MCAL User Manual for Spi

32-bit TriCore™ AURIX™ TC3xx microcontroller

About this document

Scope and purpose

This User Manual is intended to enable users to integrate the Microcontroller Abstraction Layer (MCAL) software for the TriCore™ AURIX™ family of 32-bit microcontrollers.

This document describes responsibilities of integrator in-charge of integrating MCAL software with the basic software (BSW) stack. This document also provides detailed information on safety, configuration and functions along with examples of usage of significant features.

Note: Detailed information about package installation, safety and other generic information that are common across all modules are provided in MCAL User Manual General.

Intended audience

This document is intended for anyone using the Spi module of the TC3xx MCAL software.

Document conventions

Table 1 Conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, table and figure captions, screen names, windows, dialog boxes, menus, sub-menus
Italics	Denotes variable(s) and reference(s)
`Courier New`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets
>	Indicates that a cascading sub-menu opens when you select a menu item
[cover parentID=<alpha numeric value>]	Used for traceability completeness. Reader should ignore these.

Reference documents

This User Manual should be read in conjunction with the following documents:

- AURIX™ TC3xx MCAL User Manual General
- Specification of SPI Driver, AUTOSAR_SWS_SPI_Driver, AUTOSAR Release 4.2.2
- Specification of SPI Driver, AUTOSAR_SWS_SPI_Driver, AUTOSAR Release 4.4.0

Table of contents

1 Spi driver

1.1 User information

1.1.1 Description

The SPI driver operates in the master and full duplex communication modes only. The driver supports synchronous and asynchronous communication supporting Level-0, Level-1 and Level-2 type configurations.

1.1.2 Hardware-software mapping

This section describes the system view of the SPI driver and peripherals administered by it.

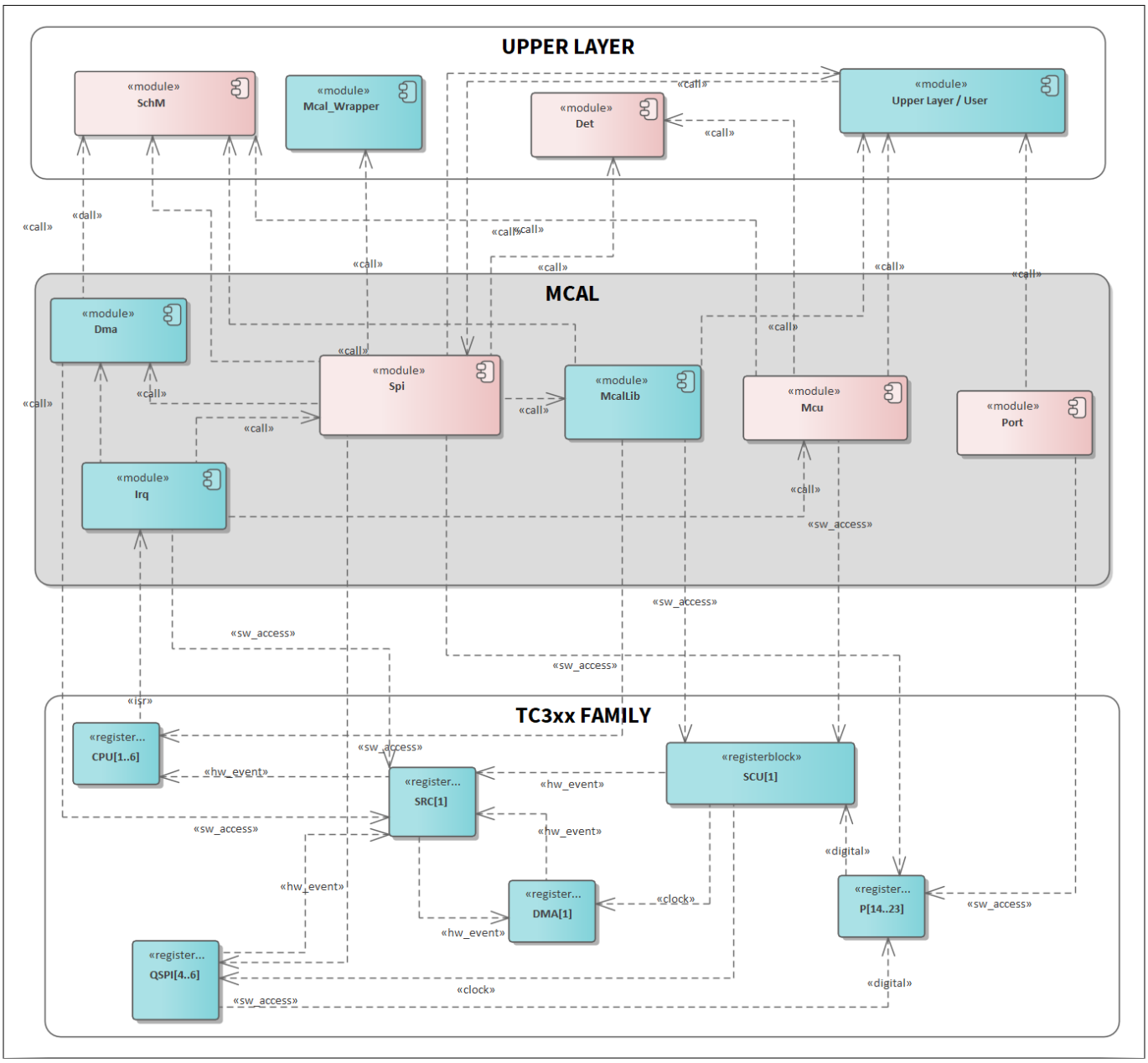


Figure 1 Mapping of hardware-software interfaces

1.1.2.1 QSPI: primary hardware peripheral

Hardware functional features

The SPI driver uses the QSPI for Synchronous and Asynchronous data transfer. The key hardware functional features used by the driver are:

- QSPI FIFOs (Tx and Rx) are configured to work in the continuous data mode
- QSPI FIFOs (Tx and Rx) interrupts are configured to work in the single move mode
- SPI driver uses the QSPI move counter mode during asynchronous data transfer

The unsupported features of the QSPI are:

- High speed input capture
- Slave mode
- Long data block transfer
- ASCLIN
- MIX entry

Users of the hardware

The SPI driver exclusively utilizes the QSPI module.

Hardware diagnostic features

The SMU alarms configured for the QSPI are not monitored by the SPI driver.

Hardware events

The SPI driver uses the following hardware events from the QSPI IP:

- On a transmitter FIFO event - TXF
- On a receiver FIFO event - RXF
- On an error condition (TxFIFO underflow / overflow, RxFIFO underflow / overflow, Expect timeout, parity error) - ERRORFLAGS
- On phase transition (end of frame) - PT2

1.1.2.2 SRC: dependent hardware peripheral

Hardware functional features

The SPI driver depends on the interrupt router for raising an interrupt to the CPU or DMA based on the transmit FIFO event, receive FIFO event, error conditions and Phase transition, which indicates the status of data transmission and reception.

Users of the hardware

The interrupt router is configured either by the IRQ driver or the user software.

Hardware diagnostic features

The SMU alarms configured for the interrupt router are not monitored by the SPI driver.

Hardware events

The interrupt events raised by the interrupt router are serviced by the CPU or DMA. The SPI driver provides interrupt handlers as software interfaces, which must be invoked from the ISR.

1.1.2.3 DMA: dependent hardware peripheral

Hardware functional features

The SPI driver uses the DMA in the Linked list mode for the transmission and reception of data in the Asynchronous mode (Level-1, 2) of transfer. The SPI driver uses the interface APIs provided by the DMA driver to use the DMA functionality.

Users of the hardware

The DMA module is exclusively owned by the DMA driver, but the functionality is shared by many MCAL drivers. The DMA module is triggered for every element transmitted or received on the QSPI interface.

Hardware diagnostic features

The move engine (ME) error is enabled during the data transmission.

Hardware events

If any ME error is encountered during the data transfer then the DMA raises an error which is handled by the DMA driver.

If a channel transfer completion event occurs, DMA notifies SPI module by invoking Spi_QspiDmaCallout which triggers the next SPI channel transmission.

1.1.2.4 SCU: dependent hardware peripheral

Hardware functional features

The SPI driver depends on the SCU IP for the clock, ENDINIT and reset functionalities. The driver requires the fSPB, fQSPI clock signals for functioning.

Users of the hardware

The SCU IP supplies clock for all the peripherals and the MCU driver is responsible for configuring the clock tree. To avoid conflicts due to simultaneous writes, update to all the ENDINIT protected registers is performed using the MCALLIB APIs.

Hardware diagnostic features

The SMU alarms configured for the SCU IP are not monitored by the SPI driver.

Hardware events

Hardware events from the SCU are not used by the SPI driver.

1.1.2.5 PORT: dependent hardware peripheral

Hardware functional features

- The MOSI, MISO, SCLK and SLSO signals are routed to the QSPI through the port pads. MOSI, MISO, SCLK and SLSO configured and enabled through the PORT driver
- For CS_VIA_GPIO, the PORT registers are directly accessed by the SPI driver for asserting/de asserting the chip select (SLSO)

Users of the hardware

The port pads are configured by the PORT driver.

Hardware diagnostic features

Not applicable.

Hardware events

Hardware events from port pads are not used by the SPI driver.

1.1.3 File structure

1.1.3.1 C file structure

This section provides details of the C files of the SPI driver.

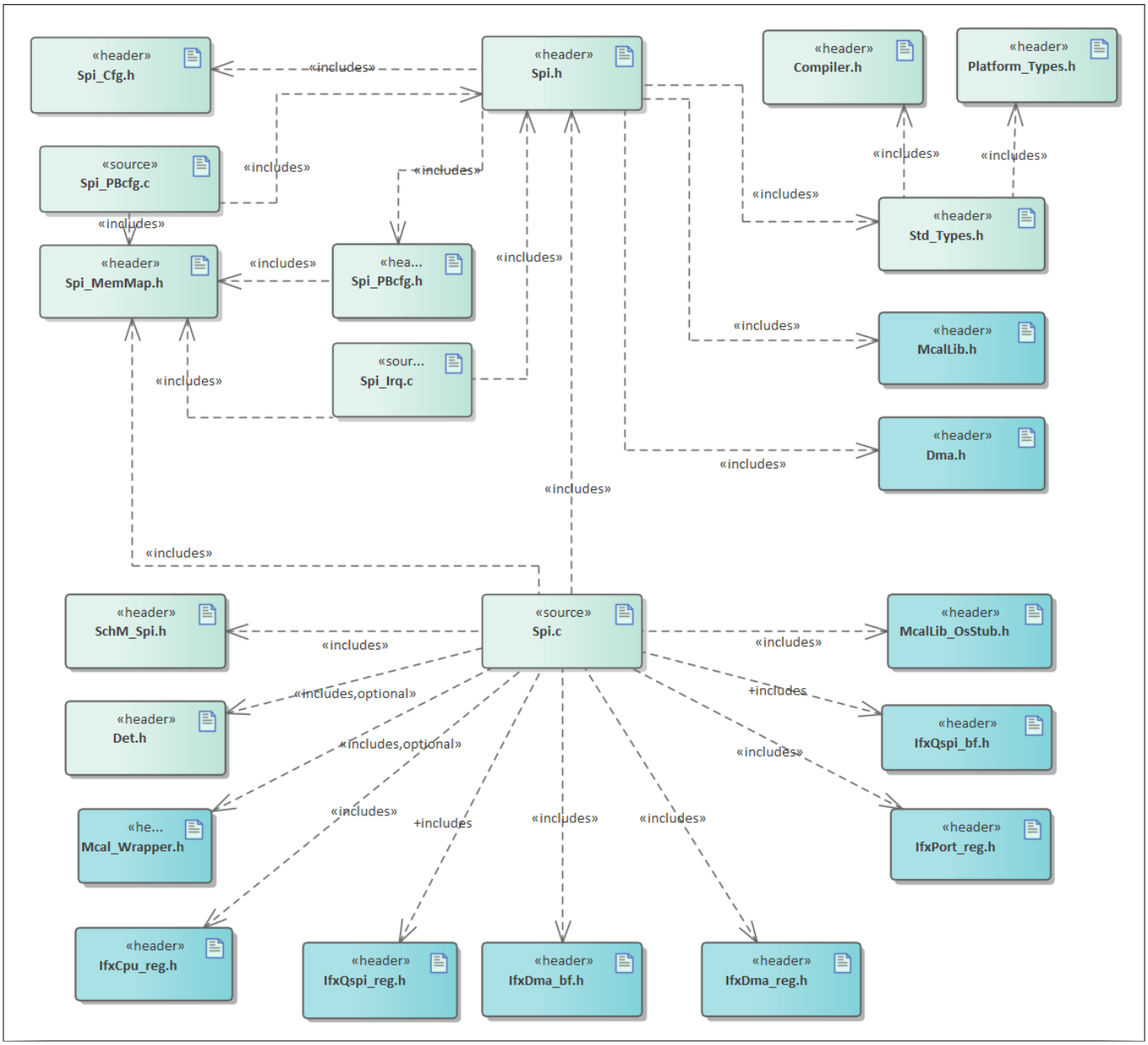


Figure 2 **Spi_C_File_Structure-1.png**

Table 2 C file structure

File name	Description
`Compiler.h`	Provides abstraction from compiler-specific keywords
`Det.h`	Provides the exported interfaces of Development Error Tracer
`Dma.h`	Header file (static) defining prototypes of data structures and APIs
`IfxCpu_reg.h`	SFR header file for CPU
`IfxDma_bf.h`	SFR header file for DMA
`IfxDma_reg.h`	SFR header file for DMA
`IfxPort_reg.h`	SFR header file for Port
`IfxQspi_bf.h`	SFR header file for QSPI
`IfxQspi_reg.h`	SFR header file for QSPI
`McalLib.h`	Static header file defining prototypes of data structure and APIs exported by the MCALLIB.
`McalLib_OsStub.h`	McalLib_OsStub.h provides macros to support user mode of Tricore. This shall be included by other drivers to call OS APIs.
`Mcal_Wrapper.h`	Provides the exported interfaces for Production Error and Runtime Development Errors. Implemented by default to include functions of Dem.h and Det.h files. This file can be modified by the user but function prototype is not user modifiable.
`Platform_Types.h`	Platform-specific type declaration file as defined by AUTOSAR
`SchM_Spi.h`	Export Header for Schm functions of SPI driver. Functions to protect the critical sections.
`Spi.c`	File (Static) containing implementation of APIs
`Spi.h`	Header file (Static) defining prototypes of data structures and APIs
`Spi_Cfg.h`	Header file (Generated) containing constants and pre-processor macros
`Spi_Irq.c`	IRQ file for handling all QSPI interrupts.
`Spi_MemMap.h`	Memmap file is used to define the section of memory to which variables or constants will be placed
`Spi_PBcfg.c`	File (Generated) containing objects to data structures
`Spi_PBcfg.h`	File (Generated) containing declaration of the post-build configuration data structures
`Std_Types.h`	Standard type declaration file as defined by AUTOSAR. It is independent of compiler or platform.

1.1.3.2 Code generator plugin files

This section provides details of the code generator plugin files of the SPI driver.

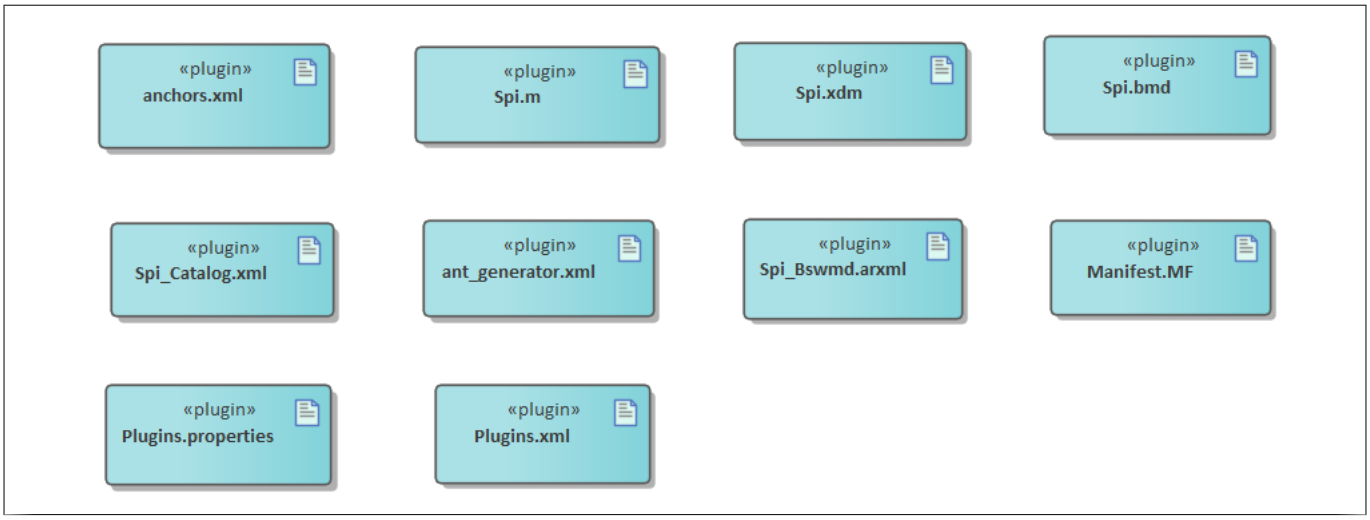


Figure 3 **Spi_Code_Generator_Plugin_Files-1.png**

Table 3 **Code generator plugin files**

File name	Description
`Manifest.MF`	Tresos plugin support file containing the metadata for SPI driver
`Plugins.properties`	Tresos plugin support file for the SPI driver
`Plugins.xml`	Tresos plugin support file for the SPI driver
`Spi.bmd`	AUTOSAR format XML data model schema
`Spi.m`	Macros for XDM logic verification
`Spi.xdm`	Tresos format XML data model schema file
`Spi_Bswmd.arxml`	AUTOSAR format module description file
`Spi_Catalog.xml`	AUTOSAR format catalogue file
`anchors.xml`	Tresos anchors support file for the SPI driver
`ant_generator.xml`	Tresos support file to generate and rename multiple post-build configuration when using variation point

1.1.4 Integration hints

This section describes the key points that an integrator or user of the SPI driver must consider.

1.1.4.1 Integration with AUTOSAR stack

This section lists the modules, which are not part of the MCAL, but are required to integrate the SPI driver.

- **EcuM**

  The ECU Manager module is a part of the AUTOSAR stack that manages common aspects of ECU. Specifically, in the context of MCAL, EcuM is used for initialization and de-initialization of the software drivers. The EcuM module provided in the MCAL package is a stub code and needs to be replaced with a complete EcuM module during the integration phase.

- **Memory mapping**

  Memory mapping is a concept from AUTOSAR that allows relocation of text, variables, constants and configuration data to user-specific memory regions.To achieve this, all the re-locatable elements of the

driver are encapsulated in different memory-section macros. These macros are defined in the `Spi_MemMap.h` file.

The `Spi_MemMap.h` file is provided in the MCAL package as a stub code. The integrator must place appropriate compiler pragmas within the memory-section macros. The pragmas ensure that the elements are re-located to the correct memory region. A sample implementation listing the memory-section macros is shown as follows.

```
#if defined SPI_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED
 /*****User pragmas here for Non-cached LMU*****/
 #undef SPI_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED
 #undef MEMMAP_ERROR
 #elif defined SPI_STOP_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED
 #ifdef _TASKING_C_TRICORE_
 /*****User pragmas here for Non-cached LMU*****/
 #undef SPI_STOP_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED
 #undef MEMMAP_ERROR
 /**** CORE[x] CONFIG DATA -- PF[x] ****/ /*[x]=0..5*/
 #elif defined SPI_START_SEC_CONFIG_DATA_ASIL_B_CORE[x]_UNSPECIFIED
 /*****User pragmas here for PF[x]*****/
 #undef SPI_START_SEC_CONFIG_DATA_ASIL_B_CORE0_UNSPECIFIED
 #undef MEMMAP_ERROR
 #elif defined SPI_STOP_SEC_CONFIG_DATA_ASIL_B_CORE0_UNSPECIFIED
 /*****User pragmas here for PF[x]*****/
 #undef SPI_STOP_SEC_CONFIG_DATA_ASIL_B_CORE0_UNSPECIFIED
 #undef MEMMAP_ERROR
 /**** CODE -- PF[x] ****/
 #elif defined SPI_START_SEC_CODE_ASIL_B_GLOBAL
 /*****User pragmas here for PF[x]*****/
 #undef SPI_START_SEC_CODE_ASIL_B_GLOBAL
 #undef MEMMAP_ERROR
 #elif defined SPI_STOP_SEC_CODE_ASIL_B_GLOBAL
 /*****User pragmas here for PF[x]*****/
 #undef SPI_STOP_SEC_CODE_ASIL_B_GLOBAL
 #undef MEMMAP_ERROR
 #endif

 #if defined MEMMAP_ERROR
 #error "SPI_MemMap.h, wrong pragma command"
 #endif
```

- **DET:**

  The DET module is a part of the AUTOSAR stack that handles all the development errors reported by the BSW modules. The SPI driver reports all the development errors to the DET module through the `Det_ReportError()` API . The user of the SPI driver must process all the errors reported to the DET module through the `Det_ReportError()` API . The files `Det.h` and `Det.c` are provided in the MCAL package as a stub code and needs to be replaced with a complete DET module during the integration phase.

- **Mcal_Wrapper:**

  This Driver performs reporting of the Production and Runtime errors. The handling of the reported errors shall be done by the user. The `Mcal_Wrapper_Det_ReportRuntimeError()` API, `Mcal_Wrapper_Dem_ReportErrorStatus()` API and `Mcal_Wrapper_Dem_SetEventStatus()` API are provided in the `Mcal_Wrapper.h` and `Mcal_Wrapper.c` files as a stub code and can be updated by the integrator to handle

the reported errors. The files `Mcal_Wrapper.h` and `Mcal_Wrapper.c` are user modifiable but the function prototype is not user modifiable and by default the Mcal_Wrapper function shall call AUTOSAR DEM and DET modules.

The user of the SPI driver shall process all the production errors (fail/pass) and Runtime errors reported to the Mcal_Wrapper module. The interface used for reporting Production error in AUTOSAR version 4.2.2 is `Mcal_Wrapper_Dem_ReportErrorStatus()` and for AUTOSAR version 4.4.0 is `Mcal_Wrapper_Dem_SetEventStatus()`, for reporting runtime error is `Mcal_Wrapper_Det_ReportRuntimeError()` API. The `Mcal_Wrapper.h` and `Mcal_Wrapper.c` files are provided in the MCAL package as a stub code and can be replaced with user specific production and runtime error handling module/s during the integration phase.

Note: Reentrancy of the `Spi_SyncTransmit` API is dependent on the reentrancy of `Mcal_Wrapper_Dem_ReportErrorStatus()` API in AUTOSAR version 4.2.2 and `Mcal_Wrapper_Dem_SetEventStatus()` API in AUTOSAR version 4.4.0. As per their design, the modules APIs are reentrant for different hardware units. However, in case `Mcal_Wrapper_Dem_ReportErrorStatus()` API and `Mcal_Wrapper_Dem_SetEventStatus()` API is implemented as non-reentrant, the APIs inherit the property of the same.

- **SchM:**

  The SchM module is a part of the RTE that manages the BSW Scheduler. The SPI driver uses the exclusive areas defined in `SchM_Spi.h` file to protect the SFRs and variables from concurrent accesses from different threads. The SchMs identified for the SPI driver are:

  - Queue_Update

  - SyncLock

  The files `SchM_Spi.h` and `SchM_Spi.c` are provided in the MCAL package as an example code and needs to updated by the integrator. The user must implement the SchM functions defined by the SPI driver as

suspend / resume of interrupts for the CPU on which the API is invoked. A sample implementation of the SchM functions is shown as follows.

```
/**** Sample implementation of SchM_Spi.c ****/

void SchM_Enter_Spi_Queue_Update(void)
{
/* Start of Critical Section */
SuspendAllInterrupts(); /* Suspend CPU core interrupt */
}

void SchM_Exit_Spi_Queue_Update(void)
{
/* End of Critical Section */
ResumeAllInterrupts(); /* Resume CPU core interrupt */
}

void SchM_Enter_Spi_SyncLock(void)
{
/* Start of Critical Section */
SuspendAllInterrupts(); /* Suspend CPU core interrupt */
}

void SchM_Exit_Spi_SyncLock(void)
{
/* End of Critical Section */
ResumeAllInterrupts(); /* Resume CPU core interrupt */
}
```

- **Safety error:**

  The SPI driver reports all the detected safety errors through the `Mcal_ReportSafetyError()` API.

  The driver performs only detection and reporting of the safety errors. The handling of the reported errors shall be done by the user. The `Mcal_ReportSafetyError()` API is provided in the files `Mcal_SafetyError.c` and `Mcal_SafetyError.h` as a stub code, and must be updated by the integrator to handle the reported errors.

  Note: All DET errors are also reported as safety errors (error code used is same as DET).

- **Notifications and callbacks:**

  The SPI driver implements notification functions `Spi_JobEndNotification` and `Spi_SeqEndNotification` for job and sequence completion respectively. These notification functions can be configured by the user in the EB tresos tool for each job and sequence separately.

  In Asynchronous communication, user should configure `Spi_QspiDmaCallout` function as the DMA call-back for RX channel in the respective DMA channel configuration. The configured call-back function `Spi_QspiDmaCallout` is triggered by DMA driver after completion of each channel transmission for updating BACON and start the transfer for successive channel.

  Note: Job and Sequence end notifications are only available for asynchronous communication.

- **Operating system(OS)**

  The OS or the application must ensure correct type of service and interrupt priority is configured in the SR register. Enabling and disabling of interrupts must also be managed by the OS or application.

  The OS files provided by the MCAL package are only an example code and must be updated by the integrator with the actual OS files for the desired function.

1.1.4.2 Multicore and Resource Manager

The SPI driver supports execution of its APIs simultaneously from all CPU cores. The user should allocate resources of the SPI to the CPU cores at pre-compile time using the Resource Manager module. The following are the key points to be considered with respect to multicore in the driver:

- A kernel can be assigned to only one core and cannot be shared between cores. Multiple kernels can be assigned to a core.
- Channels can be re-used within cores, however, protection of data must be taken care by the application code.
- Application must ensure that the channel, job and sequence numbers passed to API belong to same core, else respective DET is triggered from the driver.
- Interrupts raised by the hardware must be serviced by the CPU core to which the kernel is allocated to.
- Locating of constants, variables and configuration data to the correct memory space should be done by the user. Memory sections are marked GLOBAL (common to all cores) and CORE[x](specific to a CPU core). The following should be considered by the user to ensure better performance of the driver:

  Code section:

  The executable code of the SPI driver is placed under single MemMap section. It can be relocated to any PFlash region.

  Data section:

  The RAM variable memory sections marked as specific to a core should be re-located to the DSPR/DLMU of the same core. The sections marked as global should be relocated to the non-cached LMU region.

  Configuration data and constants:

  The configuration data sections marked as specific to a core should be re-located to the PFlash of the same core. The sections marked as global should be relocated to the PFlash of the master core.

  Note 1: Relocating of code, data or constants to a distant memory region would impact execution timings.

  Note 2: If the driver operates from single (master) core, all the sections may be relocated to the PFlash/DSPR/DLMU of the same CPU core.

1.1.4.3 MCU support

The SPI driver is dependent on the MCU driver for clock configuration. The initialization of the SPI driver must be started only after completing the MCU initialization. Configuration parameters `McuQspiClockSourceSelection` and `McuQspiFrequency` need to be considered while for QSPI driver in EB tresos.

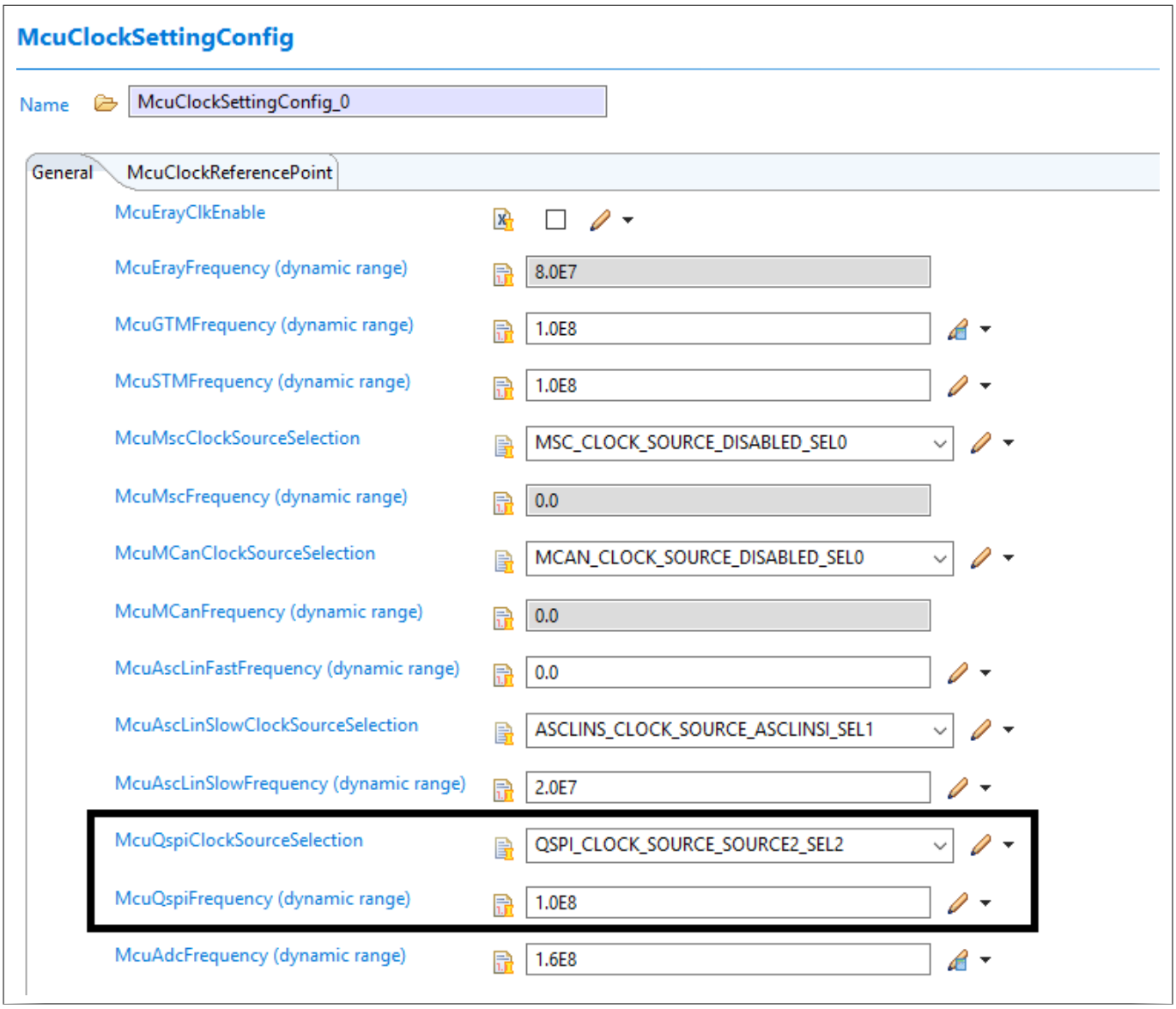


Figure 4 **QSPI clock / Frequency selection**

1.1.4.4 Port support

The PORT driver configures the port pins of the entire microcontroller. The user must configure port pins used by the SPI driver through the PORT configuration and initialize the port pins prior to invoking the SPI initialization.

- MRST - master receive slave transmit
- MTSR - master Transmit slave receive
- CLOCK - clock pin
- SLSO - hardware driven chip select OR GPIO to be operated as chip select

An example configuration for QSPI-0 is shown in the following diagram. Note that the MRST should be configured in SPI driver configuration.

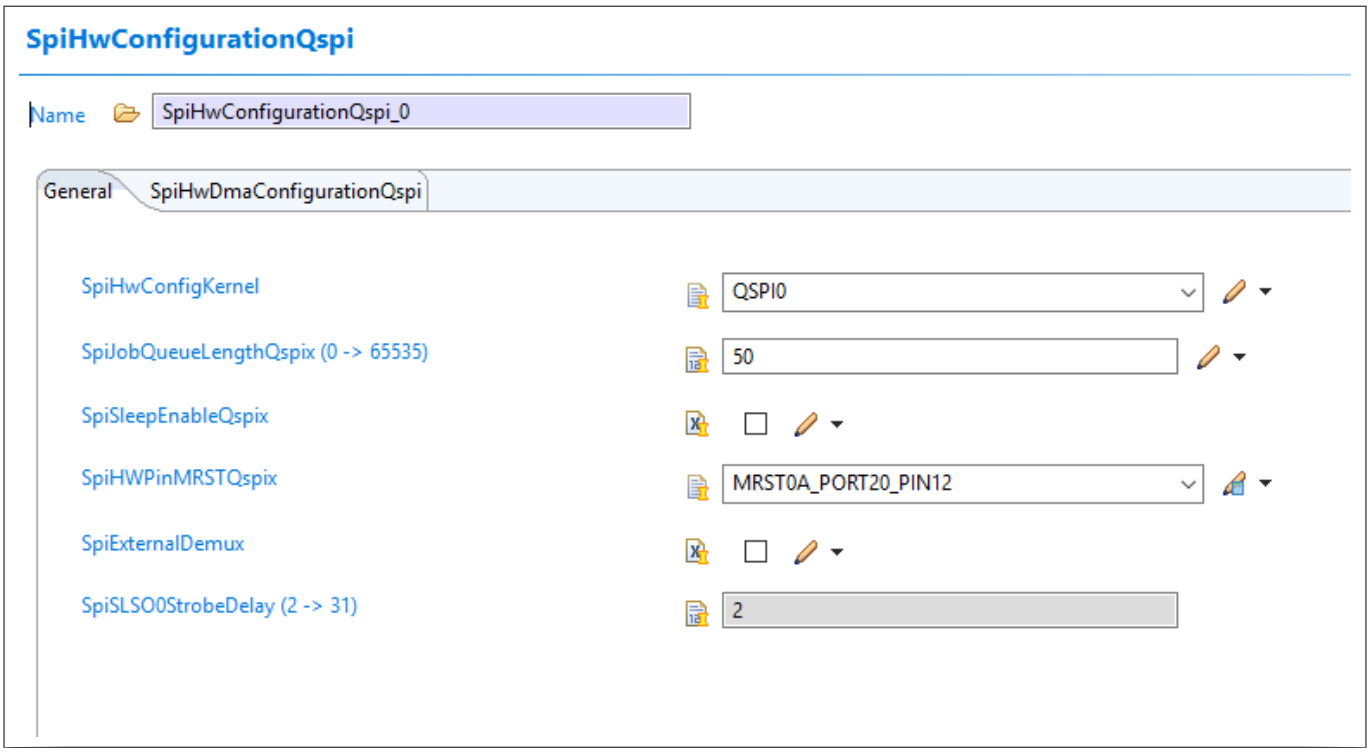
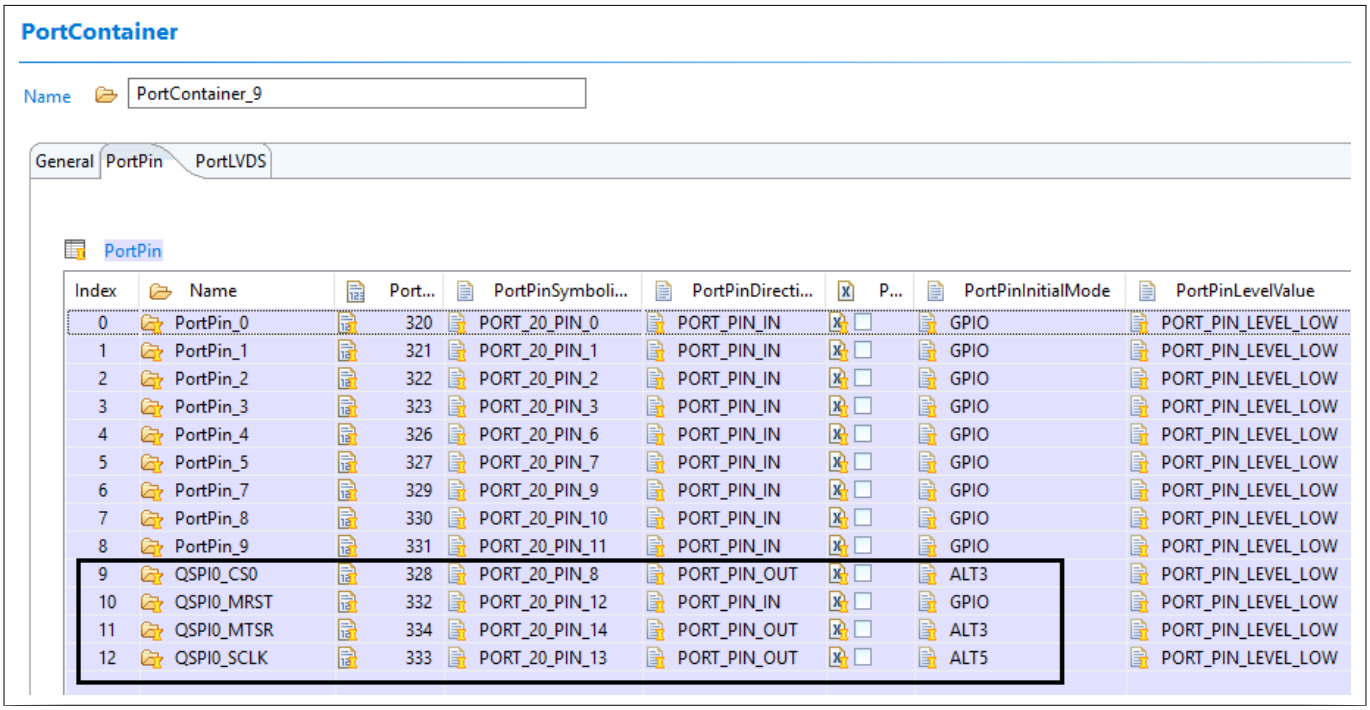


Figure 5 **QSPI MRST Configuration**



Index	Name	Port...	PortPinSymboli...	PortPinDirecti...	P...	PortPinInitialMode	PortPinLevelValue
0	PortPin_0	320	PORT_20_PIN_0	PORT_PIN_IN		GPIO	PORT_PIN_LEVEL_LOW
1	PortPin_1	321	PORT_20_PIN_1	PORT_PIN_IN		GPIO	PORT_PIN_LEVEL_LOW
2	PortPin_2	322	PORT_20_PIN_2	PORT_PIN_IN		GPIO	PORT_PIN_LEVEL_LOW
3	PortPin_3	323	PORT_20_PIN_3	PORT_PIN_IN		GPIO	PORT_PIN_LEVEL_LOW
4	PortPin_4	326	PORT_20_PIN_6	PORT_PIN_IN		GPIO	PORT_PIN_LEVEL_LOW
5	PortPin_5	327	PORT_20_PIN_7	PORT_PIN_IN		GPIO	PORT_PIN_LEVEL_LOW
6	PortPin_7	329	PORT_20_PIN_9	PORT_PIN_IN		GPIO	PORT_PIN_LEVEL_LOW
7	PortPin_8	330	PORT_20_PIN_10	PORT_PIN_IN		GPIO	PORT_PIN_LEVEL_LOW
8	PortPin_9	331	PORT_20_PIN_11	PORT_PIN_IN		GPIO	PORT_PIN_LEVEL_LOW
9	QSPI0_CS0	328	PORT_20_PIN_8	PORT_PIN_OUT		ALT3	PORT_PIN_LEVEL_LOW
10	QSPI0_MRST	332	PORT_20_PIN_12	PORT_PIN_IN		GPIO	PORT_PIN_LEVEL_LOW
11	QSPI0_MTSR	334	PORT_20_PIN_14	PORT_PIN_OUT		ALT3	PORT_PIN_LEVEL_LOW
12	QSPI0_SCLK	333	PORT_20_PIN_13	PORT_PIN_OUT		ALT5	PORT_PIN_LEVEL_LOW

Figure 6 **Configure MTSR, SLSO, SCLK**

1.1.4.5 DMA support

DMA channels should be configured when the QSPI is operated in the Level-1 or Level-2 asynchronous mode. QSPI uses two DMA channels one for RX and another for TX of QSPI. These DMA channels must be reserved for the QSPI communication only and cannot be reused.

In the DMA, in the General configuration section, enable `DmaTriggerApi` as minimum configuration. Enable other configuration items as required by application. No other configurations are required in DMA. Transaction control set configurations for DMA are handled in SPI module and does not need any configuration in DMA module.

Note: Add the respective DMA channel and configure the notification for the DMA Receive channel and error callout as shown in the image below.

For internal buffers (Spi_TxIBBufferCorex, Spi_RxIBBufferCorex) and External buffers, Address space 0xD and 0xC shall not be used for DMA related usage. MemMap sections allocating memory in scratch pad RAM should always generate global addresses instead of local addresses.

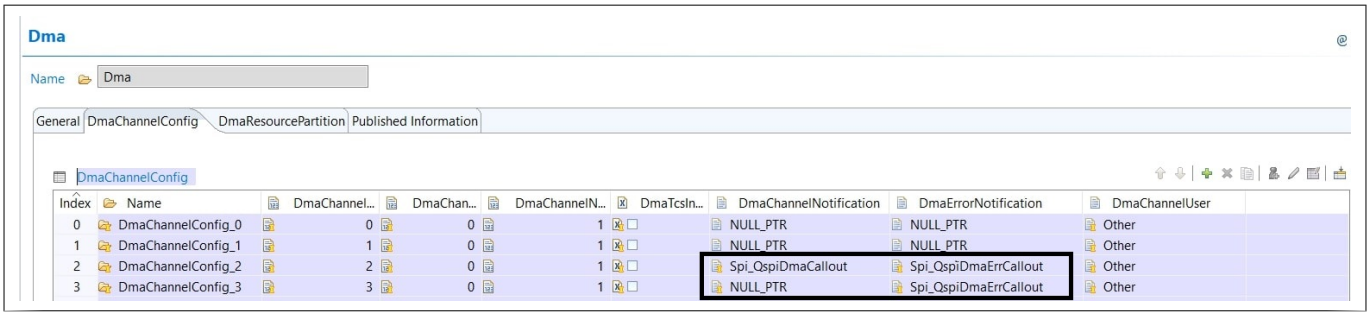


Figure 7 DMA channel configuration

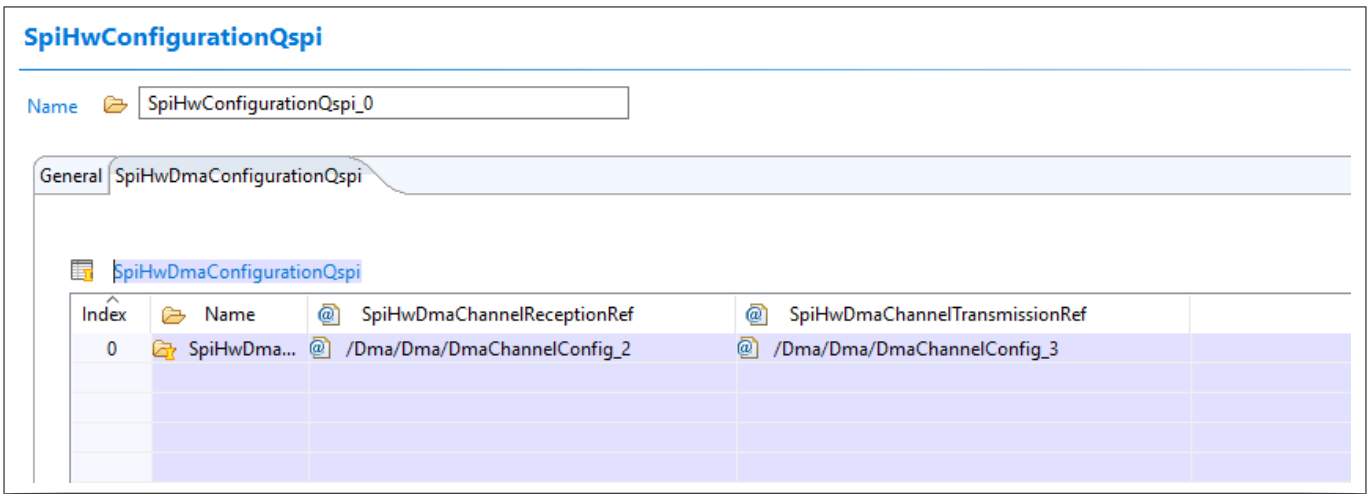


Figure 8 DMA Channel Assignment - SpiHwConfigurationQspi container

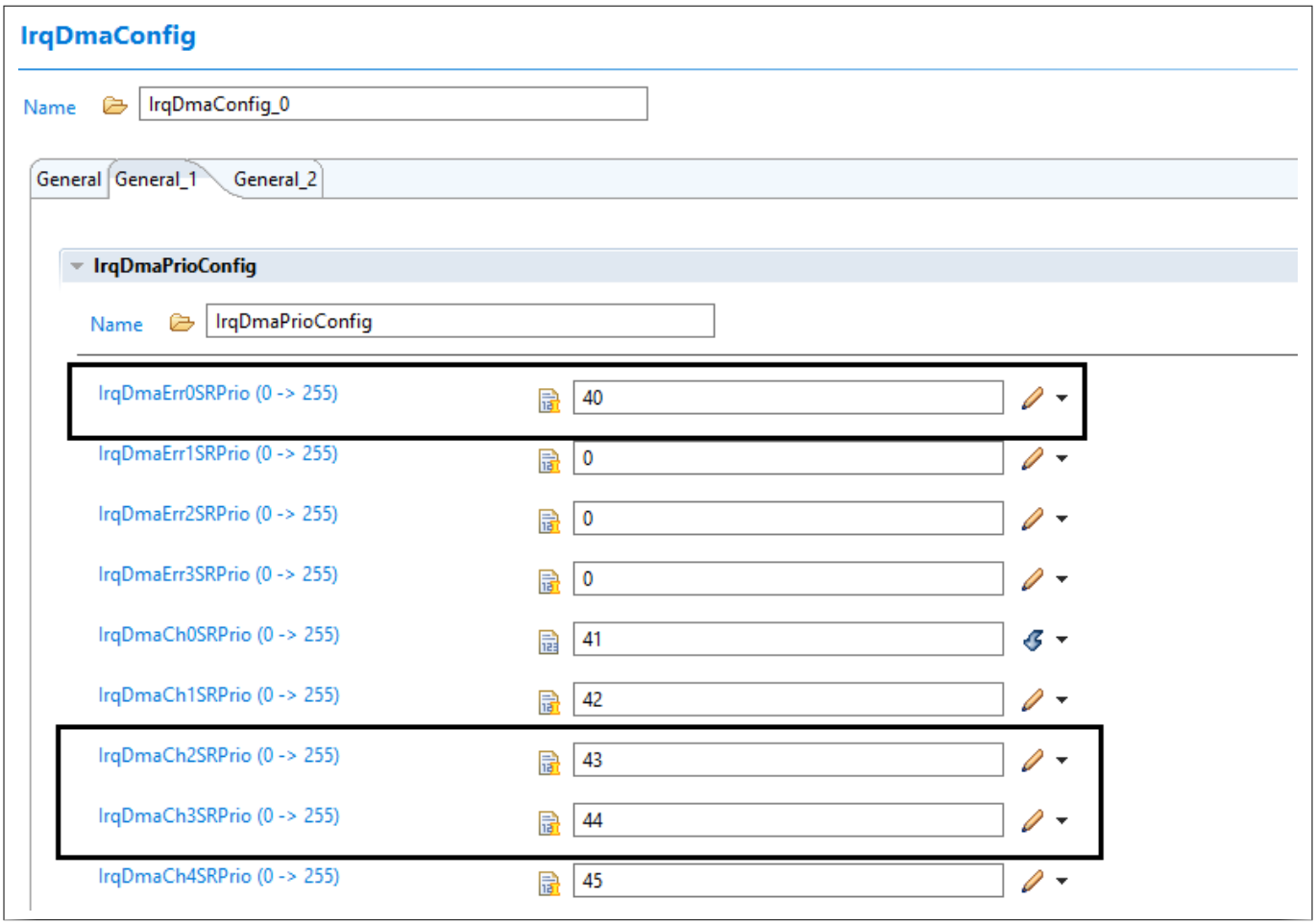


Figure 9 Configure Error and channel interrupts for DMA

ESM - DMA Error handling and Supervision:

The following are the safety measures for the user, for the handling of DMA error during asynchronous transmission:

If DMA channel used by the SPI driver encounters an error, then the DMA driver notifies the error along with the channel information to the SPI driver.

Following steps are recommended as part of Production error Handling:

1. Reset the channel using `Dma_ChStopTransfer` API
2. Reinitialize the channel using `Dma_ChInit` API

Note1: The channel would have reinitialized to the initialization values which were provided during configuration.

Note2: If there are multiple DMA errors on the same resource partition, then due to the HW limitation only the last reported error will be processed by DMA and the intermediate errors will be lost.

1.1.4.6 Interrupt connections

The interrupt connections of the SPI driver are described in this section.

QSPI TX and RX interrupt triggers the DMA channel. The DMA triggers a callout at the end of channel transmission and BACON is updated for next successive channel transmission. PT2 interrupt indicates job frame complete. It is triggered at the end of frame transmission.

Note 1: QSPI TX and RX interrupt priority - respective DMA channel numbers are to be allocated as shown in the sample.

Note 2: Priority number order must be as follows DMA error > QSPI error > IRQ DMA-Ch TX > IRQ DMA-Ch RX > QSPI PT2. All the above interrupts are configured for asynchronous communication only.

Configure the QSPI and DMA priority numbers as shown in the images below.

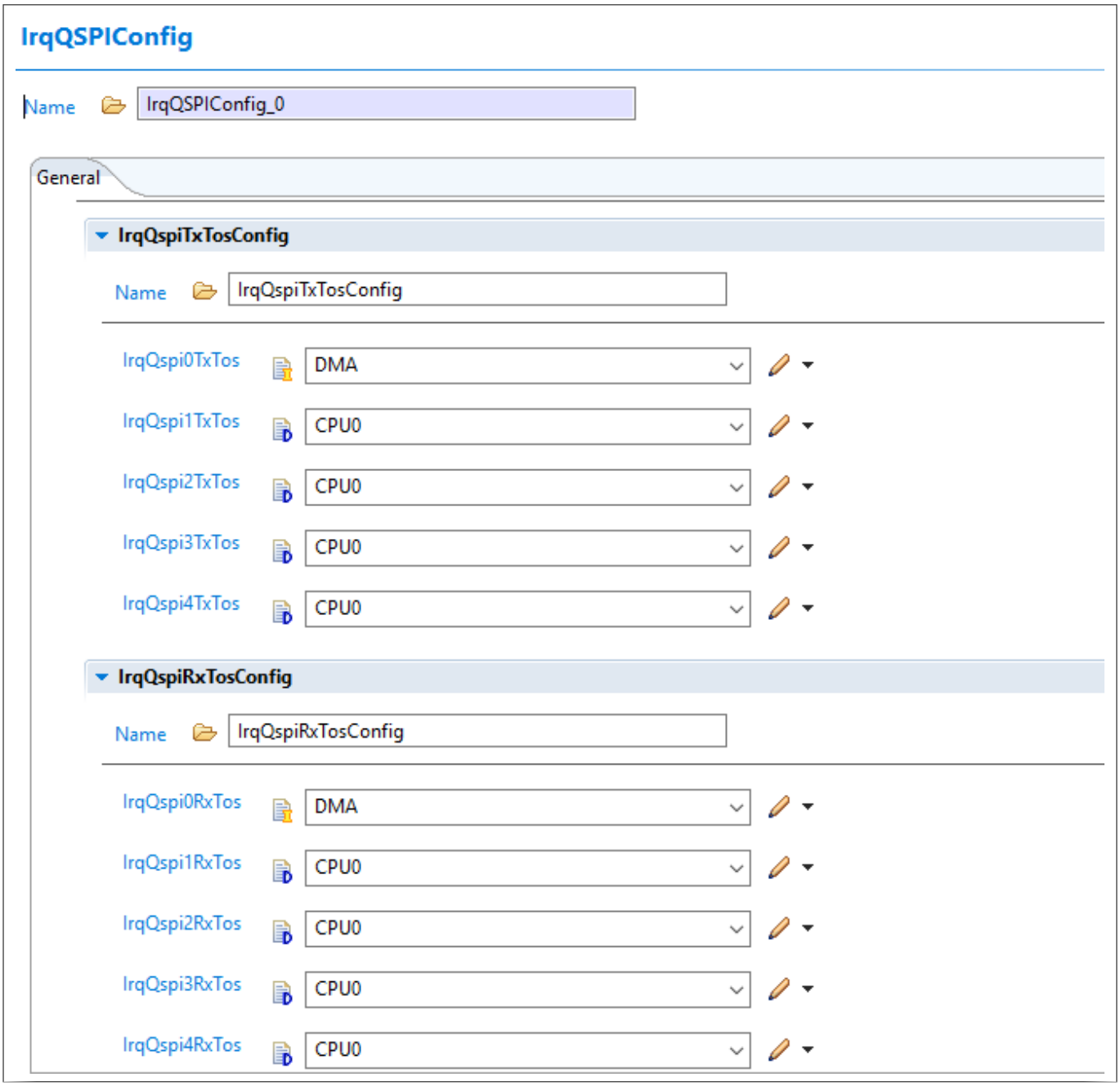


Figure 10 Configure type of service (TOS) - DMA / CPU

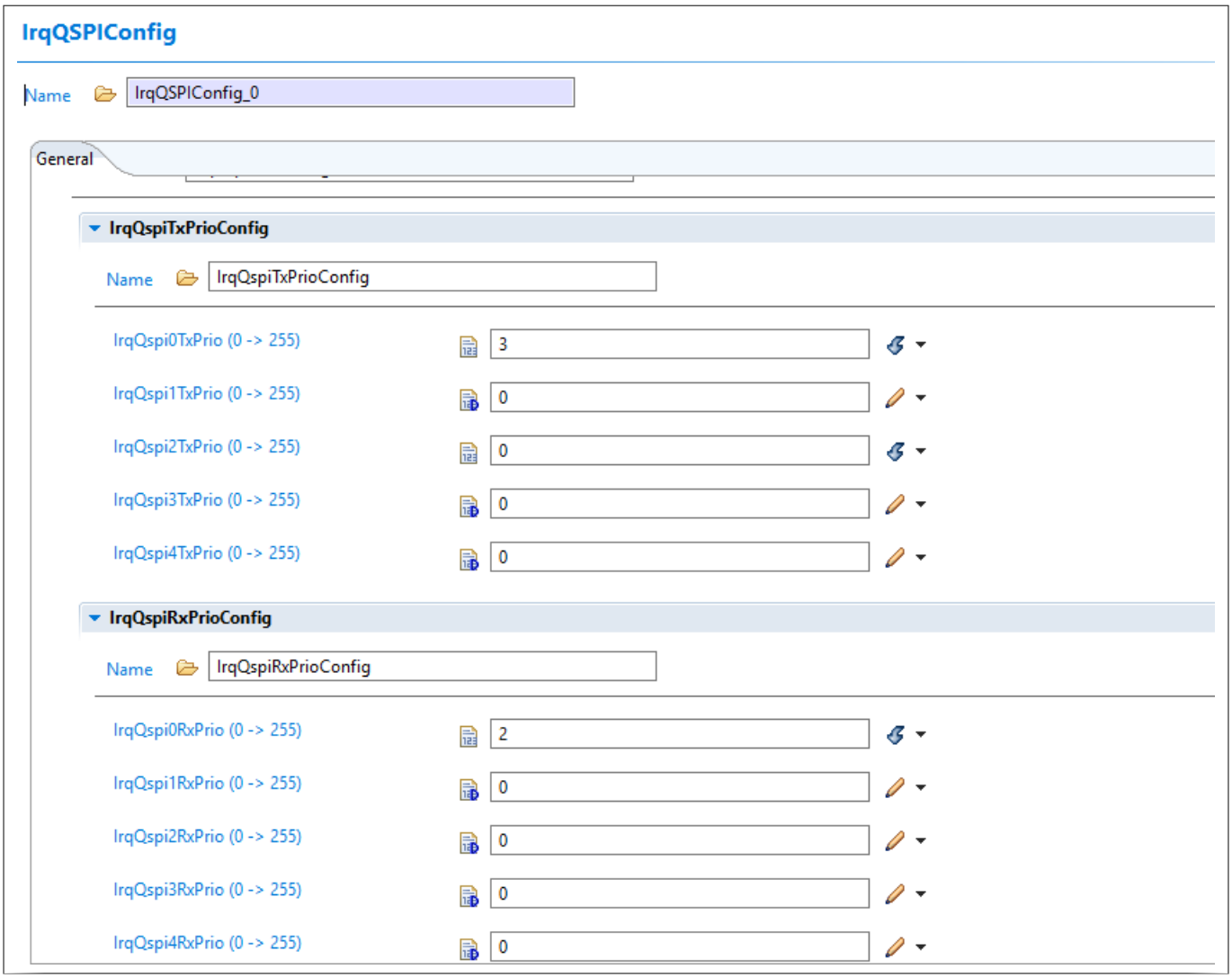


Figure 11 **Configure Tx and Rx interrupt of QSPI**

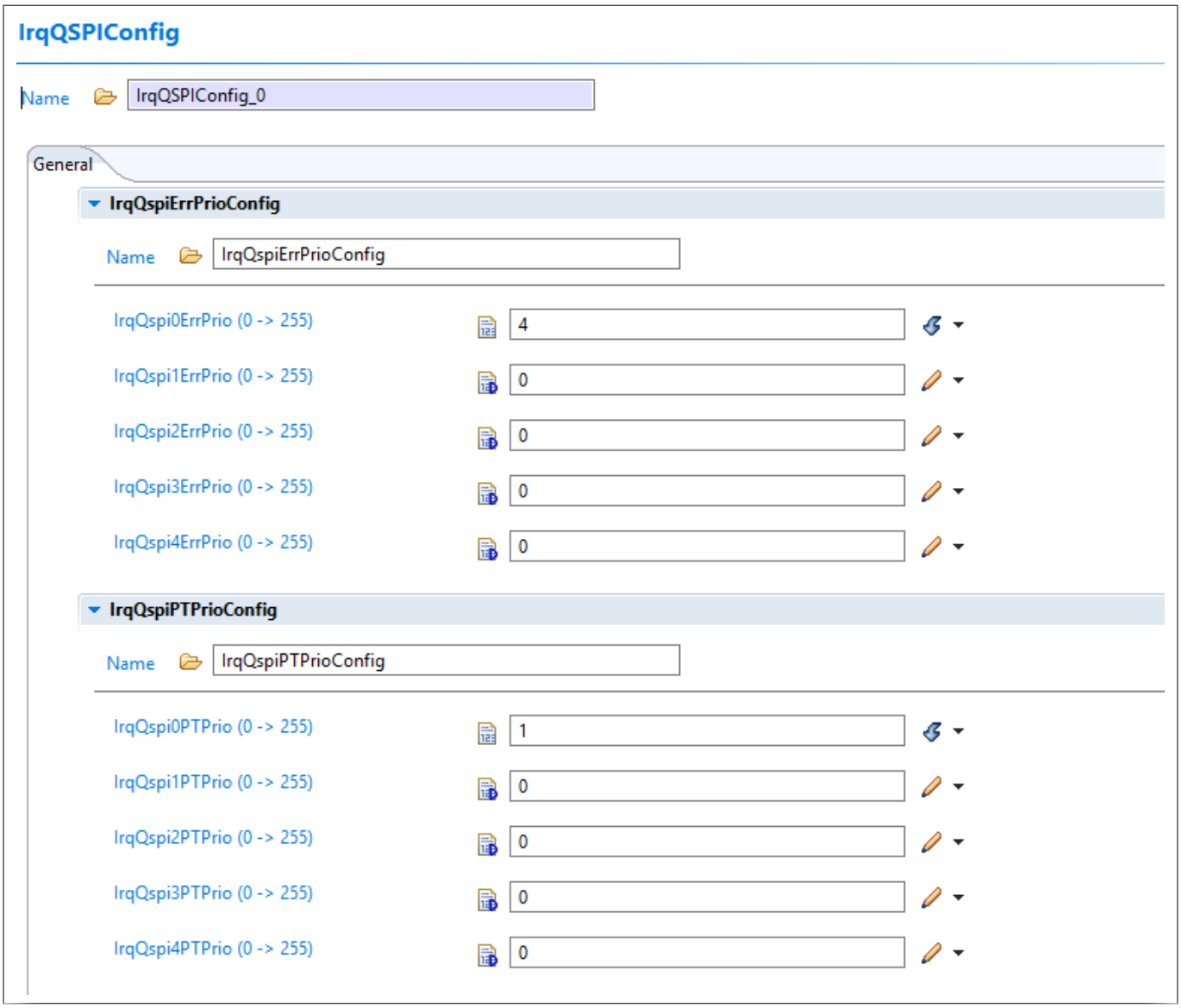


Figure 12 Configure Error and PT interrupt of QSPI

Note 1: IrqQspi0ErrTos and IrqQspi0PTTos configured for CPUx.

Note 2: All interrupts must be configured for Asynchronous transmission (Level-1 and Level-2).

A sample interrupt handler for QSPI0 kernel is depicted in the following code snippet:

```
/* Module header file inclusion */
#include "Spi.h"

ISR(QSPI0ERR_ISR)
{
 /* Call QSPI0 Error Interrupt handler */
 Spi_IsrQspiError(SPI_QSPI0_INDEX);
}

ISR(QSPI0PT_ISR)
{
 /* Call QSPI0 PT2 interrupt handler for frame completion */
 Spi_IsrQspiPT2(SPI_QSPI0_INDEX);
}
```

A sample invocation of interrupts for DMA is depicted as follows (applicable for Level-1 and Level-2)

```
ISR(DMAERR0SR_ISR)
{
 /* Handle error through respective DMA ME */
 Dma_MEInterruptDispatcher();
}

ISR(DMACH0SR_ISR)
{
 /* DMA RX interrupt handler, SPI callback will be called through this interrupt */
 Dma_ChInterruptHandler(0U);
}
```

Note: The following API calls are allowed to use within the SPI callback notifications

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

All other SPI handler/driver APIs must not be called.

1.1.4.7 Example usage

The following are some of the key use cases of the SPI driver.

Note 1: Refer to the comments in the code snippets for additional information.

Note 2: Refer to integration hints of SPI driver and add all the dependent modules.

Note 3: DMA and IRQ module configuration is applicable only for Level-1 and Level-2. IRQ driver is not a productive module and code provided is only a sample code.

Initialization of the SPI driver

The sample code sequence for initializing the SPI driver is as follows:

```
#include "Mcu.h"
#include "Spi.h"
#include "Port.h"
#include "Irq.h"
#if (SPI_LEVEL_DELIVERED != 0)
#include "Dma.h"
#endif

int core0_main (void)
{
/* Initialize all dependent modules */
/* MCU Initialization */
Mcu_Init(&Mcu_Config);
Mcu_InitClock(0U);
while(Mcu_GetPllStatus() != MCU_PLL_LOCKED);
Mcu_DistributePllClock();

/* Port Initialization */
Port_Init(&Port_Config);

#if (SPI_LEVEL_DELIVERED != 0)
 /* Initialize IRQ module */
 IrqDma_Init();
 IrqSpi_Init();
 /* DMA initialization */
 Dma_Init(&Dma_Config);

 /* Enable service request for all the configured interrupts */
 SRC_DMACH2.U |= 0x400U;
 SRC_DMACH3.U |= 0x400U;
 SRC_QSPI0RX.U |= 0x400;
 SRC_QSPI0ERR.U |= 0x400;
 SRC_QSPI0TX.U |= 0x400;
 SRC_QSPI0PT.U |= 0x400;
#endif

/* Initialize SPI module */
Spi_Init(&Spi_Config);
}
```

Sample configuration for Level-0

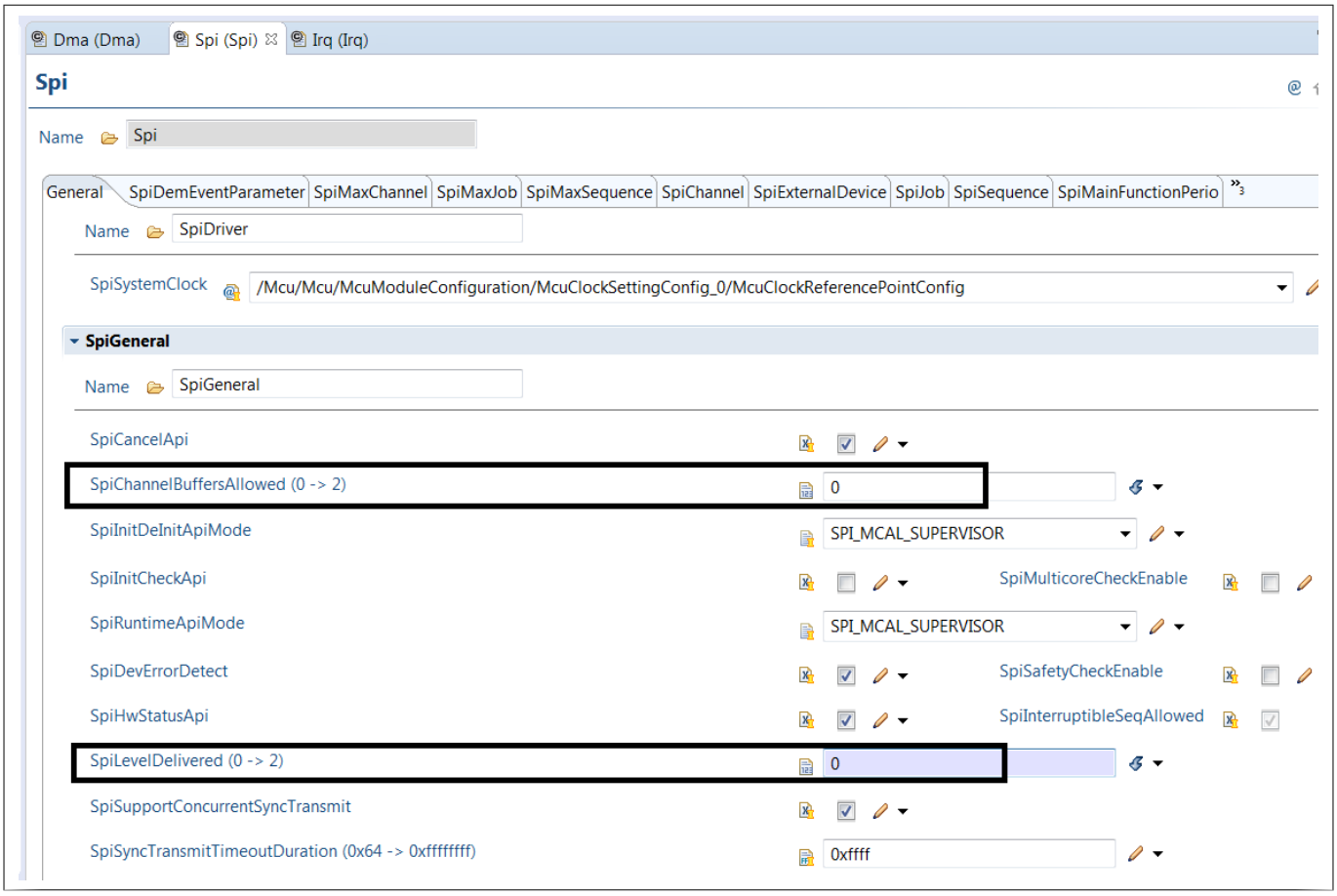


Figure 13 **Create a configuration for Level - 0 with buffers as IB**

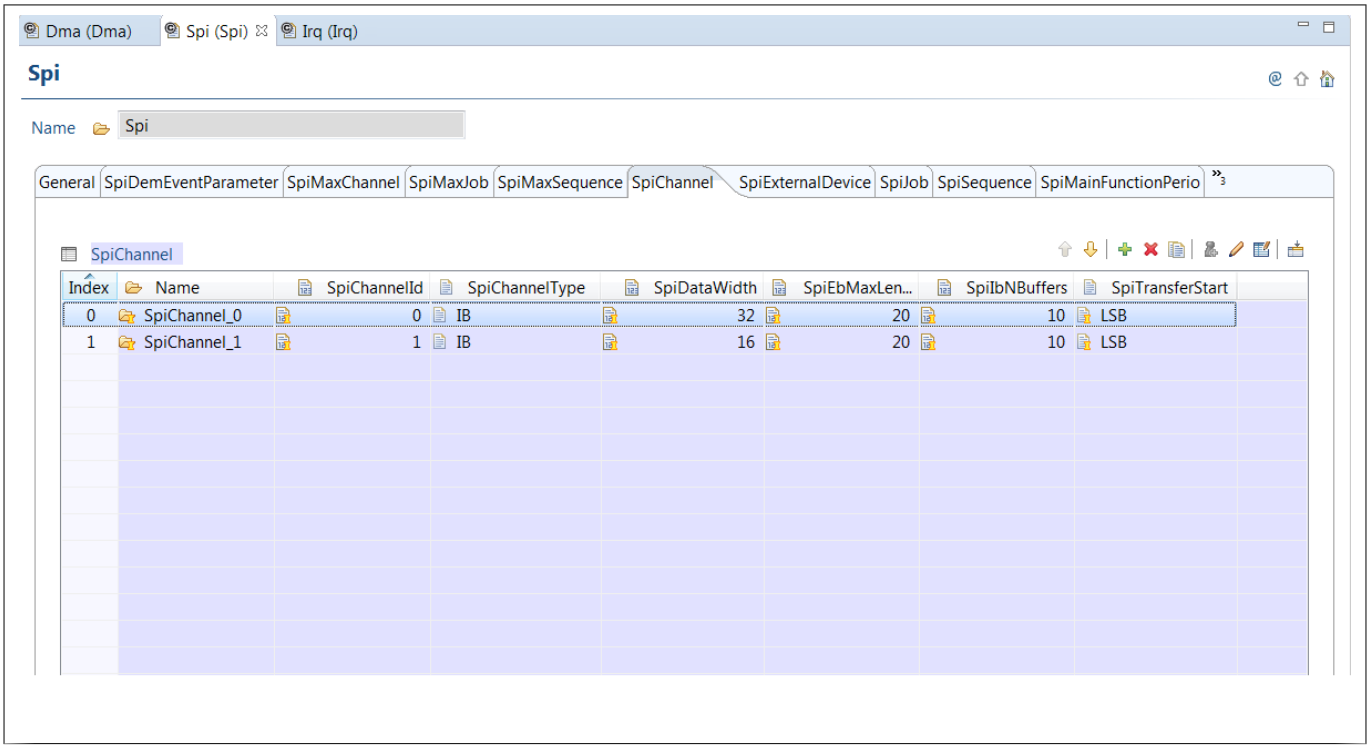


Figure 14 **Create a channels of IB type and define datawidth and length**

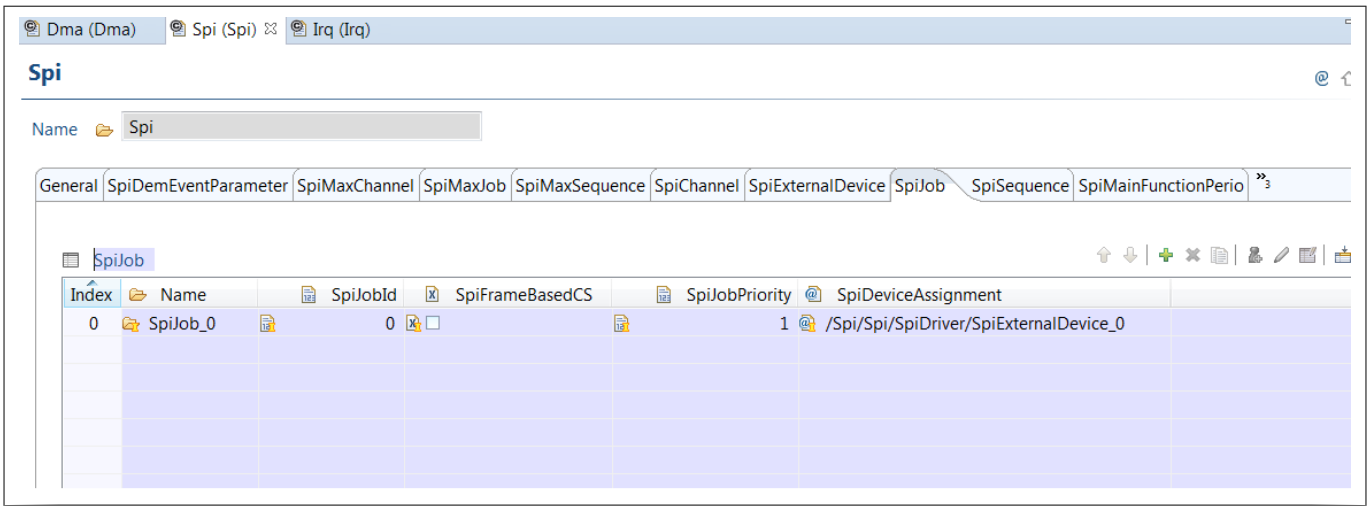


Figure 15 **Choose the external device added in the configuration**

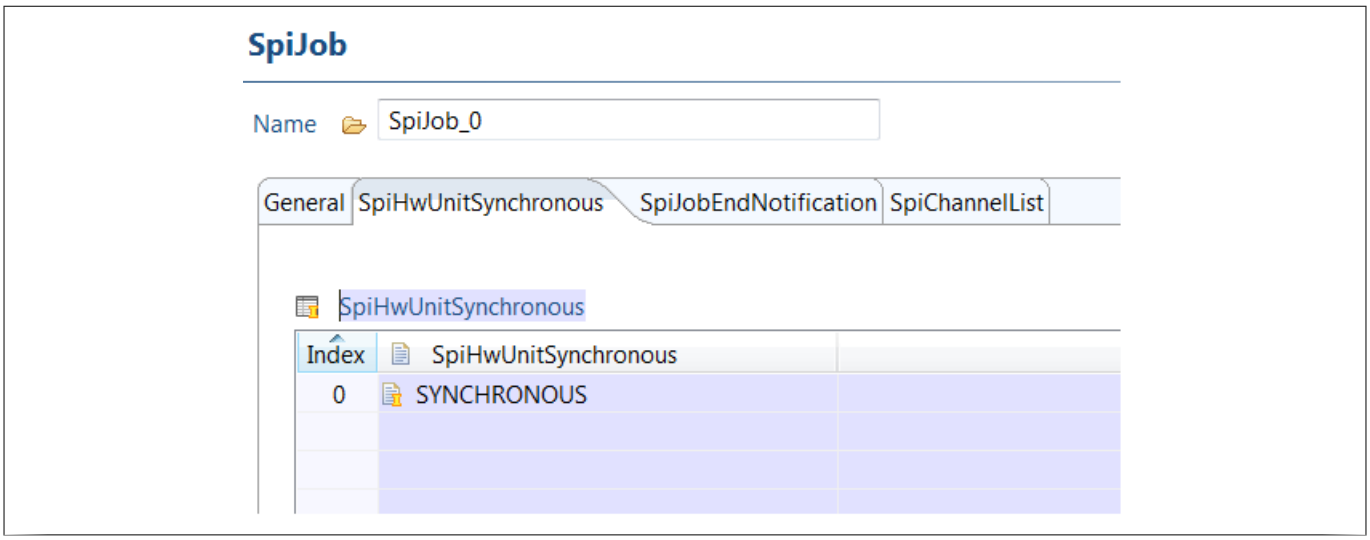


Figure 16 **Choose job to be synchronous (Applicable for 4.2.2 AUTOSAR version)**

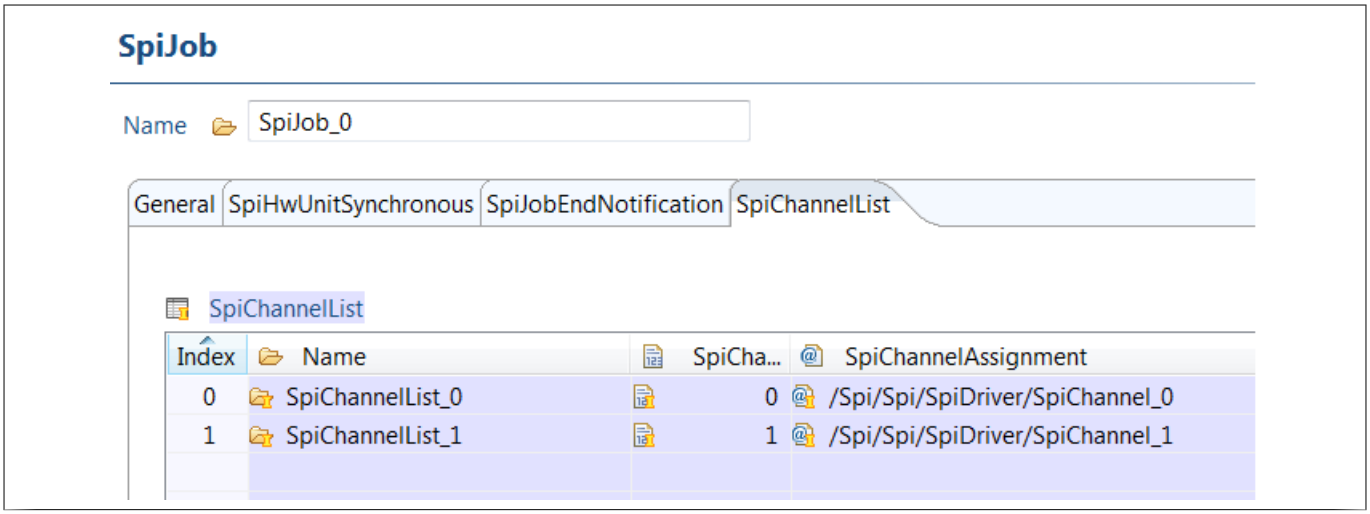


Figure 17 **Choose the channels to be added for job**

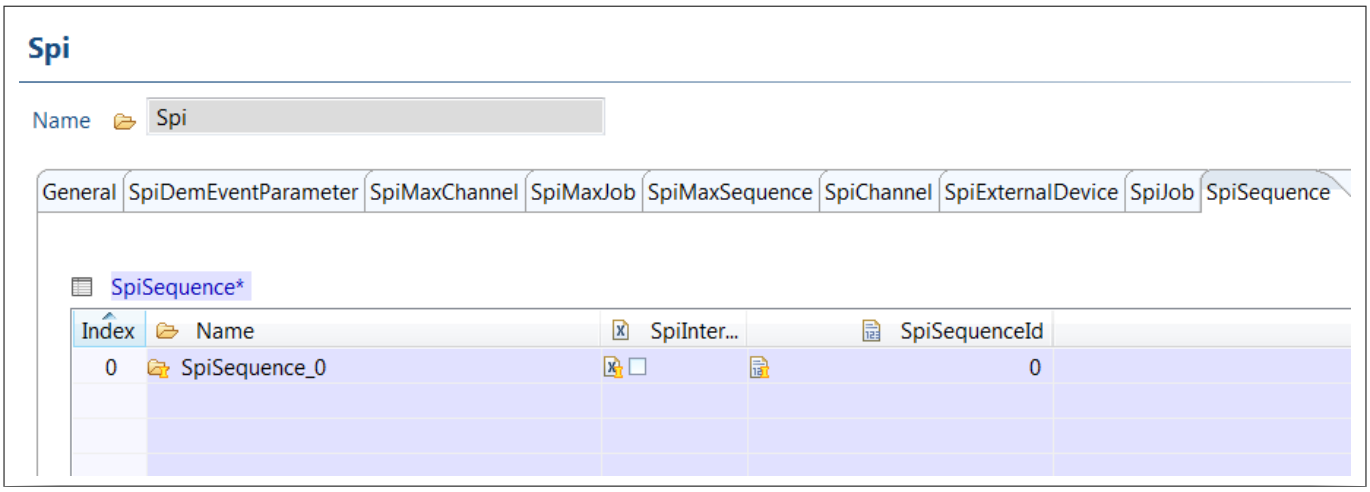


Figure 18 **Add a sequence with jobs assigned**

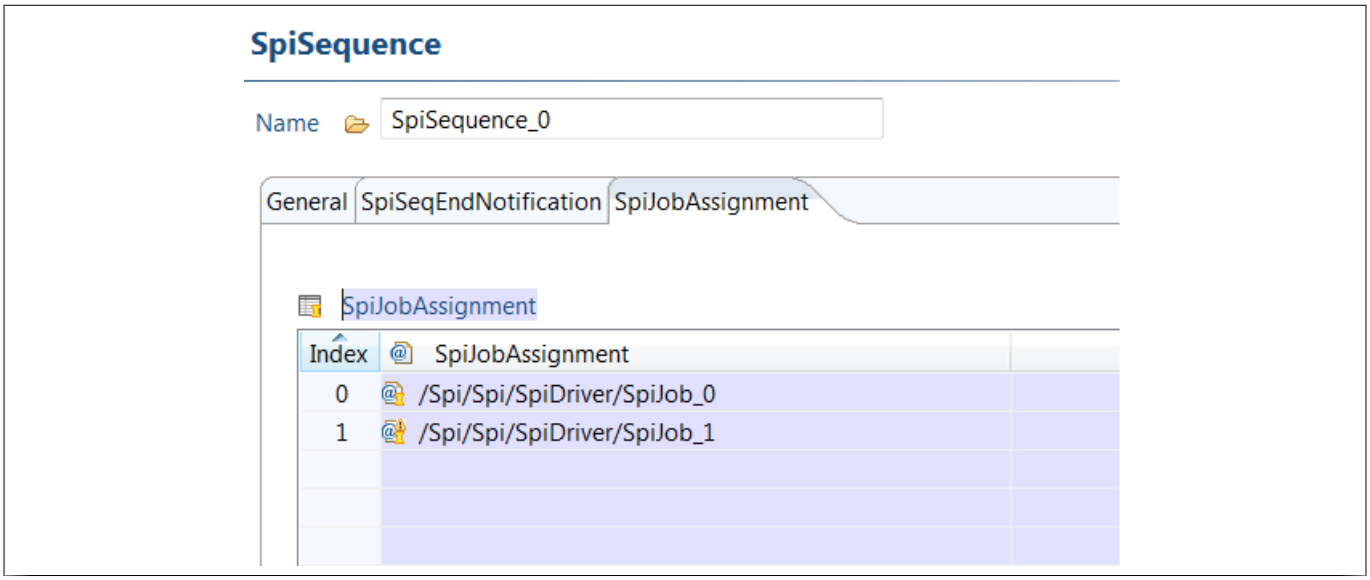


Figure 19 **Configure jobs in Sequence**

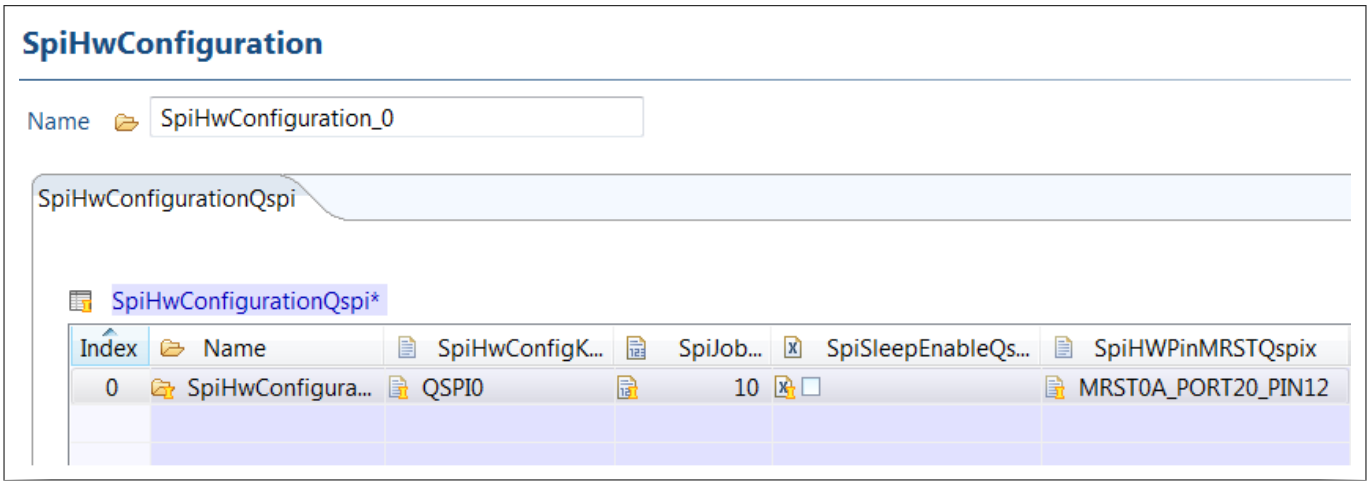


Figure 20 **Configure QSPI Hardware**

Sample code snippets

Setting up internal buffer for IB channels - synchronous transmission

The steps for setting up the IB are as follows:

1. Configure the source buffer to be transmitted via the API Spi_WriteIB.
2. After the buffers are setup for IB channel, transmit API should be invoked.
3. After transmission is completed, the received data should be read back from the internal buffer via the API Spi_ReadIB

```
/* Align data buffers to 4 byte boundary */
#define SPI_START_SEC_VAR_INIT_ASIL_B_CORE0_32
#include "Spi_MemMap.h"

/* Source buffers */
Spi_DataBufferType Spi_SrcBuf0[BUFFER_LENGTH] = {
 0x11111111,
 0x22222222,
 0xAAAAAAAA,
 0x55555555};

Spi_DataBufferType Spi_SrcBuf1[BUFFER_LENGTH] = {
 0x22222222,
 0x11111111,
 0x77777777,
 0xAAAAAAAA};

#define SPI_STOP_SEC_VAR_INIT_ASIL_B_CORE0_32
#include "Spi_MemMap.h"

/* Align data buffers to 4 byte boundary */
#define SPI_START_SEC_VAR_CLEARED_ASIL_B_CORE0_32
#include "Spi_MemMap.h"

/* Destination buffers */
Spi_DataBufferType Spi_DestBuf0[BUFFER_LENGTH];
Spi_DataBufferType Spi_DestBuf1[BUFFER_LENGTH];

#define SPI_STOP_SEC_VAR_CLEARED_ASIL_B_CORE0_32
#include "Spi_MemMap.h"

/* Initialize source buffers */
Spi_WriteIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_0,Spi_SrcBuf0);
Spi_WriteIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_1,Spi_SrcBuf1);

/* Transmit data */
u8returnvalue = Spi_SyncTransmit(SpiConf_SpiSequence_SpiSequence_0);

/* Read the received data from IB */
Spi_ReadIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_0,Spi_DestBuf0);
Spi_ReadIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_1,Spi_DestBuf1);
```

Setting up Internal Buffer for IB Channels - Asynchronous transmission

```
/* Align data buffers to 4 byte boundary */
#define SPI_START_SEC_VAR_INIT_ASIL_B_CORE0_32
#include "Spi_MemMap.h"

/* Source buffers */
Spi_DataBufferType Spi_SrcBuf0[BUFFER_LENGTH] = {
 0x11111111,
 0x22222222,
 0xAAAAAAAA,
 0x55555555};

Spi_DataBufferType Spi_SrcBuf1[BUFFER_LENGTH] = {
 0x22222222,
 0x11111111,
 0x77777777,
 0xAAAAAAAA};

#define SPI_STOP_SEC_VAR_INIT_ASIL_B_CORE0_32
#include "Spi_MemMap.h"

/* Align data buffers to 4 byte boundary */
#define SPI_START_SEC_VAR_CLEARED_ASIL_B_CORE0_32
#include "Spi_MemMap.h"

/* if Level-1 then support is only interrupt mode
*  if Level-2 then ensure to call Spi_SetAsyncMode() to switch to interrupt mode
*/
#if(SPI_LEVEL_DELIVERED == 2U)
Spi_SetAsyncMode((Spi_AsyncModeType)1U);
#endif

/* Destination buffer */
Spi_DataBufferType Spi_DestBuf0[BUFFER_LENGTH];
Spi_DataBufferType Spi_DestBuf1[BUFFER_LENGTH];

#define SPI_STOP_SEC_VAR_CLEARED_ASIL_B_CORE0_32
#include "Spi_MemMap.h"

/* Write data to IB buffer */
Spi_WriteIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_0,Spi_SrcBuf0);
Spi_WriteIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_1,Spi_SrcBuf1);

/* Start data transmission */
u8returnvalue = Spi_AsyncTransmit(SpiConf_SpiSequence_SpiSequence_0);

/* Wait till the transmission is complete */
while(Spi_GetStatus() == SPI_BUSY);

/* Read the received data from IB buffer */
```

```
Spi_ReadIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_0,Spi_DestBuf0);
Spi_ReadIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_1,Spi_DestBuf1);
```

Polling transmission for asynchronous transmission

```
/* In Level-2, set the asynchronous transmission mode to interrupt (1) / polling (0) */
Spi_SetAsyncMode((Spi_AsyncModeType)0U);

/* Write data to IB buffer */
Spi_WriteIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_0,Spi_SrcBuf0);
Spi_WriteIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_1,Spi_SrcBuf1);

/* start data transmission */
u8returnvalue = Spi_AsyncTransmit(SpiConf_SpiSequence_SpiSequence_0);

/* poll till the transmission completes */
while(Spi_GetStatus() == SPI_BUSY)
{
 Spi_MainFunction_Handling();
}

/* Read data from IB buffer */
Spi_ReadIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_0,Spi_DestBuf0);
Spi_ReadIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_1,Spi_DestBuf1);
```

SyncTransmit API trigerring queued jobs:

Note : This is applicable only in Autosar version 4.4.0

1. SyncTransmit API is called with a sequence.

2. During ongoing Spi_SyncTransmit, a request for Spi_AsyncTransmit for another sequence is accepted and queued for same QSPI hardware. These jobs will be triggered for transmission once the synchronous transmission is completed.

```
/* Align data buffers to 4 byte boundary */
#define SPI_START_SEC_VAR_INIT_ASIL_B_CORE0_32
#include "Spi_MemMap.h"

/* Source buffers */
Spi_DataBufferType Spi_SrcBuf0[BUFFER_LENGTH] = {
0x11111111,
0x22222222,
0xAAAAAAAA,
0x55555555};
Spi_DataBufferType Spi_SrcBuf1[BUFFER_LENGTH] = {
0x22222222,
0x11111111,
0x77777777,
0xAAAAAAAA};
Spi_DataBufferType Spi_SrcBuf2[BUFFER_LENGTH] = {
0x33333333,
0x66666666,
0x77777777,
0xBBBBBBBB};
Spi_DataBufferType Spi_SrcBuf3[BUFFER_LENGTH] = {
0x44444444,
0x88888888,
0x11111111,
0xCCCCCCCC};

#define SPI_STOP_SEC_VAR_INIT_ASIL_B_CORE0_32
#include "Spi_MemMap.h"

/* Align data buffers to 4 byte boundary */
#define SPI_START_SEC_VAR_CLEARED_ASIL_B_CORE0_32
#include "Spi_MemMap.h"

/* Destination buffer */
Spi_DataBufferType Spi_DestBuf0[BUFFER_LENGTH];
Spi_DataBufferType Spi_DestBuf1[BUFFER_LENGTH];
Spi_DataBufferType Spi_DestBuf2[BUFFER_LENGTH];
Spi_DataBufferType Spi_DestBuf3[BUFFER_LENGTH];

#define SPI_STOP_SEC_VAR_CLEARED_ASIL_B_CORE0_32
#include "Spi_MemMap.h"

/* Write data to IB buffer */
Spi_WriteIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_0,Spi_SrcBuf0);
Spi_WriteIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_1,Spi_SrcBuf1);
Spi_WriteIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_2,Spi_SrcBuf2);
Spi_WriteIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_3,Spi_SrcBuf3);

/* Thread 1: Start data transmission Synchronous*/
```

```
u8returnvalue0 = Spi_SyncTransmit(SpiConf_SpiSequence_SpiSequence_0);

....
/* Thread 2: Request Asynchronous Transmission on same QSPI HW */
u8returnvalue1 = Spi_AsyncTransmit(SpiConf_SpiSequence_SpiSequence_1);

/* Wait until Transmission is complete */
while(Spi_GetStatus() == SPI_BUSY);

/* Read the received data from IB buffer */
Spi_ReadIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_0,Spi_DestBuf0);
Spi_ReadIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_1,Spi_DestBuf1);
Spi_ReadIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_2,Spi_DestBuf2);
Spi_ReadIB((Spi_ChannelType)SpiConf_SpiChannel_SpiChannel_3,Spi_DestBuf3);
```

Number and position of interrupts for a sample Spi sequence configuration

Number and position of Spi interrupts		
Example configuration Of Spi Sequence	**Spi_AsyncTransmit**	
	Interrupt	**Polling**
Number of Jobs = 2 Number of Channels = 3 FramebasedCs = FALSE	Number of interrupts: 1 DMA Rx transfer complete interrupt / channel, hence 3 interrupts for 3 channels. 1 PT2 interrupt / job, hence 2 interrupts for 2 jobs. 1 DMA TRL interrupt / channel, hence 3 interrupts for 3 channels. Total number of interrupts = 8.	Number of interrupts: QSPI Tx and Rx interrupts must be still configured to trigger DMA channels – but they do not interrupt CPU. Hence we do not count them here. However the DMA TRL interrupt still occurs. 1 DMA TRL interrupt / channel, hence 3 interrupts for 3 channels. Total number of interrupts = 3.
	Position/event of interrupts relative to SPI transmission on bus: DMA Rx interrupt at the end of each channel transmission. DMA TRL interrupt at the end of each channel transmission. PT2 interrupt at the end of each job (i.e. last channel transmission in job)	Position/event of interrupts relative to SPI transmission on bus: DMA TRL interrupt at the end of each channel transmission.

Figure 21 Number and position of Spi interrupts

Note 1: When `DmaTcsInterruptTransactionLoss` is enabled in DMA, TRL events are passed to SPI driver. If this field is disabled, TRL event is suppressed at the DMA driver.

Note 2: The TRL interrupt should be configured to be ignored in the DMA channel configuration.

Note 3: If FramebasedCs = TRUE, then PT2 interrupt is not used and the transition to next job happens in the `Spi_QspiDmaCallout()` API.

Note 4: In case of errors during asynchronous transmission, either QSPI or DMA error interrupt will be triggered.

1.1.5 Key architectural considerations

1.1.5.1 Transmission modes supported

The following transmission modes are supported:

- **Level 0:** Supports only synchronous transmission

In this mode, data to be transmitted is directly copied to TX FIFO. Data is transmitted in the order defined in the configuration. Note that this functionality is a blocking call, that is, until the transmission completes or error occurs, the API will not return the status.

- **Level 1:** Supports only asynchronous transmission

In this mode, the respective interrupts are configured and DMA is configured in Linklist mode for data transfer. The following interrupts are configured in Level 1 mode: TXF, RXF, PT2 and Error. The DMA driver invokes the callback function registered by the QSPI module on completion of channel transmission. Note that all the jobs of sequences that can be interrupted will be placed in priority order if interruptible sequence feature is enabled. Note that for asynchronous transmission each kernel is allocated with an independent queue to handle jobs as per priority.

- **Level 2:** Supports both synchronous and asynchronous transmission

Asynchronous transmission is further supported with either of Interrupt and polling mode. In the polling mode, DMA is still used for TX and RX transfers in interrupt mode, however the RX complete event is polled to check if channel transmission is complete to trigger the start of next channel transmission. Error and PT2 flags are polled to indicate if the frame is complete. Interrupt mode in Level 2 is same as Level 1 implementation. Note that In level-2 mode either synchronous or asynchronous sequences can be configured for transmission.

1.1.5.2 General configuration

Decision on configuration of hardware

FIFO configuration

- TXFIFO / RXFIFO are configured for the Single move mode
- Move counter mode enabled with contentious move mode for transmission of data

Asynchronous communication (L1 and L2 mode)

By using the Move counter, SLSO de-assertion is handled by the hardware when the MCCOUNT reaches to "0". The following interrupts/callbacks are configured for QSPI for Asynchronous mode:

- TXF - Handled by DMA - Transmit FIFO interrupt - Request for feeding FIFO
- RXF - Handled by DMA - Receive FIFO interrupt - Request for emptying the FIFO
- ERROR - Handled by CPU - On QSPI Hardware error
- DMA Callback - Handled by CPU - Occurs on completion of channel data transmission
- DMA Error Callback - Handled by CPU - On DMA transfer errors (Move engine errors).
- PT2 interrupt - Handled by CPU - Occurs on complete transmission of job. Note that PT2 interrupt is needed to know the end of frame and counter can be loaded only when the IP is in the IDLE state

DMA usage / configuration

- Do note that two dedicated DMA channels should be assigned /QSPI each for TX and RX
- DMA is configured for asynchronous communication only and is configured to one DMA move (that is, one DMA transfer has one DMA move). DMA is configured to work in DMA linklist mode and two list are maintained for transmission and reception accordingly
- TCS memory for DMA linklist is allocated in the QSPI module and shall be passed to DMA by configuring suitably based on channel configuration by SPI module

- Any errors during the DMA transfer from move engine will be handled by the DMA module and should callback the error handler in the respective core to which kernel is assigned

- In order to achieve asynchronous communication priority Queue is implemented which is applicable for asynchronous communication only. Note that the jobs in Queue are maintained in priority order and each kernel is assigned with a Queue to maintain job id and its properties

Synchronous communication (L0 mode)

- TX FIFO is directly fed with data without using the DMA.

- In Synchronous communication, transfer of data is done in blocking call, and the transmit function waits for data to be received or till the timeout occurs.

- No Queuing mechanism used for synchronous communication.

1.1.5.3 Multicore decision

TC3xx is designed to have maximum six instances of QSPI and this varies based on the device variant. Each instance is defined to be a kernel and same is used through the document to represent single instance of QSPI IP. A kernel can be assigned to one of the cores and cannot be shared between cores. Multiple kernels can be assigned to a core.

All cores will work independent of each other, so configuration of master core is not applicable for this driver. However, if kernel is configured and not assigned to any core then configuration would be generated for master core. `Spi_Init` and `Spi_Deinit` can be called by any core and same will not affect the operation on other cores except for the one being called.

All APIs will be able to access the information configured for the local core only. For example, `Spi_GetJobResult` and `Spi_GetSequenceResult` APIs can return status for the jobs and sequence assigned to same core only, APIs cannot return results for the sequence assigned to different cores respective DETs will be raised if cross-core information is requested.

The `Spi_GetStatus` API will return the status local core. For example, if two kernels are assigned to core 1, if communication is in progress for core 2, and if `Spi_GetStatus` is called on core 1, IDLE will be returned if no communication on core 1. If the `Spi_GetStatus` is called on core 2, BUSY will be returned.

1.1.5.4 Sequence, jobs and channels

Jobs cannot be shared between different sequences across different kernels within a sequence all the jobs should belong to the same hardware kernel.

Channels can be shared between two jobs within the same core. However protecting the content of the channel is the responsibility of the application code.

Maximum of 8190 elements can be transmitted for a job. This is the limitation of the counter that has been used by QSPI. However this limitation does not apply for Frame-based CS logic. For frame based CS, maximum elements of 16383 elements can be transmitted in a channel. For synchronous transfer, maximum of 65534 elements can be transmitted.

1.1.5.5 Lookup tables

Lookup tables are added to expedite the access in configuration structures. Lookup tables are added for Sequence Ids, Job ids and Channel ids. Ids map the physical index to application Ids (Index in Lookup) in core configuration.

In the multicore environment since the configuration is spread across cores, accessing specific information for sequence, job and channel is time consuming since the applications ids are different from the physical location in core configuration. These tables are generated during code generation and these tables are placed in a core Flash accessible by all the cores.

1.1.5.6 Interruptible sequence behavior

When SPI_INTERRUPTIBLE_SEQ_ALLOWED is ON

- If the incoming sequence is interruptible, the Individual jobs of a sequence are arranged in queue as per the priority of the job.

Note: The order of placing the jobs in queue cannot guarantee that all the jobs of a sequence are placed in consecutive locations.

- If the incoming sequence is non-interruptible, Instead of individual jobs, the entire sequence is placed in queue as per the priority of the first job in sequence.

Note: The order of placing the jobs in queue guarantees that all the jobs of a sequence are placed in consecutive locations.

When SPI_INTERRUPTIBLE_SEQ_ALLOWED is OFF

All the incoming sequences are considered as non-interruptible and entire sequence is placed in queue as per the priority of first job in sequence.

Note: The order of placing the jobs in queue guarantees that all the jobs of a sequence are placed in consecutive locations.

1.1.5.7 External demultiplexer feature

If a QSPI is configured to operate in external demultiplexer mode, the SLSO1 to SLSO4 are driven with the value configured for configuration parameter `SpiCsIdentifier` of external device. To ensure glitch free selection, a strobe signal SLSO0 is provided and the value of the strobe delay is configured using configuration parameter`SpiSLSO0StrobeDelay` . The polarity for all these SLSO lines(SLSO0..SLSO4) are configured with the same value given for configuration parameter `SpiCsPolarity` of external device.

At any given point of time, a QSPI can be operated either in external demultiplexer mode or normal mode.

1.2 Assumptions of Use (AoU)

The AoU for the SPI driver are as follows.

- **SPI API Sequence**

Integrator shall make sure the following sequence of APIs are followed before calling the Spi_Init API:

L0 Mode:

1. Mcu_Init(&Mcu_Config)
2. Mcu_InitClock(0U)
3. Mcu_DistributePllClock()
4. Port_Init(&Port_Config)
5. Spi_Init(&Spi_Config)

L1, L2 mode:

1. Mcu_Init(&Mcu_Config)
2. Mcu_InitClock(0U)
3. Mcu_DistributePllClock()
4. IrqDma_Init()
5. IrqSpi_Init()
6. Dma_Init(&Dma_Config)
7. Port_Init(&Port_Config)
8. Spi_Init(&Spi_Config)

[cover parentID SPI={7C102304-D585-410a-9D19-2A54F6076E91}]

- **Spi_InitCheck**

Integrator shall make sure that all APIs of the SPI driver except for Spi_init, Spi_GetversionInfo API shall be called only after successful execution of the Spi_InitCheck API. SFRs related to other modules getting modified cannot be verified. This should be checked by dependent modules InitCheck APIs.

Rationale: InitCheck API shall ensure all the pointers used for cores and kernels are properly initialized and state of driver is in known state before calling any API.
[cover parentID SPI={6073DDFE-0B7F-43e9-95C4-197677F5BC53}]

- **Global configuration pointer**

Global configuration pointer passed for the Spi_Init API should be same across all the cores.

Rationale: If other pointer is passed configuration is corrupted and behaviour is unpredictable.

[cover parentID SPI={4603B79C-742E-464e-9824-B8C498554055}]

- **SchM implementation**

All Schm_x function calls are non-productive functions and are to be implemented from the application developer. Core-related interrupts are expected to be disabled between the entry and exit of the Schm calls.

Following listed Schm functions are implemented in the SPI module:

- SchM_Enter_Spi_Queue_Update() / SchM_Exit_Spi_Queue_Update()
- SchM_Enter_Spi_SyncLock() / SchM_Exit_Spi_SyncLock()

Rationale: Schm calls are made to protect the global variables shared across interrupt / different API context.
[cover parentID SPI={C087EEC6-8339-403c-A251-FB28C8CB6F9B}]

- **DMA Rx notification**

The SPI is dependent on DMA to do transfer of data. The SPI driver does not perform interrupt source check, it is the responsibility of the dependent module like DMA shall perform the interrupt source check before calling the respective channel notification / handlers.

Rationale: Registers of dependent module are accessed only by module driver only.
[cover parentID SPI={84215E23-909B-46ae-A499-07FA622AE7FA}]

- **DMA resource allocation to SPI module**

The integrator shall ensure that two dedicated DMA channels are to be allocated for TX and RX of QSPI and these channels cannot be shared with any peripherals or changed dynamically.

Rationale: DMA channels cannot be shared once allocated to QSPI.
[cover parentID SPI={43520DE0-4A16-427b-84AF-13A616D8B977}]

- **Watchdog triggering**

The integration should make sure that watchdog is enabled and is triggered in case of no response or driver is stuck in a busy state.

Rationale : Due to hardware faults, interrupts may not get triggered and driver may be in BUSY state.
[cover parentID SPI={041495D5-3E7A-4708-8608-D817B5F51EC0}]

1.3 Reference information

1.3.1 Configuration interfaces

Supported configuration variant: Post-Build

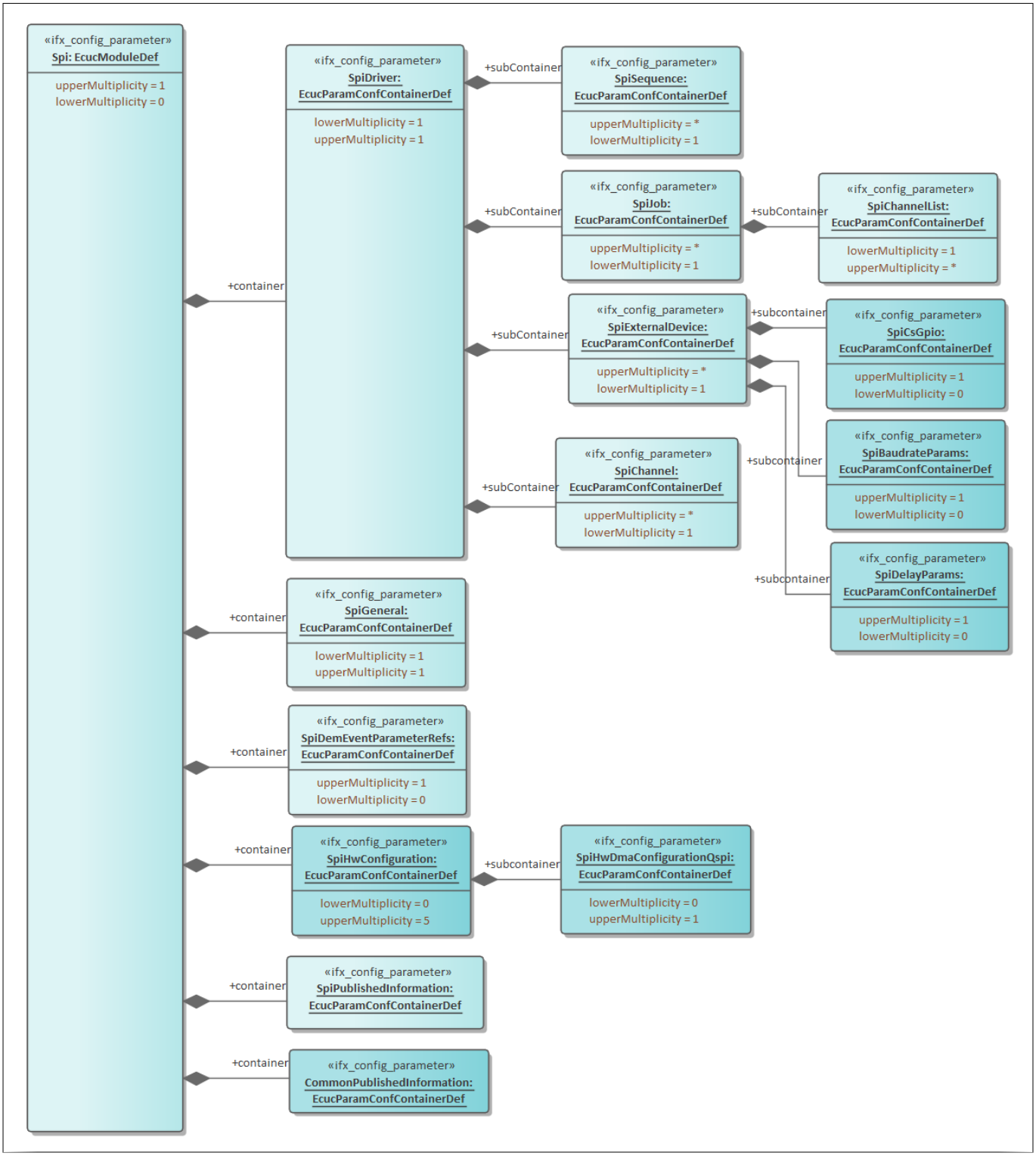


Figure 22 **Container hierarchy along with their configuration parameters**

1.3.1.1 Container: CommonPublishedInformation

Container holding all SPI specific published information parameters

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.1.1 ArMajorVersion

Table 4 Specification for ArMajorVersion

Name	`ArMajorVersion`		
Description	Parameter provides the major version of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	4		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.2 ArMinorVersion

Table 5 Specification for ArMinorVersion

Name	`ArMinorVersion`		
Description	Parameter provides the minor version of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per the selected Autosar version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.3 ArPatchVersion

Table 6 Specification for ArPatchVersion

Name	`ArPatchVersion`		
Description	Parameter provides the patch version of the AUTOSAR specification.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per the selected Autosar version		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.4 Module ID

Table 7 Specification for Module ID

Name	`Module ID`		
Description	Module id of SPI - 83		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	83		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.5 Release

Table 8 Specification for Release

Name	`Release`
Description	This parameter indicates the TC3xx device derivative used for the implementation.

(table continues...)

Table 8 (continued) Specification for Release

Multiplicity	1..1	**Type**	EcucStringParamDef
Range	String		
Default value	As per Hardware unit configured		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.6 SwMajorVersion

Table 9 Specification for SwMajorVersion

Name	`SwMajorVersion`		
Description	Module Majorversion		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 255		
Default value	As per driver		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.7 SwMinorVersion

Table 10 Specification for SwMinorVersion

Name	`SwMinorVersion`		
Description	Module Minor version		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per Driver		

(table continues...)

Table 10 (continued) Specification for SwMinorVersion

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.8 SwPatchVersion

Table 11 Specification for SwPatchVersion

Name	`SwPatchVersion`		
Description	Module Patch version		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	As per Driver		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.1.9 Vendor ID

Table 12 Specification for Vendor ID

Name	`Vendor ID`		
Description	IFX Vendor ID - 17		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	17		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 12 (continued) Specification for Vendor ID

Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.2 Container: Spi

Configuration of the Spi (Serial Peripheral Interface) module.

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.3 Container: SpiBaudrateParams

Container to hold the configuration elements required for configuring the right baudrate.

This container is Applicable only when the parameter SpiAutoCalcBaudParams is set to FALSE

Note: The Multiplicity for the container is 0..1

Post-Build Variant Multiplicity: TRUE

Multiplicity Configuration Class: Post-Build

1.3.1.3.1 SpiBaudParamA

Table 13 Specification for SpiBaudParamA

Name	`SpiBaudParamA`		
Description	Bit Segment 1 Length expressed in quanta of Q b00 - 1 b01 - 2 b10 - 3 b11 - 4 Applicable only when the parameter SpiAutoCalcBaudParams is set to false. *Note: This configuration parameter is used to configure the baudrate in the ECON register of the QSPI kernel. Default value 1, is chosen to select the wide range of baudrate.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 3		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-

(table continues...)

Table 13 (continued) Specification for SpiBaudParamA

Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.2 SpiBaudParamB

Table 14 Specification for SpiBaudParamB

Name	`SpiBaudParamB`		
Description	Bit Segment 2 Length expressed in quanta of Q b00 - 0 b01 - 1 b10 - 2 b11 - 3 Applicable only when the parameter SpiAutoCalcBaudParams is set to false. *Note: This configuration parameter is used to configure the baudrate in the ECON register of the QSPI kernel. Default value 0, is chosen to select the wide range of baudrate.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 3		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.3 SpiBaudParamC

Table 15 Specification for SpiBaudParamC

Name	`SpiBaudParamC`

(table continues...)

Table 15 (continued) Specification for SpiBaudParamC

Description	Bit Segment 3 Length expressed in quanta of Q b00 - 0 (if B=0, than C is minimum 1 per hardware) b01 - 1 b10 - 2 b11 - 3 Applicable only when the parameter SpiAutoCalcBaudParams is set to false. *Note1:* *- If SpiBaudParamB = 0, then SpiBaudParamC should have minimum of value 1* *- When SpiBaudParamB =0 and SpiBaudParamC =0 , no configuration error is reported because HW takes care of setting the SpiBaudParamC to 1* *Note2: This configuration parameter is used to configure the baudrate in the ECON register of the QSPI kernel. Default value 1, is chosen to select the wide range of baudrate.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 3		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.4 SpiBaudParamQ

Table 16 Specification for SpiBaudParamQ

Name	`SpiBaudParamQ`
Description	This defines the time quantum length used by A, B, and C to define the baud rate and duty cycle b000000 - 1 b000001 - 2 .... b111111 - 64 Applicable only when the parameter SpiAutoCalcBaudParams is set to false. *Note: This configuration parameter is used to configure the baudrate in the ECON register of the QSPI kernel. Default value 10, is chosen to select the wide range of baudrate.*

(table continues...)

Table 16 (continued) Specification for SpiBaudParamQ

Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 63		
Default value	10		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.3.5 SpiBaudParamTQ

Table 17 Specification for SpiBaudParamTQ

Name	`SpiBaudParamTQ`		
Description	Global Time Quantum Length Common n-divider scaling the baud rates of all channels in direction of higher or lower baud rates. 0 - division by 1 1 - division by 2 ... 255 - division by 256 Applicable only when the parameter SpiAutoCalcBaudParams is set to false. *Note: This configuration parameter is used to configure the baudrate in the GLOBALCON register of the QSPI kernel. Default value 2, is chosen to select the wide range of baudrate.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 255		
Default value	2		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4 Container: SpiChannel

This container contains the configuration parameters to describe a channel.

Lower multiplicity is 1 and upper multiplicity is 255.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Post-Build

1.3.1.4.1 SpiChannelId

Table 18 Specification for SpiChannelId

Name	`SpiChannelId`		
Description	This is ID number assigned to SPI channel. By default SpiChannelId is set to 0, however this number is auto incremented on adding successive channels. *Note:* *- Due to Multi-core implementation physical and logical channel ids are to be generated. Logical channel ids are numbered sequentially, however in the configuration, physical memory location can be different due to assignment of channels to jobs of different cores* *- Application should always use the same number generated as per the Spi_Cfg.h file for accessing channel buffers. ex: SpiConf_SpiChannel_(x). x is derived from the name provided by the user in the configuration* *- ID - 0xFF (255) value is used as delimiter to indicate the end of channel-id list*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 254		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.2 SpiChannelType

Table 19 Specification for SpiChannelType

Name	`SpiChannelType`

(table continues...)

Table 19 (continued) Specification for SpiChannelType

Description	This parameter specifies the buffer type (External Buffer / Internal Buffer) used by the channel. IB - Channel of internal buffer type EB - Channel of external buffer type By Default buffer type is set to EB since most applications prefer using EB over IB. *Note: The value configuration class and postBuildVariantValue is deviated from AUTOSAR due to this parameter is used in generating a macro for total number of IB channels, EB channels and total IB buffer size configured for the core.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	EB: External Buffer IB: Internal Buffer		
Default value	EB		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiChannelBuffersAllowed		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.3 SpiDataWidth

Table 20 Specification for SpiDataWidth

Name	`SpiDataWidth`		
Description	This parameter specifies the width of the data to be transmitted in terms of bits. *Note: The QSPI supports the datawidth from 2 to 32 bits. Deviated from AUTOSAR 4.4.0 and AUTOSAR 4.2.2* Default value for SpiDataWidth is set to 8 since the frames provided by any application would be defined in bytes (8-bits). *Note: The value configuration class and postBuildVariantValue is deviated from AUTOSAR due to this parameter is used in generating a macro for total number of IB channels, EB channels and total IB buffer size configured for the core.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	2 - 32		
Default value	8		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 20 (continued) Specification for SpiDataWidth

Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.4 SpiDefaultData

Table 21 Specification for SpiDefaultData

Name	`SpiDefaultData`		
Description	Data to be transmitted if source buffer is defined to be NULL for Spi_SetupEB or Spi_WriteIB APIs. Defaultdata is set to '0' since this is application specific and needs to be defined as supported by external devices.		
Multiplicity	0..1	**Type**	EcucIntegerParamDef
Range	0 - 4294967295		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	TRUE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiDataWidth		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.5 SpiEbMaxLength

Table 22 Specification for SpiEbMaxLength

Name	`SpiEbMaxLength`

(table continues...)

Table 22 (continued) Specification for SpiEbMaxLength

Description	This parameter defines number of data to be transmitted for EB channel. Available only if (SpiChannelBuffersAllowed is 1 or 2) and (SpiChannelType is EB) By default SpiEBMaxLength is set to 8190 elements. For synchronous transfer, the maximum of 65535 elements can be transmitted per channel. For Asynchronous transfer: 1.For jobs with Frame based Cs is disabled, Move counter mode is used for transmission of the QSPI data, maximum of 8190 elements can be transmitted i.e. for a job containing multiple channels sum of all elements in a job cannot be greater than 8190 elements *Note:* *- If the sum of data elements from all the channels exceeds 8190, warning is reported during code generation.* *Application developer / integrator has to take care of checking if sum of all elements of EB does not exceed 8190 elements* *- In AUTOSAR 4.2.2, check for 8190 elements is applicable for Jobs for which configuration parameter SpiHwUnitSynchronous is ASYNCHRONOUS(both in Level-1 and Level-2)* *- In AUTOSAR 4.4.0, check for 8190 elements is applicable for all jobs where SpiLevelDelivered=1 or SpiLevelDelivered=2* 2.For jobs with FrameBasedCS enabled, maximum of 16383 elements can be transmitted per channel. This limitation is due to the DMA CHCFGR register TREL bit field which is written with the number of elements to be transmitted. *The range for this parameter is deviated from AUTOSAR 4.4.0 and AUTOSAR 4.2.2.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 65535		
Default value	8190		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiChannelBuffersAllowed, SpiChannelType		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.6 SpiIbNBuffers

Table 23 Specification for SpiIbNBuffers

Name	`SpiIbNBuffers`

(table continues...)

Table 23 (continued) Specification for SpiIbNBuffers

Description	Defines the size of channel buffer for IB channels. By Default size of IB buffer is chosen to be 10 elements as many data packets to SPI external devices can fit with-in this data length. *Note:* *- The value configuration class and postBuildVariantValue is deviated from AUTOSAR due to this parameter is used in generating a macro for total number of IB channels, EB channels and total IB buffer size configured for the core* *- Channel width less than 8 bit is considered as 8-bit channel, 8-bit to 15-bit (inclusive) is considered as 16-bit channel and above 16-bit is considered as 32-bit channel. So depending on data width, size of the buffer will be generated* *- Available only if (SpiChannelBuffersAllowed is 0 or 2) and (SpiChannelType is IB)* For synchronous transfer, the maximum of 65535 elements can be transmitted per channel. For Asynchronous transfer: 1. For jobs with Frame based Cs is disabled, Move counter mode is used for transmission of the QSPI data, maximum of 8190 elements can be transmitted i.e. for a job containing multiple channels sum of all elements in a job cannot be greater than 8190 elements. Application developer / integrator has to take care of checking if sum of all elements of IB does not exceed 8190 elements. *Note:* *- For AUTOSAR 4.2.2, a code generation error will be thrown if the sum of elements in a job crosses 8190. Applicable for all asynchronous sequences* *- For AUTOSAR 4.4.0, Warning will be reported if the sum of elements in a job crosses 8190. Applicable for SpiLevelDelivered= 1 and SpiLevelDelivered=2* 2. For jobs with FrameBasedCS enabled, maximum of 16383 elements can be transmitted per channel. This limitation is due to the DMA CHCFGR register TREL bit field which is written with the number of elements to be transmitted. The range for this parameter is deviated from AUTOSAR 4.4.0 and AUTOSAR 4.2.2.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 65535		
Default value	10		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiChannelBuffersAllowed, SpiChannelType		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.4.7 SpiTransferStart

Table 24 Specification for SpiTransferStart

Name	`SpiTransferStart`		
Description	This parameter specifies whether, Least Significant Bit (LSB) is transmitted first or Most Significant bit (MSB) transmitted first. By Default TransferStart is set to LSB, since most devices communicate over LSB frames.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	LSB: Transmission starts with the Least Significant Bit first MSB: Transmission starts with the Most Significant Bit first		
Default value	LSB		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5 Container: SpiChannelList

References to SPI channels and their order within the Job.

Lower multiplicity is 1 and upper multiplicity is 255(Max supported channels)
Post-Build Variant Multiplicity: FALSE
Multiplicity Configuration Class: Post-Build

1.3.1.5.1 SpiChannelAssignment

Table 25 Specification for SpiChannelAssignment

Name	`SpiChannelAssignment`		
Description	This parameter specifies the channel linked to this Job container. *Note: The value configuration class and postBuildVariantValue is deviated from AUTOSAR due to this parameter is used in generating a macro for total number of IB channels, EB channels and total IB buffer size configured for the core.*		
Multiplicity	1..1	**Type**	EcucReferenceDef
Range	Reference to Node: SpiChannel		
Default value	None		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 25 (continued) Specification for SpiChannelAssignment

Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.5.2 SpiChannelIndex

Table 26 Specification for SpiChannelIndex

Name	`SpiChannelIndex`		
Description	This parameter specifies the order of Channels within the Job. *Note: The value 255 is used as delimiter to indicate that all the channels assigned are completed.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 254		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.6 Container: SpiCsGpio

This container specifies the port pin which is used for Chip Select assertion/De-assertion.

It is applicable if SpiCsSelection is CS_VIA_GPIO and has multiplicity of 0…1.

Post-Build Variant Multiplicity: TRUE
Multiplicity Configuration Class: Post-Build

1.3.1.6.1 SpiCsGpioPinSelection

Table 27 Specification for SpiCsGpioPinSelection

Name	`SpiCsGpioPinSelection`

(table continues...)

Table 27 (continued) Specification for SpiCsGpioPinSelection

Description	This parameter specifies the pin number of the port specified by SpiCsGpioPortSelection, used as Chip select, which is activated/de-activated by the SPI driver. Enabled only if SpiEnableCs is true and SpiCsSelection is CS_VIA_GPIO *Note: The range of port pins would vary across variants, refer to respective DS to check for the exact number of port-pins. A port can contain maximum of 16 pins.* By Default the port-pin and port is set to 1, this has to be modified by application as per the Hardware mapping.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 15		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiCsSelection, SpiEnableCs		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.6.2 SpiCsGpioPortSelection

Table 28 Specification for SpiCsGpioPortSelection

Name	`SpiCsGpioPortSelection`		
Description	This parameter specifies the port number whose one of the pin is used as Chip select, which is activated/de-activated by the SPI driver. Enabled only if SpiEnableCs is true and SpiCsSelection is CS_VIA_GPIO *Note: The port number can vary from 0 to 41 depending on the device variant. Refer to respective DS to know the exact number of port available.* By Default the port-pin and port is set to 1, this has to be modified by application as per the Hardware mapping.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 41		
Default value	1		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL

(table continues...)

Table 28 (continued) Specification for SpiCsGpioPortSelection

Dependency	SpiCsSelection, SpiEnableCs
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.1.7 Container: SpiDelayParams

This container is Applicable only when the parameter SpiAutoCalcDelayParams is set to FALSE

Defines the basic configuration for the current slave select. The parameters are vendor specific for use by SPI Hardware. Default values for this container is set to obtain a minimum delay in nanoseconds.

Note: The Multiplicity for the container is 0..1

Post-Build Variant Multiplicity: TRUE

Multiplicity Configuration Class: Pre-Compile

1.3.1.7.1 SpiDelayParamIdleLength

Table 29 Specification for SpiDelayParamIdleLength

Name	`SpiDelayParamIdleLength`		
Description	Idle Delay Length Defines the length of both idle delays, IDLEA and IDLEB, in Tqspi units pre scaled with IPRE 0 represents 1 units 1 represents 2 unit ... 7 represents 8 units Applicable only when the parameter SpiAutoCalcDelayParams is set to false and CS is driven by HW engine. The default value is set to obtain a minimum delay in nanoseconds.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 7		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcDelayParams, SpiCsSelection		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.7.2 SpiDelayParamIdlePre

Table 30 Specification for SpiDelayParamIdlePre

Name	`SpiDelayParamIdlePre`		
Description	Pre-scalar for the Idle Delay Length in Tqspi units b000 represents 1 b001 represents 4 b010 represents 16 ... b111 represents 16384 Applicable only when the parameter SpiAutoCalcDelayParams is set to false and CS is driven by HW engine. The default value is set to obtain a minimum delay in nanoseconds.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 7		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcDelayParams, SpiCsSelection		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.7.3 SpiDelayParamLeadLength

Table 31 Specification for SpiDelayParamLeadLength

Name	`SpiDelayParamLeadLength`		
Description	Leading Delay Length Defines the length of the leading delay, in Tqspi units pre scaled with LPRE 0 - 1 units 1 - 2 unit ... 7 - 8 units Applicable only when the parameter SpiAutoCalcDelayParams is set to false and CS is driven by HW engine. The default value is set to obtain a minimum delay in nanoseconds.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 7		
Default value	0		

(table continues...)

Table 31 (continued) Specification for SpiDelayParamLeadLength

Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcDelayParams, SpiCsSelection		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.7.4 SpiDelayParamLeadPre

Table 32 Specification for SpiDelayParamLeadPre

Name	`SpiDelayParamLeadPre`		
Description	Prescaler for the Leading Delay Length in Tqspi units b000 represents 1 b001 represents 4 b010 represents 16 ... b111 represents 16384 Applicable only when the parameter SpiAutoCalcDelayParams is set to false and CS is driven by HW engine. The default value is set to obtain a minimum delay in nanoseconds.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 7		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcDelayParams, SpiCsSelection		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.7.5 SpiDelayParamTrailLength

Table 33 Specification for SpiDelayParamTrailLength

Name	`SpiDelayParamTrailLength`

(table continues...)

Table 33 (continued) Specification for SpiDelayParamTrailLength

Description	Trailing Delay Length Defines the length of the trailing delay, in Tqspi units pre scaled with TPRE 0 - 1 units 1 - 2 unit ... 7 - 8 units Applicable only when the parameter SpiAutoCalcDelayParams is set to false and CS is driven by HW engine. The default value is set to obtain a minimum delay in nanoseconds.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 7		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcDelayParams, SpiCsSelection		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.7.6 SpiDelayParamTrailPre

Table 34 Specification for SpiDelayParamTrailPre

Name	`SpiDelayParamTrailPre`		
Description	Prescaler for the Trailing Delay Length in Tqspi units b000 represents 1 b001 represents 4 b010 represents 16 ... b111 represents 16384 Applicable only when the parameter SpiAutoCalcDelayParams is set to false and CS is driven by HW engine.The default value is set to obtain a minimum delay in nanoseconds.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 7		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-

(table continues...)

Table 34 (continued) Specification for SpiDelayParamTrailPre

Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcDelayParams, SpiCsSelection		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.8 Container: SpiDemEventParameterRefs

Container lists the production errors supported by SPI driver. The standardized errors are provided in the container and can be extended by vendor specific error references. Multiplicity of the container is 0..1

Note: The configuration of DEM is mandatory when SpiSafetyEnable is true.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.8.1 SPI_E_HARDWARE_ERROR

Table 35 Specification for SPI_E_HARDWARE_ERROR

Name	`SPI_E_HARDWARE_ERROR`		
Description	Reference to configured DEM event to report Hardware failure. If the reference is not configured the error will not be reported.		
Multiplicity	0..1	**Type**	EcucSymbolicNameReferenceDef
Range	Reference to Node:		
Default value	NULL		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiSafetyEnable		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.9 Container: SpiDriver

This container contains the configuration parameters and sub containers of the AUTOSAR Spi module.

Multiplicity is 1..1

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.9.1 SpiMaxChannel

Table 36 Specification for SpiMaxChannel

Name	`SpiMaxChannel`		
Description	This parameter represents number of channels allocated to all the cores. It will be gathered by tools during the configuration stage and hence not editable. This parameter is visible in Tresos only if the field is added after adding all channels, however code is generated for application use. *Note:* *- When there are no channels configured, the parameter is set with value 0 and however the code generation throws an error* *- Range for this parameter is deviated from AUTOSAR 4.4.0 and AUTOSAR 4.2.2* *- The postBuildVariantValue is set to FALSE because changing the value across the variants irrespective of number of channels configured has an impact on the total number of channels configured per core* Rationale: Due to the limitation of Tresos tool, it is not possble to derive the maximum number of channels configured across the variants.		
Multiplicity	0..1	**Type**	EcucIntegerParamDef
Range	1 - 255		
Default value	Depends on Max channels added		
Post-build variant value	FALSE	**Post-build variant multiplicity**	TRUE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.9.2 SpiMaxJob

Table 37 Specification for SpiMaxJob

Name	`SpiMaxJob`

(table continues...)

Table 37 (continued) Specification for SpiMaxJob

Description	This parameter represents the total number of jobs allocated across cores. It will be gathered by tool during code generation stage and hence is not editable. This field is visible only if added after all jobs are added in system, however in code generation value is generated properly and available to application use. *Note:* *- When there are no jobs configured, the parameter is set with value 0 and however the code generation throws an error* *- Range for this parameter is deviated from AUTOSAR 4.4.0 and AUTOSAR 4.2.2* *- The postBuildVariantValue is set to FALSE because changing the value across the variants irrespective of number of jobs configured has an impact on the total number of jobs configured per core* Rationale: Due to the limitation of Tresos tool, it is not possble to derive the maximum number of jobs configured across the variants.		
Multiplicity	0..1	**Type**	EcucIntegerParamDef
Range	1 - 5000		
Default value	Depends on total jobs added		
Post-build variant value	FALSE	**Post-build variant multiplicity**	TRUE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.9.3 SpiMaxSequence

Table 38 Specification for SpiMaxSequence

Name	`SpiMaxSequence`

(table continues...)

Table 38 (continued) Specification for SpiMaxSequence

Description	This parameter represents the total number of sequence allocated across cores. It will be gathered by tool during configuration stage and hence is not editable. This parameter is visible in Tresos only if the field is added after adding all sequences in list, however value is generated in code generation for application use. *Note:* *- When there are no sequences configured, the parameter is set with value 0 and however the code generation throws an error* *- Range for this parameter is deviated from AUTOSAR 4.4.0 and AUTOSAR 4.2.2* *- The postBuildVariantValue is set to FALSE because changing the value across the variants irrespective of number of sequences configured has an impact on the total number of sequences configured per core* Rationale: Due to the limitation of Tresos tool, it is not possble to derive the maximum number of sequences configured across the variants.		
Multiplicity	0..1	**Type**	EcucIntegerParamDef
Range	1 - 255		
Default value	Depends on total sequences added		
Post-build variant value	FALSE	**Post-build variant multiplicity**	TRUE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.9.4 SpiSystemclock

Table 39 Specification for SpiSystemclock

Name	`SpiSystemclock`		
Description	This parameter refers to the system clock configured by MCU driver. It could refer to Fspb or Fqspi . This reference is used for BaudRate computation Reference to parameter of type McuClockSettingConf		
Multiplicity	1..1	**Type**	EcucReferenceDef
Range	Reference to Node: McuClockReferencePointConfig		
Default value	None		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-

(table continues...)

Table 39 (continued) Specification for SpiSystemclock

Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	McuClockSettingConfig		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10 Container: SpiExternalDevice

This container contains the configuration parameters to describe the external device (slave) properties. This container is attached/referenced to a job.

Multiplicity of the container is 1 to many.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.10.1 SpiAutoCalcBaudParams

Table 40 Specification for SpiAutoCalcBaudParams

Name	`SpiAutoCalcBaudParams`		
Description	If the parameter is set to TRUE, then the configuration tool will automatically generate the Baudrate parameters (TQ, Q, A, B, C) based on the parameter SpiBaudrate. True: Automatically calculate the Baudrate parameters False: Manually enter the Baudrate parameters By default value is set to true to calculate the register value for ECON and GLOBALCON register of QSPI to get the right baudrate as configured by the application.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.2 SpiAutoCalcDelayParams

Table 41 Specification for SpiAutoCalcDelayParams

Name	`SpiAutoCalcDelayParams`		
Description	If the parameter is set to TRUE, then the configuration tool will automatically generate the delay parameters (IPRE, IDLE, LPRE, LEAD, TPRE, TRAIL) based on the parameters SpiIdleTime, SpiTimeClk2CS, SpiTrailingTime. True: Automatically calculate the delay parameters False: Manually enter the Delay parameters By default value is set to true to calculate the delay parameters automatically during code generation for activation of chipselect.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.3 SpiBaudrate

Table 42 Specification for SpiBaudrate

Name	`SpiBaudrate`		
Description	This parameter defines the QSPI communication baudrate. By default the baudrate is set to 640kHz to support wide spread of devices, however this values are to be modified as per the application need and speed supported. Error: Configuration error is reported, if the baudrate value is incorrect for the given frequency(Fqspi). *Note:* *The HW supports baudrate until 50MHz(for simplex communication) and the maximum baudrate supported in Full duplex is upto 33MHz. Hence the Baudrate is limited to maximum value supported. This is a deviation from AUTOSAR, since the high limit is set to Infinity and low limit is set to 0Hz as per AUTOSAR.*		
Multiplicity	1..1	**Type**	EcucFloatParamDef

(table continues...)

Table 42 (continued) Specification for SpiBaudrate

Range	9600 - 33 MHz		
Default value	640000		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiAutoCalcBaudParams		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.4 SpiCsIdentifier

Table 43 Specification for SpiCsIdentifier

Name	`SpiCsIdentifier`		
Description	This parameter specifies the Chip Select (CS) for the hardware specified by SpiHwUnit. Can range from Channel0 (0) to Channel15 (15). By Default SpiCsIdentifier is set to CHANNEL0. *Note: This parameter is deviated from AUTOSAR and its type is Enumeration instead of String.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	CHANNEL0: CHANNEL10: CHANNEL11: CHANNEL12: CHANNEL13: CHANNEL14: CHANNEL15: CHANNEL1: CHANNEL2: CHANNEL3: CHANNEL4: CHANNEL5: CHANNEL6: CHANNEL7: CHANNEL8: CHANNEL9:		
Default value	CHANNEL0		

(table continues...)

Table 43 (continued) Specification for SpiCsIdentifier

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.5 SpiCsPolarity

Table 44 Specification for SpiCsPolarity

Name	`SpiCsPolarity`		
Description	This parameter defines the active polarity of chip select. By Default SpiCsPolarity is set to LOW to support wide variety of devices.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	HIGH: LOW:		
Default value	LOW		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.6 SpiCsSelection

Table 45 Specification for SpiCsSelection

Name	`SpiCsSelection`
Description	Indicates if the chip select is either through GPIO (Driven from the SPI driver) or SLSO (Driven by the QSPI Hardware). By Default SpiCsSelection is set to CS_VIA_PERIPHERAL_ENGINE so that using SLSO is the recommended mechanism to have better performance.

(table continues...)

Table 45 (continued) Specification for SpiCsSelection

Multiplicity	0..1	**Type**	EcucEnumerationParamDef
Range	CS_VIA_GPIO: Chip select handled via GPIO by SPI driver. CS_VIA_PERIPHERAL_ENGINE: Chip select is handled via Peripheral Hardware engine.		
Default value	CS_VIA_PERIPHERAL_ENGINE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	TRUE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiEnableCs		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.7 SpiDataShiftEdge

Table 46 Specification for SpiDataShiftEdge

Name	`SpiDataShiftEdge`		
Description	The data can be shifted on either leading edge or on trailing edge of the shift clock. This parameter defines the data shift with leading or trailing edge. *Note: The default value is configured as LEADING edge.*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	LEADING: First data shift edge is a leading edge TRAILING: First data shift edge is a trailing edge		
Default value	LEADING		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.8 SpiDeviceEcucPartitionRef

Table 47 Specification for SpiDeviceEcucPartitionRef

Name	`SpiDeviceEcucPartitionRef`		
Description	Parameter maps an SPI external device to zero or multiple ECUC partitions to limit the access to this external device. The ECUC partitions referenced are a subset of the ECUC partitions where the SPI driver is mapped to. *Note: Parameter support is added only for AUTOSAR schema compliance, this parameter is not used in code generation logic, hence this parameter is made editable false.*		
Multiplicity	0..*	**Type**	EcucReferenceDef
Range	Reference to Node:		
Default value	NULL		
Post-build variant value	TRUE	**Post-build variant multiplicity**	TRUE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.10.9 SpiEnableCs

Table 48 Specification for SpiEnableCs

Name	`SpiEnableCs`		
Description	This parameter specifies the chip select handling functions. If this parameter is enabled then parameter SpiCsSelection further details the type of chip selection. False: No Chip select is enabled True: Chip select is enabled By Default SpiEnableCs is set to False.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-

(table continues...)

Table 48 (continued) Specification for SpiEnableCs

Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.10 SpiHwUnit

Table 49 Specification for SpiHwUnit

Name	`SpiHwUnit`		
Description	This parameter specifies the SPI hardware microcontroller peripheral allocated for transmission. QSPI0, QSPI1, QSPI2, QSPI3, QSPI4 and QSPI5 kernels can be configured. By Default value of this parameter is set to QSPI0. *Note:* *- As per AUTOSAR this parameter is defined as enum with names CSIBx, however to match the AURIX Hardware capability QSPIx is defined to replace CSIBx* *- The value configuration class and postBuildVariantValue is deviated from AUTOSAR, as this parameter is used in generating a macro for total number of IB channels, EB channels and total IB buffer size configured for the core*		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	QSPI0: QSPI Kernel 0 QSPI1: QSPI Kernel 1 QSPI2: QSPI Kernel 2 QSPI3: QSPI Kernel 3 QSPI4: QSPI Kernel 4 QSPI5: QSPI Kernel 5		
Default value	QSPI0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.11 SpiIdleTime

Table 50 Specification for SpiIdleTime

Name	`SpiIdleTime`		
Description	This parameter IDLEA/IDLEB time is the QSPI hardware delay after which SLSO will be activated by Hardware. The default value is chosen to support most of the baudrate configurations.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	0.00000004 - 0.098304		
Default value	0.0000001		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcDelayParams, SpiCsSelection		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.12 SpiParitySupport

Table 51 Specification for SpiParitySupport

Name	`SpiParitySupport`		
Description	This parameter indicates whether the parity feature to be enabled in Hardware or not. Default value for this parameter is set to UNUSED to support wide varieties of external devices.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	EVEN: Parity Bit added during Transmission to make the data as Even Parity ODD: Parity Bit is added during Transmission to make the data as Odd Parity UNUSED: Parity not configured		
Default value	UNUSED		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.13 SpiShiftClockIdleLevel

Table 52 Specification for SpiShiftClockIdleLevel

Name	`SpiShiftClockIdleLevel`		
Description	This parameter defines the idle level of shift clock. The idle level of the shift clock can be configured to be idle level low or idle level high. By default SpiShiftClockIdlelevel is set to LOW to support wide range of application.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	HIGH: Shift clock idle level is a high voltage level LOW: Shift Clock idle level is low		
Default value	LOW		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.14 SpiTimeClk2Cs

Table 53 Specification for SpiTimeClk2Cs

Name	`SpiTimeClk2Cs`		
Description	This parameter is the minimum time (in seconds) between clock and chip select. This parameter is used to calculate the QSPI Hardware lead delay parameters LPRE and LEAD.The default value is chosen to support most of the baudrate configurations. Enabled only when the parameter SpiAutoCalcDelayParams is set to TRUE. Provide configuration error if the delay value is incorrect for the given frequency(Fqspi). *Note:* *- This value is set to default when the CS_VIA_GPIO is selected as the option since the timing cannot be guaranteed when GPIO is used as chip select* *- Range for this parameter is deviated from AUTOSAR 4.2.2 and AUTOSAR 4.4.0* Min value of this parameter is derived through the capability of AURIX Hardware and derived based on the clock that can be supplied to peripheral, note that parameter values are deviated from AUTOSAR defined values.		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	0.00000004 - 0.0001		

(table continues...)

Table 53 (continued) Specification for SpiTimeClk2Cs

Default value	0.0000001		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiAutoCalcDelayParams, SpiCsSelection		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.10.15 SpiTrailingTime

Table 54 Specification for SpiTrailingTime

Name	`SpiTrailingTime`		
Description	Delay expected at the trailing phase of data transmission. This field introduces the delay after every data element transmission. Applicable only when the parameter SpiAutoCalcDelayParams is set to true. This parameter is IFX specific to make use of Hardware capability. The default value is chosen to support most of the baudrate configurations. Error: Provide configuration error if the delay value is incorrect for the given frequency(Fqspi)		
Multiplicity	1..1	**Type**	EcucFloatParamDef
Range	0.00000004 - 0.098304		
Default value	0.0000001		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiAutoCalcDelayParams, SpiCsSelection		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11 Container: SpiGeneral

General configuration settings for SPI-Handler

Multiplicity is 1..1

Post-Build Variant Multiplicity: -

Multiplicity Configuration Class: -

1.3.1.11.1 SpiCancelApi

Table 55 Specification for SpiCancelApi

Name	`SpiCancelApi`		
Description	This parameter specifies the availability of the API Spi_Cancel. This parameter is realized as, #define SPI_CANCEL_API (STD_ON / STD_OFF) True: Spi_Cancel API is available False: Spi_Cancel API is not available By Default this feature is disabled, must be enabled if application demands to cancel a sequence at runtime.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.2 SpiChannelBuffersAllowed

Table 56 Specification for SpiChannelBuffersAllowed

Name	`SpiChannelBuffersAllowed`		
Description	This parameter specifies the type of buffers available for the user This parameter is realized as, #define SPI_CHANNEL_BUFFERS_ALLOWED (0U / 1U / 2U) 0 - Only internal buffers are selected in handler/driver, 1 - Only external buffers are selected in handler/driver, 2 - Both internal and external buffers are selected in handler/driver. By Default value of this parameter is set to 1 to support EB by default since widely used by application.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 2		
Default value	1		

(table continues...)

Table 56 (continued) Specification for SpiChannelBuffersAllowed

Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.3 SpiDevErrorDetect

Table 57 Specification for SpiDevErrorDetect

Name	`SpiDevErrorDetect`		
Description	This parameter enables/disables development error detections. The default value of this parameter is set to FALSE to minimize the executable code size.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.4 SpiEcucPartitionRef

Table 58 Specification for SpiEcucPartitionRef

Name	`SpiEcucPartitionRef`		
Description	Parameter maps the SPI driver to zero or multiple ECUC partitions to make the driver API available in the according partition. *Note: Parameter support is added only for AUTOSAR schema compliance, this parameter is not used in code generation logic, hence this parameter is made editable false.*		
Multiplicity	0..*	**Type**	EcucReferenceDef

(table continues...)

Table 58 (continued) Specification for SpiEcucPartitionRef

Range	Reference to Node:		
Default value	NULL		
Post-build variant value	TRUE	**Post-build variant multiplicity**	TRUE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.11.5 SpiEnableLoopBackApi

Table 59 Specification for SpiEnableLoopBackApi

Name	`SpiEnableLoopBackApi`		
Description	Switches the Spi_ControlLoopBack function ON or OFF. The default value of this parameter is set to FALSE to minimize the executable code size.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.6 SpiHwStatusApi

Table 60 Specification for SpiHwStatusApi

Name	`SpiHwStatusApi`

(table continues...)

Table 60 (continued) Specification for SpiHwStatusApi

Description	This parameter specifies whether API Spi_GetHWUnitStatus is available or not. This parameter is realized as, #define SPI_HW_STATUS_API (STD_ON / STD_OFF) By Default this feature is disabled, must be enabled if application demands to know individual status of hardware.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.7 SpiInitCheckApi

Table 61 Specification for SpiInitCheckApi

Name	`SpiInitCheckApi`		
Description	Switches the Spi_InitCheck () API ON or OFF. By Default this feature is disabled, must be enabled if application demands safety features and needs to verify initialization sequence.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.8 SpiInitDeInitApiMode

Table 62 Specification for SpiInitDeInitApiMode

Name	`SpiInitDeInitApiMode`		
Description	This configuration parameter gives the mode in which Spi_Init and Spi_deinit API will be used. If this parameter is configured to SPI_MCAL_SUPERVISOR then Spi module directly writes to SFRs without using OS function. By Default mode is set to SPI_MCAL_SUPERVISOR since driver code executes in supervisor mode in most cases.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	SPI_MCAL_SUPERVISOR: SPI_MCAL_USER1:		
Default value	SPI_MCAL_SUPERVISOR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.9 SpiInterruptibleSeqAllowed

Table 63 Specification for SpiInterruptibleSeqAllowed

Name	`SpiInterruptibleSeqAllowed`		
Description	This parameter specifies whether interruptible sequences are allowed or not, if this field is STD_OFF and a sequence is defined to be interruptible, such sequence would be treated as non-interruptible. Significant only if SpiLevelDelivered is 1 or 2 This parameter is realized as, #define SPI_INTERRUPTIBLE_SEQ_ALLOWED (STD_ON / STD_OFF) By Default this feature is disabled since having this feature will have slight performance impact due to sorting of queue involved at runtime and many applications do not need to have this feature enabled.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		

(table continues...)

Table 63 (continued) Specification for SpiInterruptibleSeqAllowed

Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiLevelDelivered		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.10 SpiKernelEcucPartitionRef

Table 64 Specification for SpiKernelEcucPartitionRef

Name	`SpiKernelEcucPartitionRef`		
Description	Parameter Maps the SPI kernel to zero or one ECUC partitions to assign the driver kernel to a certain core. The ECUC partition referenced is a subset of the ECUC partitions where the SPI driver is mapped to. *Note: Parameter support is added only for AUTOSAR schema compliance, this parameter is not used in code generation logic, hence this parameter is made editable false.*		
Multiplicity	0..1	**Type**	EcucReferenceDef
Range	Reference to Node:		
Default value	NULL		
Post-build variant value	TRUE	**Post-build variant multiplicity**	TRUE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.11.11 SpiLevelDelivered

Table 65 Specification for SpiLevelDelivered

Name	`SpiLevelDelivered`

(table continues...)

Table 65 (continued) Specification for SpiLevelDelivered

Description	This parameter is to select the type of communication driver has to support. L0 - Synchronous transmission (No DMA usage, blocking call) L1 - Asynchronous communication configuring related interrupts (DMA used, non-blocking) L2 - Handles both Synchronous and Asynchronous communication. For asynchronous transmissions polling and Interrupt modes are supported. (DMA used for asynchronous communication, non-blocking) This parameter is realized as, #define SPI_LEVEL_DELIVERED (0 / 1 / 2) By Default value of this parameter is set to 1 to Level-1 asynchronous communication.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 2		
Default value	1		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.12 SpiMainFunctionPeriod

Table 66 Specification for SpiMainFunctionPeriod

Name	`SpiMainFunctionPeriod`		
Description	This parameter defines the interval in which application has to call Spi_MainFunction_Handling. This function is used by the upper layer / application. The macro SPI_MAIN_FUNCTION_PERIOD is generated only for Level-2 communication. For Level-0 and Level-1, the macro is not generated even though some value is configured for the parameter. Default value for polling is set to 10milliseconds to support wide applications. *Note: Range for this parameter is deviated from AUTOSAR 4.4.0.*		
Multiplicity	0..1	**Type**	EcucFloatParamDef
Range	0.0000001 second - 1 second		
Default value	0.01 second		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE

(table continues...)

Table 66 (continued) Specification for SpiMainFunctionPeriod

Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiLevelDelivered		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.13 SpiMultiCoreErrorDetect

Table 67 Specification for SpiMultiCoreErrorDetect

Name	`SpiMultiCoreErrorDetect`		
Description	This parameter enables or disables the Multicore related default error tracer (DET) detection and reporting. It is applicable only when DETs are enabled. When set to TRUE, detection and reporting of multi-core related errors is enabled. The default value of this parameter is set to FALSE to minimize the executable code size.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiDevErrorDetect		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.14 SpiRunTimeErrorDetect

Table 68 Specification for SpiRunTimeErrorDetect

Name	`SpiRunTimeErrorDetect`
Description	This parameter enables or disables the Runtime errors reporting. When this parameter is set to TRUE, this enables the runtime errors reporting. The default value of this parameter is set to TRUE to ensure the runtime error detection during the product lifecycle. *Note: When SpiSafetyEnable is TRUE, this parameter must be set to TRUE.*

(table continues...)

Table 68 (continued) Specification for SpiRunTimeErrorDetect

Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiSafetyEnable		
Autosar Version	Applicable for Autosar version 4.4.0.		

1.3.1.11.15 SpiRuntimeApiMode

Table 69 Specification for SpiRuntimeApiMode

Name	`SpiRuntimeApiMode`		
Description	This configuration parameter gives the mode in which the Runtime API will be used. If this parameter is configured to SPI_MCAL_SUPERVISOR then Spi module directly writes to SFRs without using OS function. By Default mode is set to SPI_MCAL_SUPERVISOR since driver code executes in supervisor mode in most cases.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	SPI_MCAL_SUPERVISOR: SPI_MCAL_USER1:		
Default value	SPI_MCAL_SUPERVISOR		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiInitDeInitApiMode		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.16 SpiSafetyEnable

Table 70 Specification for SpiSafetyEnable

Name	`SpiSafetyEnable`		
Description	Enables / disables safety related checks. The detection of safety related errors is enabled, by default, to ensure that safety issues are addressed during the product lifecycle. #define SPI_SAFETY_ENABLE (STD_ON / STD_OFF)		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	TRUE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.17 SpiSupportConcurrentSyncTransmit

Table 71 Specification for SpiSupportConcurrentSyncTransmit

Name	`SpiSupportConcurrentSyncTransmit`		
Description	This parameter specifies whether concurrent synchronous transmission is allowed or not. When SpiSupportConcurrentSyncTransmit is TRUE: - If a QSPI is busy in transmission, then transmission request on the same kernel is blocked - If a QSPI is busy in transmission, then transmission request on the other kernels(configured to same core) will be accepted When SpiSupportConcurrentSyncTransmit is FALSE: If atleast one QSPI(configured to same core) is busy in transmission, the parallel requests on same or other kernels will be blocked. *Note: This parameter is available when SpiLevelDelivered is 0 or 2, for synchronous communication.* The default value of this parameter is set to FALSE. It must be set to TRUE, if application demands to transmit synchronous sequences on multiple QSPI at same time.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef

(table continues...)

Table 71 (continued) Specification for SpiSupportConcurrentSyncTransmit

Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiLevelDelivered		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.18 SpiSyncTransmittimeoutDuration

Table 72 Specification for SpiSyncTransmittimeoutDuration

Name	`SpiSyncTransmittimeoutDuration`		
Description	The parameter is used as timeout loop counter during synchronous transmission while waiting for data reception after transmission. Value is user configurable and can be changed as per the need of application. Timeout value is generated as part of root configuration accessible by all cores. Dependent on SpiLevelDelivered for L0 and L2.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0x64 - 0xFFFFFFFF		
Default value	0xFFFF		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiLevelDelivered		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.11.19 SpiUserCallbackHeaderFile

Table 73 Specification for SpiUserCallbackHeaderFile

Name	`SpiUserCallbackHeaderFile`

(table continues...)

Table 73 (continued) Specification for SpiUserCallbackHeaderFile

Description	Header file name which will be included by the Spi. This parameter value must not represent a path. *Note: The default value is only a representational value, this needs to be edited as per application need.*		
Multiplicity	0..*	**Type**	EcucStringParamDef
Range	String		
Default value	Spi_UserDefined_Cbk.h		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar version 4.2.2.		

1.3.1.11.20 SpiVersionInfoApi

Table 74 Specification for SpiVersionInfoApi

Name	`SpiVersionInfoApi`		
Description	Pre-processor switch to enable / disable the API to read out the driver version information The parameter is realized as, #define SPI_VERSION_INFO_API (STD_ON / STD_OFF) By Default this feature is disabled, must be enabled if application demands to know the version of driver at runtime.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12 Container: SpiHwConfiguration

Hw configuration for QSPI and multiplicity is 1..1

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.12.1 SpiExternalDemux

Table 75 Specification for SpiExternalDemux

Name	`SpiExternalDemux`		
Description	This parameter enables/disables the Spi External Demultiplexer. In the External Demultiplexer mode, SLSO1 to SLSO4 are used to drive the QSPI HW channel (0 to 15) and SLSO0 is used to as strobe signal in order to ensure glitch free selection. The default value of this parameter is set to FALSE.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiHwConfigKernel		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.2 SpiHWPinMRSTQspix

Table 76 Specification for SpiHWPinMRSTQspix

Name	`SpiHWPinMRSTQspix`		
Description	Port Pin selection for Master Receive Slave Transmit. Refer DS for the list of pins applicable for specific QSPI, format of pin description is as below: QSPIx_MRSTy x - represents 0 to 6 based on the AURIX variant y - represents A, B, C, DN, DP, CN Respective Alt-x function to be selected from the configuration. This parameter is IFX specific to make use of Hardware provided capability for selecting the right MRST pins.		
Multiplicity	1..1	**Type**	EcucStringParamDef

(table continues...)

Table 76 (continued) Specification for SpiHWPinMRSTQspix

Range	String		
Default value	Depends on Micro variant		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiHwConfigKernel		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.3 SpiHwConfigKernel

Table 77 Specification for SpiHwConfigKernel

Name	`SpiHwConfigKernel`		
Description	This parameter is the symbolic name to identify the Kernel ID configuration This parameter is IFX specific to list all SPI kernels, by default set to QSPI0 needs to be modified as per the Hardware mapped.		
Multiplicity	1..1	**Type**	EcucEnumerationParamDef
Range	QSPI0: QSPI Kernel 0 QSPI1: QSPI Kernel 1 QSPI2: QSPI Kernel 2 QSPI3: QSPI Kernel 3 QSPI4: QSPI Kernel 4 QSPI5: QSPI Kernel 5		
Default value	QSPI0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.4 SpiJobQueueLengthQspix

Table 78 Specification for SpiJobQueueLengthQspix

Name	`SpiJobQueueLengthQspix`		
Description	This parameter specifies the maximum jobs that can be held in Queue for transmission at a time, once a job is transmitted location is freed-up for successive sequence. This Macro is generated for every QSPI kernel configured so that each kernel is independent and transmission happen parallel. *Note1:* *- Significant only for SpiLevelDelivered is 1 or 2* *- The SPI Job queue and the sequence queue are the circular queue. Ideally the queue size should be maximum jobs configured* The Syntax of the generated parameter for default value is as below: #define SPI_JOB_QUEUE_LENGTH_QSPIx (2U) If jobs are added for transmission and a cancel API is called on a sequence, if jobs are placed as per priority, jobs related to sequence is skipped and the location are not freed up in this special case. *Note2: For AUTOSAR 4.4.0, Configuration of this parameter is mandatory in both Level-1 and Level-2.*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 65535		
Default value	2		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiLevelDelivered		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.5 SpiSLSO0StrobeDelay

Table 79 Specification for SpiSLSO0StrobeDelay

Name	`SpiSLSO0StrobeDelay`		
Description	This parameter is used to configure the strobe delay. In order to ensure glitch free selection, a strobe signal is provided, driven at SLSO0 pin. This signal is delayed relative to the SLSO1 to SLSO4 signals for LS (Lead_Strobe) and TS (Trail_Strobe) delays(in TQ time units). Default value is set to minimum possible value.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef

(table continues...)

Table 79 (continued) Specification for SpiSLSO0StrobeDelay

Range	2 - 31		
Default value	2		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	SpiExternalDemux, SpiHwConfigKernel		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.12.6 SpiSleepEnableQspix

Table 80 Specification for SpiSleepEnableQspix

Name	`SpiSleepEnableQspix`		
Description	Disable/enable entering into sleep mode upon sleep request from MCU. This parameter is used during spi initialization to configure the CLC register. True: QSPIx enters sleep mode upon sleep request from MCU False: QSPIx does not enter sleep mode upon sleep request from MCU This parameter is IFX specific to make use of Hardware provided capability. By default this feature is disabled and must be enabled if application demands.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.13 Container: SpiHwDmaConfigurationQspi

Contains the references of DMA channels for QSPI Tx and Rx channels in configuration.

This parameter is IFX specific to make use of Hardware provided capability to configure the associated DMA channels for SPI.

Multiplicity is 0..1

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.13.1 SpiHwDmaChannelReceptionRef

Table 81 Specification for SpiHwDmaChannelReceptionRef

Name	`SpiHwDmaChannelReceptionRef`		
Description	This parameter refers to the DmaConfiguration. Channel is a reference field available through configured list of channels in DMA module. Available only if SpiLevelDelivered is 1 or 2.		
Multiplicity	0..1	**Type**	EcucReferenceDef
Range	Reference to Node: SpiHwDmaConfigurationQspi		
Default value	None		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile
Origin	IFX	**Scope**	LOCAL
Dependency	SpiLevelDelivered		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.13.2 SpiHwDmaChannelTransmissionRef

Table 82 Specification for SpiHwDmaChannelTransmissionRef

Name	`SpiHwDmaChannelTransmissionRef`		
Description	This parameter refers to the DmaConfiguration. Channel is a reference field available through configured list of channels in DMA module. Available only if SpiLevelDelivered is 1 or 2.		
Multiplicity	0..1	**Type**	EcucReferenceDef
Range	Reference to Node: SpiHwDmaConfigurationQspi		
Default value	None		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Pre-Compile	**Multiplicity configuration class**	Pre-Compile

(table continues...)

Table 82 (continued) Specification for SpiHwDmaChannelTransmissionRef

Origin	IFX	**Scope**	LOCAL
Dependency	SpiLevelDelivered		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14 Container: SpiJob

This container contains the configuration parameters to describe a job.

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

Lower Multiplicity is 1 and upper multiplicity is 5000.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

1.3.1.14.1 SpiDeviceAssignment

Table 83 Specification for SpiDeviceAssignment

Name	`SpiDeviceAssignment`		
Description	This parameter is a reference parameter to the external device container. *Note: In AUTOSAR 4.2.2, if the same External Device assigned for both Synchronous and Asynchronous Jobs, a configuration error is reported during code generation.*		
Multiplicity	1..1	**Type**	EcucReferenceDef
Range	Reference to Node: SpiExternalDevice		
Default value	None		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.2 SpiFrameBasedCS

Table 84 Specification for SpiFrameBasedCS

Name	`SpiFrameBasedCS`

(table continues...)

Table 84 (continued) Specification for SpiFrameBasedCS

Description	If Frame based CS feature is enabled, It asserts and de-asserts the SLSO for every data element being transferred through SPI interface. If Frame based CS feature is disabled, the assertion of SLSO happens at the start of job transmission and de-assertion of SLSO happens at the end of job transmission. *Note: Frame based CS is allowed to enable only when SpiCsSelection = CS_VIA_PERIPHERAL_ENGINE.* This parameter is IFX specific to make use of Hardware provided capability. By Default this feature is disabled since this is application specific and not frequently used.		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	IFX	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.3 SpiHwUnitSynchronous

Table 85 Specification for SpiHwUnitSynchronous

Name	`SpiHwUnitSynchronous`

(table continues...)

Table 85 (continued) Specification for SpiHwUnitSynchronous

Description	If SpiHwUnitSynchronous is set to SYNCHRONOUS, then the job is used in a synchronous manner and vice versa. Since AUTOSAR requirement says to pre-assigned SPI buses required for transmission, this parameter is created to pre-assign the available SPI bus to Sync or Async. This parameter is mandatory in Level-2 configuration where as it is optional in Level-0 and Level-1. - In case, if the configuration parameter is added in Level-0, then the value for the configuration parameter must be selected as Synchronous. - In case, if the configuration parameter is added in Level-1, then the value for the configuration parameter must be selected as Asynchronous. A configuration error is reported in case of inappropriate value. By default value is set to ASYNCHRONOUS since most application use the L2 configuration and use asynchronous transfers. *Note: The value configuration class, postBuildVariantValue and postBuildVariantMultiplicity is deviated from AUTOSAR due to this parameter is used for generating the macro which indicates the type of communication is allowed on QSPI HW.*		
Multiplicity	0..1	**Type**	EcucEnumerationParamDef
Range	ASYNCHRONOUS: SYNCHRONOUS:		
Default value	ASYNCHRONOUS		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiLevelDelivered		
Autosar Version	Applicable for Autosar version 4.2.2.		

1.3.1.14.4 SpiJobEndNotification

Table 86 Specification for SpiJobEndNotification

Name	`SpiJobEndNotification`

(table continues...)

Table 86 (continued) Specification for SpiJobEndNotification

Description	This parameter defines the notification function name or function address. In case of function name, a function by this name must be defined in an application and will be called at end of job transmission. Significant if SpiLevelDelivered is 1 or 2. *Note:* *- In Level-2, notification will be provided only for asynchronous communications as the configuration of notification is enabled by default. This is applicable for 4.4.0* *- This configuration parameter depends on SpiHwUnitSynchronous, only in AUTOSAR 4.2.2* *- The Spi driver does not validate the configured function name or address for correctness and the responsibility is on the user* *- If SpiJobEndNotification is enabled, the value for the configuration parameter cannot be left blank. It should always contain a valid identifier name(Following C identifier naming rules) or it can hold a integer value or it can have a default value NULL_PTR*		
Multiplicity	0..1	**Type**	EcucFunctionNameDef
Range	String		
Default value	NULL_PTR		
Post-build variant value	TRUE	**Post-build variant multiplicity**	TRUE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiHwUnitSynchronous, SpiLevelDelivered		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.5 SpiJobId

Table 87 Specification for SpiJobId

Name	`SpiJobId`

(table continues...)

Table 87 (continued) Specification for SpiJobId

Description	This ID is assigned to job. *Note:* *- Due to Multi-core implementation physical and logical job ids are to be generated. Logical job ids are numbered sequentially, however in the configuration physical memory location can be different due to assignment of jobs to sequences of different cores* *- Application should always use the same job id generated as per the Spi_Cfg.h file for accessing job information. ex: SpiConf_SpiJobs_(x). x is derived from the name provided by the user in the configuration* *- ID - 0xFFFF (65535) value is used as delimiter to indicate the end of job-id list* By Default value of ID is set to '0' however the value is auto-incremented if more than one job is added.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 4999		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.14.6 SpiJobPriority

Table 88 Specification for SpiJobPriority

Name	`SpiJobPriority`		
Description	This parameter defines the priority of a job. 0 - lowest, 3 - highest priority By default priority of all jobs are set to '0' so that all jobs are scheduled as round-robin.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 3		
Default value	0		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL

(table continues...)

Table 88 (continued) Specification for SpiJobPriority

Dependency	-
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.1.15 Container: SpiPublishedInformation

Post-Build Variant Multiplicity: -
Multiplicity Configuration Class: -

1.3.1.15.1 SpiMaxHwUnit

Table 89 Specification for SpiMaxHwUnit

Name	`SpiMaxHwUnit`		
Description	Total QSPI IP kernels available in the selected resource. This value would change based on the microcontroller variant.		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	1 - 6		
Default value	Depends on the Hardware variant		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Published-Information	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16 Container: SpiSequence

This container contains the configuration parameters to describe a sequence.

A sequence must contain at-least 1 job, if it contains more than one, all Jobs contained have the same Sequence properties during transmission and will be linked together statically.

Lower multiplicity is 1 and upper multiplicity is 255.

Post-Build Variant Multiplicity: FALSE
Multiplicity Configuration Class: Pre-Compile

1.3.1.16.1 SpiInterruptibleSequence

Table 90 Specification for SpiInterruptibleSequence

Name	`SpiInterruptibleSequence`

(table continues...)

Table 90 (continued) Specification for SpiInterruptibleSequence

Description	If feature is enabled, jobs are transmitted based on the priority of jobs across multiples sequences. Significant only if SpiLevelDelivered is 1 or 2 and if SpiInterruptibleSequenceAllowed is true. True: Sequence could be interrupted False: Sequence is not interrupted By Default this feature is disabled so this field is set to false. *Note: This parameter has a dependency on configuration parameter SpiHwUnitSynchronous only in AUTOSAR 4.2.2.*		
Multiplicity	1..1	**Type**	EcucBooleanParamDef
Range	TRUE FALSE		
Default value	FALSE		
Post-build variant value	TRUE	**Post-build variant multiplicity**	-
Value configuration class	Post-Build	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiHwUnitSynchronous, SpiInterruptibleSeqAllowed, SpiLevelDelivered		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.2 SpiJobAssignment

Table 91 Specification for SpiJobAssignment

Name	`SpiJobAssignment`		
Description	This parameter should reference to a list of jobs. Jobs have priorities assigned. Jobs linked in a Sequence must have decreasing priorities. That means the first Job must have the highest priority of all Jobs within the Sequence *Note1: The postBuildVariantMultiplicity is set to FALSE because this parameter is used to generate macro for total number of sequences configured per core.* Error: A sequence cannot have Jobs on different QSPI Hardware modules otherwise configuration error is shown to rectify the same. *Note2: If the jobs assigned are mapped to both Asynchronous and Synchronous bus then configuration error will be reported.This is applicable only for AUTOSAR 4.2.2.*		
Multiplicity	1..*	**Type**	EcucReferenceDef
Range	Reference to Node: SpiJob		

(table continues...)

Table 91 (continued) Specification for SpiJobAssignment

Default value	None		
Post-build variant value	FALSE	**Post-build variant multiplicity**	FALSE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.3 SpiSeqEndNotification

Table 92 Specification for SpiSeqEndNotification

Name	`SpiSeqEndNotification`		
Description	This parameter defines the notification function name or function address called after sequence transmission. A function by this type must be defined in an application and defined in Tresos. Significant only if SpiLevelDelivered is 1 or 2. *Note:* *- In Level-2, notification will be provided only for asynchronous communications as the configuration of notification is enabled by default. This is applicable for 4.4.0* *- The Spi driver does not validate the configured function name or address for correctness and the responsibility is on the user* *- If SpiSeqEndNotification is enabled, the value for the configuration parameter cannot be left blank. It should always contain a valid identifier name(Following C identifier naming rules) or it can hold an integer value or it can have a default value NULL_PTR* *- This parameter has a dependency on SpiHwUnitSynchronous configuration parameter only in AUTOSAR 4.2.2*		
Multiplicity	0..1	**Type**	EcucFunctionNameDef
Range	String		
Default value	NULL_PTR		
Post-build variant value	TRUE	**Post-build variant multiplicity**	TRUE
Value configuration class	Post-Build	**Multiplicity configuration class**	Post-Build
Origin	AUTOSAR_ECUC	**Scope**	LOCAL
Dependency	SpiHwUnitSynchronous, SpiLevelDelivered		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.1.16.4 SpiSequenceId

Table 93 Specification for SpiSequenceId

Name	`SpiSequenceId`		
Description	This ID is assigned to sequence. Default value is set to '0', however on adding multiple sequences id is automatically incremented. *Note:* *- Due to Multi-core implementation physical and logical sequence ids are to be generated. Logical job ids are numbered sequentially, however in the configuration physical memory location can be different due to assignment of sequence to different cores / kernels* *- Application should always use the same sequence id generated as per the Spi_Cfg.h file for accessing sequence information. ex: SpiConf_SpiSequence_(x). x is derived from the name provided by the user in the configuration* *- 0xFF (255) is used as delimiter, this value cannot be used for ID*		
Multiplicity	1..1	**Type**	EcucIntegerParamDef
Range	0 - 254		
Default value	0		
Post-build variant value	FALSE	**Post-build variant multiplicity**	-
Value configuration class	Pre-Compile	**Multiplicity configuration class**	-
Origin	AUTOSAR_ECUC	**Scope**	ECU
Dependency	-		
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.2 Functions - Type definitions

This section lists all the data types of the SPI driver.

1.3.2.1 Spi_AsyncModeType

Table 94 Specification for Spi_AsyncModeType

Syntax	`Spi_AsyncModeType`	
Type	Enumeration	
File	`Spi.h`	
Range	0 - SPI_POLLING_MODE	The asynchronous mechanism is ensured by polling, so interrupts (PT2, Error and DMA channel completion) related to the SPI busses are disabled.

(table continues...)

Table 94 (continued) Specification for Spi_AsyncModeType

	1 - SPI_INTERRUPT_MODE	The asynchronous mechanism is ensured by interrupts, so interrupts (PT2, Error and DMA channel completion) related to the SPI busses are enabled.
Description	This variable indicates if all the kernels are executing in POLLING or INTERRUPT mode. This data type will be available or not according to the pre compile time parameter SPI_LEVEL_DELIVERED. This type is only relevant for LEVEL 2.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.2 Spi_ConfigType

Table 95 Specification for Spi_ConfigType

Syntax	`Spi_ConfigType`	
Type	Structure	
File	`Spi.h`	
Range	-[]	None
Description	This structure holds the configuration of all the cores containing sequence, job, channel and Hardware unit information required to configure Hardware and transmit data over SPI interface.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.3 Spi_JobEndNotificationType

Table 96 Specification for Spi_JobEndNotificationType

Syntax	`Spi_JobEndNotificationType`
Type	Pointer to a function of type void Function_Name (void)
File	`Spi.h`
Description	Callback routine type for each Job to notify the caller that a job has been finished.
Source	AUTOSAR
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.2.4 Spi_JobResultType

Table 97 Specification for Spi_JobResultType

Syntax	`Spi_JobResultType`

(table continues...)

Table 97 (continued) Specification for Spi_JobResultType

Type	Enumeration	
File	`Spi.h`	
Range	0 - SPI_JOB_OK	The last transmission of the job has been finished successfully.
	1 - SPI_JOB_PENDING	The SPI Handler/Driver is performing (transmitting) a specific SPI job. The meaning of this status is equal to SPI_BUSY.
	2 - SPI_JOB_FAILED	The last transmission of the job has failed.
	3 - SPI_JOB_QUEUED	An asynchronous transmission of a Job has been accepted, while actual transmission for this Job has not started yet.
Description	This type defines a range of specific Jobs status for SPI Handler/Driver. It informs about a SPI Handler/Driver job status and can be obtained calling the API service Spi_GetJobResult with the job ID.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.5 Spi_LoopBackType

Table 98 Specification for Spi_LoopBackType

Syntax	`Spi_LoopBackType`	
Type	Enumeration	
File	`Spi.h`	
Range	0 - SPI_LOOPBACK_DISABLE	Disables the Loopback mode
	1 - SPI_LOOPBACK_ENABLE	Enables the Loopback mode
Description	This type is used to enable/disable the loopback feature.	
Source	IFX	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.6 Spi_SeqEndNotificationType

Table 99 Specification for Spi_SeqEndNotificationType

Syntax	`Spi_SeqEndNotificationType`
Type	Pointer to a function of type void Function_Name (void)
File	`Spi.h`

(table continues...)

Table 99 (continued) Specification for Spi_SeqEndNotificationType

Description	Callback routine type for each Sequence to notify the caller that a sequence has been finished.
Source	AUTOSAR
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.2.7 Spi_SeqResultType

Table 100 Specification for Spi_SeqResultType

Syntax	`Spi_SeqResultType`	
Type	Enumeration	
File	`Spi.h`	
Range	0 - SPI_SEQ_OK	The last transmission of the sequence has been finished successfully.
	1 - SPI_SEQ_PENDING	The SPI Handler/Driver is performing a specific SPI sequence. The meaning of this status is equal to SPI_BUSY.
	2 - SPI_SEQ_FAILED	The last transmission of the sequence has failed.
	3 - SPI_SEQ_CANCELLED	The last transmission of the sequence has been cancelled by user
Description	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.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.8 Spi_StatusType

Table 101 Specification for Spi_StatusType

Syntax	`Spi_StatusType`	
Type	Enumeration	
File	`Spi.h`	
Range	0 - SPI_UNINIT	The SPI Handler/Driver is not initialized or not usable (state after reset).
	1 - SPI_IDLE	The SPI Handler/Driver is not currently transmitting any jobs.
	2 - SPI_BUSY	The SPI Handler/Driver is performing a SPI job (transmit).

(table continues...)

Table 101 (continued) Specification for Spi_StatusType

Description	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.
Source	AUTOSAR
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.2.9 Spi_DataBufferType

Table 102 Specification for Spi_DataBufferType

Syntax	`Spi_DataBufferType`	
Type	uint8	
File	`Spi.h`	
Range	0-255	
Description	Type of application data buffer elements. *Note:* *- Channel width > 8, the SPI driver uses type uint16 to read or write the buffer. Similarly if channel width > 16, the SPI driver uses type uint32 to read or write the buffer. The SPI Driver will access the buffer as per little endian format (in accordance to architecture)* *- For 16-bit/32-bit transfer (Channel data width >8) user data have to be word aligned*	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.10 Spi_NumberOfDataType

Table 103 Specification for Spi_NumberOfDataType

Syntax	`Spi_NumberOfDataType`	
Type	uint16	
File	`Spi.h`	
Range	0-65535	
Description	Type for defining the number of data elements of the type Spi_DataBufferType to send and/or receive by channel. For Range details refer below parameters: - SpiIBNBuffers - SpiEBMaxLength	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.11 Spi_ChannelType

Table 104 Specification for Spi_ChannelType

Syntax	`Spi_ChannelType`	
Type	uint8	
File	`Spi.h`	
Range	0-255	
Description	Specifies the identification (ID) for a Channel. Channel ID can be from 0-254. Value 255 is used as delimiter to identify if last channel is reached in the channel list. This is a deviation from AUTOSAR.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.12 Spi_JobType

Table 105 Specification for Spi_JobType

Syntax	`Spi_JobType`	
Type	uint16	
File	`Spi.h`	
Range	0-65535	
Description	Specifies the identification (ID) for a Job. Job ID can be from 0-65535. Value 65535 is used as delimiter to identify if last job in a job list. This is a deviation from AUTOSAR.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.13 Spi_SequenceType

Table 106 Specification for Spi_SequenceType

Syntax	`Spi_SequenceType`	
Type	Spi_SequenceType	
File	`Spi.h`	
Range	0-255	
Description	Specifies the identification (ID) for a sequence of jobs. Sequence ID can be from 0-254. Value 255 is used as delimiter to identify last sequence in sequence list. This is a deviation from AUTOSAR.	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.2.14 Spi_HWUnitType

Table 107 Specification for Spi_HWUnitType

Syntax	`Spi_HWUnitType`	
Type	uint8	
File	`Spi.h`	
Range	0-255	
Description	Specifies the identification (ID) for a SPI Hardware microcontroller peripheral (unit). Range details: - Module no (bit [0:3]): QSPI**x**, where the range of **x** depends on the microcontroller derivative - Channel no (bit [4:7]): Channel0-channel15	
Source	AUTOSAR	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3 Functions - APIs

This section lists all the APIs of the SPI driver.

1.3.3.1 Spi_AsyncTransmit

Table 108 Specification for `Spi_AsyncTransmit` API

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

Service ID	0x03	
Sync/Async	Asynchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Sequence	Sequence ID.
Parameters (out)	-	-
Parameters (in - out)	-	-

(table continues...)

Table 108 (continued) Specification for `Spi_AsyncTransmit` API

Return	Std_ReturnType	E_OK: Transmit request accepted E_NOT_OK: In the cases as follows -Driver is not initialized -Invalid sequence parameter -Job is not able to fit in Job queue(Job queue is full) -Sequence is already in SPI_SEQ_PENDING state -Given Sequence shares the job with another sequence which is in SPI_SEQ_PENDING state -If DMA channels are not configured, applicable only for AUTOSAR 4.4.0 -If Move counter limit exceeds, applicable only for AUTOSAR 4..4.0
Description	This API transmits the sequence asynchronously over the QSPI interface. This API is asynchronous, which means the application invoking the API is not blocked till the sequence is transmitted completely and completion of transmission would be notified (if configured). *Note:* *- Before calling Spi_AsyncTransmit() API, user must ensure to call Spi_SetupEB for EB channels or Spi_WriteIB for IB channels(but before the Spi_ReadIB call)* *- The API is enabled only if SpiLevelDelivered is 1 or 2* *- From multicore perspective, sequences assigned to core can only be transmitted else SPI_E_NOT_CONFIGURED DET will be returned* *- Sequence ID to be used by application will be of format - SpiConf_SpiSequence_(x), x is user defined string for ex: SpiConf_SpiSequence_EEPROM_Write* *- Spi_AsyncTransmit() API will enable the interrupts only during asynchronous transmission and once the transmission is complete, interrupts are disabled*	
Source	AUTOSAR	
Error handling	SPI_E_PARAM_SEQ, SPI_E_UNINIT, SPI_E_SEQ_PENDING, SPI_E_NOT_CONFIGURED, SPI_E_QUEUE_FULL, SPI_E_DMA_CHANNEL_NOT_CONFIGURED, SPI_E_MOVECOUNTER_LIMIT_EXCEEDED	
Configuration dependencies	SpiLevelDelivered	
User hints	-	
SFR accessed	CPU_CORE_ID(r), DMA_CH_ADICR(rw), DMA_CH_CHCFGR(w), DMA_CH_CHCSR(w), DMA_CH_DADR(w), DMA_CH_RDCRCR(w), DMA_CH_SADR(w), DMA_CH_SDCRCR(w), DMA_CH_SHADR(rw), DMA_TSR(rw), P_OMR(w), QSPI_BACONENTRY(w), QSPI_ECON(w), QSPI_FLAGSCLEAR(w), QSPI_GLOBALCON(rw), QSPI_GLOBALCON1(w), QSPI_MC(w), QSPI_MCCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.2 Spi_Cancel

Table 109 Specification for `Spi_Cancel` API

Syntax	`void Spi_Cancel ( const Spi_SequenceType Sequence )`	
Service ID	0x0C	
Sync/Async	Asynchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Sequence	Sequence ID.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This API cancels the on-going sequence transmission. Sets the sequence status to SPI_SEQ_CANCELLED and job status to SPI_JOB_OK. If any sequence notification is set, same will be called after completion of the ongoing job. In Multicore context, only the sequence assigned to core in which the API has been called can only be cancelled else respective error will be returned. *Note:* *Calling Spi_Cancel() cancels the successive job of ongoing sequence being transmitted and updates the sequence status to cancelled. However updating the QSPI HW unit status to IDLE only happens once the ongoing job transmission is completed. So calling Spi_GetHwUnitStatus or Spi_GetStatus immediately after Spi_Cancel will return SPI_BUSY due to ongoing job.*	
Source	AUTOSAR	
Error handling	SPI_E_UNINIT, SPI_E_NOT_CONFIGURED, SPI_E_PARAM_SEQ	
Configuration dependencies	SpiCancelApi	
User hints	-	
SFR accessed	CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.3 Spi_ControlLoopBack

Table 110 Specification for `Spi_ControlLoopBack` API

Syntax	`Std_ReturnType Spi_ControlLoopBack ( const Spi_HWUnitType HWUnit, const Spi_LoopBackType EnableOrDisable )`	
Service ID	0x25	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different QSPI HW	
Parameters (in)	HWUnit EnableOrDisable	Specifies the QSPI HW unit Specifies enable/disable the loopback mode
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: If the loopback mode enable/disable is successful E_NOT_OK: In case of - Driver is not initialized - Invalid parameters - HW unit is busy
Description	This API enables/disables the Loopback mode. *Note :* *When Spi_ControlLoopBack API is invoked, upper layer must ensure that no other thread is starting a new sequence on the same HW until Spi_ControlLoopBack API has completed its execution.*	
Source	IFX	
Error handling	SPI_E_PARAM_UNIT, SPI_E_UNINIT, SPI_E_SAFETY_INVALID_PARAM	
Configuration dependencies	SpiEnableLoopBackApi	
User hints	-	
SFR accessed	CPU_CORE_ID(r), QSPI_GLOBALCON(rw) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.4 Spi_DeInit

Table 111 Specification for `Spi_DeInit` API

Syntax	`Std_ReturnType Spi_DeInit ( void )`	
Service ID	0x01	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: de-initialization command has been accepted E_NOT_OK: In case of - Driver is not initialized - Driver is busy in transmission
Description	This API de-initializes the hardware and global variables related to SPI driver. The API must be called only after module initialization and is accepted to be processed only when the device is in IDLE state. In multicore context, this API can be called by any core, only the kernel information associated with the caller core will be de-initialized other core still continue to work. *Note: This API resets all the registers including runtime registers as well.*	
Source	AUTOSAR	
Error handling	SPI_E_UNINIT	
Configuration dependencies	-	
User hints	-	
SFR accessed	CPU_CORE_ID(r), QSPI_CLC(w), QSPI_ECON(w), QSPI_FLAGSCLEAR(w), QSPI_GLOBALCON(w), QSPI_GLOBALCON1(w), QSPI_PISEL(w), QSPI_SSOC(w), SCU_CCUCON0(r), SCU_EICON0(rw), SCU_OSCCON(r), SCU_SYSPLLCON0(r), SCU_SYSPLLCON1(r), STM_TIM0(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.5 Spi_GetHWUnitStatus

Table 112 Specification for `Spi_GetHWUnitStatus` API

Syntax	`Spi_StatusType Spi_GetHWUnitStatus ( const Spi_HWUnitType HWUnit )`	
Service ID	0x0B	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	HWUnit	QSPI kernel Id
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Spi_StatusType	SPI_UNINIT: In case of - Driver is not initialized - Invalid HW unit parameter - HW unit is not configured for the core SPI_IDLE: The QSPI Hardware is not currently transmitting any Job SPI_BUSY: The QSPI Hardware is performing a SPI Job (transmit)
Description	This API returns the status of the Hardware kernel requested for, If the QSPI kernel is busy transmitting a sequence SPI_BUSY is returned else SPI_IDLE is returned. In multicore context, API will be able to get the status of the kernels assigned to core on which the request is made, kernel assigned to different cores status cannot be obtained and returns un-predictable results.	
Source	AUTOSAR	
Error handling	SPI_E_PARAM_UNIT, SPI_E_UNINIT	
Configuration dependencies	SpiHwStatusApi	
User hints	-	
SFR accessed	CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.6 Spi_GetJobResult

Table 113 Specification for `Spi_GetJobResult` API

Syntax	`Spi_JobResultType Spi_GetJobResult ( const Spi_JobType Job )`	
Service ID	0x07	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Job	Job ID.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Spi_JobResultType	SPI_JOB_OK: The last transmission of the job has been finished successfully SPI_JOB_PENDING: The SPI Handler/Driver is performing (transmitting) a specific SPI job. The meaning of this status is equal to SPI_BUSY SPI_JOB_FAILED: In case of - Driver is not initialized - Job is not within the configured job range for core - Job is not assigned to the core - The last transmission of the job has failed SPI_JOB_QUEUED: An asynchronous transmission of a Job has been accepted, while actual transmission for this Job has not started yet
Description	This service returns the last transmission result of the specified Job. API returns the status of the job depending on whether job is queued, failed, pending or successful. In multicore context, API can only return the status of the jobs that are assigned to core in which API is called. *Note: It is recommended to call this API from the notification function when notification is enabled to avoid the stale data problems.*	
Source	AUTOSAR	
Error handling	SPI_E_PARAM_JOB, SPI_E_UNINIT, SPI_E_NOT_CONFIGURED	
Configuration dependencies	-	
User hints	-	

(table continues...)

Table 113 (continued) Specification for `Spi_GetJobResult` API

SFR accessed	CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.7 Spi_GetSequenceResult

Table 114 Specification for `Spi_GetSequenceResult` API

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

Service ID	0x08	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Sequence	Sequence Id for which the status to be returned.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Spi_SeqResultType	SPI_SEQ_OK: The last transmission of the sequence has been finished successfully SPI_SEQ_PENDING: The SPI Handler/Driver is performing a specific SPI sequence. The meaning of this status is equal to SPI_BUSY SPI_SEQ_FAILED: In case of - Driver is not initialized - Sequence is not configured for the Core - Sequence is not within the configured range for the core - The last transmission of the sequence has failed SPI_SEQ_CANCELLED: The last transmission of the sequence has been cancelled by user

(table continues...)

Table 114 (continued) Specification for `Spi_GetSequenceResult` API

Description	This service returns the last transmission result of the specified Sequence. API returns the status of the sequence depending on whether sequence is queued, failed, pending or successful. In multicore context, API can only return the status of the sequence that are assigned to core in which API is called. *Note: It is recommended to call this API from the notification function when notification is enabled to avoid the stale data problems.*
Source	AUTOSAR
Error handling	SPI_E_UNINIT, SPI_E_PARAM_SEQ, SPI_E_NOT_CONFIGURED
Configuration dependencies	-
User hints	-
SFR accessed	CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.8 Spi_GetStatus

Table 115 Specification for `Spi_GetStatus` API

Syntax	`Spi_StatusType Spi_GetStatus ( void )`	
Service ID	0x06	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-

(table continues...)

Table 115 (continued) Specification for `Spi_GetStatus` API

Return	Spi_StatusType	SPI_UNINIT: The SPI Handler/driver is not initialized SPI_IDLE: The SPI Handler/Driver is not currently transmitting any Job SPI_BUSY: The SPI Handler/Driver is performing a SPI Job (transmit)
Description	This API returns the status of the driver as whole including synchronous and asynchronous transmissions (if configured). After reset and before Spi_Init() API is invoked, the status of the driver will be SPI_UNINIT. In Multicore context, API will return the status of the driver for the kernels assigned to core only.	
Source	AUTOSAR	
Error handling	SPI_E_UNINIT	
Configuration dependencies	-	
User hints	-	
SFR accessed	CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.9 Spi_GetVersionInfo

Table 116 Specification for `Spi_GetVersionInfo` API

Syntax	`void Spi_GetVersionInfo`
	`(`
	`  Std_VersionInfoType * const versioninfo`
	`)`	

Service ID	0x09	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	-	-
Parameters (out)	versioninfo	Pointer to the address where the driver information should be stored
Parameters (in - out)	-	-

(table continues...)

Table 116 (continued) Specification for `Spi_GetVersionInfo` API

Return	void	-
Description	This API updates the pointer address with the driver version information.	
Source	AUTOSAR	
Error handling	SPI_E_PARAM_POINTER	
Configuration dependencies	SpiVersionInfoApi	
User hints	-	
SFR accessed	-	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.10 Spi_Init

Table 117 Specification for `Spi_Init` API

Syntax	`void Spi_Init ( const Spi_ConfigType * const ConfigPtr )`	
Service ID	0x00	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non Reentrant	
Parameters (in)	ConfigPtr	Pointer to configuration set
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	The API Spi_Init initializes all the SFRs of QSPI kernels assigned to core, resets the global variables and sets the status to IDLE. In multicore context, Only the kernels assigned to core will be initialized. *Note: The Spi_Init API initializes the asynchronous mode to polling mode.*	
Source	AUTOSAR	
Error handling	SPI_E_ALREADY_INITIALIZED, SPI_E_PARAM_POINTER, SPI_E_INIT_FAILED	
Configuration dependencies	-	
User hints	-	

(table continues...)

Table 117 (continued) Specification for `Spi_Init` API

SFR accessed	CPU_CORE_ID(r), QSPI_CLC(w), QSPI_ECON(w), QSPI_FLAGSCLEAR(w), QSPI_GLOBALCON(w), QSPI_GLOBALCON1(w), QSPI_PISEL(w), QSPI_SSOC(w), SCU_CCUCON0(r), SCU_EICON0(rw), SCU_OSCCON(r), SCU_SYSPLLCON0(r), SCU_SYSPLLCON1(r), STM_TIM0(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.11 Spi_InitCheck

Table 118 Specification for `Spi_InitCheck` API

Syntax	`Std_ReturnType Spi_InitCheck`
	`(`
	`  const Spi_ConfigType * const ConfigPtr`
	`)`	

Service ID	0x20	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different core	
Parameters (in)	ConfigPtr	Pointer to Configuration to be checked against
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: On successfully checking the mentioned global variables / SFRs E_NOT_OK: In case of - Driver is not initialized - Input config Pointer is Null - Input config pointer is other than the one used for Init - Global Variables or SFR is not set as expected

(table continues...)

Table 118 (continued) Specification for `Spi_InitCheck` API

Description	InitCheck API will check the registers and the critical variables are initialized as expected. In Multicore context, only the globals and SFRs assigned to core will be verified in InitCheck API. *Note:* *- API except Spi_Init and Spi_GetVersionInfo must be called only after successful return of initCheck when Safety is enabled* *- Following pointers and SFR are verified in InitCheck API: ConfigPtr, Global core pointer, Global kernel pointer, Queue pointer for each kernel, buffer pointer in case of IB, Overall status of driver and kernel (should be IDLE), all QSPI related registers are set to reset state and all DMA RAM TCS elements source and destination address is updated during initialization* *- Following global elements are not verified to be in reset state: content of IB buffer itself, content of the Queue variables and TCS memory itself, however critical variables of Queue like Queue index, start and end index are verified to be in reset state* Rationale: These variables will be updated at run-time and previous values will not affect any functionality since these will be updated for every transfers.
Source	IFX
Error handling	-
Configuration dependencies	SpiInitCheckApi
User hints	-
SFR accessed	CPU_CORE_ID(r), QSPI_ECON(r), QSPI_GLOBALCON(r), QSPI_GLOBALCON1(r), QSPI_PISEL(r), QSPI_SSOC(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.12 Spi_ReadIB

Table 119 Specification for `Spi_ReadIB` API

Syntax	`Std_ReturnType Spi_ReadIB`
	`(`
	`  const Spi_ChannelType Channel,`
	`  Spi_DataBufferType * const DataBufferPointer`
	`)`

Service ID	0x04
Sync/Async	Synchronous
Safety Level	Refer to the release notes for the safety related info
Re-entrancy	Reentrant

(table continues...)

Table 119 (continued) Specification for `Spi_ReadIB` API

Parameters (in)	Channel	Channel ID.
Parameters (out)	DataBufferPointer	This is pointer to the destination buffer pointer to where the received data is copied
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Data from Internal buffer to the destination buffer is copied successfully E_NOT_OK: In case of - Driver is not initialized - Invalid channel or DataBufferPointer is NULL
Description	Service for reading synchronously the received channel data from Internal buffer to the destination buffer passed by application. In Multicore context, ReadIB will be successful for the channels assigned to core in which the request is made else the result is un-predictable. *Note: Application should take care of protecting the buffer since driver do not use any protection mechanism to protect data. i.e. Application should sequence the call for writing to same channel only after reading the data else buffer corruption could occur.*	
Source	AUTOSAR	
Error handling	SPI_E_UNINIT, SPI_E_PARAM_CHANNEL, SPI_E_NOT_CONFIGURED, SPI_E_PARAM_POINTER	
Configuration dependencies	SpiChannelBuffersAllowed	
User hints	-	
SFR accessed	CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.13 Spi_SetAsyncMode

Table 120 Specification for `Spi_SetAsyncMode` API

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

Service ID	0x0d
Sync/Async	Synchronous
Safety Level	Refer to the release notes for the safety related info

(table continues...)

Table 120 (continued) Specification for `Spi_SetAsyncMode` API

Re-entrancy	Non-Reentrant	
Parameters (in)	Mode	Specifies the asynchronous mechanism mode for SPI busses handled asynchronously in Level 2
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Switching to new Mode is successful E_NOT_OK: In case of - Driver is not initialized - Invalid parameter - One of the QSPI kernel assigned to core is busy in asynchronous transmission
Description	This API sets the asynchronous mode of handling transmission of sequences to either Polling mode or Interrupt mode. Available only in Level 2, mode cannot be updated / changed if any of the kernel assigned to core is in BUSY state.	
Source	AUTOSAR	
Error handling	SPI_E_UNINIT, SPI_E_SAFETY_INVALID_PARAM	
Configuration dependencies	SpiLevelDelivered	
User hints	-	
SFR accessed	CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.3.14 Spi_SetupEB

Table 121 Specification for `Spi_SetupEB` API

Syntax	`Std_ReturnType Spi_SetupEB`
	`(`
	`const Spi_ChannelType Channel,`
	`const Spi_DataBufferType * const SrcDataBufferPtr,`
	`const Spi_DataBufferType * const DesDataBufferPtr,`
	`const Spi_NumberOfDataType Length`
	`)`

Service ID	0x05

(table continues...)

Table 121 (continued) Specification for `Spi_SetupEB` API

Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different channels	

Parameters (in)	Channel
	SrcDataBufferPtr
	DesDataBufferPtr
	Length
		Channel ID of the respective EB channel
		This is the pointer to source buffer for the EB channel
		This is the pointer to destination buffer to where the received data is copied
		Number of data elements to be transmitted.
		i.e.,
		for 8-bit channel, if length is 2, then
		2 * 8-bit = 16 bits will be transferred,
		for 16-bit channel, if length is 2, then
		2 * 16-bit = 32 bits will be transferred and
		for 32-bit channel, if length is 2, then
		2 * 32-bit = 64 bits will be transferred

Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: Buffers has been setup for the EB channel E_NOT_OK: In case of - Driver is not initialized - Invalid channel parameter - Invalid length parameter
Description	This API updates the source pointer, destination pointer and transfer length of channel as passed by API. No DET is raised for pointers being NULL, if Source pointer is NULL indicates that default data to be used for transmission. If Destination address is NULL, then ignore the received data. In Multicore context, channel assigned to core can only be accessed.	
Source	AUTOSAR	
Error handling	SPI_E_PARAM_LENGTH, SPI_E_PARAM_CHANNEL, SPI_E_NOT_CONFIGURED, SPI_E_UNINIT	
Configuration dependencies	SpiChannelBuffersAllowed	
User hints	-	
SFR accessed	CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	

(table continues...)

Table 121 (continued) Specification for `Spi_SetupEB` API

Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.15 Spi_SyncTransmit

Table 122 Specification for `Spi_SyncTransmit` API

Syntax	`Std_ReturnType Spi_SyncTransmit ( const Spi_SequenceType Sequence )`	
Service ID	0x0A	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant	
Parameters (in)	Sequence	Sequence ID.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK : Synchronous transmission request is accepted E_NOT_OK: In case of - Driver is not initialized - Invalid sequence parameter - QSPI kernel is already busy in transmission when Concurrent synchronous transmission is enabled - One of the synchronous kernel configured for core is busy in transmission - HW error occurs during transmission - Requested kernel is busy in asynchronous transmission. This is applicable only for AUTOSAR 4.4.0

(table continues...)

Table 122 (continued) Specification for `Spi_SyncTransmit` API

Description	This API transmits the sequence synchronously over the QSPI bus. Note that the API is a blocking API which means execution is blocked till the sequence is transmitted completely. - This API is enabled only if SpiLevelDelivered is 0 or 2 - If the QSPI HW unit is already in BUSY state(transmission ongoing), then new synchronous request on the same QSPI HW is not accepted - In AUTOSAR 4.4.0 and Level2 configuration, When a QSPI is busy in synchronous transmission and an asynchronous transmission of a sequence is requested on the same QSPI, all jobs belonging to sequence are queued and transmission of the queued jobs start after synchronous transmission is over *Note1:* *- When concurrent transmission is enabled, The HW units are allowed to operate parallel* *- When concurrent transmission is disabled, The HW units are not allowed to operate parallel* *Note2:* *- From multicore perspective, sequences assigned to core can only be transmitted else SPI_E_NOT_CONFIGURED DET will be returned* *- Sequence ID to be used by application will be of format - SpiConf_SpiSequence_(x), x is user defined string for ex: SpiConf_SpiSequence_EEPROM_Write*
Source	AUTOSAR
Error handling	SPI_E_HARDWARE_ERROR, SPI_E_NOT_CONFIGURED, SPI_E_PARAM_SEQ, SPI_E_SEQ_IN_PROCESS, SPI_E_UNINIT
Configuration dependencies	SpiLevelDelivered
User hints	-
SFR accessed	CPU_CORE_ID(r), DMA_CH_ADICR(rw), DMA_CH_CHCFGR(w), DMA_CH_CHCSR(w), DMA_CH_DADR(w), DMA_CH_RDCRCR(w), DMA_CH_SADR(w), DMA_CH_SDCRCR(w), DMA_CH_SHADR(rw), DMA_TSR(rw), P_OMR(w), QSPI_BACONENTRY(w), QSPI_DATAENTRY(w), QSPI_ECON(w), QSPI_FLAGSCLEAR(w), QSPI_GLOBALCON(rw), QSPI_GLOBALCON1(w), QSPI_MC(w), QSPI_MCCON(w), QSPI_RXEXIT(r), QSPI_STATUS(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.

1.3.3.16 Spi_WriteIB

Table 123 Specification for `Spi_WriteIB` API

Syntax	`Std_ReturnType Spi_WriteIB`
	`(`
	`  const Spi_ChannelType Channel,`
	`  const Spi_DataBufferType * const DataBufferPtr`
	`)`

(table continues...)

Table 123 (continued) Specification for `Spi_WriteIB` API

Service ID	0x02	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different channels	
Parameters (in)	Channel DataBufferPtr	Channel ID. Pointer to source data buffer. If this pointer is null, it is assumed that the data to be transmitted is not relevant and the default transmit value of this channel will be used instead.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	Std_ReturnType	E_OK: specifies the data in the source buffer is copied into the local internal buffer E_NOT_OK: In case of - Driver is not initialized - Invalid channel parameter
Description	This API copies the data to be transmitted in transmit buffer from the source pointer passed in the API by application. Dependency: If SpiChannelBufferAllowed is set to 0 OR 2 this API will be enabled during compilation. In multicore context, only the channels assigned to core can be written.	
Source	AUTOSAR	
Error handling	SPI_E_UNINIT, SPI_E_PARAM_CHANNEL, SPI_E_NOT_CONFIGURED	
Configuration dependencies	SpiChannelBuffersAllowed	
User hints	-	
SFR accessed	CPU_CORE_ID(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.4 Notifications and Callbacks

This section lists all the notification and callbacks of the SPI driver.

1.3.4.1 Spi_QspiDmaCallout

Table 124 Specification for `Spi_QspiDmaCallout` API

Syntax	`void Spi_QspiDmaCallout ( const uint8 Channel, const uint32 Event )`	
Service ID	0x21	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different HW unit	
Parameters (in)	Channel Event	Channel number [0-127] DMA events
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	DMA callback is called at end of every channel transmission, during this callback channel is re-configured and respective DMA channels are re-triggered to start the next channel transfer. This callback cannot be avoided since each channels can have different data length and same has to be re-configured in the BACON register.	
Source	IFX	
Error handling	SPI_E_SAFETY_INVALID_PARAM	
Configuration dependencies	SpiLevelDelivered	
User hints	-	
SFR accessed	CPU_CORE_ID(r), DMA_CH_ADICR(rw), DMA_CH_CHCFGR(w), DMA_CH_CHCSR(w), DMA_CH_DADR(w), DMA_CH_RDCRCR(w), DMA_CH_SADR(w), DMA_CH_SDCRCR(w), DMA_CH_SHADR(rw), DMA_TSR(rw), P_OMR(w), QSPI_BACONENTRY(w), QSPI_ECON(w), QSPI_FLAGSCLEAR(w), QSPI_GLOBALCON(rw), QSPI_GLOBALCON1(rw), QSPI_MC(w), QSPI_MCCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.4.2 Spi_QspiDmaErrCallout

Table 125 Specification for `Spi_QspiDmaErrCallout` API

Syntax	`void Spi_QspiDmaErrCallout ( const uint8 Channel, const uint32 Event )`	
Service ID	0x24	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different HW unit	
Parameters (in)	Channel Event	Channel number [0-127] DMA events.
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This function is called from DMA module on detecting a move engine error during DMA transfer.	
Source	IFX	
Error handling	SPI_E_HARDWARE_ERROR, SPI_E_SAFETY_INVALID_PARAM	
Configuration dependencies	SpiLevelDelivered	
User hints	-	
SFR accessed	CPU_CORE_ID(r), DMA_CH_ADICR(rw), DMA_CH_CHCFGR(w), DMA_CH_CHCSR(w), DMA_CH_DADR(w), DMA_CH_RDCRCR(w), DMA_CH_SADR(w), DMA_CH_SDCRCR(w), DMA_CH_SHADR(rw), DMA_TSR(rw), P_OMR(w), QSPI_BACONENTRY(w), QSPI_ECON(w), QSPI_FLAGSCLEAR(w), QSPI_GLOBALCON(rw), QSPI_GLOBALCON1(rw), QSPI_MC(w), QSPI_MCCON(w) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.5 Scheduled functions

This section lists all the scheduled functions of SPI driver.

1.3.5.1 Spi_MainFunction_Handling

Table 126 Specification for `Spi_MainFunction_Handling` API

Syntax	`void Spi_MainFunction_Handling ( void )`	
Service ID	0x10	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Non-Reentrant	
Parameters (in)	-	-
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This function polls the SPI interrupt flags linked to QSPI Hardware units. This function will be called by application at regular interval as defined in the application.	
Source	AUTOSAR	
Error handling	-	
Configuration dependencies	SpiLevelDelivered	
User hints	-	
SFR accessed	CPU_CORE_ID(r), DMA_CH_ADICR(rw), DMA_CH_CHCFGR(w), DMA_CH_CHCSR(rw), DMA_CH_DADR(w), DMA_CH_RDCRCR(w), DMA_CH_SADR(w), DMA_CH_SDCRCR(w), DMA_CH_SHADR(rw), DMA_ME_ERRSR(r), DMA_TSR(rw), P_OMR(w), QSPI_BACONENTRY(w), QSPI_ECON(w), QSPI_FLAGSCLEAR(w), QSPI_GLOBALCON(rw), QSPI_GLOBALCON1(rw), QSPI_MC(w), QSPI_MCCON(w), QSPI_STATUS(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.6 Interrupt service routines

This section lists all the interrupt handlers of the SPI driver.

1.3.6.1 Spi_IsrQspiError

Table 127 Specification for `Spi_IsrQspiError` API

Syntax	`void Spi_IsrQspiError ( const uint8 Module )`	
Service ID	0x23	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different HW unit	
Parameters (in)	Module	Kernel number 0-5
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	This interrupt service routine handles the QSPI errors during asynchronous transmission. Sets the status of the Sequence to SPI_SEQ_FAILED and job status belonging to the sequence except which are completed to SPI_JOB_FAILED.	
Source	IFX	
Error handling	SPI_E_SAFETY_INVALID_PARAM, SPI_E_HARDWARE_ERROR, SPI_E_SAFETY_SPURIOUS_INTERRUPT	
Configuration dependencies	SpiLevelDelivered	
User hints	-	
SFR accessed	CPU_CORE_ID(r), DMA_CH_ADICR(rw), DMA_CH_CHCFGR(w), DMA_CH_CHCSR(w), DMA_CH_DADR(w), DMA_CH_RDCRCR(w), DMA_CH_SADR(w), DMA_CH_SDCRCR(w), DMA_CH_SHADR(rw), DMA_TSR(rw), P_OMR(w), QSPI_BACONENTRY(w), QSPI_ECON(w), QSPI_FLAGSCLEAR(w), QSPI_GLOBALCON(rw), QSPI_GLOBALCON1(rw), QSPI_MC(w), QSPI_MCCON(w), QSPI_STATUS(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.6.2 Spi_IsrQspiPT2

Table 128 Specification for `Spi_IsrQspiPT2` API

Syntax	`void Spi_IsrQspiPT2 ( const uint8 Module )`	
Service ID	0x22	
Sync/Async	Synchronous	
Safety Level	Refer to the release notes for the safety related info	
Re-entrancy	Reentrant for different HW unit	
Parameters (in)	Module	QSPI module index [0 – 5]
Parameters (out)	-	-
Parameters (in - out)	-	-
Return	void	-
Description	PT2 - Phase transition interrupt signals one out of all phases of Hardware state transition. This Interrupt Service routine marks the end of the frame transmission and is triggered only at the end of the job transmission. Total number of elements to be transmitted in a next job is updated in the MCCOUNT during this interrupt and respective DMA channels are re-triggered.	
Source	IFX	
Error handling	SPI_E_HARDWARE_ERROR, SPI_E_SAFETY_INVALID_PARAM, SPI_E_SAFETY_SPURIOUS_INTERRUPT	
Configuration dependencies	SpiLevelDelivered	
User hints	-	
SFR accessed	CPU_CORE_ID(r), DMA_CH_ADICR(rw), DMA_CH_CHCFGR(w), DMA_CH_CHCSR(w), DMA_CH_DADR(w), DMA_CH_RDCRCR(w), DMA_CH_SADR(w), DMA_CH_SDCRCR(w), DMA_CH_SHADR(rw), DMA_TSR(rw), P_OMR(w), QSPI_BACONENTRY(w), QSPI_ECON(w), QSPI_FLAGSCLEAR(w), QSPI_GLOBALCON(rw), QSPI_GLOBALCON1(rw), QSPI_MC(w), QSPI_MCCON(w), QSPI_STATUS(r) *Note : The list includes all the SFRs accessed in the context of the API. It lists the SFRs accessed by the driver and called interfaces from other drivers. During runtime, the SFRs accessed from this list may vary based on configuration and execution context.*	
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.	

1.3.7 Callout

The driver does not support any callout functions.

1.3.8 Errors Handling

This section describes the various errors reported by the SPI driver.

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
SPI_E_ALREADY_INITIALIZED: API SPI_Init service called while the SPI driver has already been initialized.	AUTOSAR	0x4A	DET_SAFETY	0x4A	DET_SAFETY
SPI_E_DMA_CHANNEL_NOT_CONFIGURED: DMA channels are not configured for a QSPI. *Note: This is applicable only for AUTOSAR 4.4.0(Level-2 Configuration).*	IFX	NA	NA	0x67	DET_SAFETY

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
SPI_E_HARDWARE_ERROR: In AUTOSAR 4.2.2: - On any error bit set in status register: Mcal_Wrapper_Dem_ReportErrorStatus (SPI_E_HARDWARE_ERROR, DEM_EVENT_STATUS_FAILED) - If no error is reported and successful transmission: Mcal_Wrapper_Dem_ReportErrorStatus (SPI_E_HARDWARE_ERROR, DEM_EVENT_STATUS_PASSED) In AUTOSAR 4.4.0: - On any error bit set in status register: Mcal_Wrapper_Dem_SetEventStatus (SPI_E_HARDWARE_ERROR, DEM_EVENT_STATUS_FAILED) - If no error is reported and successful transmission: Mcal_Wrapper_Dem_SetEventStatus (SPI_E_HARDWARE_ERROR, DEM_EVENT_STATUS_PASSED) If any error is reported, application needs to monitor and take appropriate action. *Note: The SPI_E_HARDWARE_ERROR Production Error is raised for QSPI Hardware errors or DMA ME errors.*	AUTOSAR	Assigned by DEM	Production Error	Assigned by DEM	Production Error
SPI_E_INIT_FAILED: The DET SPI_E_INIT_FAILED is reported when the configuration set used during initialization is not same across all cores.	AUTOSAR	0x64	DET_SAFETY	0x64	DET_SAFETY

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
SPI_E_MOVECOUNTER_LIMIT_EXCEEDED: When the cumulative number of data elements of all the channels configured for a job, crosses 8190 elements. *Note: This is applicable only for LEVEL-2 configuration in both AUTOSAR 4.2.2 and AUTOSAR 4.4.0*	IFX	0x68	DET_SAFETY	0x68	DET_SAFETY
SPI_E_NOT_CONFIGURED: Sequence / job / channel number passed is not configured to core on which request is made but with-in the max range of sequence. *Note: This is not applicable for APIs Spi_AsyncTransmit and Spi_SyncTransmit in AUTOSAR 4.4.0.*	AUTOSAR	0x4B	DET_SAFETY	0x4B	DET_SAFETY
SPI_E_PARAM_CHANNEL: Incorrect parameter passed in API.	AUTOSAR	0x0A	DET_SAFETY	0x0A	DET_SAFETY
SPI_E_PARAM_JOB: API service called with wrong parameter.	AUTOSAR	0x0B	DET_SAFETY	0x0B	DET_SAFETY
SPI_E_PARAM_LENGTH: Length parameter is greater than the defined limit.	AUTOSAR	0x0D	DET_SAFETY	0x0D	DET_SAFETY
SPI_E_PARAM_POINTER: APIs called with a null pointer.	AUTOSAR	0X10	DET_SAFETY	0X10	DET_SAFETY
SPI_E_PARAM_SEQ: API service called if the sequence ID is not in the range of the total sequence numbers allocated to all the cores.	AUTOSAR	0X0C	DET_SAFETY	0X0C	DET_SAFETY
SPI_E_PARAM_UNIT: API service called with wrong parameter.	AUTOSAR	0x0E	DET_SAFETY	0x0E	DET_SAFETY

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
SPI_E_QUEUE_FULL: If a new sequence is requested to be transmitted and if slots are less than number of jobs in Queue this DET is raised. On this DET user can increase the Queue length in SpiJobQueueLengthQspix field.	AUTOSAR	0x4C	DET_SAFETY	0x4C	DET_SAFETY
SPI_E_SAFETY_INVALID_PARAM: - If an invalid parameter is passed to Spi_SetAsyncMode API, the mode will be changed to the polling mode. As a safety check this DET is added - If an invalid parameter is passed to ISR like wrong kernel which is not in range this safety DET will be triggered - A safety check DET is reported when an invalid parameter is passed to the EnableOrDisable parameter of the Spi_ControlLoopBack API - A safety check DET is reported when an invalid DMA channel number passed or invalid DMA channel event passed to Spi_QspiDmaCallout - A safety check DET is reported when an invalid DMA channel number passed to Spi_QspiDmaErrCallout	IFX	0x65	SAFETY	0x65	SAFETY
SPI_E_SAFETY_SPURIOUS_INTERRUPT: For every interrupt triggered, source of interrupt will be checked, if no source can be detected this safety error will be triggered.	IFX	0x66	SAFETY	0x66	SAFETY
SPI_E_SEQ_IN_PROCESS: Synchronous transmission service called at wrong time.	AUTOSAR	0X3A	DET_SAFETY	0X3A	RUNTIME
SPI_E_SEQ_PENDING: Indicates that sequence is in Pending state and requested action cannot be performed.	AUTOSAR	0X2A	DET_SAFETY	0X2A	RUNTIME

Error Name: Description	Source	Error ID (AS422)	Type (AS422)	Error ID (AS440)	Type (AS440)
SPI_E_UNINIT: API service used without module initialization.	AUTOSAR	0X1A	DET_SAFETY	0X1A	DET_SAFETY

1.3.9 Deviations and limitations

The section describes the deviations and limitations of the SPI driver.

1.3.9.1 Deviations

The section describes the deviations of the SPI driver.

1.3.9.1.1 Software specification deviations

This section describes the deviations from software specification.

Table 129 Known deviations

Reference	Deviation
For all requirements related to Production/Runtime errors	Reporting of Production error: Dem_ReportErrorStatus is done through Mcal_Wrapper_Dem_ReportErrorStatus interface for AUTOSAR 4.2.2 and Dem_SetEventStatus is done through Mcal_Wrapper_Dem_SetEventStatus interface for AUTOSAR 4.4.0. Reporting of Runtime error: Det_ReportRuntimeError is done through Mcal_Wrapper_Det_ReportRuntimeError interface. This is applicable for only AUTOSAR 4.4.0. All production and runtime related datatypes and modified interfaces inclusion shall be done via Mcal_Wrapper.h

1.3.9.1.2 AMDC Violations

The SPI driver does not have any AMDC violations.

1.3.9.1.3 VSMD Violations

This section describes the violations reported by the EB VSMD checker tool with respect to AUTOSAR.

Violations reported by VSMD checker tool for TpsEcuc_06051_ASR41

Table 130 Table: Violations reported by VSMD checker tool for TpsEcuc_06051_ASR41

Rule ID:	TpsEcuc_06051_ASR41

(table continues...)

Table 130 (continued) Table: Violations reported by VSMD checker tool for TpsEcuc_06051_ASR41

VSMD Node(s):	/AURIX2G/EcucDefs/Spi/POST_BUILD_VARIANT_USED /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiChannelType /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiDataWidth /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiIbNBuffers /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiHwUnit /AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiChannelList/SpiChannelAssignment /AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiHwUnitSynchronous
Description:	The implementationConfigClass of an EcucParameterDef or EcucAbstractReferenceDef in VSMD shall be the same or higher (where PreCompile configuration class is considered to be the lowest and PostBuild the highest) as in StMD with respect to the selected subset defined by the actually implemented supportedConfigVariant.

(table continues...)

Table 130 (continued) Table: Violations reported by VSMD checker tool for TpsEcuc_06051_ASR41

Additional information:	The implementationConfigClass value is deviated from AUTOSAR due to the following reasons: 1. `SpiChannelType`: This parameter specifies the buffer type (External Buffer / Internal Buffer) used by the channel. The value configured to this parameter may change only at pre-compile time since it is coupled with buffer size which cannot be changed across the variants. This parameter is used to generate a macro for total IB channels, EB channels and total IB buffer size configured for a core. Hence, the implementationConfigClass is pre-compile instead of post-build. 2. `SpiDataWidth`: This parameter specifies the width of the data to be transmitted in terms of bits. The QSPI supports the data width from 2 to 32 bits. The value configured to this parameter may change only at pre-compile time. This parameter is used to generate a macro for total IB channels, EB channels and total IB buffer size configured for a core. Hence, the implementationConfigClass is pre-compile instead of post-build. 3. `SpiIbNBuffers`: This parameter specifies the buffer size of the IB channels. The value configured to this parameter may change only at pre-compile time. This parameter is used to generate a macro for total IB channels, EB channels and total IB buffer size configured for a core. Hence, the implementationConfigClass is pre-compile instead of post-build. 4. `SpiHwUnit`: This parameter specifies the SPI hardware microcontroller peripheral allocated for transmission. The value configured to this parameter may change only at pre-compile time. This parameter is used to generate a macro for total IB channels, EB channels and total IB buffer size configured for a core. Hence, the implementationConfigClass is pre-compile instead of post-build. 5. `SpiChannelAssignment`: This parameter specifies the channel linked to this Job container. The value configured to this parameter may change only at pre-compile time. This parameter is used to generate a macro for total IB channels, EB channels and total IB buffer size configured for a core. Hence, the implementationConfigClass is pre-compile instead of post-build. 6. `SpiHwUnitSynchronous`: This parameter specifies if a job is Synchronous or Asynchronous. The

(table continues...)

Table 130 (continued) Table: Violations reported by VSMD checker tool for TpsEcuc_06051_ASR41

	value configured to this parameter may change only at pre-compile time. This parameter is used to generate a macro which indicates the type of communication is allowed on QSPI HW. Hence, the implementationConfigClass is pre-compile instead of post-build (applicable for AUTOSAR 4.2.2 only).

Table 131 Table: Violations reported by VSMD checker tool for TpsEcuc_08032

Rule ID:	TpsEcuc_08032
VSMD Node(s):	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiChannelType /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiDataWidth /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiIbNBuffers /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiHwUnit /AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiChannelList/SpiChannelAssignment /AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiHwUnitSynchronous /AURIX2G/EcucDefs/Spi/SpiDriver/SpiMaxChannel /AURIX2G/EcucDefs/Spi/SpiDriver/SpiMaxJob /AURIX2G/EcucDefs/Spi/SpiDriver/SpiMaxSequence /AURIX2G/EcucDefs/Spi/SpiDriver/SpiSequence/SpiJobAssignment
Description:	If the EcucModuleDef.postBuildVariantSupport is set to true and the postBuildVariantValue for an EcucParameterDef or an EcucAbstractReferenceDef in this EcucModuleDef in the StMD is set to true, the corresponding VSMD shall also set it to true.

(table continues...)

Table 131 (continued) Table: Violations reported by VSMD checker tool for TpsEcuc_08032

Additional information:	1. `SpiMaxChannel`: The postBuildVariantValue is set to FALSE because changing the value across the variants irrespective of number of channels configured has an impact on the total number of channels configured per core. Since the EB Tresos tool has a limitation, we cannot derive the maximum number of channels configured across the variants. 2. `SpiMaxJob`: The postBuildVariantValue is set to FALSE because changing the value across the variants irrespective of number of jobs configured has an impact on the total number of jobs configured per core. Since the EB Tresos tool has a limitation, we cannot derive the maximum number of jobs configured across the variants. 3. `SpiMaxSequence`: The postBuildVariantValue is set to FALSE because changing the value across the variants irrespective of number of sequences configured has an impact on the total number of sequences configured per core. Since the EB Tresos tool has a limitation, we cannot derive the maximum number of sequences configured across the variants. 4. `SpiChannelType, SpiDataWidth, SpiIbNBuffers, SpiHwUnit, SpiChannelAssignment`: The postBuildVariantValue is set to FALSE because these parameters are used for generating a macro for total number of IB channels and EB channels configured per core and total IB buffer size required per core. 5. `SpiJobAssignment`: The postBuildVariantValue is set to FALSE because this parameter is used to generate macro for total number of sequences configured per core. 6. `SpiHwUnitSynchronous`: The postBuildVariantValue is set to FALSE because this parameter is used for generating the macro which indicates the type of communication(Asynchronous/Synchronous) is allowed on QSPI HW.

Table 132 Table: Violations reported by VSMD checker tool for TpsEcuc_08033

Rule ID:	TpsEcuc_08033
VSMD Node(s):	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiHwUnitSynchronous /AURIX2G/EcucDefs/Spi/SpiDriver/SpiSequence/SpiJobAssignment

(table continues...)

Table 132 (continued) Table: Violations reported by VSMD checker tool for TpsEcuc_08033

Description:	If the EcucModuleDef.postBuildVariantSupport is set to true and the postBuildVariantMultiplicity for an EcucParameterDef or an EcucAbstractReferenceDef in this EcucModuleDef in the StMD is set to true, the corresponding VSMD shall also set it to true.
Additional information:	1. `SpiJobAssignment`: The postBuildVariantMultiplicity is set to FALSE because this parameter is used to generate macro for total number of sequences configured per core. 2. `SpiHwUnitSynchronous`: The postBuildVariantMultiplicity is set to FALSE because this parameter is used to generate macro for total number of EB and IB channels configured and also IB buffer size allocated per core.

Table 133 Table: Violations reported by VSMD checker tool for TpsEcuc_08038

Rule ID:	TpsEcuc_08038
VSMD Node(s):	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiChannelType /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiDataWidth /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiIbNBuffers /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiHwUnit /AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiChannelList/SpiChannelAssignment /AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiHwUnitSynchronous
Description:	If the valueConfigClass attribute for an EcucParameterDef or an EcucAbstractReferenceDef is defined in the StMD,valueConfigClass.configClass for each valueConfigClass.configVariant in the VSMD shall be the same or higher as in the StMDwith respect to the selected subset defined by the actually implemented supportedConfigVariant of the corresponding EcucModuleDef.

(table continues...)

Table 133 (continued) Table: Violations reported by VSMD checker tool for TpsEcuc_08038

Additional information:	The value configuration class for the above configuration parameters is deviated(Changed to Precompile) from AUTOSAR due to the following reasons: 1. `SpiChannelType`: This parameter specifies the buffer type (External Buffer / Internal Buffer) used by the channel. The value configured to this parameter may change only at pre-compile time since it is coupled with buffer size which cannot be changed across the variants. This parameter is used to generate a macro for total IB channels, EB channels and total IB buffer size configured for a core. Hence, the valueConfigClass.configClass is pre-compile instead of post-build. 2. `SpiDataWidth`: This parameter specifies the width of the data to be transmitted in terms of bits. The QSPI supports the data width from 2 to 32 bits. The value configured to this parameter may change only at pre-compile time. This parameter is used to generate a macro for total IB channels, EB channels and total IB buffer size configured for a core. Hence, the valueConfigClass.configClass is pre-compile instead of post-build. 3. `SpiIbNBuffers`: This parameter specifies the buffer size of the IB channels. The value configured to this parameter may change only at pre-compile time. This parameter is used to generate a macro for total IB channels, EB channels and total IB buffer size configured for a core. Hence, the valueConfigClass.configClass is pre-compile instead of post-build. 4. `SpiHwUnit`: This parameter specifies the SPI hardware microcontroller peripheral allocated for transmission. The value configured to this parameter may change only at pre-compile time. This parameter is used to generate a macro for total IB channels, EB channels and total IB buffer size configured for a core. Hence, the valueConfigClass.configClass is pre-compile instead of post-build. 5. `SpiChannelAssignment`: This parameter specifies the channel linked to this Job container. The value configured to this parameter may change only at pre-compile time. This parameter is used to generate a macro for total IB channels, EB channels and total IB buffer size configured for a core. Hence, the valueConfigClass.configClass is pre-compile instead of post-build.

(table continues...)

Table 133 (continued) Table: Violations reported by VSMD checker tool for TpsEcuc_08038

	6. `SpiHwUnitSynchronous`: This parameter specifies if a job is Synchronous or Asynchronous. The value configured to this parameter may change only at pre-compile time. This parameter is used to generate a macro which indicates the type of communication is allowed on QSPI HW. Hence, the valueConfigClass.configClass is pre-compile instead of post-build (applicable for AUTOSAR 4.2.2 only).

Table 134 Violations reported by VSMD checker tool for EB03

Rule ID:	EB03
VSMD Node(s):	/AURIX2G/EcucDefs/Spi/SpiDemEventParameterRefs /AURIX2G/EcucDefs/Spi/SpiDemEventParameterRefs/SPI_E_HARDWARE_ERROR /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiDefaultData /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiCsSelection /AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiHwUnitSynchronous /AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiJobEndNotification /AURIX2G/EcucDefs/Spi/SpiDriver/SpiMaxChannel /AURIX2G/EcucDefs/Spi/SpiDriver/SpiMaxJob /AURIX2G/EcucDefs/Spi/SpiDriver/SpiMaxSequence /AURIX2G/EcucDefs/Spi/SpiDriver/SpiSequence/SpiSeqEndNotification /AURIX2G/EcucDefs/Spi/SpiGeneral/SpiKernelEcucPartitionRef /AURIX2G/EcucDefs/Spi/SpiGeneral/SpiMainFunctionPeriod
Description:	The StMD node has LOWER-MULTIPLICITY=0 and UPPER-MULTIPLICITY=1. The VSMD-node shall get the OPTIONAL-attribute instead of creating a list!
Additional information:	-

Table 135 Violations reported by VSMD checker tool for EB09

Rule ID:	EB09
VSMD Node(s):	/AURIX2G/EcucDefs/Spi
Description:	EB specific rule to check consistency of parameter postBuildVariantUsed.
Additional information:	-

Table 136 Violations reported by VSMD checker tool for EcucSws_1014

Rule ID:	EcucSws_1014
VSMD Node(s):	/AURIX2G/EcucDefs/Spi /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice /AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob /AURIX2G/EcucDefs/Spi/SpiGeneral
Description:	Additional vendor specific parameter definitions (using ParameterTypes), container definitions and references shall be added to the VSMD according to the alphabetical order.
Additional information:	-

Table 137 Violations reported by VSMD checker tool for EcucSws_1035

Rule ID:	EcucSws_1035

(table continues...)

Table 137 (continued) Violations reported by VSMD checker tool for EcucSws_1035

VSMD Node(s):	/AURIX2G/EcucDefs/Spi /AURIX2G/EcucDefs/Spi/SpiDemEventParameterRefs /AURIX2G/EcucDefs/Spi/SpiDemEventParameterRefs/SPI_E_HARDWARE_ERROR /AURIX2G/EcucDefs/Spi/SpiDriver /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiChannelId /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiChannelType /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiDataWidth /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiDefaultData /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiEbMaxLength /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiIbNBuffers /AURIX2G/EcucDefs/Spi/SpiDriver/SpiChannel/SpiTransferStart /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiBaudrate /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiCsIdentifier /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiCsPolarity /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiCsSelection /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiDataShiftEdge /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiDeviceEcucPartitionRef /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiEnableCs /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiHwUnit /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiShiftClockIdleLevel /AURIX2G/EcucDefs/Spi/SpiDriver/SpiExternalDevice/SpiTimeClk2Cs /AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob /AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiChannelList

(table continues...)

Table 137 (continued) Violations reported by VSMD checker tool for EcucSws_1035

	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiChannelList/SpiChannelAssignment
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiChannelList/SpiChannelIndex
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiDeviceAssignment
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiHwUnitSynchronous
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiJobEndNotification
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiJobId
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiJob/SpiJobPriority
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiMaxChannel
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiMaxJob
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiMaxSequence
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiSequence
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiSequence/SpiInterruptibleSequence
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiSequence/SpiJobAssignment
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiSequence/SpiSeqEndNotification
	/AURIX2G/EcucDefs/Spi/SpiDriver/SpiSequence/SpiSequenceId
	/AURIX2G/EcucDefs/Spi/SpiGeneral
	/AURIX2G/EcucDefs/Spi/SpiGeneral/SpiCancelApi
	/AURIX2G/EcucDefs/Spi/SpiGeneral/SpiChannelBuffersAllowed
	/AURIX2G/EcucDefs/Spi/SpiGeneral/SpiDevErrorDetect
	/AURIX2G/EcucDefs/Spi/SpiGeneral/SpiEcucPartitionRef
	/AURIX2G/EcucDefs/Spi/SpiGeneral/SpiHwStatusApi
	/AURIX2G/EcucDefs/Spi/SpiGeneral/SpiInterruptibleSeqAllowed
	/AURIX2G/EcucDefs/Spi/SpiGeneral/SpiKernelEcucPartitionRef
	/AURIX2G/EcucDefs/Spi/SpiGeneral/SpiLevelDelivered
	/AURIX2G/EcucDefs/Spi/SpiGeneral/SpiMainFunctionPeriod
	/AURIX2G/EcucDefs/Spi/SpiGeneral/SpiSupportConcurrentSyncTransmit

(table continues...)

Table 137 (continued) Violations reported by VSMD checker tool for EcucSws_1035

	/AURIX2G/EcucDefs/Spi/SpiGeneral/SpiUserCallbackHeaderFile /AURIX2G/EcucDefs/Spi/SpiGeneral/SpiVersionInfoApi /AURIX2G/EcucDefs/Spi/SpiPublishedInformation /AURIX2G/EcucDefs/Spi/SpiPublishedInformation/SpiMaxHwUnit
Description:	For Containers, Parameters and References elements UUID must be unique (also between StMD and VSMD).
Additional information:	-

Table 138 Violations reported by VSMD checker tool for EcucSws_2101

Rule ID:	EcucSws_2101
VSMD Node(s):	/AURIX2G/EcucDefs/Spi/POST_BUILD_VARIANT_USED
Description:	For each ConfigurationVariant supported by the ModuleDef, there must be one ImplementationConfigClass element. In VSMD, the ImplementationConfigClass is mandatory.
Additional information:	-

Table 139 Violations reported by VSMD checker tool for EcucSws_6003

Rule ID:	EcucSws_6003
VSMD Node(s):	/AURIX2G/EcucDefs/Spi
Description:	The SHORT-NAME of the AR-PACKAGEs of StMD and VSMD must be different to ensure a unique SHORT-NAME-path.
Additional information:	-

1.3.9.2 Limitations

This section describes the limitations of SPI driver.

Table 140 Known limitations

Reference	Limitation
DMA Transaction control set (TCS) Memory alignment	SPI driver uses the DMA to perform asynchronous data transfer which inturn uses TCS to store the configuration of DMA. Behaviour of the DMA / SPI driver is un-predictable if the TCS is not aligned to 32-byte boundary. Ex: `Spi_DmaTxControlSetArrayQSPI0` Rationale: DMA Hardware mandates TCS to be 32-byte aligned.

(table continues...)

Table 140 (continued) Known limitations

Reference	Limitation
Buffer alignment	All buffers that are used for transmission and reception of data shall be aligned to 4-byte boundary. Rationale: Odd address access would create a trap.
Sleep mode	Application should ensure that there is no ongoing SPI communication when CPU is requested for sleep. Rationale: If CPU enters sleep mode, clock to peripheral modules would be turned OFF.
Allowed IB buffer size per Core	The maximum allowed IB buffer size per Core is 65535. This means user must ensure the sum of generated macros `SPI_ASYNC_IB_BUFFER_SIZE_COREx` and `SPI_SYNC_IB_BUFFER_SIZE_COREx` (where 'x' represents the coreId) should not cross 65536(Word aligned). In case of violation, Code generation throws an error.
Cache and non-cache sections of memory	The Global buffers `Spi_TxIBBufferCorex`, `Spi_RxIBBufferCorex`, `Spi_DmaTxControlSetArrayQSPIx` and `Spi_DmaRxControlSetArrayQSPIx` are extensively used by DMA for data transfer and for TCS, so memory sections for these global must not be cached. Note that EB buffers that are used by applications are to be allocated in non-cached section of memory.
Unexpected Spurious interrupt during Error Handler	If SPI driver encounters multiple errors during the transmission of a sequence, it could result in `SPI_E_SAFETY_SPURIOUS_INTERRUPT` error reported by the driver. Application needs to ignore any such spurious interrupt generated during transmission. Rationale: The error ISR is triggered more than once due to the pending interrupt in SRC register. This is seen when SPI driver encounters multiple errors during the transmission of a sequence, for the same HW module.
SLSO on Power-on	On power-up, before the first frame is transmitted to any slave device CS lines are held in low state. Before the start of first frame all the SLSO pins are de-asserted and only the selected slave is asserted. For Successive frames SLSO levels will behave as expected and this is observed for hardware triggered SLSO only. No functional impact is expected from this behaviour.
DMA TRL (Transaction request lost) event	TRL events are safety feature of DMA hardware and cannot be suppressed. TRL event will be triggered on completion of channel transfer. If the configuration parameter `DmaTcsInterruptTransactionLoss` is disabled in DMA module, though TRL interrupt event is triggered by hardware, the DMA driver processes the event but does not provide any notification to user application.

(table continues...)

Table 140 (continued) Known limitations

Reference	Limitation
Handling OS calls invoked by SPI Interrupt service routine in CAT1 context	If the runtime API mode (SpiRuntimeApiMode) is configured to SPI_MCAL_USER1, the SPI interrupt handler uses OS service to access supervisor privileged SFRs. Due to this, if the SPI interrupt handlers are invoked in CAT1 context, the application software must handle the OS service call invoked from SPI handler.
Address and Data CRC	The SPI driver does not implement the CRC mechanism provided by the DMA driver. It is the responsibility of the upper layer to perform the integrity check of the data by using additional CRC mechanisms.
SPI level 1 implementation	The SPI driver supports only interrupt mode with DMA in Level 1 implementation (no support for polling mode provided).

Revision history

Table 141 Revision history

Date	Version	Description
2023-06-30	7.0	Document is released
2023-06-12	6.1	• Updated the section 1.1.2.Hardware-software mapping to include Mcal_Wrapper module and removed Dem module. • Updated the Section 1.1.3.C file Structure to include Mcal_Wrapper.h and removed Dem.h. • In Section 1.1.4.1.Integration with AUTOSAR stack, the following points are modified - Instead of DEM Module, Mcal_Wrapper Section is added. - Moved Runtime Error description from DET to Mcal_Wrapper Module. • In Section 1.3.8.Error Handling for SPI_E_HARDWARE_ERROR parameter Description is updated with Mcal_Wrapper API. • DEM has been modified to Production error in Section: 1.1.4.5.DMA support, 1.3.8.Error Handling. • Updated the section 1.3.9.1.1: Software Specification Deviations for Autosar requirements. - Added the Reference "For all requirements related to Production/Runtime errors". - Updated Description to add Mcal_Wrapper Module Information. • ASIL Level has been updated to Safety level in Section 1.3.3.Functions - APIs, 1.3.4. Notifications and Callbacks, 1.3.5. Scheduled Functions and 1.3.6. Interrupt service routines. • Updated the description of return type E_NOT_OK information in section 1.3.3.11.Spi_InitCheck() API.
2022-07-06	6.0	Document is released
2022-07-04	5.1	-SPI level 1 implementation Limitation is updated. -Default Value of following Configuration parameters has been changed 1.SpiDelayParamIdleLength 2.SpiDelayParamIdlePre 3.SpiDelayParamLeadLength 4.SpiDelayParamLeadPre 5.SpiDelayParamTrailLength 6.SpiDelayParamTrailPre
2021-11-18	5.0	Document is released
2021-11-12	4.1	- Config variant attribute table information is removed and added this information in 'Configuration interfaces' section - Updated example code snippet - Example usage section updated for the number and position of interrupts for a sample Spi sequence configuration

(table continues...)

Table 141 (continued) Revision history

2021-03-12	4.0	- Channellock SchM references removed from the integration hints and AOU sections. - Released version.
2021-02-25	3.0	- Limitation updated for TRL event behaviour. - Description of configuration parameter SpiSupportConcurrentSyncTransmit is updated. - Released version.
2020-11-23	2.0	Review comments Fixed. Released version.
2020-11-18	1.1	- Added reference to Dem_SetEventStatus API for AUTOSAR 4.4.0. - Limitation on Handling of DMA error during asynchronous transmission is removed. Information captured under 'Integration Hints section'. - Limitation on Status request removed, as not applicable. - Limitation added on Unexpected Spurious interrupt during Error Handler. - Generic AOU on Trap handler and protection of global variable removed. - Updated description for SPI_E_SAFETY_INVALID_PARAM error. - DMA channel configuration image updated under 'DMA Support' section. - Example usage section updated for Triggering of queued jobs in SyncTransmit API. - 'Reference documents' section updated to include AUTOSAR 4.4.0 SPI SWS.
2020-08-17	1.0	Document is released
2020-08-05	0.1	- Initial Version - SPI driver chapter moved from MC-ISAR_TC3xx_UM_Basic to this document. - Updated the order of the priority numbers for interrupts under Interrupt connections sub chapter. - Limitation updated for error handling in SPI driver when DMA is used for asynchronous transmission. - Added AMDC violations. - Added VSMD violations. - Unsupported HW features removed. All information captured under 'Hardware-Software mapping' section. - Limitation added when DMA is used by SPI for asynchronous communication. - Harmonization and format update in all the section.

Edition 2023-06-30

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-ocr1484806431059