# MICROSAR SPI

## Technical Reference

MCAL Emulation in VTT

Version 1.1.0

|  |  |
|---|---|
| Authors | Christian Leder |
| Status | Released |

# Document Information

## History

| Author | Date | Version | Remarks |
|---|---|---|---|
| Christian Leder | 2014-03-03 | 1.00.00 | Initial version |
| Christian Leder | 2015-02-15 | 1.01.00 | > Global renaming of Vip to Vtt<br>> Usage of template 5.11.0 for the Technical reference |

## Reference Documents

| No. | Source | Title | Version |
|---|---|---|---|
| [1] | AUTOSAR | AUTOSAR_SWS_SPIHandlerDriver.pdf | V3.2.0 |
| [2] | AUTOSAR | AUTOSAR_SWS_DevelopmentErrorTracer.pdf | V3.2.0 |
| [3] | AUTOSAR | AUTOSAR_SWS_DiagnosticEventManager.pdf | V4.2.0 |
| [4] | AUTOSAR | AUTOSAR_TR_BSWModuleList.pdf | V1.6.0 |

**Caution**
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

## Contents

## Illustrations

## Tables

# 1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

| Component Version | New Features |
|---|---|
| 1.0.x | Initial version of the Vip SPI driver |
| 2.0.x | Global renaming of Vip to Vtt |

Table 1-1 Component history

# 2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module SPI as specified in [1].

| | | |
|---|---|---|
| **Supported AUTOSAR Release*:** | 4 | |
| **Supported Configuration Variants:** | pre-compile | |
| **Vendor ID:** | SPI_VENDOR_ID | 30 decimal<br>(= Vector-Informatik, according to HIS) |
| **Module ID:** | SPI_MODULE_ID | 083 decimal<br>(according to ref. [4]) |

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

The SPI driver provides services for basic communication with external components. In this case, the SPI environment is emulated. Instead of communicating with components, the module does not read or write anything. Actually, the SPI driver fulfills development error checks and serves as a stub.

The main tasks of the SPI normally are:

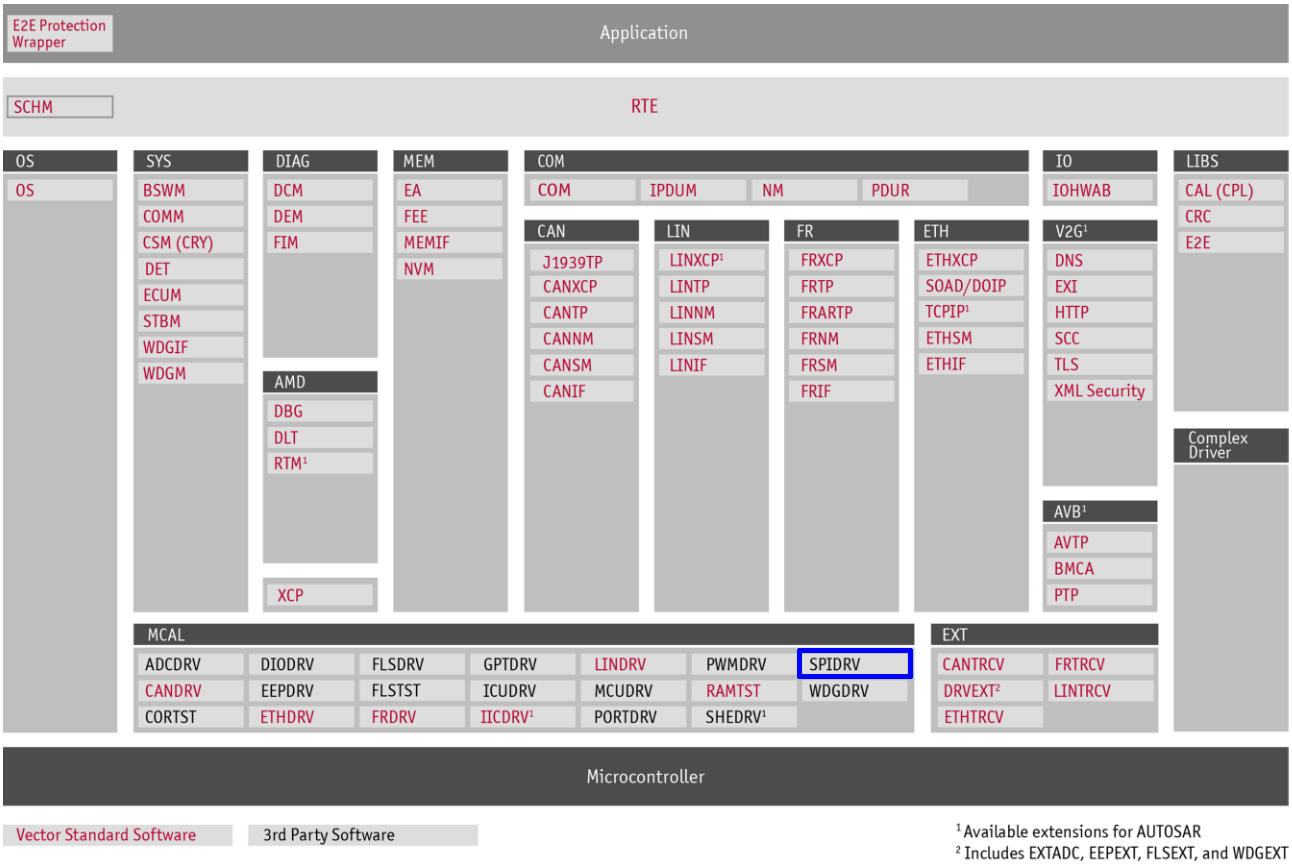
- Handle the SPI hardware units onboard
- Handle data transmission to the components connected via SPI
- Take care of the settings required by external components (baud rate etc.)

The development of the emulation of this component has not finished yet. In the future any functionality will be implemented.

**Caution**
Please notify that this software implementation does not have any transmission functionality!

## 2.1 Architecture Overview

The following figure shows where the SPI is located in the AUTOSAR architecture.



[1] Available extensions for AUTOSAR

[2] Includes EXTADC, EEPEXT, FLSEXT, and WDGEXT

Figure 2-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the SPI. These interfaces are described in chapter 5.

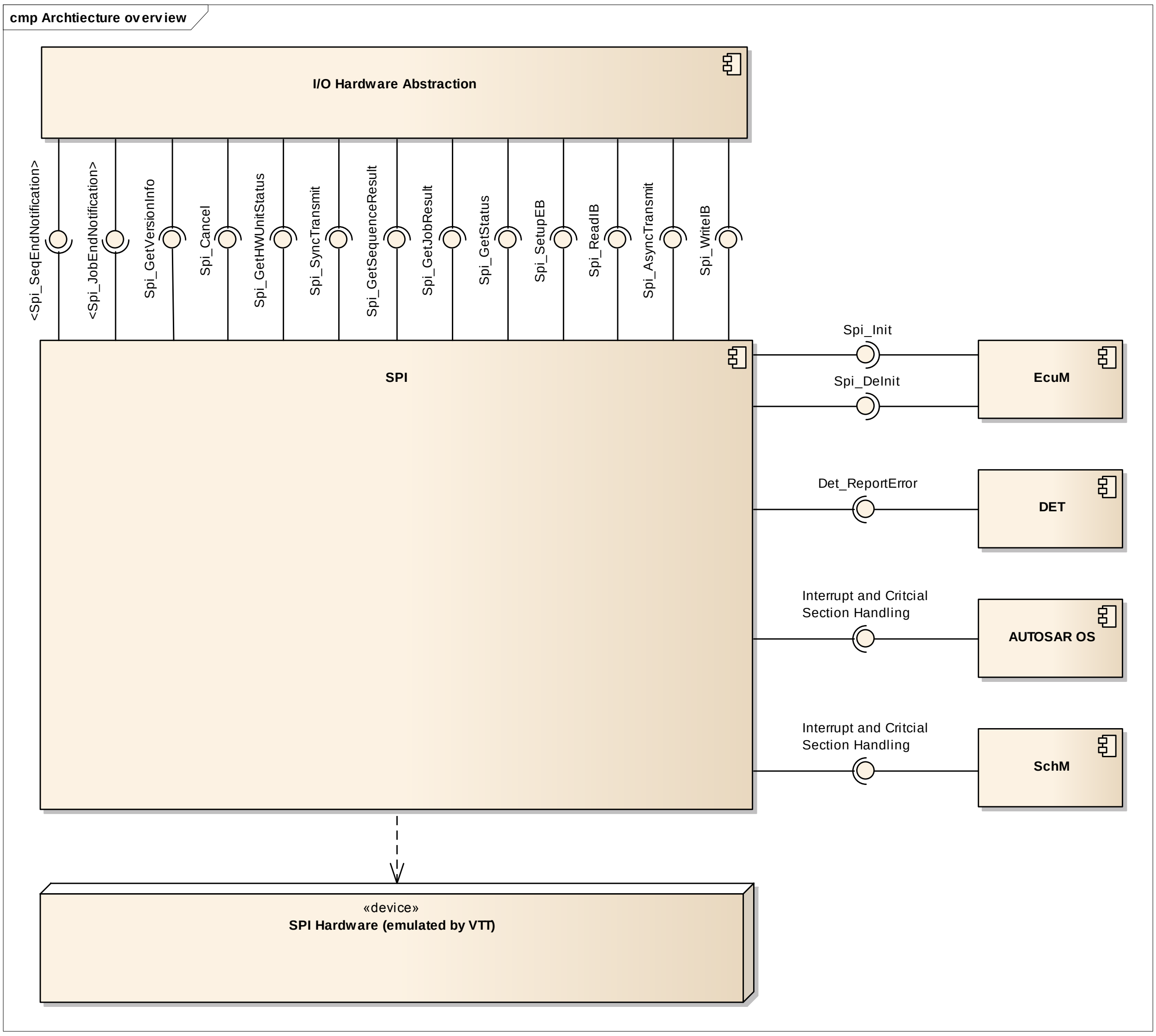


Figure 2-2 Interfaces to adjacent modules of the SPI

# 3 Functional Description

## 3.1 Features

The features listed in the following tables cover the complete functionality specified for the SPI.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further SPI functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

| Supported AUTOSAR Standard Conform Features |
|---|
| Configure SPI with<br>> External devices<br>> Channels<br>> Jobs<br>> Sequences |
| Configure physical units and callback functions |
| Configure error detection (DET) |
| Configure implementation features like:<br>> Spi scalability level(s). The AUTOSAR specification defined three levels. LEVEL0 offers only synchronous transfer of sequences. LEVEL1 offers only asynchronous sequence transfers. LEVEL2 includes both modes to be used (not supported).<br>> Spi channel buffers<br>> Spi Interrupts |

Table 3-1 Supported AUTOSAR standard conform features

### 3.1.1 Deviations

The following features specified in [1] are not supported:

| Not Supported AUTOSAR Standard Conform Features |
|---|
| The implemented software does not have any functionality unless development error checks |

Table 3-2 Not supported AUTOSAR standard conform features

### 3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

| Features Provided Beyond The AUTOSAR Standard |
|---|
| In addition to the existing checks required by the AUTOSAR standard, the parameter versioninfo passed to the service `Spi_GetVersionInfo()` is checked for not referencing `NULL_PTR`. If it does, the error `SPI_E_PARAM_VINFO` is reported to DET instead of `SPI_E_PARAM_POINTER` |
| In addition, if the parameter passed to the service `Spi_Init` references `NULL_PTR`, the error `SPI_E_PARAM_CONFIG` is reported to DET instead of `SPI_E_PARAM_POINTER` |
| In addition, if the parameter passed to the service `Spi_SetAsyncMode` contains a value unequal `SPI_POLLING_MODE` or `SPI_INTERRUPT_MODE`, the error `SPI_E_PARAM_MODE` is reported to DET |

Table 3-3 Features provided beyond the AUTOSAR standard

### 3.1.3 Limitations

**Caution**
No transmission functionality is implemented in this software version.

## 3.2 Emulation

This driver is an emulation of an SPI module.

**Caution**
Be careful using while loops in order to poll any status.

The user has to ensure, that the application does not block the emulation. So, within every while loop the following function call has to be called:

```
while(ANY_STATUS == temp_status)
{
   Schedule();
}
```

Use the function call Schedule() which is available once the header file of the module SPI is included.

## 3.3 Initialization

The SPI module is being initialized by calling `Spi_Init(&SpiChannelConfigSet)`. All global variables are initialized by calling `Spi_InitMemory()`. So, `Spi_InitMemory()` has to be called prior to `Spi_Init()`.

## 3.4 States

The SPI maintains states for:

- The SPI driver itself
- The configured jobs
- The configured sequences
- The hardware

These states can be obtained by:

- Spi_GetStatus
- Spi_GetJobResult
- Spi_GetSequenceResult
- Spi_GetHWUnitStatus

## 3.5 Main Functions

`Spi_MainFunction_Handling` has to be called cyclically for processing asynchronous sequences. This is typically done by the Schedule Manager SchM.

## 3.6 Error Handling

### 3.6.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `SPI_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported SPI ID is 083.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

| Service ID | Service |
|---|---|
| 0x00 | Spi_Init |
| 0x01 | Spi_DeInit |
| 0x02 | Spi_WriteIB |
| 0x03 | Spi_AsyncTransmit |
| 0x04 | Spi_ReadIB |
| 0x05 | Spi_SetupEB |
| 0x06 | Spi_GetStatus |
| 0x07 | Spi_GetJobResult |
| 0x08 | Spi_GetSequenceResult |

| Service ID | Service |
|---|---|
| 0x09 | Spi_GetVersionInfo |
| 0x0A | Spi_SyncTransmit |
| 0x0B | Spi_GetHWUnitStatus |
| 0x0C | Spi_Cancel |
| 0x0D | Spi_SetAsyncMode |
| 0x10 | Spi_MainFunction_Handling |

Table 3-4 Service IDs

The errors reported to DET are described in the following table:

| Error Code | | Description |
|---|---|---|
| 0x0A | `SPI_E_PARAM_CHANNEL` | API service called with wrong parameter |
| 0x0B | `SPI_E_PARAM_JOB` | API service called with wrong parameter |
| 0x0C | `SPI_E_PARAM_SEQ` | API service called with wrong parameter |
| 0x0D | `SPI_E_PARAM_LENGTH` | API service called with wrong parameter |
| 0x0E | `SPI_E_PARAM_UNIT` | API service called with wrong parameter |
| 0x0F | `SPI_E_PARAM_CONFIG` | API service called with wrong parameter |
| 0x10 | `SPI_E_PARAM_POINTER` | API service called with parameter referencing `NULL_PTR` |
| 0x11 | `SPI_E_PARAM_MODE` | API service called with wrong parameter |
| 0x12 | `SPI_E_PARAM_VINFO` | API `Spi_GetVersionInfo` service called with parameter referencing `NULL_PTR` |
| 0x1A | `SPI_E_UNINIT` | API service used without module initialization |
| 0x2A | `SPI_E_SEQ_PENDING` | API called if the sequence is already in state `SPI_SEQ_PENDING` or the requested sequence shares jobs with another sequence that is in state `SPI_SEQ_PENDING` |
| 0x3A | `SPI_E_SEQ_IN_PROCESS` | Synchronous transmission service called at wrong time |
| 0x4A | `SPI_E_ALREADY_INITIALIZED` | API `Spi_Init` service called while the SPI driver has already been initialized |

Table 3-5 Errors reported to DET

### 3.6.1.1 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 3-6 are internal parameter checks of the API functions. These checks are for development error reporting and can be en-/disabled.

The following table shows which parameter checks are performed on which services:

| Check<br>Service | SPI_E_PARAM_CHANNEL | SPI_E_PARAM_JOB | SPI_E_PARAM_SEQ | SPI_E_PARAM_LENGTH | SPI_E_PARAM_UNIT | SPI_E_PARAM_CONFIG | SPI_E_PARAM_POINTER | SPI_E_PARAM_MODE | SPI_E_PARAM_VINFO | SPI_E_UNINIT | SPI_E_SEQ_PENDING | SPI_E_SEQ_IN_PROCESS | SPI_E_ALREADY_INITIALIZED |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Spi_Init | | | | | | ■ | | | | | | | ■ |
| Spi_DeInit | | | | | | | | | | ■ | | | |
| Spi_WriteIB | ■ | | | | | | | | | ■ | | | |
| Spi_AsyncTransmit | | | ■ | | | | | | | ■ | ■ | | |
| Spi_ReadIB | ■ | | | | | | ■ | | | ■ | | | |
| Spi_SetupEB | ■ | | | ■ | | | | | | ■ | | | |
| Spi_GetStatus | | | | | | | | | | | | | |
| Spi_GetJobResult | | ■ | | | | | | | | ■ | | | |
| Spi_GetSequenceResult | | | ■ | | | | | | | ■ | | | |
| Spi_GetVersionInfo | | | | | | | | | ■ | | | | |
| Spi_SyncTransmit | | | ■ | | | | | | | ■ | ■ | | |
| Spi_GetHWUnitStatus | | | | | ■ | | | | | ■ | | | |
| Spi_Cancel | | | ■ | | | | | | | ■ | | | |
| Spi_SetAsncMode | | | | | | | | ■ | | ■ | | | |
| Spi_MainFunction_Handling | | | | | | | | | | ■ | | | |
| Spi_MainFunction_Driving | | | | | | | | | | | | | |

Table 3-6 Development Error Reporting: Assignment of checks to services

### 3.6.2 Production Code Error Reporting

**Info**
Production errors are not supported in this emulation.

# 4 Integration

This chapter gives necessary information for the integration of the MICROSAR SPI into an application environment of an ECU.

## 4.1 Scope of Delivery

The delivery of the SPI contains the files which are described in the chapters 4.1.1 and 4.1.2:

### 4.1.1 Static Files

| File Name | Description |
|---|---|
| Spi.h | The module header defines the interface of the SPI. This file must be included by upper layer software components |
| Spi.c | This C-source contains the implementation of the module’s functionalities |
| DrvSpi_VttCanoe01Asr.jar | This jar-file contains the generator and the validator for the DaVinci Configurator |
| VTTSpi_bswmd.arxml | Basic Software Module Description according to AUTOSAR for VTT Emulation |
| Spi_bswmd.arxml | Optional Basic Software Module Description. Placeholder for real target (semiconductor manufacturer) in VTT only use case |

Table 4-1 Static files

### 4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator.

| File Name | Description |
|---|---|
| Spi_Cfg.h | The configuration-header contains the static configuration part of this module |
| Spi_PBcfg.c | The configuration-source contains the object independent part of the runtime configuration |

Table 4-2 Generated files

## 4.2 Include Structure

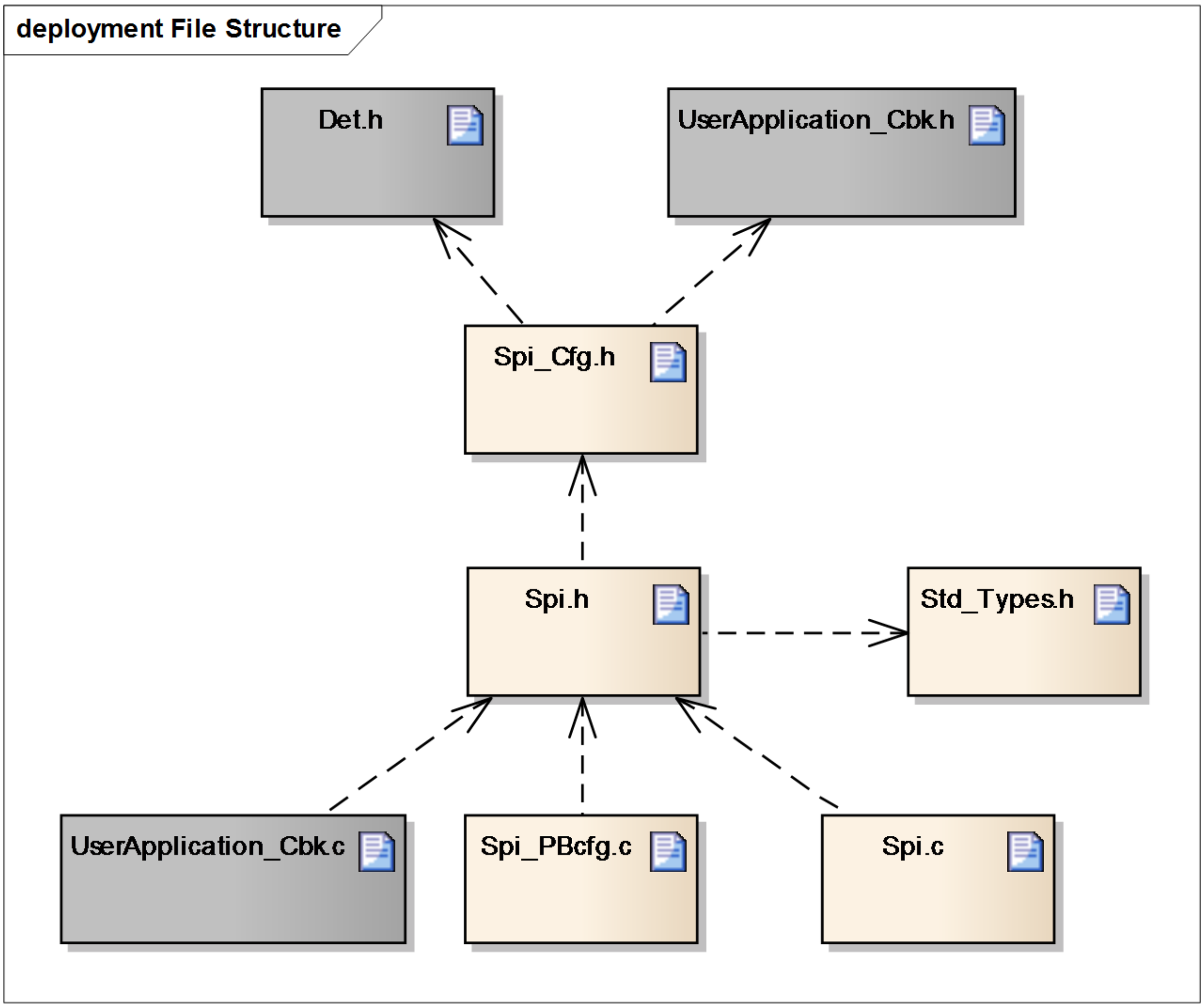


Figure 4-1 Include Structure

## 4.3 Dependencies on SW Modules

### 4.3.1 AUTOSAR OS (Optional)

An operating system can be used for task scheduling, interrupt handling, global suspend and restore of interrupts and creating of the Interrupt Vector Table.

### 4.3.2 DET (Optional)

The SPI module depends on the DET (by default) in order to report development errors. Detection and reporting of development errors can be enabled or disabled by the switch "Enable Development Error Detection".

### 4.3.3 SchM (Optional)

Beside the AUTOSAR OS the Schedule Manager provides functions that module SPI calls at begin and end of critical sections. Besides, the Schedule Manager is responsible for calling the main functions.

### 4.3.4 EcuM (Optional)

The EcuM cares for the initialization of the module SPI.

# 5 API Description

For an interfaces overview please see Figure 2-2.

## 5.1 Type Definitions

The types defined by the SPI are described in this chapter.

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| Spi_StatusType | enum | States of the SPI driver | `SPI_UNINIT = 0`, driver is not initialized |
| | | | `SPI_IDLE = 1`, driver is IDLE |
| | | | `SPI_BUSY = 2`, driver is BUSY |
| Spi_DataModeType | enum | Data amount processing mode | `SPI_DATA_8BIT` = 0 |
| | | | `SPI_DATA_16BIT` = 1 |
| Spi_JobResultType | enum | The result / status of the job | `SPI_JOB_OK = 0`, job is IDLE or finished successfully |
| | | | `SPI_JOB_PENDING = 1`,job is running now |
| | | | `SPI_JOB_FAILED = 2`, job failed |
| Spi_SeqResultType | enum | The result / status of the sequence | `SPI_SEQ_OK = 0`, sequence is IDLE or finished successfully |
| | | | `SPI_SEQ_PENDING = 1`, sequence is running now |
| | | | `SPI_SEQ_FAILED = 2`, sequence failed |
| | | | `SPI_SEQ_CANCELLED = 3`, sequence has been aborted |
| Spi_AsyncModeType | enum | Data processing mode | `SPI_POLLING_MODE` |
| | | | `SPI_INTERRUPT_MODE` |
| Spi_NumberOfDataType | uint16 | Type for any length parameters | 0-65535 |
| Spi_JobType | uint8 | Type for job ID | 0-255 |
| Spi_DataType | uint8 | Type for data transmission, reception | 0-255 |
| Spi_ChannelType | uint8 | Type for channel ID | 0-255 |
| Spi_SequenceType | uint8 | Type for sequence ID | 0-255 |
| Spi_HWUnitType | uint8 | Type for Hw Unit ID | 0-255 |
| Spi_SrcPtrType | Spi_Data | Type for source | Address of a Spi_Datatype |

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| | Type* | pointer address | |
| Spi_DstPtrType | Spi_Data Type* | Type for destination pointer address | Address of a Spi_Datatype |
| Spi_DlcType | uint8 | Type for a HW Unit | Not used, currently always 0 |

## 5.2 Services provided by SPI

### 5.2.1 Spi_InitMemory

| Prototype | |
|---|---|
| `void Spi_InitMemory (void)` | |
| **Parameter** | |
| - | - |
| **Return code** | |
| - | - |
| **Functional Description** | |
| This service initializes the global variables in case the startup code does not work | |
| **Particularities and Limitations** | |
| > This function is synchronous.<br>> This function is non reentrant.<br>> Module must not be initialized. | |
| Expected Caller Context | |
| > Called during startup | |

Table 5-1 Spi_InitMemory

### 5.2.2 Spi_Init

| Prototype | |
|---|---|
| `void Spi_Init (P2CONST(Spi_ConfigType, AUTOMATIC, SPI_APPL_CONST) ConfigPtr)` | |
| **Parameter** | |
| ConfigPtr | Pointer to SPI driver configuration set |
| **Return code** | |
| - | - |
| **Functional Description** | |
| The initialization has to be called to operate the SPI driver. Other API services cannot be executed if the Spi_Init was not executed before. | |

| **Particularities and Limitations** |
|---|
| > Service ID: see table 'Service IDs'<br>> This function is synchronous.<br>> This function is non reentrant.<br>> Module must not be initialized. |
| Expected Caller Context |
| > ECU State Manager or comparable software module, responsible for driver initialization after startup. |

Table 5-2 Spi_Init

### 5.2.3 Spi_DeInit

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`Spi_DeInit`** `(void)` | |
| **Parameter** | |
| - | - |
| **Return code** | |
| Std_ReturnType | `E_OK`, success<br>`E_NOT_OK`, request rejected |
| **Functional Description** | |
| The de-initialization can be called to shut down the SPI driver. | |
| **Particularities and Limitations** | |
| > Service ID: see table 'Service IDs'<br>> This function is synchronous.<br>> This function is non reentrant.<br>> This function should not be called during a running operation.<br>> Module should be initialized. | |
| Call context | |
| > ECU State Manager or comparable software module, responsible for driver initialization after startup. | |

Table 5-3 Spi_DeInit

### 5.2.4 Spi_WriteIB

| **Prototype** | |
|---|---|
| `Std_ReturnType` **`Spi_WriteIB`**<br>`(`<br>`  Spi_ChannelType Channel,`<br>`  const Spi_DataType* DataBufferPtr`<br>`)` | |
| **Parameter** | |
| `Channel` | ID of the channel which stores the data for transmission. |

| | |
|---|---|
| DataBufferPtr | Pointer to the a constant buffer which holds the data. |
| **Return code** | |
| Std_ReturnType | E_OK, buffer write executed.<br>E_NOT_OK, request rejected. |
| **Functional Description** | |
| This function is used to set up an internal buffer before transmitting data. The passed data buffer is copied into the internal buffer (of the respective Channel) for later transmission. | |
| **Particularities and Limitations** | |
| > Service ID: see table 'Service IDs'<br>> This function is synchronous.<br>> This function is reentrant for different channel numbers.<br>> Module should be initialized. | |
| Expected Caller Context | |
| > Task Context | |

Table 5-4 Spi_WriteIB

**Caution**
Use this service to setup *internal* buffers before calling one of the transmit services.

### 5.2.5 Spi_AsyncTransmit

| | |
|---|---|
| **Prototype** | |
| Std_ReturnType **Spi_AsyncTransmit** (Spi_SequenceType Sequence) | |
| **Parameter** | |
| Sequence | The transmission and reception for this Sequence ID is engaged. |
| **Return code** | |
| Std_ReturnType | E_OK, request accepted, transmission will be processed<br>E_NOT_OK, request rejected |
| **Functional Description** | |
| The passed Sequence will trigger an asynchronous transmission on the SPI bus. If Spi_MainFunctionHandling is called the next time, the sequence is transmitted. | |
| **Particularities and Limitations** | |
| > Service ID: see table 'Service IDs'<br>> This function is asynchronous.<br>> This function is reentrant for different channel numbers.<br>> This function is configurable.<br>> Module should be initialized. | |
| Expected Caller Context | |

| |
|---|
| - Task Context |

Table 5-5 Spi_AsyncTransmit

### 5.2.6 Spi_ReadIB

| Prototype | |
|---|---|
| `Std_ReturnType` **`Spi_ReadIB`**<br>`(`<br>`  Spi_ChannelType Channel,`<br>`  Spi_DataType* DataBufferPtr`<br>`)` | |
| **Parameter** | |
| `Channel` | ID of the channel which stores the data for transmission. |
| `DataBufferPtr` | Pointer to the buffer which holds the data. |
| **Return code** | |
| `Std_ReturnType` | `E_OK`, data has been retrieved<br>`E_NOT_OK`, request rejected |
| **Functional Description** | |
| Service for reading an internal buffer of a certain `Channel` synchronously. The content of the internal buffer is copied to the location where `DataBufferPtr` points to. | |
| **Particularities and Limitations** | |
| - Service ID: see table 'Service IDs'<br>- This function is synchronous.<br>- This function is reentrant for different channel numbers.<br>- Module should be initialized. | |
| Expected Caller Context | |
| - Task Context | |

Table 5-6 Spi_ReadIB

### 5.2.7 Spi_SetupEB

| Prototype | |
|---|---|
| Std_ReturnType **Spi_SetupEB**<br>(<br>  Spi_ChannelType Channel,<br>  const Spi_DataType* SrcDataBufferPtr,<br>  Spi_DataType* DesDataBufferPtr,<br>  Spi_NumberOfDataType Length<br>) | |
| **Parameter** | |
| Channel | ID of the channel whose buffers should  be set up. |
| SrcDataBufferPtr | Pointer to source data buffer. |
| DesDataBufferPtr | Pointer to destination data buffer in RAM. |
| Length | Length (in bytes) of the data to be transmitted from SrcDataBufferPtr and/or received from DesDataBufferPtr<br>Min.: 1<br>Max.: Max of data specified at configuration for this channel |
| **Return code** | |
| Std_ReturnType | E_OK, buffers have been set up<br>E_NOT_OK, buffers have not been set up |
| **Functional Description** | |
| The function sets an external source buffer and an external destination buffer for the respective Channel. It also specifies the Length of the transmission and/or reception. | |
| **Particularities and Limitations** | |
| > Service ID: see table 'Service IDs'<br>> This function is synchronous.<br>> This function is reentrant for different channel numbers.<br>> Module should be initialized. | |
| Expected Caller Context | |
| > Task Context | |

Table 5-7 Spi_SetupEB

**Caution**
Use this service to setup external buffers before calling one of the transmit services.

### 5.2.8 Spi_GetStatus

| | |
|---|---|
| **Prototype** | |
| Spi_StatusType **Spi_GetStatus** (void) | |
| **Parameter** | |
| - | - |
| **Return code** | |
| Spi_StatusType | SPI_UNINIT, driver is not initialized.<br>SPI_IDLE, driver waiting for sequences to process.<br>SPI_BUSY, driver is processing a sequence(s). |
| **Functional Description** | |
| Returns the current driver status. | |
| **Particularities and Limitations** | |
| > Service ID: see table 'Service IDs'<br>> This function is synchronous.<br>> This function is reentrant.<br>> Module should be initialized. | |
| Expected Caller Context | |
| > Task Context | |

Table 5-8 Spi_GetStatus

### 5.2.9 Spi_GetJobResult

| | |
|---|---|
| **Prototype** | |
| Spi_JobResultType **Spi_GetJobResult** (Spi_JobType Job) | |
| **Parameter** | |
| Job | ID of the job. |
| **Return code** | |
| Spi_JobResultType | SPI_JOB_OK, Job successfully finished or is idle.<br>SPI_JOB_PENDING, Job is processing a transfer.<br>SPI_JOB_FAILED, An error occurred during transmission. |
| **Functional Description** | |
| Returns the current job status. | |
| **Particularities and Limitations** | |
| > Service ID: see table 'Service IDs'<br>> This function is synchronous and is reentrant.<br>> Module should be initialized. | |
| Expected Caller Context | |
| > Task Context | |

Table 5-9 Spi_GetJobresult

### 5.2.10 Spi_GetSequenceResult

| **Prototype** | |
|---|---|
| `Spi_SeqResultType` **`Spi_GetSequenceResult`** `(Spi_SequenceType Seq)` | |
| **Parameter** | |
| `Seq` | ID of the sequence. |
| **Return code** | |
| `Spi_SeqResultType` | `SPI_SEQ_OK`, Sequence is idle or has finished.<br>`SPI_SEQ_PENDING`, Sequence is waiting for being serviced.<br>`SPI_SEQ_FAILED`, Sequence aborted due to an error.<br>`SPI_SEQ_CANCELLED`, Sequence cancelled by user. |
| **Functional Description** | |
| This service returns the last transmission result of the specified sequence. | |
| **Particularities and Limitations** | |
| > Service ID: see table 'Service IDs'<br>> This function is synchronous.<br>> This function is reentrant.<br>> Module should be initialized. | |
| Expected Caller Context | |
| > Task Context | |

Table 5-10 Spi_GetSequenceResult

### 5.2.11 Spi_GetVersionInfo

| **Prototype** | |
|---|---|
| `void Spi_GetVersionInfo`<br>`(`<br>`  P2VAR(Std_VersionInfoType, AUTOMATIC, SPI_APPL_DATA) versioninfo`<br>`)` | |
| **Parameter** | |
| `VersioninfoPtr` | Pointer to version information. |
| **Return code** | |
| - | - |
| **Functional Description** | |
| This function returns the version information of the module.<br>The version information includes:<br>> Module Id<br>> Vendor Id<br>> Software version numbers | |

| Particularities and Limitations |
|---|
| > Service ID: see table 'Service IDs'<br>> This function is synchronous.<br>> This function is reentrant.<br>> Module should be initialized. |
| Expected Caller Context |
| > Task Context |

Table 5-11 GetVersionInfo

### 5.2.12 Spi_SyncTransmit

| Prototype | |
|---|---|
| Std_ReturnType **Spi_SyncTransmit** (Spi_SequenceType Sequence) | |
| **Parameter** | |
| Sequence | The transmission and reception for this Sequence ID is engaged. |
| **Return code** | |
| Std_ReturnType | E_OK, request accepted, transmission will be processed.<br>E_NOT_OK, request rejected. |
| **Functional Description** | |
| The passed sequence will trigger a transmission on the SPI bus. | |
| **Particularities and Limitations** | |
| > Service ID: see table 'Service IDs'<br>> This function is synchronous.<br>> This function is reentrant for different sequence numbers.<br>> Module should be initialized.<br>> This function is configurable | |
| Expected Caller Context | |
| > Task Context | |

Table 5-12 Spi_SyncTransmit

### 5.2.13 Spi_GetHWUnitStatus

| Prototype | |
|---|---|
| Spi_StatusType **Spi_GetHWUnitStatus** (Spi_HWUnitType HWUnit) | |
| **Parameter** | |
| HWUnit | Hardware unit ID. |
| **Return code** | |
| Spi_StatusType | SPI_UNINIT, The SPI Handler/Driver is not initialized or not usable.<br>SPI_IDLE, The SPI Handler/Driver is not currently transmitting any Job.<br>SPI_BUSY, The SPI Handler/Driver is performing a SPI Job (transmit). |

| **Functional Description** |
|---|
| Returns the status of the Spi hardware. |
| **Particularities and Limitations** |
| > Service ID: see table 'Service IDs'<br>> This function is synchronous.<br>> This function is reentrant.<br>> Module should be initialized.<br>> This function is configurable |
| Expected Caller Context |
| > Task Context |

Table 5-13 Spi_GetHWUnitStatus

### 5.2.14 Spi_Cancel

| **Prototype** | |
|---|---|
| `void **Spi_Cancel** (Spi_SequenceType Sequence)` | |
| **Parameter** | |
| `Sequence` | The transmission and reception for this Sequence ID is canceled. |
| **Return code** | |
| - | - |
| **Functional Description** | |
| Cancels an ongoing sequence. If a job processing is ongoing, the job is finished and the sequence is aborted. The user will get a notification (if configured) after the job has finished. | |
| **Particularities and Limitations** | |
| > Service ID: see table 'Service IDs'<br>> This function is synchronous.<br>> This function is reentrant for different sequence numbers.<br>> Module should be initialized.<br>> This function is configurable | |
| Expected Caller Context | |
| > Task Context | |

Table 5-14 Spi_Cancel

### 5.2.15 Spi_MainFunctionHandling

| **Prototype** | |
|---|---|
| `void **Spi_MainFunction_Handling** (void)` | |
| **Parameter** | |
| - | - |

| Return code | |
|---|---|
| - | - |
| **Functional Description** | |
| The main function handles cyclic procedures if required by the driver. Cyclic functions are i.e. the 10ms task of the OSEK operating system. | |
| **Particularities and Limitations** | |
| > Service ID: see table 'Service IDs' | |
| Expected Caller Context | |
| > Expected to be called in scheduler context, during operational phase. | |

Table 5-15 Spi_ MainFunctionHandling

### 5.2.16 Spi_MainFunctionDriving

| Prototype | |
|---|---|
| `void` **`Spi_MainFunction_Driving`** `( void )` | |
| **Parameter** | |
| - | - |
| **Return code** | |
| - | - |
| **Functional Description** | |
| The main driving function handles cyclic procedures if required by the hardware driver (not needed in this emulation). | |
| **Particularities and Limitations** | |
| > Service ID: see table 'Service IDs' | |
| Expected Caller Context | |
| > Expected to be called in scheduler context, during operational phase. | |

Table 5-16 Spi_MainFunctionDriving

**Note**
This function is implemented, but as empty body.

## 5.3 Services used by SPI

In the following table services provided by other components, which are used by the SPI are listed. For details about prototype and functionality refer to the documentation of the providing component.

| Component | API |
|---|---|
| DET | Det_ReportError |

Table 5-17 Services used by the SPI

## 5.4 Configurable Interfaces

### 5.4.1 Notifications

At its configurable interfaces the SPI defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the SPI but can be performed at configuration time. The function prototypes that can be used for the configuration have to match the appropriate function prototype signatures, which are described in the following sub-chapters.

#### 5.4.1.1 Spi_JobEndNotification

| Prototype | |
|---|---|
| void **<Spi_JobEndNotification>** (void) | |
| **Parameter** | |
| - | - |
| **Return code** | |
| - | - |
| **Functional Description** | |
| Job end notification function. | |
| **Particularities and Limitations** | |
| > This function is synchronous.<br>> The notification functions can be configured in the configuration tool | |
| Call context | |
| > Interrupt Context | |

Table 5-18 Spi_JobEndNotification

#### 5.4.1.2 Spi_SequenceEndNotification

| Prototype | |
|---|---|
| void **<Spi_SequenceEndNotification>** (void) | |
| **Parameter** | |
| - | - |
| **Return code** | |
| - | - |

| **Functional Description** |
|---|
| Sequence end notification function. |
| **Particularities and Limitations** |
| > This function is synchronous.<br>> The notification functions can be configured in the configuration tool |
| Call context |
| > Interrupt Context |

Table 5-19 Spi_SequenceEndNotification

# 6 Configuration

## 6.1 Configuration Variants

The SPI supports the configuration variants

- `VARIANT-PRE-COMPILE`

The configuration classes of the SPI parameters depend on the supported configuration variants. For their definitions please see the VTTSpi_bswmd.arxml file.

## 6.2 Configuration with DaVinci Configurator 5

The SPI module is configured with the help of the configuration tool DaVinci Configurator 5 (CFG5). The definition of each parameter is given in the corresponding BSWMD file.

# 7 Glossary and Abbreviations

## 7.1 Glossary

| Term | Description |
|---|---|
| CANoe | Tool for simulation and testing of networks and electronic control units. |
| DaVinci Configurator | Configuration and generation tool for MICROSAR components |

Table 7-1 Glossary

## 7.2 Abbreviations

| Abbreviation | Description |
|---|---|
| API | Application Programming Interface |
| AUTOSAR | Automotive Open System Architecture |
| BSW | Basis Software |
| DEM | Diagnostic Event Manager |
| DET | Development Error Tracer |
| ECU | Electronic Control Unit |
| EcuM | ECU State Manager |
| IoHwAb | BSW Module I/O Hardware Abstraction (Connection to RTE) |
| ISR | Interrupt Service Routine |
| MICROSAR | Microcontroller Open System Architecture (the Vector AUTOSAR solution) |
| OS | Operating System |
| RTE | Runtime Environment |
| SchM | BSW Module Scheduler |
| SPI | Serial Peripheral Interface |
| VTT | vVIRTUALtarget |

Table 7-2 Abbreviations

# 8 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com