		
Description	This parameter should reference to a list of linked Channels. The type is off in order to optimize the source code and because this parameter is only used in the SPI Handler/Driver module.		
Type	--		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	ECU		
Dependency	none		

10.2.6 SpiSequenceConfiguration

SWS Item	SPI106:
Container Name	SpiSequenceConfiguration
Description	This container contains the configuration (parameters) to describe a Sequence as defined above in this document.
Configuration Parameters	

Name	SPI_SEQUENCE_NAME		
Description	This parameter is the symbolic name to be used on the API. This symbolic name allows accessing Sequence information.		
Type	Spi_SequenceType		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	all Variants
	Link time	--	--
	Post Build	--	--
Scope	ECU		
Dependency	none		

Name	SPI_INTERRUPTIBLE_SEQUENCE		
Description	This parameter allows or not this Sequence to be suspended by another one.		
Type	--		
Unit	boolean		
Range	TRUE	Sequence could be suspended	
	FALSE	Sequence is not suspendable	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	module		
Dependency	This SPI_INTERRUPTIBLE_SEQ_ALLOWED parameter as to be configured as ON.		

Name	SPI_SEQ_END_NOTIFICATION		
Description	This parameter is a reference to a notification function.		
Type	Function pointer		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB
Scope	ECU		
Dependency	This parameter depends on SPI_LEVEL_DELIVERED value.It is only used for SPI_LEVEL_DELIVERED configured to 1 or 2		

Name	SPI_JOB_LINKING		
Description	This parameter should reference to a list of Jobs. The type is off in order to optimize the source code and because this parameter is only used in the SPI Handler/Driver module.		
Type	--		
Unit	--		
Range	--	--	
Configuration Class	***Pre-compile***	x	Variant PC
	Link time	x	Variant LT
	Post Build	x	Variant PB

Scope	ECU
Dependency	none

10.3 Published parameters

SPI089: The following table specifies information that are 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.

SPI068: This published information is provided in the module's description for use by configuration tools. Further hardware / implementation specific parameters can be added if necessary.

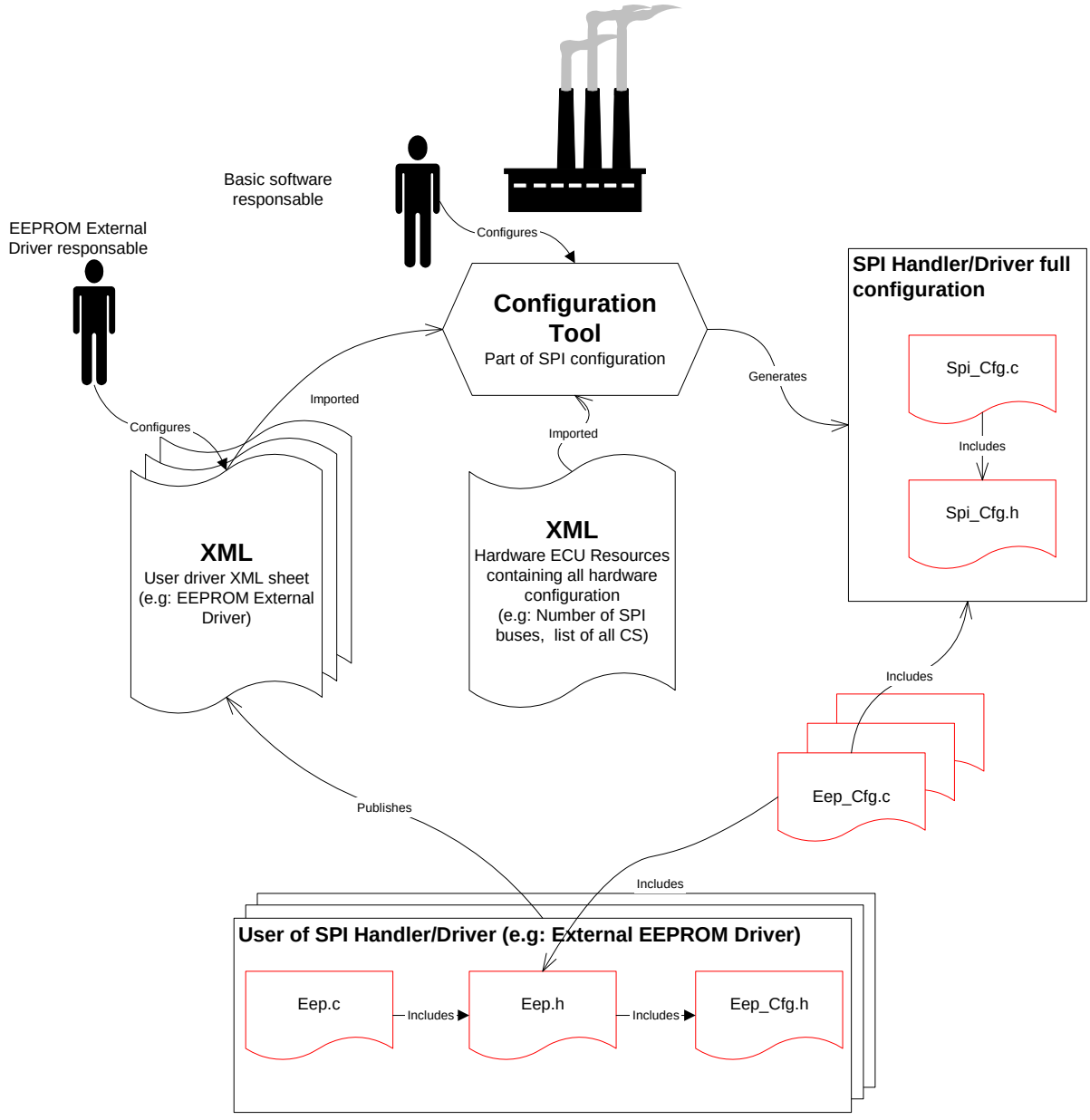
SWS Item	SPI094:	
Information elements		
Information element name	***Type / Range***	***Information element description***
`SPI_VENDOR_ID`	`#define/ uint16`	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
`SPI_MODULE_ID`	`#define/ uint8`	Module ID of this module from Module List
`SPI_AR_MAJOR_VERSION`	`#define/ uint8`	Major version number of AUTOSAR specification on which the appropriate implementation is based on.
`SPI_AR_MINOR_VERSION`	`#define/ uint8`	Minor version number of AUTOSAR specification on which the appropriate implementation is based on.
`SPI_AR_PATCH_VERSION`	`#define/ uint8`	Patch level version number of AUTOSAR specification on which the appropriate implementation is based on.
`SPI_SW_MAJOR_VERSION`	`#define/ uint8`	Major version number of the vendor specific implementation of the module. The numbering is vendor specific.
`SPI_SW_MINOR_VERSION`	`#define/ uint8`	Minor version number of the vendor specific implementation of the module. The numbering is vendor specific.
`SPI_SW_PATCH_VERSION`	`#define/ uint8`	Patch level version number of the vendor specific implementation of the module. The numbering is vendor specific.
`SPI_HW_UNIT_SYNCHRONOUS`	`-- / --`	Prearranged information of the SPI bus dedicated for synchronous transfers in SPI Handler/Driver LEVEL 2. It will help for configuration and usage.
`SPI_MAX_HW_UNIT`	`#define/ --`	Number of different SPI hardware microcontroller peripherals (units/busses) available and handled by this SPI Handler/Driver module. It will help for configuration and usage.

10.4 Configuration concept

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

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

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

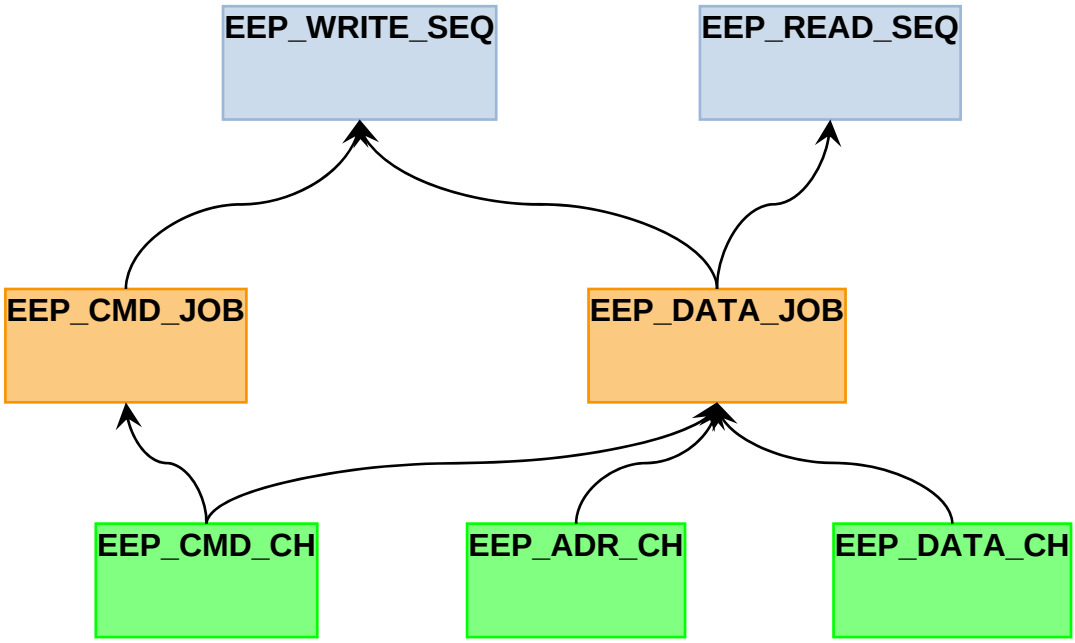


The steps on the diagrams are:

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

11 Appendix

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



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

12 Changes to Release 1

12.1 Deleted SWS Items

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

12.2 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.

12.3 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 according to a bug from Bugzilla accepted.
SPI052	Changes according to a bug from Bugzilla accepted.
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

12.4 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 SpiChannelConfiguration container with all its parameters.
SPI105	Creation of SpiJobConfiguration container with all its parameters.
SPI106	Creation of SpiSequenceConfiguration with all its parameters.
SPI107	Introduction of the selectable levels of functionalities concept.
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 LEVEL 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 behavior.
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.
SPI132	Applicability of requirements coming from LEVEL 0 and LEVEL 1.
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 interrupt 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.