AUTOSAR SPI Handler Driver

User Guide

Version 1.3

Nov 30, 2016

Table of Contents

1. Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module SPI.

Supported AUTOSAR Release	4.03	
Supported Configuration Variants	Post-build	
Vendor ID	SPI_VENDOR_ID	44
Module ID	SPI_MODULE_ID	83
Release version	02.22.00	

This document describes the functionalities, the configuration and the API of the SPI driver for the microcontroller Family TMS570LSx from Texas Instruments.

The Spi driver provides services for basic communication with external components. These components can be used by an application.
The main tasks of the Spi 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.)

1.1. Architecture Overview

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

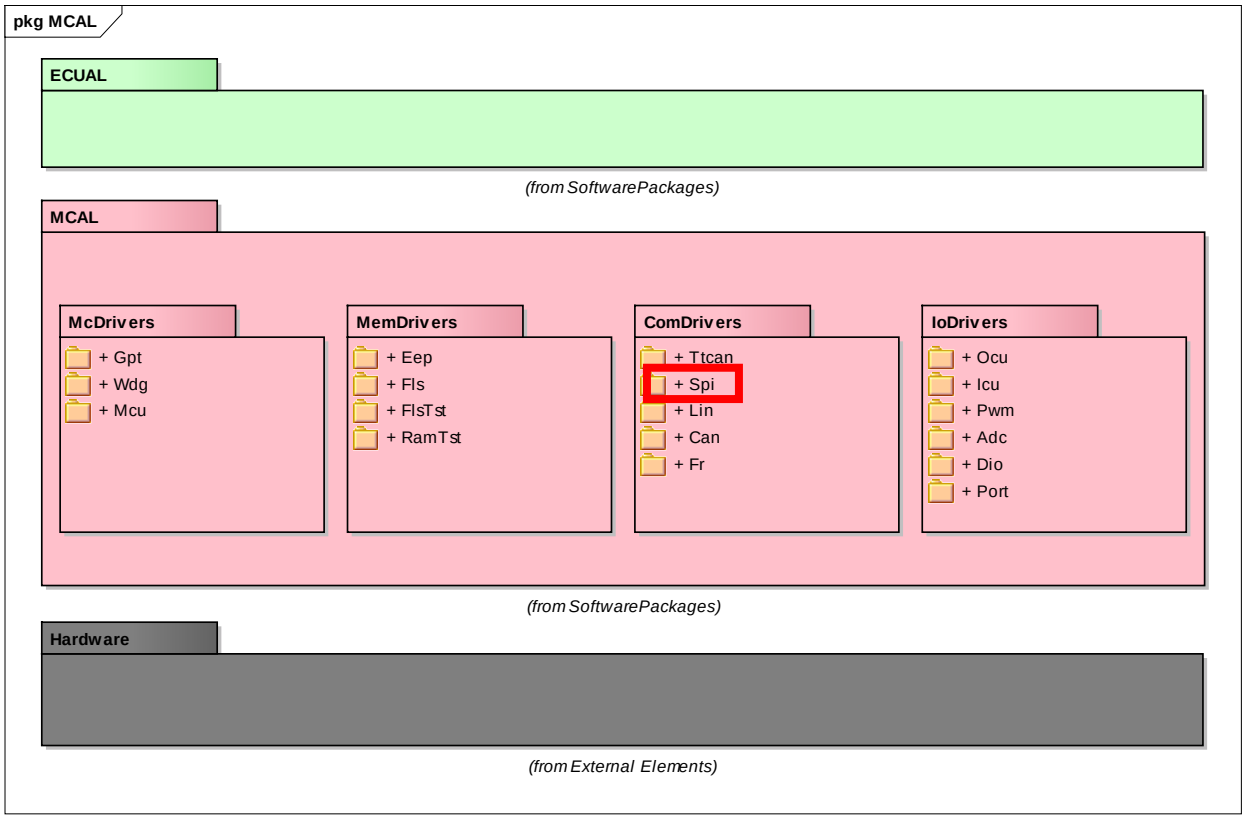

Figure 1-1 AUTOSAR Architecture

Below figure shows the interfaces to adjacent modules of the Spi.

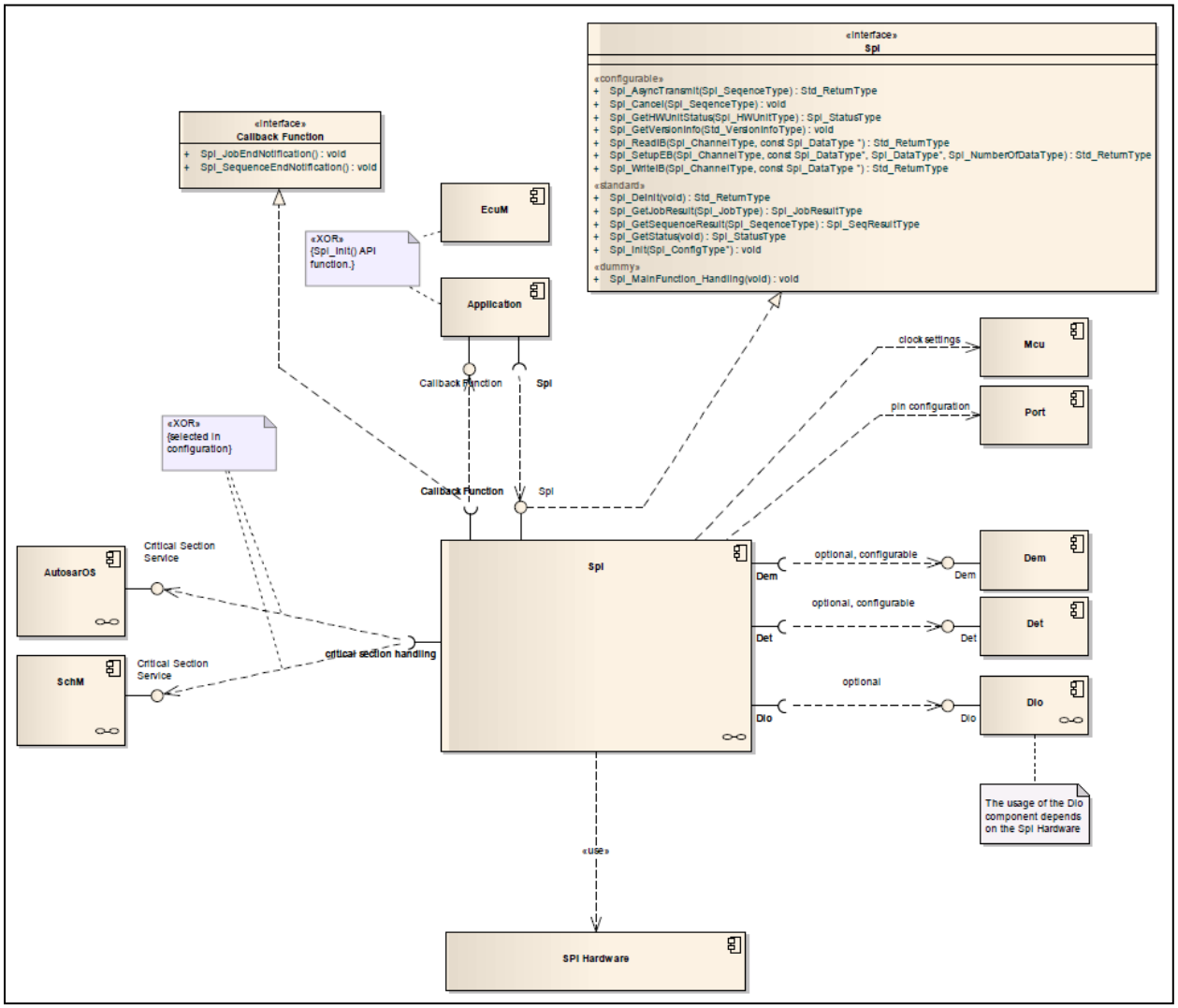

Figure 1-2 Interfaces to adjacent modules of the Spi

2. Functional Description

2.1. Features

The "supported" and "not supported" features are presented in the following two tables. For further information of not supported features also see chapter 5.

Supported Features
Configure Spi with ■ External devices ■ Channels ■ Jobs ■ Sequences
Configure physical units and callback functions
Configure error detection (DEM and DET)
Configure implementation features like: ■ Spi scalability level(s). ■ Spi channel buffers ■ Spi Interrupts ■ Spi frame transfers with 8 or 16bit clock frames
Select simple or extended API
Interruptible Sequences

Table 2-1 Supported SWS features

Not Supported Features
32bit clock frames are not supported in DMA mode
DMA mode supported only on SPI2 and SPI4 instances
Synchronous mode
Internal buffers not supported

Table 2-2 Not Supported SWS features

2.2. Initialization

The driver Spi is initialized by calling `Spi_Init()` with a pointer to a configuration as parameter. To re-initialize call `Spi_DeInit()` and then `Spi_Init()` again with a different configuration. The service for de-initialization is provided with the function `Spi_DeInit()`.

2.3. States

The Spi maintains states for:

- The Spi driver itself,
- the configured jobs,
- the configured sequences

- and Hardware units.

These states can be obtained by the:

- `Spi_GetStatus`,
- `Spi_GetJobResult`,
- `Spi_GetSequenceResult`,
- `Spi_GetHWUnitStatus` functions, so the user knows the ongoing action of the driver during a transmission.

2.4. Error Handling

2.4.1. Development Error Reporting

By default, development errors are reported to the DET using the service `DET_ReportError()` 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 service IDs identify the services which are described in 3.8. The following table presents the service IDs and the related services:

Service ID	Service
SPI_SID_INIT = 0x00	Spi_Init
SPI_SID_DEINIT = 0x01	Spi_DeInit
SPI_SID_WRITE_IB = 0x02	Spi_WriteIB
SPI_SID_ASYNC_TRANSMIT = 0x03	Spi_AsyncTransmit
SPI_SID_READ_IB = 0x04	Spi_ReadIB
SPI_SID_SETUP_EB = 0x05	Spi_SetupEB
SPI_SID_GET_STATUS = 0x06	Spi_GetStatus
SPI_SID_GET_JOB_RESULT = 0x07	Spi_GetJobResult
SPI_SID_GET_SEQ_RESULT = 0x08	Spi_GetSequenceResult
SPI_SID_GETVERSION_INFO = 0x09	Spi_GetVersionInfo
SPI_SID_SYNC_TRANSMIT = 0x0A	Spi_SyncTransmit
SPI_SID_GET_UNIT_STATUS = 0x0B	Spi_GetHWUnitStatus
SPI_SID_CANCEL = 0x0C	Spi_Cancel
SPI_SID_SET_ASYNC_MODE = 0x0D	Spi_SetAsyncMode
SPI_SID_MAINFUNCTION_HANDLING = 0x10	Spi_MainFunction_Handling

Table 2-3 Mapping of service IDs to services

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

Error Code		Description
[0x0A]	SPI_E_PARAM_CHANNEL	channel out of bounds, exceeds the maximum number of

		configured channels
[0x0B]	SPI_E_PARAM_JOB	Job out of bounds, exceeds the maximum nuber of configured jobs
[0x0C]	SPI_E_PARAM_SEQ	Sequence out of bounds, exceeds the maximum number of configured sequences
[0x0D]	SPI_E_PARAM_LENGTH	Length out of bounds, exceeds the maximum number of configured EB- or IB- buffer length
[0x0E]	SPI_E_PARAM_UNIT	The requested hardware unit does not exist
[0x10]	SPI_E_PARAM_POINTER	An invalid configuration has been passed (i.e. a NULL_PTR). This is an extension to AUTOSAR.
[0x1A]	SPI_E_UNINIT	A service was requested, but the driver has not been initialized
[0x2A]	SPI_E_SEQ_PENDING	The requested sequence is still pending
[0x3A]	SPI_E_SEQ_IN_PROCESS	Transmission of synchronous sequence in progress (not supported)
[0x4A]	SPI_E_ALREADY_INITIALIZED	The driver is already initialized.

Table 2-4 Error Reported to DET

2.4.1.1. Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 3-5 Development Error Reporting: Assignment of checks to services are internal parameter checks of the API functions. These checks are for development error reporting and can be en-/disabled separately. The configuration of en-/disabling the checks is described in chapter 6.1. En-/disabling of single checks is an addition to the AUTOSAR standard which requires to en-/disable the complete parameter checking via the parameter `SPI_DEV_ERROR_DETECT`. The following table shows which parameter checks are performed on which services:

Check / Services	SPI_E_PARAM_CHANNEL	SPI_E_PARAM_JOB	SPI_E_PARAM_SEQ	SPI_E_PARAM_LENGTH	SPI_E_PARAM_UNIT	SPI_E_PARAM_POINTER	SPI_E_UNINIT	SPI_E_SEQ_PENDING	SPI_E_SEQ_IN_PROCESS	SPI_E_ALREADY_INITIALIZED
Spi_Init						●				●
Spi_DeInit							●			
Spi_AsyncTransmit			●				●	●		
Spi_SetupEB	●			●		●	●			
Spi_GetStatus										
Spi_GetJobResult		●					●			
Spi_GetSequenceResult			●				●			

Spi_GetVersionInfo										
Spi_GetHWUnitStatus					•		•			
Spi_Cancel			•				•			

Table 2-5 Development Error Reporting: Assignment of checks to services

2.4.2. Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `DEM_ReportErrorStatus()`.

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

Error Code		Description
Assigned by DEM	SPI_E_BIT_ERROR	A bit error was detected.
Assigned by DEM	SPI_E_DESYNC_ERROR	Slave desynchronized.
Assigned by DEM	SPI_E_DLEN_ERROR	Data length mismatch detected during transmission.
Assigned by DEM	SPI_E_TXSTALL_ERROR	This error occurs if the driver tries to write data which has not been transmitted yet.
Assigned by DEM	SPI_E_RXSTALL_ERROR	The driver tries to read data, but no data has been provided by the hardware.
Assigned by DEM	SPI_E_DOVR_ERROR	The hardware reported an internal receive buffer overrun
Assigned by DEM	SPI_E_TO_ERROR	The hardware reported an Timeout error because of non-activation of ENA pin.

Table 2-6 Errors reported to DEM

2.5. Scope of Delivery

The delivery of the Spi contains the files which are described below:

2.5.1. Static Files

File Name	Description
Spi.c	Main Spi file, contains the API for the external visible services
Spi.h	Contains the type and function declarations for the Spi driver
Spi_Hw.c	This layer handles the execution of jobs and sequences
Spi_Hw.h	Contains the type and function declarations for the Spi_Hw module
Spi_Dlc.c	Driver layer, handles channels and all hardware related tasks
Spi_Dlc.h	Contains the type and function declarations for the Spi_Dlc module. Also contains the register definitions.
Spi_Irq.c	Source file that contain the ISR's
Spi_Irq.h	Header file that contains the declaration of the ISR's

plugin.xml	XML file used to register resources with EB Tresos Studio.
MANIFEST.MF	Manifest
Spi.xdm	Contains the formal notation of all information, which belongs to the Spi driver.
Spi_Rec.xdm	Contains values to a recommended default configuration of the Spi.

Table 2-7 Static files

2.5.2. Dynamic Files

The dynamic files are generated by the configuration tool

File Name	Description
Spi_Cfg.h	Pre-compile parameters are generated into this file.
Spi_PBcfg.c	Post-build configuration parameters are generated into this file.

Table 2-8 Dynamic files

2.6. Include Structure

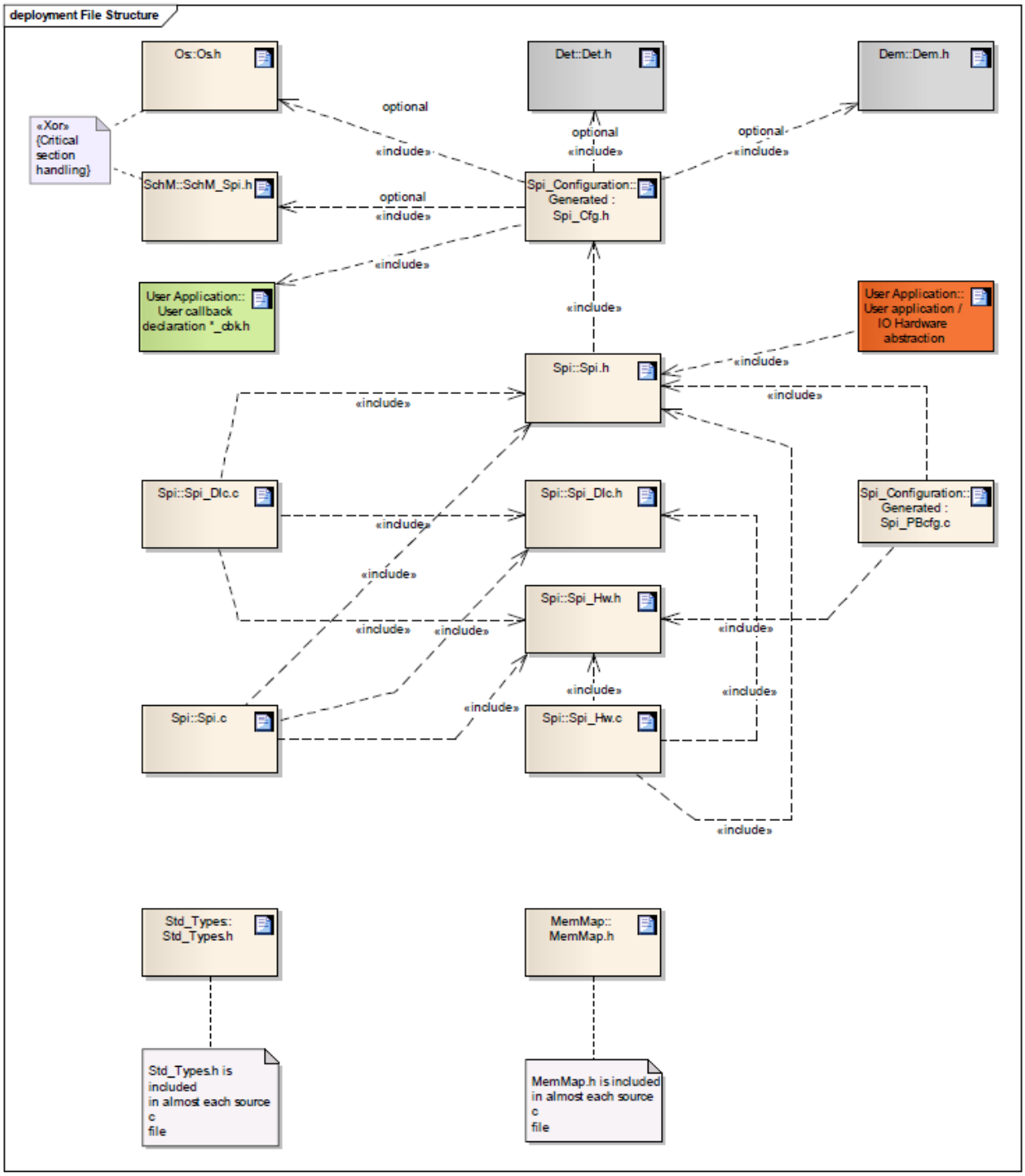

Figure 2-1 AUTOSAR Architecture

2.7. Compiler Abstraction and Memory Mapping

Although the Compiler Abstraction is implemented in the sources, there is no compiler specific keyword that could be assigned to the Compiler Abstraction definitions mentioned above. The TMS570 platform's

CPU only supports 32bit-addresses. Because of this, it is recommended to define the Compiler Abstraction definitions required by this module as empty macros in Compiler_Cfg.h.

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section. The following table contains the memory section names and the compiler abstraction definitions defined for the Spi and illustrate their assignment among each other.

Memory Mapping Sections / Compiler Abstraction Definitions	SPI_CODE	SPI_VAR	SPI_VAR_NOINIT	SPI_VAR_ZERO_INIT	SPI_CONST	SPI_PBCFG	SPI_APPL_DATA	SPI_APPL_CODE
SPI_START_SEC_CONST_32BIT (.const) SPI_STOP_SEC_CONST_32BIT					●			
SPI_START_SEC_PBCFG_ROOT (.const) SPI_STOP_SEC_PBCFG_ROOT						●		
SPI_START_SEC_PBCFG (.const) SPI_STOP_SEC_PBCFG						●		
SPI_START_SEC_CONST_UNSPECIFIED (.const) SPI_STOP_SEC_CONST_UNSPECIFIED					●			
SPI_START_SEC_CONST_32BIT (.const) SPI_STOP_SEC_CONST_32BIT	●							
SPI_START_SEC_CODE (.text) SPI_STOP_SEC_CODE	●							
SPI_START_SEC_CODE_ISR (.text) SPI_STOP_SEC_CODE_ISR			●					
SPI_START_SEC_VAR_NO_INIT_UNSPECIFIED (.bss) SPI_STOP_SEC_VAR_NO_INIT_UNSPECIFIED			●					
SPI_START_SEC_VAR_INIT_UNSPECIFIED (.data) SPI_STOP_SEC_VAR_INIT_UNSPECIFIED		●						
SPI_START_SEC_VAR_ZERO_INIT_UNSPECIFIED (.data) SPI_STOP_SEC_VAR_ZERO_INIT_UNSPECIFIED				●				
For application buffers passed to SPI							●	
For application code used by Spi(like callback functions)								●

Table 2-9 Compiler abstraction and memory mapping

2.8. Dependencies on SW modules

2.8.1. OSEK/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. The Spi module may use OSEK/AUTOSAR OS to suspend and restore global interrupts.

2.8.2. DET

The module Spi 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" on the tab "General Settings" within the module Spi.

2.8.3. DEM

The module Spi depends on the DEM in order to report diagnostic errors. Detection and reporting of diagnostic errors cannot be disabled.

2.8.4. MCU

The module MCU powers up the microcontroller's peripherals at startup time. Since the peripherals are also containing the registers for I/O functionality they have to be activated if it is intended to use them.

2.8.5. PORT

The module PORT enables the Spi lines at startup time. To operate the Spi properly, the PORT driver has to be configured. The PORT driver sets the PINs to the required values for the Spi to operate. The following figure shows an example configuration of the PORT.

2.8.6. SchM (Optional)

Beside the OSEK / AUTOSAR OS the Schedule Manager provides functions that module Spi calls at begin and end of critical sections.

2.9. Dependencies on HW modules

The module Spi driver is compatible to TMS570LS12x

3. API Description

3.1. Interfaces Overview

3.2. Imported Type Definitions

Type name	C-Type	Description	Value Range
Dem_EventIdType	uint16	Identification of an event by assigned EventId. The EventId is assigned by the Dem	N/A
Dem_EventStatusType	uint8	This type contains all monitor test result values, which can be reported via Dem_ReportErrorStatus() and Dem_SetEventStatus()	DEM_EVENT_STATUS_PASSED DEM_EVENT_STATUS_FAILED
Std_ReturnType	uint8	This type can be used as standard API return type which is shared between the RTE and the BSW modules	E_OK E_NOT_OK
Std_VersionInfoType	struct	This type shall be used to request the version of a BSW module using the <Module name>_GetVersionInfo() function.	N/A

Table 3-1 Imported Type definitions

3.3. Exported Type Definitions

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_JobResultType	Enum	The result / status of the job	SPI_JOB_OK = 0, job is IDLE or finished successfully (default value)
			SPI_JOB_QUEUED = 1, job is queued now
			SPI_JOB_PENDING = 2, job is about to be transmitted

			SPI_JOB_ACTIVE = 3, job is running (internal state)
			SPI_JOB_FAILED = 4, job failed
Spi_SeqResultType	Enum	The result / status of the sequence	SPI_SEQ_OK = 0, sequence is IDLE or finished successfully (default value)
			SPI_SEQ_PENDING = 1, sequence is running now
			SPI_SEQ_FAILED = 2, sequence failed
			SPI_SEQ_CANCELLED = 3, sequence has been aborted
Spi_HWUnitResultType	Enum	The result / status of the hardware	SPI_DLC_OK = 0, driver hardware is IDLE or finished successfully
			SPI_DLC_PENDING = 1, driver hardware is running now
			SPI_DLC_FAILED = 2, driver hardware failed
Spi_NumberOfDataType	uint16	Type for channel length parameter	Range (0-65535)
Spi_JobType	uint16	Type for job ID	Range (0-255), limited to 255.
Spi_DataType	uint8	Type for data transmission, reception	Range (0-255)
Spi_ChannelType	uint8	Type for channel ID	Range (0-255)
Spi_SequenceType	uint8	Type for sequence ID	Range (0-255)
Spi_HWUnitType	uint8	Type for hardware unit ID	Range (0-4)
Spi_ExternalDeviceType	uint8	Type for external device ID	Range (0-255)
Spi_ConfigType	Struct	This is the configuration type of the Spi. A reference of this type is generated in the Spi_PBcfg.c file.	Configuration of external devices
			Configuration of channels
			Configuration of job settings
			Configuration of sequence settings
			Configuration of hardware units
Spi_LoopbackModeType	enum	Type to select Loopback Mode	SPI_DIGITAL_LPBK
			SPI_ANALOG_LPBK
Spi_RegisterReadbackType	struct	Structure to report current status of configuration	GCR0, GCR1, DEF, EN

		registers	
Spi_ExternalDeviceConfigType	Struct	Type containing details about the external device assigned to a job	Dio Pin
			CS Property
			Baudrate
			Mode
			Clk Delays
			CS Idle Time
			CS Mode
			HW Instance
Spi_ChannelConfigType	Struct	Type containing details about Channel configuration	Channel ID
			Buffer Type
			Data Width
			Default TX Data
			Buffer Length
			TX Start (MSB or LSB)
Spi_JobConfigType	Struct	Type containing details about Job configuration	Job notification
			Channel Index List
			External device assigned
			Job ID
			Job Priority
			HW Unit
Spi_SequenceConfigType	Struct	Type containing details about Sequence configuration	Sequence Notification
			Job Index list
			Sequence ID
			Sequence Interruptible

Table 3-2 Type definitions

3.4. Exported Objects

Type name	Type	Description	Value Range
SpiRuntime	Spi_ConfigType	SPI configuration structure	N/A

Table 3-4 Exported Objects

3.5. Interrupt Service Routines provided by Spi

The Spi provides, depending on the available hardware units, up to six interrupt service routines. Three are designed for normal operation of each hardware unit (i.e. MIBSPI1, MIBSPI3 or MIBSPI5) and the others are used as error interrupts.

Note: Final integrator is required to enable and associate the interrupt channels in VIM to the corresponding ISRs.

3.5.1. Spi error interrupt service

Prototype	
void Spi_IrqUnit[X]_TxRx_ERR (void)	
Parameter	
None	
Return code	
None	
Functional Description	
Interrupt service for Tx and Rx error handling of hardware Unit [X] X = 0..4	
Particularities and Limitations	
The error interrupts are mapped to the high priority Interrupt level0 line of each module	

Table 3-6 Spi_IrqUnit[X]_TxRx_ERR

3.5.2. Spi data exchange interrupt service

Prototype	
void **Spi_IrqUnit[X]_TxRx** (void)	
Parameter	
None	
Return code	
None	
Functional Description	
Interrupt service for Tx and Rx handling of MIBSPI[X]	
Particularities and Limitations	
The interrupts are mapped to the low priority Interrupt level1 line of each module	

Table 3-7 Spi_IrqUnit[X]_TxRx

3.5.3. DMA block transfer end interrupt

Prototype	
void **DMA_IsrBTC** (void)	
Parameter	
None	
Return code	
None	
Functional Description	
Interrupt service for Tx and Rx handling of DMA transfers for hardware unit [X], X = 0..4	
Particularities and Limitations	
This service only has to be configured one time. It is valid for all Spi units which support DMA.	

Table 3-8 DMA_IsrBTC

3.6. API services provided by Spi

The Spi API consists of services, which are realized by function calls.

3.6.1. Spi_Init

Prototype	
void **Spi_Init** (const Spi_ConfigType* ConfigPtr)	
Parameter	
`ConfigPtr`	Pointer to Spi configuration set.
Return code	
void	--
Functional Description	
The initialization must be called to operate the Spi driver. This applies to a power on reset or a normal reset. Most API services will throw a DET error if the `Spi_Init` was not executed before. If development error detection is activated, the function performs these checks: ■ Checks if the driver has already been initialized. ■ Checks if the passed parameter is no `NULL` pointer. If a check is successful, the function executes the request. If one of the checks fails, an error is reported to the DET component.	
Particularities and Limitations	
This function is synchronous. This function is non reentrant. This function is always available. This function returns without action if a check fails This function should not be called during a running operation.	
Expected Caller Context	
Expected to be called in application context, during initialization phase.	

Table 3-9 Spi_Init

3.6.2. Spi_DeInit

Prototype	
Std_ReturnType **Spi_DeInit** (void)	
Parameter	
void	--
Return code	
Std_ReturnType	`E_OK`, success `E_NOT_OK`, request rejected.
Functional Description	
The de-initialization can be called to shut down the Spi driver. This applies to a system shutdown of the system. It can also be used to reinitialize the system in case of a critical error. Critical errors are reported to the DEM module. A critical error can be detected by evaluating a job result (status:	

SPI_JOB_FAILED). If development error detection is activated, the function performs these checks: Checks if the driver has been initialized. If a check is successful, the function executes the request. If one of the checks fails, an error is reported to the DET component. The Spi driver checks also if it is busy. The request is rejected if the driver is not idle.
Particularities and Limitations
This function is synchronous. This function is non reentrant. This function is always available. This function should not be called during a running operation.
Expected Caller Context
Expected to be called in application context, during shutdown phase.

Table 3-10 Spi_DeInit

3.6.3. 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. E_NOT_OK, request rejected.
Functional Description	
The passed sequence will perform transmission on the Spi bus. If development error detection is activated, the function performs these checks: ■ Checks if the driver has been initialized. ■ Checks if the sequence parameter is correct. ■ Checks if a sequence is pending. If a check is successful, the function executes the request. If one of the checks fails, an error is reported to the DET component.	
Particularities and Limitations	
■ This function is synchronous. ■ This function is non reentrant. ■ This function is configurable (Spi scalability level).	
Expected Caller Context	
Expected to be called in application context, during operational phase.	

Table 3-11 Spi_SyncTransmit

3.6.4. 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. E_NOT_OK, request rejected.
Functional Description	
The passed sequence will trigger a transmission on the Spi bus. If development error detection is activated, the function performs these checks: ■ Checks if the driver has been initialized. ■ Checks if the sequence parameter is correct. ■ Checks if a sequence is pending. If a check is successful, the function executes the request. If one of the checks fails, an error is reported to the DET component.	
Particularities and Limitations	
■ This function is asynchronous. ■ This function is non reentrant. ■ This function is configurable (Spi scalability level).	
Expected Caller Context	
Expected to be called in application context, during operational phase.	

Table 3-11 Spi_AsyncTransmit

3.6.5. Spi_SetupEB

Prototype	
Std_ReturnType **Spi_SetupEB** (Spi_ChannelType Channel, const Spi_DataType* SrcDataBufferPtr, Spi_DataType* DesDataBufferPtr, Spi_NumberOfDataType Length)	
Parameter	
Channel SrcDataBufferPtr DesDataBufferPtr Length	ID of the channel which stores the data for transmission. Pointer to the buffer which holds the data. Pointer to the buffer which gets the data. Length to transmit in bytes.
Return code	
Std_ReturnType	E_OK, data has been retrieved a copied to application buffer E_NOT_OK, request rejected.
Functional Description	
The function prepares a buffer, which is passed by the application for transmission. If development error detection is activated, the function performs these checks: ■ Checks if the channel is valid.	

■ Checks if the length does not exceed SpiEbMaxLength or yield 0. ■ Checks if the driver has been initialized. If a check is successful, the function executes the request. If one of the checks fails, an error is reported to the DET component.
Particularities and Limitations
■ This function is synchronous. ■ This function is non reentrant. ■ This function is configurable (Spi scalability level).
Expected Caller Context
Expected to be called in application context, during operational phase.

Table 3-2 Spi_SetupEB

3.6.6. Spi_GetStatus

Prototype	
Spi_StatusType **Spi_GetStatus** (void)	
Parameter	
void	--
Return code	
Spi_StatusType	Status of the Spi driver. SPI_UNINIT Driver is uninitialized SPI_IDLE Driver waiting for sequences to process. SPI_BUSY Driver is processing a sequence(s). .
Functional Description	
Returns the current driver status.	
Particularities and Limitations	
■ This function is synchronous. ■ This function is non reentrant. ■ This function is configurable (Spi scalability level).	
Expected Caller Context	
Expected to be called in application context, during operational phase.	

Table 3-33 Spi_GetStatus

3.6.7. Spi_GetJobResult

Prototype	
Spi_JobResultType **Spi_GetJobResult** (Spi_JobType Job)	
Parameter	
Job	ID of the job.
Return code	
Spi_JobResultType	Result of the job operation.

	SPI_JOB_OK Job successfully finished or is idle. SPI_JOB_PENDING Job is processing a transfer. SPI_JOB_FAILED An error occurred during transmission.
Functional Description	
Returns the current job status. If development error detection is activated, the function performs these checks: ■ Check if the job ID is valid ■ Check if the driver is initialized If a check is successful, the function executes the request. If one of the checks fails, an error is reported to the DET component.	
Particularities and Limitations	
■ This function is synchronous. ■ This function is non reentrant. ■ This function is always available.	
Expected Caller Context	
Expected to be called in application context, during operational phase.	

Table 3-44 Spi_GetJobResult

3.6.8. Spi_GetSequenceResult

Prototype	
Spi_SeqResultType **Spi_GetSequenceResult** (Spi_SequenceType Seq)	
Parameter	
Seq	ID of the sequence.
Return code	
Spi_SeqResultType	Result of the sequence operation. SPI_SEQ_OK sequence is idle or has finished. SPI_SEQ_PENDING sequence is waiting for being serviced. SPI_SEQ_FAILED sequence aborted due to an error. SPI_SEQ_CANCELLED sequence cancelled by user.
Functional Description	
Returns the current sequence status. If development error detection is activated, the function performs these checks: ■ Check if the sequence ID is valid. ■ Check if the driver is initialized. If a check is successful, the function executes the request. If one of the checks fails, an error is reported to the DET component.	
Particularities and Limitations	
■ This function is synchronous. ■ This function is non reentrant. ■ This function is always available.	

Expected Caller Context
Expected to be called in application context, during operational phase.

Table 3-55 Spi_GetSequenceResult

3.6.9. Spi_GetVersionInfo

Prototype	
void **Spi_GetVersionInfo** (Std_VersionInfoType * VersioninfoPtr)	
Parameter	
VersioninfoPtr	Pointer to version information.
Return code	
void	--
Functional Description	
Returns the version information of the driver. If development error detection is activated, the function performs these checks: ■ Check if the pointer is valid – no NULL_PTR. If a check is successful, the function executes the request. If one of the checks fails, an error is reported to the DET component.	
Particularities and Limitations	
■ This function is synchronous. ■ This function is reentrant. ■ This function is configurable.	
Expected Caller Context	
Expected to be called in application context, during operational phase.	

Table 3-66 Spi_GetVersionInfo

3.6.10. Spi_GetHWUnitStatus

Prototype	
Spi_StatusType **Spi_GetHWUnitStatus** (Spi_HWUnitType HWUnit)	
Parameter	
HWUnit	Hardware unit ID.
Return code	
Spi_StatusType	Status of the hardware driver. SPI_UNINIT Driver is not initialized SPI_IDLE Driver is idle SPI_BUSY Driver is processing data.
Functional Description	
Returns the status of the Spi hardware. If development error detection is activated, the function performs these checks:	

■ Checks if the driver has been initialized. ■ Checks if the unit parameter is correct If a check is successful, the function executes the request. If one of the checks fails, an error is reported to the DET component.
Particularities and Limitations
■ This function is synchronous. ■ This function is configurable.
Expected Caller Context
Expected to be called in application context, during operational phase.

Table 3-77 Spi_GetHWUnitStatus

3.6.11. Spi_Cancel

Prototype	
void **Spi_Cancel** (Spi_SequenceType Sequence)	
Parameter	
Sequence	The transmission and reception for this Sequence ID shall be canceled.
Return code	
void	--
Functional Description	
Cancels an actual 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. If development error detection is activated, the function performs these checks: ■ Checks if the driver has been initialized or is busy. ■ Checks if the sequence parameter is correct If a check is successful, the function executes the request. If one of the checks fails, an error is reported to the DET component.	
Particularities and Limitations	
■ This function is synchronous. ■ This function is non reentrant. ■ This function is configurable.	
Expected Caller Context	
Expected to be called in application context, during operational phase.	

Table 3-88 Spi_Cancel

3.6.12. Spi_EnableLoopbackMode

Prototype	
Std_ReturnType Spi_EnableLoopbackMode(Spi_HWUnitType HWUnit,Spi_LoopbackModeType LBMode)	
Parameter	
Spi_HWUnitType	HW Unit to check (Range 0-4)

Spi_LoopbackModeType	Selected Loopback Mode (Analog, Digital)
Return code	
Std_ReturnType	E_OK, request accepted.. E_NOT_OK, request rejected.
Functional Description	
Function to enable Loopback mode for SPI module test purposes	
Particularities and Limitations	
■ This function is synchronous. ■ This function is non reentrant. ■ This function is configurable.	
Expected Caller Context	
Expected to be called in application context for diagnostic use only.	

Table 3-19 Spi_EnableLoopbackMode

3.6.13. Spi_DisableLoopbackMode

Prototype	
Std_ReturnType Spi_DisableLoopbackMode(Spi_HWUnitType)	
Parameter	
Spi_HWUnitType	HW Unit to check (Range 0-4)
Return code	
Std_ReturnType	E_OK, request accepted. E_NOT_OK, request rejected.
Functional Description	
Function to disable Loopback mode for SPI module test purposes	
Particularities and Limitations	
■ This function is synchronous. ■ This function is non reentrant. ■ This function is configurable.	
Expected Caller Context	
Expected to be called in application context for diagnostic use only.	

Table 3-20 Spi_DisableLoopbackMode

3.6.14. Spi_RegisterReadback

Prototype	
Std_ReturnType Spi_RegisterReadback(Spi_RegisterReadbackType RegisterReadbackPtr, Spi_HWUnitType HWUnit)	
Parameter	
Spi_RegisterReadbackType	Pointer to structure

Spi_HWUnitType	HW Unit to check (Range 0-4)
Return code	
Std_ReturnType	E_OK, request accepted.. E_NOT_OK, request rejected.
Functional Description	
Function to read back the current value stored in configuration registers for test purposes	
Particularities and Limitations	
■ This function is synchronous. ■ This function is non reentrant. ■ This function is configurable.	
Expected Caller Context	
Expected to be called in application context to detect any unexpected change in Configuration.	

Table 3-21 Spi_RegisterReadback

3.7. 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
DEM	DEM_ReportErrorStatus

Table 3-22 Services used by Spi

3.8. Callback Functions

Spi does not have any callbacks which can be invoked by other modules.

3.9. Configurable Interfaces

3.9.1. 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.

3.9.1.1. Spi_JobEndNotification

Prototype	
void **(*Spi_JobEndNotification)** (void)	
Parameter	
Void	--
Return code	

void	--
Functional Description	
Job end notification function. Can be configured in the configuration tool.	
Particularities and Limitations	
■ No Limitations	
Expected Caller Context	
The job notification will be called from interrupt context.	

Table 3-23 Spi_JobEndNotification

3.9.1.2. Spi_SequenceEndNotification

Prototype	
void **(*Spi_SequenceEndNotification)** (void)	
Parameter	
Void	--
Return code	
void	--
Functional Description	
Sequence end notification function. Can be configured in the configuration tool.	
Particularities and Limitations	
■ No Limitations	
Expected Caller Context	
The sequence notification will be called from interrupt context	

Table 3-24 Spi_SequenceEndNotification

4. Configuration

The module Spi can easily be configured using the EB TRESOS configuration tool. The outputs of the configuration and generation process are the configuration source files. This chapter describes the graphical elements of the view of the Spi, to help you to pre-estimate the results of the configuration.

4.1. Configuration of the Spi driver

4.1.1. Spi configuration

Select one of the four tabs to configure:

- External device
- Channels
- Jobs
- Sequences
- Hardware Units

4.1.1.1. Spi external device configuration

With the buttons [Insert] / [delete], a new device can be added/removed from the configuration. A default name is automatically provided, but can be changed by clicking the “DeviceName” field.

Attribute Name	Value Type	Values The default value is written in bold	Description
SpiBaudrate	Float	**500**	Set the number of bits to be transmitted per second. Unit is in KBits/s.
SpiCsIdentifier	Enum	**SCS0,** SCS1 SCS2, SCS3	This parameter is the symbolic name to identify the Chip Select (CS) allocated to this Job. **Some SPI HW Units don’t support all different Chip Selects.**
SpiCsPolarity	Enum	**High** Low	This parameter defines the active polarity of Chip Select
SpiDataShiftEdge	Enum	**LEADING** TRAILING	This parameter defines the SPI data shift edge.
SpiEnableCs	Enum	**ON** OFF	Enables or disables the chip select function.
SpiHwUnit	Enum	**CSIB0** CSIB1 CSIB2 CSIB3 CSIB4	This parameter is the symbolic name to identify the HW SPI Hardware. SPI1 – CSIB0 to SPI5 – CSIB4. MIBSPI1 → CSIB0 SPI2 → CSIB1

			MIBSPI3 → CSIB2 SPI4 → CSIB3 MIBSPI5 → CSIB4
SpiShiftClockIdleLevel	Enum	**High** Low	This parameter defines the SPI shift clock idle level.
SpiTimeClk2Cs	Float	**1E-5** 0-0.0001sec	Set the timing between last clock signal and CS back to idle. Unit is in nanoseconds. A value of zero disables this feature.
SpiTimeCs2Clk	Float	**1E-5** 0-0.0001sec	Chip-select-active-to-transmit-start-delay.
SpiCsMode	Enum	**CONTINUOUS,** SINGLE	At the end of a frame transfer the chip select stays active. If SINGLE is selected, the CS will go idle. A frame is the smallest data unit on the Spi bus.
SpiCsIdleTime	Float	**1E-4** 0-0.0001sec	Active if CS select mode is "SINGLE". This is the duration the CS signal will go idle. If 0 is selected the duration is at last 2xVBUS clock. The VBUS clock can be obtained from "Resulting clock for Spi in MHz"

Table 4-1 Spi external device configuration

4.1.1.2. Spi channels configuration

With the buttons [Insert] / [delete], a new channel can be added/removed from the configuration. A default name is automatically provided, but can be changed by clicking the "Channel Id" field.

Attribute Name	Value Type	Values The default value is written in bold	Description
SpiChannelId	Integer	**0** 0-255	This shows the unique channel ID. Consecutive numbers are required.
SpiChannelType	Enum	**EB** IB	Select internal (IB) or external (EB) channel buffer. Internal buffers provide a small amount of memory in the Spi driver while external buffers only use the application buffers. **IB not supported.** **Example** If a user configures a lot of internal buffers, the size of the driver inflates, while external buffers do not affect the driver size. **Caution** the use of IB and EB is overridden by the setting "Select allowed channel

			buffers" (General Settings) option in the view. This may lead to a validation error if this option has been changed.
SpiDataWidth	Enum	**8** 16 32	The size of one transmission unit. Selectable are 8, 16 and 32 bits.
SpiDefaultData	Integer	**255** 0-65535	If the function Spi_SetupEB gets a NULL_PTR as source buffer, the default data will be sent to the device. Used to read out devices.
SpiEbMaxLength	Integer	**255** 1-65535	Maximum allowed length of an EB buffer. If the passed size exceeds this parameter, a DET error is thrown (development checks have to be enabled).
SpiIbNBuffers	Integer	**10** 1-20	Maximum allowed length of an IB buffer. This parameter configures the size of the internal provided Spi memory.
SpiTransferStart	Enum	**MSB** LSB	Starts a Spi transfer with the most significant bit (MSB) first or the least significant bit first (LSB). **Example** A value of 0xF0 = 11110000 is sent as 11110000. If LSB is selected, the result is 00001111.

Table 4-2 Spi channels configuration

4.1.1.3.Spi job configuration

With the buttons [Insert] or [delete], a new job can be added/removed from the configuration. A default name is automatically provided, but can be changed by clicking the "Job Id" field.

Attribute Name	Value Type	Values The default value is written in bold	Description
SpiHwUnitSynchronous	Enum	**ASYNCHRONOUS** SYNCHRONOUS	Non editable. NOT USED
SpiJobEndNotification	String	**NULL_PTR** Valid C-function name	Select a notification callback if required by the external device driver. A job end will trigger this function.

SpiJobId	Integer	**0** 0-65535	This shows an unique job ID. This Job ID is used to determine the priority order of the jobs. Consecutive numbers are required.
SpiJobPriority	Integer	**3** 0-3	Set the priority of the Job. Priority ranges from [0-3; lowest - highest]. **Info** High priority jobs are transmitted first. They are not dependent on their assigned order in the sequence.
SpiDeviceAssignment	--	**SpiExternalDevice**	Assign a device to this job. A job can only have one device assignment.
SpiChannelList	--	**SpiChannel**	Assign one or more channels with different Id's to this job.

Table 4-3 Spi Jobs configuration

4.1.1.4. Spi sequences configuration

With the buttons [Insert]/[delete], a new Sequence can be added/removed from the configuration. A default name is automatically provided, but can be changed by clicking the "Sequence Id" field.

Attribute Name	Value Type	Values The default value is written in bold	Description
SpiInterruptibleSequence	Enum	**OFF** ON	Select this if a sequence can be interrupted by another sequence with high priority jobs (please refer also to "Global enable interruptible sequences".) **Caution** Take care that a sequence which has high priority jobs can interrupt a sequence, marked as interruptible with lower priority jobs. **Interruptible sequences not supported.**
SpiSeqEndNotification	String	**NULL_PTR** Valid C-function name	Select a notification callback if required by the external device driver. A sequence complete will trigger this function.
SpiSequenceId	Integer	**0** 0-255	This shows the unique sequence ID. Consecutive numbers are required.
SpiJobAssignment	--	**SpiJob**	Assign one or more jobs with different Id's to this sequence.

Table 4-4 Spi sequences configuration

4.1.1.5. Spi hardware unit configuration

With the buttons [Insert]/[delete], a new hardware unit can be added/removed from the configuration. A default name is automatically provided, but can be changed by clicking the "Hardware unit Id" field.

Attribute Name	Value Type	Values The default value is written in bold	Description
SpiUnitHasRam	Boolean	**ON** OFF	If a hardware unit is a MIBSPI unit (1, 3 and 5), this shall be enabled. **Do not modify.**
SpiUnitId	Integer	**0** 0-4 (This depends on the used derivative)	This shows the unique hardware unit ID. Eg: MIBSPI1 equals 0, SPI2 equals 1.. etc. **Do not modify.**
SpiUnitUsesDma	Boolean	**Off** On	This enables the DMA support for standard non MIBSPI (SPI2 and SPI4). User has to ensure Dma_IsrBTC() service is configured in the Os. **Do not modify.**
SpiUnitEnable	Boolean	**ON** OFF	This parameter enables or disables the specific SPI module. **Note:** Against **SpiHardwareUnit_3** select **OFF** for **PGE** Variant Devices since it does not support SPI4. Refer **Figure 4-1** below.

Table 4-5 Spi hardware units configuration

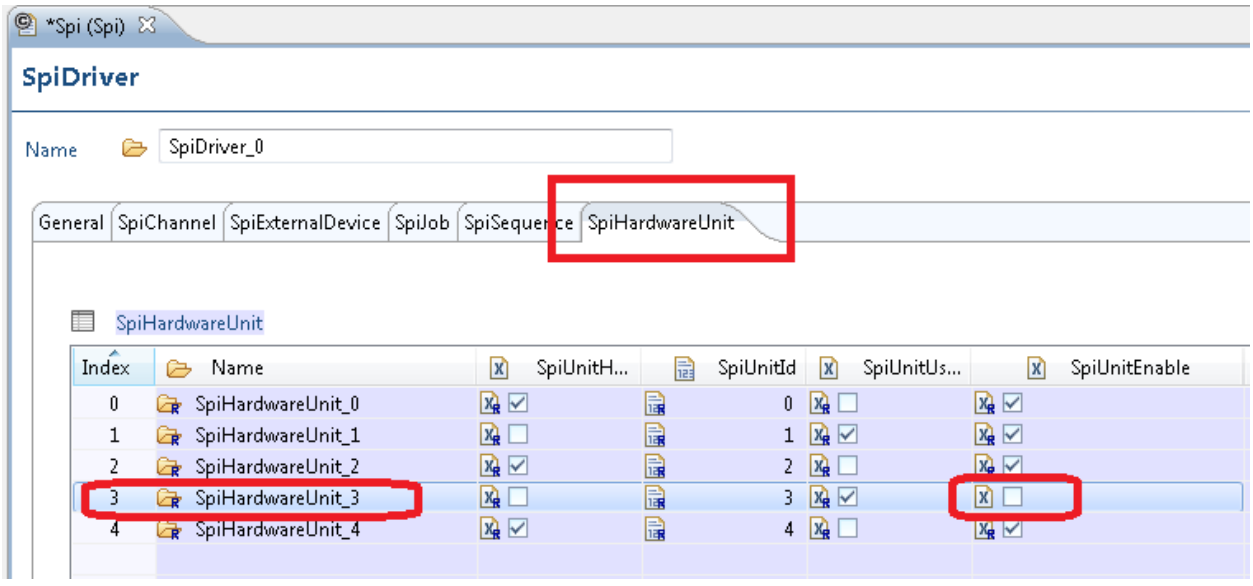

Figure 4-1 Interfaces to adjacent modules of the Spi

4.1.2. Spi general settings

Attribute Name	Value Type	Values The default value is written in bold	Description
SpiCancelApi	Boolean	**ON** OFF	Enable or disable the function `Spi_Cancel()`.
SpiChannelBuffersAllowed	Integer	**1** 0-2	Selects the SPI Handler/Driver Channel Buffers. **EB only**
SpiDevErrorDetect	Boolean	**ON** OFF	Global enable/disable error checking. **Caution** passing wrong parameters when error detection is off, might result in a system crash!
SpiHwStatusApi	Boolean	**ON** OFF	Enable or disable the function `Spi_GetHWUnitStatus()`.
SpiInterruptibleSeqAllowed	Boolean	**FALSE**	Switches the Interruptible Sequences handling functionality ON or OFF.
SpiLevelDelivered	Integer	**1** 0-2	Selects the SPI Handler/Driver level of scalable functionality.
SpiSupportConcurrentSyncTransmit	Boolean	**FALSE**	Specifies whether concurrent Spi_SyncTransmit() calls for different sequences shall be configurable. **Not supported.**
SpiVersionInfoApi	Boolean	**TRUE** FALSE	Enable or disable the function `Spi_GetVersionInfo()`.
SpiIrqType	Integer	**Category 2** Category 1 Void Func(void)	Is used for the interrupt level: `SPI_ISR_CAT2` uses the OSEK ISR() macro. `SPI_ISR_CAT1` uses the interrupt {SpiIrqfunction} definition `SPI_ISR_VOID` a simple void {SpiIrqfunction}(void) definition .
SpiNotificationHeader	String		Header file providing Job End and Sequence End notification functions definitions
SpiLpbkApi	Boolean	**FALSE** TRUE	Switches the Loopback Mode functions ON or OFF.
SpiRegRdbkApi	Boolean	**FALSE** TRUE	Enable or disable API to read current values of configuration registers

All configuration items are pre-compile

5. AUTOSAR Standard Compliance

5.1. Additions/ Extensions

5.1.1. Request queuing

Spi transmission requests can be queued up to the maximum number of configured sequences. If one sequence is cancelled, the driver continues to process the next queued sequence.

5.2. Limitations

- SPI Sync is implemented in polling mode only.
- SPI Async is implemented in Interrupt mode only.
- External Buffers only
- MIBSPI1, MIBSPI3 and MIBSPI5 hardware units use mutibuffered mode only
- SPI2 and SPI4 hardware units use DMA mode only
- Sequences not interruptible

5.2.1. Runtimes

The Spi driver provides only one runtime (runtime means only one post-build configuration).

5.2.2. Configuration

1) Sequence must contain Jobs that containing same Hardware unit.
2) Channel / Job / Sequence ID must be same as Index of that particular Channel / Job / Sequence ID.
3) Only SPI2 and SPI4 support DMA.
4) SPI2 and SPI4 cannot be used in Parallel.
5) SPI1, SPI3 and SPI 5 units use Multi-Buffered mode only

6. General Recommendations

- Use of MPU (Memory Protection Unit) is highly advised at the OS level
- Code generated using AUTOSAR Configuration Tool shall not be altered manually
- Use of Watchdog Timer and Reset cause monitoring is highly advised