Integration Manual

for MPC574XP SPI Driver

Document Number: IM30SPIASR4.0 Rev0003R2.0.0
Rev. 2.1

Contents

Section number **Title** **Page**

Chapter 1
Revision History

Chapter 2
Introduction

Chapter 3
Building the Driver

Chapter 4
Function Calls to Module

Chapter 5
Module Requirements

Section number Title Page

Chapter 6
Main API Requirements

Chapter 7
Memory Allocation

Chapter 8
Configuration Parameter Considerations

Chapter 9
Integration Steps

Chapter 10
ISR Reference

Chapter 11
External Assumptions for SPI driver

Chapter 1
Revision History

Table 1-1. Document Change History

Date	Version	Changed by	Change description
08-02-2013	1.1	Marius Rotaru	sPanther EAR 0.8.1 Release
14-08-2013	1.2	Sasu Adys-Bernard	Panther Beta 0.9.1 Release
24-10-2014	2.0	Nguyen Nguyen Van	Panther RTM 1.0.0 Release
12-12-2016	2.1	Nguyen Trung Thanh	Panther RTM 2.0.0 Release

Chapter 2
Introduction

This Integration Manual describes the integration requirements for Autosar SPI Driver for Freescale Semiconductor's MPC574XP microcontrollers .

The roadmap for the document is as follows:

Building the Driver : This section gives a brief overview of the build procedure (compiler,linker options and source files) and Plugins setup.

Function Calls to Module : This section lists the various function calls to modules during Start-up, Shutdown and Wake-up.

Module Requirements : This section specifies the various module requirements related to

- Exclusive areas to be defined in BSW scheduler
- Peripheral Hardware Requirements
- Specific interface to other modules
- ISR to configure within OS
- Dependencies with other AUTOSAR modules

Main API Requirements : This section specifies the requirements related to to the main SPI_main API and gives a brief overview of the main functions calls within BSW scheduler, API_Name Requirements and calls to notification functions, callbacks, callouts.

Memory Allocation : This section describes the memory allocation requirements namely the sections to be defined in MemMap.h and the linker command file.

Configuration Parameter Considerations : This section covers the various Pre Compile, Link Time and Post Build time configuration parameters.

Integration Steps : This section describes in brief the steps for integrating SPI module.

2.1 Supported Derivatives

The software described in this document is intented to be used with the following microcontroller devices of Freescale Semiconductor .

Table 2-1. MPC574XP Derivatives

Freescale Semiconductor	mpc5744p_144lqfp, mpc5743P_144lqfp, mpc5742P_144lqfp, mpc5741P_144lqfp, mpc5744P_257mapbga, mpc5743P_257mapbga, mpc5742P_257mapbga, mpc5741P_257mapbga

All of the above microcontroller devices are collectively named as MPC574XP .

2.2 Overview

AUTOSAR (AUTomotive Open System ARchitecture) is an industry partnership working to establish standards for software interfaces and software modules for automobile electronic control systems.

AUTOSAR

- paves the way for innovative electronic systems that further improve performance, safety and environmental friendliness.
- is a strong global partnership that creates one common standard: "Cooperate on standards, compete on implementation".
- is a key enabling technology to manage the growing electrics/electronics complexity. It aims to be prepared for the upcoming technologies and to improve cost-efficiency without making any compromise with respect to quality.
- facilitates the exchange and update of software and hardware over the service life of the vehicle.

2.3 About this Manual

This Technical Reference employs the following typographical conventions:

Boldface type: Bold is used for important terms, notes and warnings.

Italic font: Italic typeface is used for code snippets in the text. Note that C language modifiers such "const" or "volatile" are sometimes omitted to improve readability of the presented code.

Notes and warnings are shown as below:

Note

This is a note.

2.4 Acronyms and Definitions

Table 2-2. Acronyms and Definitions

Term	Definition
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSMI	Basic Software Make file Interface
CS	Chip Select
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
FIFO	First In First Out
MIDE	Multi Integrated Development Environment
MCU	Micro Controller Unit
LSB	Least Significant Bit
MSB	Most Significant Bit
RAM	Random Access Memory
SIU	Systems Integration Unit
SPI	Serial Peripheral Interface
SWS	Software Specification
VLE	Variable Length Encoding
XML	Extensible Markup Language
BSW	Basic Software
N/A	Not Applicable
ISR	Interrupt Service Routine
OS	Operating System
MCU	Microcontroller Unit
GUI	Graphical User Interface
PB Variant	Post Build Variant
PC Variant	Pre Compile Variant
LT Variant	Link Time Variant

2.5 Reference List

Table 2-3. Reference List

#	Title	Version
1	AUTOSAR 4.0 Rev0003SPI Driver Software Specification Document.	R4.0 Rev 0003
2	MPC574XP Reference Manual	Rev. 6, June 2016
3	Mask Set Errata for MASKSET 1N15P	Rev. SEP 2015

Chapter 3
Building the Driver

This section describes the source files and various compilers, linker options used for building the Autosar SPI driver for Freescale SemiconductorMPC574XP . It also explains the EB Tresos Studio plugin setup procedure.

3.1 Build Options

The SPI driver files are compiled using

- Green Hills Multi 6.1.4 / Compiler 2013.5.4 patch 9252
- Windriver DIAB DIAB_5_9_4_0-FCS_20140312_144007

The compiler, linker flags used for building the driver are explained below:

Note

The TS_T2D30M20I0R0 plugin name is composed as follow:

TS_T = Target_Id

D = Derivative_Id

M = SW_Version_Major

I = SW_Version_Minor

R = Revision

(i.e. Target_Id = 2 identifies PowerPC architecture and Derivative_Id = 30 identifies the MPC574XP)

3.1.1 GHS Compiler/Linker/Assembler Options

Table 3-1. Compiler Options

Option	Description
-cpu=ppc5746mz420	Selects target processor: ppc5746mz420
-ansi	Specifies ANSI C with extensions. This mode extends the ANSI X3.159-1989 standard with certain useful and compatible constructs.
-noSPE	Disables the use of SPE and vector floating point instructions by the compiler.
-Ospace	Optimize for size.
-sda=0	Enables the Small Data Area optimization with a threshold of 0.
-vle	Enables VLE code generation
-dual_debug	Enables the generation of DWARF, COFF, or BSD debugging information in the object file
-G	Generates source level debugging information and allows procedure call from debugger's command line.
--no_exceptions	Disables support for exception handling
-Wundef	Generates warnings for undefined symbols in preprocessor expressions
-Wimplicit-int	Issues a warning if the return type of a function is not declared before it is called
-Wshadow	Issues a warning if the declaration of a local variable shadows the declaration of a variable of the same name declared at the global scope, or at an outer scope
-Wtrigraphs	Issues a warning for any use of trigraphs
--prototype_errors	Generates errors when functions referenced or called have no prototype
--incorrect_pragma_warnings	Valid #pragma directives with wrong syntax are treated as warnings
-noslashcomment	C++ like comments will generate a compilation error
-preprocess_assembly_files	Preprocesses assembly files
-nostartfile	Do not use Start files
--short_enum	Store enumerations in the smallest possible type
-DAUTOSAR_OS_NOT_USED	-D defines a preprocessor symbol and optionally can set it to a value. AUTOSAR_OS_NOT_USED: By default in the package, the drivers are compiled to be used without Autosar OS. If the drivers are used with Autosar OS, the compiler option '-DAUTOSAR_OS_NOT_USED' must be removed from project options
-DUSE_SW_VECTOR_MODE	-D defines a preprocessor symbol and optionally can set it to a value. USE_SW_VECTOR_MODE: By default in the package, drivers are compiled to be used with interrupt controller configured to be in hardware vector mode. In case of AUTOSAR_OS_NOT_USED, the compiler option "-DUSE_SW_VECTOR_MODE" must be added to the list of compiler options to be used with interrupt controller configured to be in software vector mode.
-DDISABLE_MCAL_INTERMODULE_ASR_CHECK	-D defines a preprocessor symbol to disable the inter-module version check for AR_RELEASE versions. DISABLE_MCAL_INTERMODULE_ASR_CHECK: By default in the package, drivers are compiled to perform the inter-module version check as per Autosar BSW004. When the inter-module version check needs to be disabled then the DISABLE_MCAL_INTERMODULE_ASR_CHECK global define must be added to the list of compiler options.
-DGHS	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the GHS preprocessor symbol.

Table 3-2. Assembler Options

Option	Description
-cpu=ppc5746mz420	Selects target processor: ppc5746mz420

Table 3-3. Linker Options

Option	Description
-cpu=ppc5746mz420	Selects target processor: ppc5746mz420
-nostartfiles	Do not use Start files.
-vle	Enables VLE code generation

3.1.2 DIAB Compiler/Linker/Assembler Options

Table 3-4. Compiler Options

Option	Description
-tPPCE200Z4VEN:simple	Sets target processor to PPCE200Z4, generates ELF using EABI conventions, No floating point support (minimizes the required runtime), selects simple environment settings for Startup Module and Libraries
-Xdialect-ansi	Follow the ANSI C standard with some additions
-XO	Enables extra optimizations to produce highly optimized code
-g3	Generate symbolic debugger information and do all optimizations.
-Xsize-opt	Optimize for size rather than speed when there is a choice
-Xsmall-data=0	Set Size Limit for 'small data' Variables to zero.
-Xaddr-sconst=0x11	Specify addressing for constant static and global variables with size less than or equal to -Xsmall-const to far-absolute.
-Xaddr-sdata=0x11	Specify addressing for non-constant static and global variables with size less than or equal to -Xsmall-data in size to far-absolute.
-Xno-common	Disable use of the 'COMMON' feature so that the compiler or assembler will allocate each uninitialized public variable in the .bss section for the module defining it, and the linker will require exactly one definition of each public variable
-Xnested-interrupts	Allow nested interrupts
-Xdebug-dwarf2	Generate symbolic debug information in dwarf2 format
-Xdebug-local-all	Force generation of type information for all local variables
-Xdebug-local-cie	Create common information entry per module
-Xdebug-struct-all	Force generation of type information for all typedefs, struct, union and class types
-Xforce-declarations	Generates warnings if a function is used without a previous declaration
-ee1481	Generate an error when the function was used before it has been declared
-Xmacro-undefined-warn	Generates a warning when an undefined macro name occurs in a #if preprocessor directive
-Xlink-time-lint	Enable the checking of object and function declarations across compilation units, as well as the consistency of compiler options used to compile source files
-W:as:,-l	Pass the option '-l' (lower case letter L) to the assembler to get an assembler listing file

Table continues on the next page...

Table 3-4. Compiler Options (continued)

Option	Description
-Wa,-Xisa-vle	Instruct the assembler to expect and assemble VLE (Variable Length Encoding) instructions rather than BookE instructions.
-DAUTOSAR_OS_NOT_USED	-D defines a preprocessor symbol and optionally can set it to a value. AUTOSAR_OS_NOT_USED: By default in the package, the drivers are compiled to be used without Autosar OS. If the drivers are used with Autosar OS, the compiler option '-DAUTOSAR_OS_NOT_USED' must be removed from project options
-DUSE_SW_VECTOR_MODE	-D defines a preprocessor symbol and optionally can set it to a value. USE_SW_VECTOR_MODE: By default in the package, drivers are compiled to be used with interrupt controller configured to be in hardware vector mode. In case of AUTOSAR_OS_NOT_USED, the compiler option "-DUSE_SW_VECTOR_MODE" must be added to the list of compiler options to be used with interrupt controller configured to be in software vector mode.
-DDIAB	-D defines a preprocessor symbol and optionally can set it to a value. This one defines the DIAB preprocessor symbol.
-DDISABLE_MCAL_INTERMODULE_ASR_CHECK	-D defines a preprocessor symbol to disable the inter-module version check for AR_RELEASE versions. DISABLE_MCAL_INTERMODULE_ASR_CHECK: By default in the package, drivers are compiled to perform the inter-module version check as per Autosar BSW004. When the inter-module version check needs to be disabled then the DISABLE_MCAL_INTERMODULE_ASR_CHECK global define must be added to the list of compiler options.

NOTE

-Xc-new compiler option is used only to build the testes. The drivers delivered are compliant to ANSI standard.

Table 3-5. Assembler Options

Option	Description
-tPPCE200Z4VEG:simple	Sets target processor to PPCE200Z4, generates ELF using EABI conventions, All Single Hardware Floating Point (Single precision uses hardware, double precision is mapped to single precision), selects simple environment settings for Startup Module and Libraries
-g	Dump the symbols in the global symbol table in each archive file.
-Xisa-vle	Expect and assemble VLE (Variable Length Encoding) instructions rather than Book E instructions. The default code section is named .text_vle instead of .text, and the default code section fill "character" is set to 0x44444444 instead of 0. The .text_vle code section will have ELF section header flags marking it as VLE code, not Book E code.
-Xasm-debug-on	Generate debug line and file information
-Xdebug-dwarf2	Generate symbolic debug information in dwarf2 format

Table 3-6. Linker Options

Option	Description
-tPPCE200Z4VEG:simple	Sets target processor to PPCE200Z4, generates ELF using EABI conventions, All Single Hardware Floating Point (Single precision uses hardware, double precision is mapped to single precision), selects simple environment settings for Startup Module and Libraries
-Xelf	Generates ELF object format for output file

Table continues on the next page...

Table 3-6. Linker Options (continued)

Option	Description
-m6	Generates a detailed link map and cross reference table
-lc	Specifies to linker to search for libc.a
-Xlibc-old	Enables usage of legacy (pre-release 5.6) libraries
-Xlink-time-lint	Enable the checking of object and function declarations across compilation units, as well as the consistency of compiler options used to compile source files

3.2 Files required for Compilation

This section describes the include files required to compile, assemble (if assembler code) and link the SPI driver for MPC574XP microcontrollers.

To avoid integration of incompatible files, all the include files from other modules shall have the same AR_MAJOR_VERSION and AR_MINOR_VERSION, i.e. only files with the same AUTOSAR major and minor versions can be compiled.

SPI Files

- ..\Spi _ TS_T2D30M20I0R0 \include\Spi.h
- ..\Spi _ TS_T2D30M20I0R0 \include\Spi_IPW.h
- ..\Spi _ TS_T2D30M20I0R0 \include\Spi_IPW_Types.h
- ..\Spi _ TS_T2D30M20I0R0 \include\Spi_DSPI.h
- ..\Spi _ TS_T2D30M20I0R0 \include\Reg_eSys_DSPI.h
- ..\Spi_TS_T2D30M20I0R0\src\Spi.c
- ..\Spi_TS_T2D30M20I0R0\src\Spi_Dspi_Irq.c
- ..\Spi_TS_T2D30M20I0R0\src\Spi_DSPI.c

SPI Generated Files

- Spi_Cfg.c (For PC Variant) - This file should be generated by the user using a configuration tool for compilation
- Spi_Lcfg.c (For LT Variant) - This file should be generated by the user using a configuration tool for compilation
- Spi_PBcfg.c (For PB Variant) - This file should be generated by the user using a configuration tool for compilation
- Spi_Cfg.h - This file should be generated by the user using a configuration tool for compilation

Files from Base common folder

- ..\Base_ TS_T2D30M20I0R0 \include\Cer.h
- ..\Base_ TS_T2D30M20I0R0 \include\Compiler.h
- ..\Base_ TS_T2D30M20I0R0 \include\Compiler_Cfg.h

- ..\Base_ TS_T2D30M20I0R0 \include\ComStack_Cfg.h
- ..\Base_ TS_T2D30M20I0R0 \include\ComStack_Types.h
- ..\Base_ TS_T2D30M20I0R0 \include\Mcal.h
- ..\Base_ TS_T2D30M20I0R0 \include\MemMap.h
- ..\Base_ TS_T2D30M20I0R0 \include\Platform_Types.h
- ..\Base_ TS_T2D30M20I0R0 \include\Reg_eSys.h
- ..\Base_ TS_T2D30M20I0R0 \include\RegLockMacros.h
- ..\Base_ TS_T2D30M20I0R0 \include\SilRegMacros.h
- ..\Base_ TS_T2D30M20I0R0 \include\Soc_Ips.h
- ..\Base_ TS_T2D30M20I0R0 \include\Std_Types.h
- ..\Base_ TS_T2D30M20I0R0 \include\StdRegMacros.h
- ..\Base_ TS_T2D30M20I0R0 \generate_PC\include\StdRegMacros.h

Files from Dem folder:

- ..\Dem_ TS_T2D30M20I0R0 \include\Dem.h
- ..\Dem_ TS_T2D30M20I0R0 \include\Dem_Types.h
- ..\Dem_ TS_T2D30M20I0R0 \generate_PC\include\Dem_IntErrId.h

Files from Det folder:

- ..\Det_ TS_T2D30M20I0R0 \include\Det.h
- ..\Det_ TS_T2D30M20I0R0 \src\Det.c

Files from MCL folder (Only when DMA is used):

- ..\Mcl_ TS_T2D30M20I0R0 \include\CDD_Mcl.h

Files from SchM folder(Only when OS is used):

- ..\Rte_TS_T2D30M20I0R0\include\SchM_Spi.h
- ..\Rte_TS_T2D30M20I0R0\src\SchM_Spi.c

3.3 Setting up the Plug-ins

The SPI driver was designed to be configured by using the EB Tresos Studio (version EB tresos Studio 14.3.1 b140806-0327 or later.)

Location of various files inside the SPI module folder:

- VSMD (Vendor Specific Module Definition) file in EB tresos Studio XDM format:
    - ..\Spi_TS_T2D30M20I0R0\config\Spi.xdm

- Code Generation Templates for Pre-Compile time configuration parameters:
    - ..\Spi_TS_T2D30M20I0R0\generate_PC\src\Spi_Cfg.c
    - ..\Spi_TS_T2D30M20I0R0\generate_PC\include\Spi_Cfg.h

- Code Generation Templates for Post-Build time configuration parameters:

  - ..\Spi_TS_T2D30M20I0R0\generate_PB\src\Spi_PBcfg.c

- Code Generation Templates for Link time configuration parameters:
  - ..\Spi_TS_T2D30M20I0R0\generate_LT\src\Spi_Lcfg.c

Steps to generate the configuration:

1. Copy the module folders Spi _ TS_T2D30M20I0R0 , Mcu_ TS_T2D30M20I0R0, Mcl_ TS_T2D30M20I0R0 , Base_ TS_T2D30M20I0R0 , Resource_ TS_T2D30M20I0R0 , EcuM_ TS_T2D30M20I0R0 , Rte_ TS_T2D30M20I0R0 , Dem_ TS_T2D30M20I0R0 , Det_ TS_T2D30M20I0R0 into the Tresos plugins folder.
2. Set the desired Tresos Output location folder for the generated sources and header files.
3. Use the EB tresos Studio GUI to modify ECU configuration parameters values.
4. Generate the configuration files.

Dependencies

- **MCU** is required to use System Clock when clock source is used as Peripheral clock source to generate CAN Segment values.
- **RESOURCE** is required to select processor derivative. Current Can driver has support for the following derivatives, everyone having attached a Resource file: mpc5744p_144lqfp, mpc5743P_144lqfp, mpc5742P_144lqfp, mpc5741P_144lqfp, mpc5744P_257mapbga, mpc5743P_257mapbga, mpc5742P_257mapbga, mpc5741P_257mapbga .
- **DET** is required for signalling the development error detection (parameters out of range, null pointers, etc).
- **DEM** is required for signalling the production error detection (hardware failure, etc).
- **MCL** is required when DMA option is used.

Chapter 4
Function Calls to Module

4.1 Function Calls during Start-up

SPI shall be initialized during STARTUP phase of EcuM initialization. The API to be called for this is Spi _Init(). The MCU module should be initialized before the SPI is initialized. The API to be called for this purpose is Spi_Init(). The PORT and MCL (if the DMA option is used) modules shall be initialized before SPI is initialized.

4.2 Function Calls during Shutdown

SPI can be silenced by calling Spi_DeInit().

4.3 Function Calls during Wake-up

N/A

Chapter 5
Module Requirements

5.1 Exclusive areas to be defined in BSW scheduler

SPI_EXCLUSIVE_AREA_01: Used in function Spi_SyncTransmit, to protect the status of the given sequence result. Also it protects the global variable which contains the status of the Spi_SyncTransmit service. As stated by the Autosar, this service cannot be called when another sequence is during transmission, using this service.

SPI_EXCLUSIVE_AREA_02: Used in function Spi_SyncTransmit, to protect the status of the given sequence result. Also it protects the global variable which contains the status of the Spi_SyncTransmit service. As stated by the Autosar, this service cannot be called when another sequence is during transmission, using this service.

SPI_EXCLUSIVE_AREA_03: Used in the internal function Spi_ScheduleJob, protects the schedule mechanism for the situation when a scheduling operation determined by a pending Spi_AsyncTransmit() call may be preempted by a job scheduling requested by an ISR event. It also protect concurrent Spi_AsyncTransmit() calls to schedule in the same time different jobs on the same DSPI unit.

SPI_EXCLUSIVE_AREA_04: Used in the internal function Spi_ScheduleNextJob, protects the schedule mechanism for the situation when a scheduling operation determined by a pending Spi_AsyncTransmit() call may be preempted by a job scheduling requested by an ISR event.

SPI_EXCLUSIVE_AREA_05: Used in the internal function Spi_LockJobs, guaranties the atomicity of locking for the entire set of jobs belonging to an asynchronous sequence.

SPI_EXCLUSIVE_AREA_06: Used in the internal function Spi_UnlockRemainingJobs, guaranties the atomicity of unlocking for the entire set of jobs belonging to an asynchronous sequence.

Critical Region Exclusive Matrix

Below is the table depicting the exclusivity between different critical region IDs from the SPI driver. If there is an "X" in a table, it means that those 2 critical regions cannot interrupt each other.

The critical regions from interrupts are grouped in "Interrupt Service Routines Critical Regions (composed diagram)". If an exclusive area is "exclusive" with the composed "Interrupt Service Routines Critical Regions (composed diagram)" group, it means that it is exclusive with each one of the ISR critical regions.

Table 5-1. Exclusive Areas

	SPI_EXCLUSIVE_AREA_01	SPI_EXCLUSIVE_AREA_02	SPI_EXCLUSIVE_AREA_03	SPI_EXCLUSIVE_AREA_04	SPI_EXCLUSIVE_AREA_05	SPI_EXCLUSIVE_AREA_06	Interrupt Service Routines Critical Regions(composed diagram)
SPI_EXCLUSIVE_AREA_01	X	X			X	X	X
SPI_EXCLUSIVE_AREA_02	X	X			X	X	X
SPI_EXCLUSIVE_AREA_03			X	X	X	X	X
SPI_EXCLUSIVE_AREA_04			X	X	X	X	X
SPI_EXCLUSIVE_AREA_05	X	X	X	X	X	X	X
SPI_EXCLUSIVE_AREA_06	X	X	X	X	X	X	X
Interrupt Service Routines Critical Regions (composed diagram)	X	X	X	X	X	X	X

5.2 Peripheral Hardware Requirements

N/A

5.3 DMA configuration

This section applies only to DSPI units configured for asynchronous transmission (SpiPhyUnitSync not checked) and which use DMA for serializing/deserializing data between the hardware unit and the TX/RX buffers (SpiPhyUnitAsyncMethod = DMA).

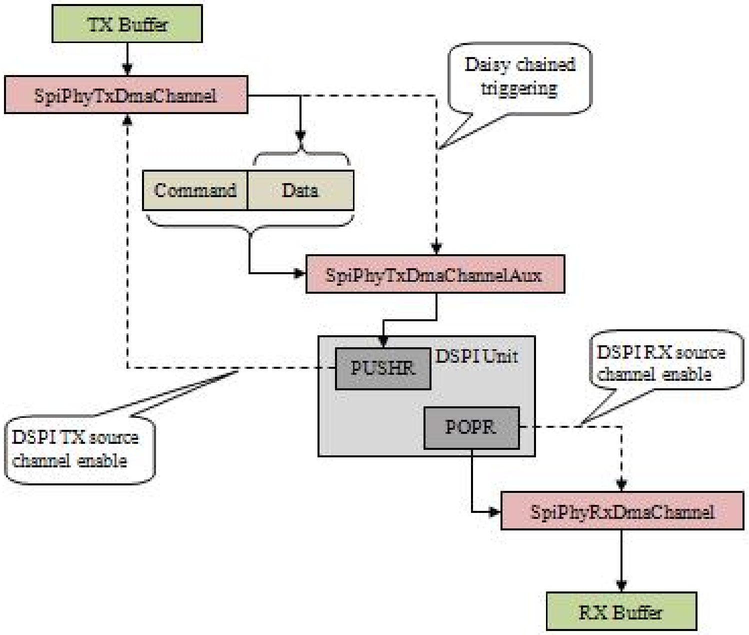


Figure 5-1. DMA transferring mode internal architecture

Each DSPI unit configured in DMA mode requires 3 distinct DMA channels from the **same** DMA Mux:

-**SpiPhyTxDmaChannel:** the master TX DMA channel used for reading data from the TX buffer, in order to prepare it for serializing into an internal dataframe; this channel is triggered by TX DSPI unit event and must be “wired” to “DSPI TX source” (configured inside the MCL module – MclDMA folder) – it must be a channel linkable to the external DMA TX source for the given DSPI unit.

-**SpiPhyTxDmaChannelAux:** secondary TX DMA channel used for sending internal prepared dataframes to the DSPI unit PUSH register. This channel transfers are triggered by SpiPhyTxDmaChannel acting as master DMA channel; no specific settings needed for it – this channel must not be linked to any peripheral DMA source and not be used in any other DMA transfer.

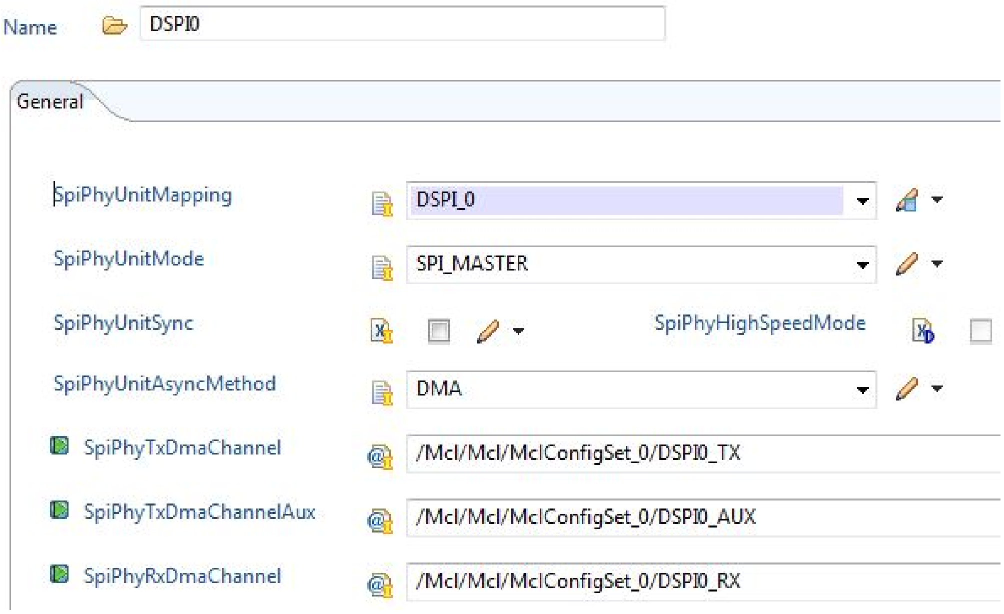
-**SpiPhyRxDmaChannel:** the RX DMA channel used for filling RX buffer with the deserialized data; this channel is triggered by RX DSPI unit event and must be “wired” to “DSPI RX source” (configured inside the MCL module – MclDMA folder) – it must be a channel linkable to the external DMA RX source for the given DSPI unit.

Note

- If DMA uses fixed priority arbitration, then **SpiPhyTxDmaChannelAux** priority must be greater than **SpiPhyTxDmaChannel** priority.
- If DMA uses round robin arbitration, no priority constraints are applied on **SpiPhyTxDmaChannelAux** and **SpiPhyTxDmaChannel** priority.

If the SPI driver is working in interrupt mode, the DMA Rx notification must be enabled for the specified Rx DMA channel. The name of the function to be used as a notification is Spi_Dspi_IsrRxDma_DSPI_X, where X is the number of DSPI unit used.

Next figures show an example of DMA configuration for DSPI0 unit.



Name: DSPI0

SpiPhyUnitMapping: DSPI_0

SpiPhyUnitMode: SPI_MASTER

SpiPhyUnitSync: □

SpiPhyHighSpeedMode: □

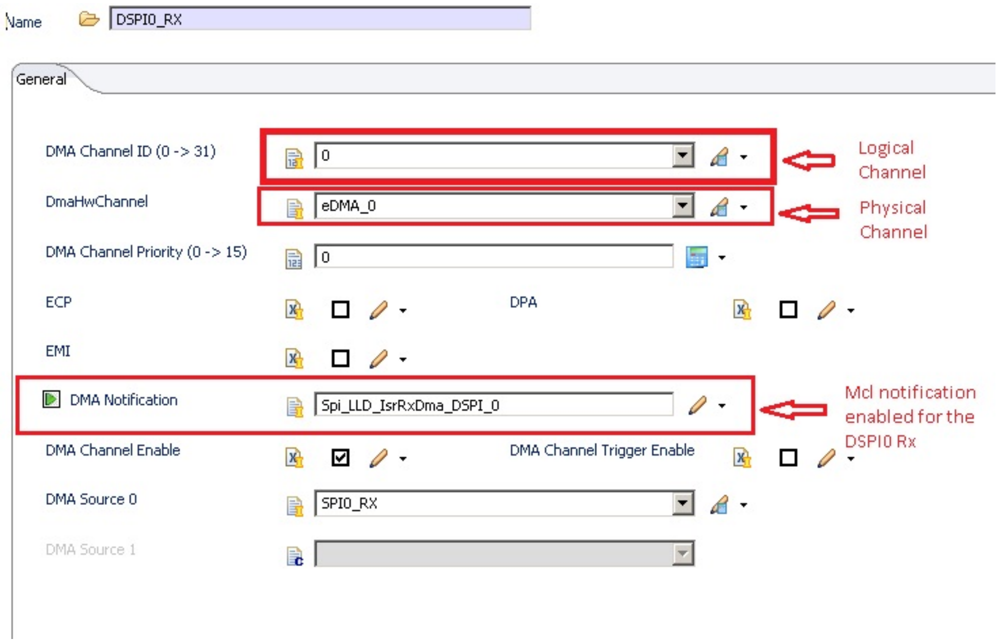
SpiPhyUnitAsyncMethod: DMA

SpiPhyTxDmaChannel: /Mcl/Mcl/MclConfigSet_0/DSPI0_TX

SpiPhyTxDmaChannelAux: /Mcl/Mcl/MclConfigSet_0/DSPI0_AUX

SpiPhyRxDmaChannel: /Mcl/Mcl/MclConfigSet_0/DSPI0_RX

Figure 5-2. DMA Configuration sample for DSPI0 Physical Unit - SPI module in tresos



Name DSPI0_RX

General

DMA Channel ID (0 -> 31): 0

DmaHwChannel: eDMA_0

DMA Channel Priority (0 -> 15): 0

ECP □ DPA □

EMI □

DMA Notification: Spi_LLD_IsrRxDma_DSPI_0

DMA Channel Enable ☑ DMA Channel Trigger Enable □

DMA Source 0: SPI0_RX

DMA Source 1

Logical Channel

Physical Channel

Mcl notification enabled for the DSPI0 Rx

Figure 5-3. DMA channels enable sample for DSPI0 Physical Unit - MCL module in tresos

5.4 ISR to configure within OS – dependencies

The following ISRs are used by the SPI driver and need to be assigned to a priority level. The interrupt vector numbers corresponding to PIO_FIFO for master&slave mode is as shown in bottom table. Int the master mode, interrupt occurs each time the EOQ bit in SR register arises, and occurs each time the RFDF bit in SR register arises with slave mode. The interrupt vector numbers of DMA channel configuration depend on the number of used DMA channel in EB Tresos configuration for SPI modules. Please see details in the reference manual.

Note

- Unused interrupts shouldn't be configured in the OS.

Table 5-2. SPI ISRs for DMA

Physical Unit	ISR Name
DSPI_0	Spi_Dspi_IsrRxDma_DSPI_0
DSPI_1	Spi_Dspi_IsrRxDma_DSPI_1
DSPI_2	Spi_Dspi_IsrRxDma_DSPI_2
DSPI_3	Spi_Dspi_IsrRxDma_DSPI_3

Table 5-3. SPI ISRs for PIO_FIFO in Master mode

Physical Unit	ISR Name	Hardware interrupt vector
DSPI_0	Spi_Dspi_IsrTCF_DSPI_0	262
DSPI_1	Spi_Dspi_IsrTCF_DSPI_1	271
DSPI_2	Spi_Dspi_IsrTCF_DSPI_2	280
DSPI_3	Spi_Dspi_IsrTCF_DSPI_3	289

Table 5-4. SPI ISRs for PIO_FIFO in Slave mode

Physical Unit	ISR Name	Hardware interrupt vector
DSPI_0	Spi_Dspi_IsrTCF_DSPI_0	263
DSPI_1	Spi_Dspi_IsrTCF_DSPI_1	272
DSPI_2	Spi_Dspi_IsrTCF_DSPI_2	281
DSPI_3	Spi_Dspi_IsrTCF_DSPI_3	290

Note: In case of AUTOSAR_OS_NOT_USED, the compiler option "-DUSE_SW_VECTOR_MODE" must be added to the list of compiler options to be used with interrupt controller configured to be in software vector mode.

5.5 ISR Macro

MCAL drivers use the ISR macro to define the functions that will process hardware interrupts. Depending on whether the OS is used or not, this macro can have different definitions:

a. OS is not used - AUTOSAR_OS_NOT_USED is defined:

i. If USE_SW_VECTOR_MODE is defined:

```
#define ISR(IsrName) void IsrName(void)
```

In this case, drivers’ interrupt handlers are normal C functions and the prolog/epilog handle the context save and restore.

ii. If USE_SW_VECTOR_MODE is not defined:

```
#define ISR(IsrName) INTERRUPT_FUNC void IsrName(void)
```

In this case, drivers’ interrupt handlers must save and restore the execution context.

Custom OS is used - AUTOSAR_OS_NOT_USED is not defined

```
#define ISR(IsrName) void OS_isr_##IsrName()
```

In this case, OS is handling the execution context when an interrupt occurs. Drivers' interrupt handlers are normal C functions.

Other vendor's OS is used - AUTOSAR_OS_NOT_USED is not defined. Please refer to the OS documentation for description of the ISR macro.

5.6 Other AUTOSAR modules - dependencies

Development Error Tracer:

This module is necessary for enabling Development error detection. The API function used is Det_ReportError(). The activation / deactivation of Development error detection is configurable using the

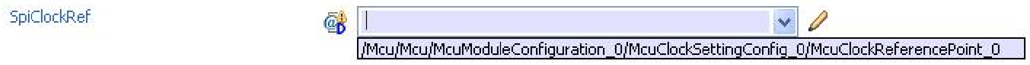
'SpiDevErrorDetect' configuration parameter.

Diagnostic Event Manager:

This module is necessary for enabling Production error detection. The API function used is Dem_ReportErrorStatus ().

Mcu:

The SPI reference clock is provided by MCU plugin. The reference is specified by the parameter SpiGeneral\SpiClockRef:



SpiClockRef

/Mcu/Mcu/McuModuleConfiguration_0/McuClockSettingConfig_0/McuClockReferencePoint_0

Figure 5-4. Spi reference clock provided by MCU plugin

Mcl:

For each DSPI in use, a transmit and a receive DMA channel need to be defined and routed through the DMA Multiplexer using MCL plugin. MCL should be initialized before SPI switch to DMA mode

The Table Table 5-5 shows an example DMA configuration. For more information, refer to section DMA configuration

Table 5-5. SPI DMA Channel Multiplexer

DMA Name	DMA Source
DSPI 0 Transmit DMA	0 (DSPI0.SpiPhyTxDmaChannel)
DSPI 0 Receive DMA	2 (DSPI0.SpiPhyRxDmaChannel)
DSPI 1 Transmit DMA	3 (DSPI1.SpiPhyTxDmaChannel)
DSPI 1 Receive DMA	5 (DSPI1.SpiPhyRxDmaChannel)
DSPI 2 Transmit DMA	6 (DSPI2.SpiPhyTxDmaChannel)
DSPI 2 Receive DMA	8 (DSPI2.SpiPhyRxDmaChannel)

PORT module: For each DSPI, the SCK, SOUT, SIN and CSx_y signals need to be configured. In the MPC574XP Reference manual there is an example of the pin configuration. Please refer to the Reference List .

5.7 Data Cache Restriction

In the DMA transfer mode, DMA transfers may issue cache coherency problems. To avoid possible coherency issues when **D-CACHE** is enabled, the user shall ensure that the buffers used as TCD source and destination are allocated in the **NON-CACHEABLE** area (by means of Memmap). Otherwise, the SPI driver has some dependencies. The user must follow the below things:

The first: Should not use the internal buffer for transmitter and receiver

The second: User must to put all variables, which were used for transmitter and receiver, to the **NON CACHEABLE** memory section in the RAM zone by the definition **SPI_START_SEC_VAR_<INIT_POLICY>_<ALIGNMENT>_NO_CACHEABLE** and **SPI_STOP_SEC_VAR_<INIT_POLICY>_<ALIGNMENT>_NO_CACHEABLE**

Chapter 6
Main API Requirements

6.1 Main functions calls within BSW scheduler

The function Spi_MainFunction_Handling() should be called periodically only if polling mode is enabled for Spi_AsyncTransmit() .

6.2 Calls to Notification Functions, Callbacks, Callouts

Call-back Notifications:

None.

User Notification:

The SPI Handler & Driver provides notifications per job and sequence in asynchronous mode. The notifications can be configured as pointers to user defined functions. If notification is not desired, the appropriate EndNotification field shall be left blank.

For asynchronous transmissions, job and sequences notifications are performed before the scheduling of the next job (contrary to the recommendation given by SPI088) . In this way, calls like Spi_SetupIB() or Spi_WriteIB() can be targeted on the next schedulable jobs, before the starting of the job transfer.

Chapter 7
Memory Allocation

7.1 Sections to be defined in MemMap.h

Table 7-1. Memory Allocation

Section name	Type of section	Description
SPI_START_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	Start of Memory Section for Config Data
SPI_STOP_SEC_CONFIG_DATA_UNSPECIFIED	Configuration Data	End of Memory Section for Config Data
SPI_START_SEC_CODE	Code	Start of memory Section for Code
SPI_STOP_SEC_CODE	Code	End of memory Section for Code
SPI_START_SEC_VAR_NO_INIT_32	Variables	Used for variables which have to be aligned to 32 bit. For instance used for variables of size 32 bit or used for composite data types: arrays, structs containing elements of maximum 32 bits. These variables are never cleared and never initialized by start-up code.
SPI_STOP_SEC_VAR_NO_INIT_32	Variables	End of above section.
SPI_START_SEC_VAR_NO_INIT_UNSPECIFIED	Variables	Used for variables, structures, arrays when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit. These variables are never cleared and never initialized by start-up code.
SPI_STOP_SEC_VAR_NO_INIT_UNSPECIFIED	Variables	End of above section.
SPI_START_SEC_VAR_NO_INIT_UNSPECIFIED_NO_CACHEABLE	Variables	Used for variables, structures, arrays when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit, and that have to be stored in a non-cacheable memory section. These variables are never cleared and never initialized by start-up code.
SPI_STOP_SEC_VAR_NO_INIT_UNSPECIFIED_NO_CACHEABLE	Variables	End of above section.
SPI_START_SEC_VAR_INIT_32	Variables	Used for variables which have to be aligned to 32 bit. For instance used for variables of size 32 bit or used for composite data types: arrays ,structs containing elements of

Table continues on the next page...

Table 7-1. Memory Allocation (continued)

Section name	Type of section	Description
		maximum 32 bits. These variables are initialized with values after every reset.
SPI_STOP_SEC_VAR_INIT_32	Variables	End of above section.
SPI_START_SEC_VAR_INIT_UNSPECIFIED	Variables	Used for variables, structures, arrays, when the SIZE (alignment) does not fit the criteria of 8,16 or 32 bit. These variables are initialized with values after every reset.
SPI_STOP_SEC_VAR_INIT_UNSPECIFIED	Variables	End of above section.
SPI_START_SEC_CONST_32	Constant Data	Used for constants that have to be aligned to 32 bit.
SPI_STOP_SEC_CONST_32	Constant Data	End of above section.

7.2 Linker command file

Memory shall be allocated for every section defined in MemMap.h.

Chapter 8
Configuration Parameter Considerations

1. Adding or removing Channels, Jobs or Sequences typically requires updating the application, rendering those parameters useless as PB option.
2. Changing the buffer type of a channel requires updating the application, rendering this parameter useless as PB option.
3. Changing the size for internal buffers post build requires a “PostBuild RAM” concept.
4. Please note that this is the peripheral clock frequency supplied to the DSPI.

8.1 Configuration Parameters

Configuration parameter class for Autosar SPI driver fall into the following variants as defined below:

Table 8-1. Configuration Parameters

Configuration Container	Configuration Parameters	Configuration Variant	Current Implementation
SpiDriver	SPI_MAX_CHANNEL	PC, LT or PB	Pre Compile (1)
	SPI_MAX_JOB	PC, LT or PB	Pre Compile (1)
	SPI_MAX_SEQUENCE	PC, LT or PB	Pre Compile (1)
SpiChannel	SpiChannelId	Pre-Compile all Variants	Pre Compile
	SpiChannelType	PC, LT or PB	Pre Compile (2)
	SpiIbNBuffers	PC, LT or PB	Pre Compile (3)
	SpiDataWidth	PC, LT or PB	Post Build
	SpiDefaultData	PC, LT or PB	Post Build
	SpiEbMaxlength	PC, LT or PB	Post Build
	SpiTransferStart	PC, LT or PB	Post Build
SpiDemEventParameterRefs	Spi_E_Hardware_Error	PC, LT or PB	Post Build
SpiExternalDevice	SpiSlaveMode	PC, LT or PB	Post Build
	TSBModeEnable	PC, LT or PB	Post Build
	ITSBModeEnable	PC, LT or PB	Post Build
	SpiBaudRate	PC, LT or PB	Post Build

Table continues on the next page...

Table 8-1. Configuration Parameters (continued)

	SpiEnableCs	PC, LT or PB	Post Build
	SpiCsIdentifier	PC, LT or PB	Post Build
	SpiCsPolarity	PC, LT or PB	Post Build
	SpiCsSelection	PC, LT or PB	Post Build
	SpiDataShiftEdge	PC, LT or PB	Post Build
	SpiHwUnit	PC, LT or PB	Post Build
	SpiShiftClockIdleLevel	PC, LT or PB	Post Build
	SpiTimeClk2Cs	PC, LT or PB	Post Build
	SpiTImeCs2Clk	Vendor specific	Post Build
	SpiTimeCs2Cs	Vendor specific	Post Build
	SpiCsContinuous	Vendor specific	Post Build
SpiJob	TSBModeEnable	Pre-Compile all Variants	Pre Compile
	ITSBModeEnable	Pre-Compile all Variants	Pre Compile
	SpiHwUnitSynchronous	PC, LT or PB	Post Build
	SpiJobEndNotification	PC, LT or PB	Post Build
	SpiJobStartNotification	PC, LT or PB	Post Build
	SpiJobId	Pre-Compile all Variants	Pre Compile
	SpiJobPriority	PC, LT or PB	Post Build
	SpiDeviceAssignment	PC, LT or PB	Post Build
	TSBFrameSize	PC, LT or PB	Post Build
	TS0_LEN	PC, LT or PB	Post Build
	TS1_LEN	PC, LT or PB	Post Build
	TS2_LEN	PC, LT or PB	Post Build
	TS3_LEN	PC, LT or PB	Post Build
	TS0_CONF	PC, LT or PB	Post Build
	TS1_CONF	PC, LT or PB	Post Build
	TS2_CONF	PC, LT or PB	Post Build
	TS3_CONF	PC, LT or PB	Post Build
	DsiCsIdentifier	PC, LT or PB	Post Build
	TransmitDataSource	PC, LT or PB	Post Build
	ChangeInDataTransfer	PC, LT or PB	Post Build
	DualReceiverSupport	Pre-Compile all Variants	Pre Compile
	SecondaryFrameSize	PC, LT or PB	Post Build
	SpiChannelAssignment	PC, LT or PB	Post Build
	SecondaryDsiCsIdentifier	PC, LT or PB	Post Build
	SpiSequenceId	Pre-Compile all Variants	Pre Compile
SpiSequence	SpiInterruptibleSequence	PC, LT or PB	Post Build
	SpiSeqEndNotification	PC, LT or PB	Post Build
	SpiJobAssignment	PC, LT or PB	Post Build
SpiGeneral	SpiCancelApi	Pre-Compile all Variants	Pre Compile

Table continues on the next page...

Table 8-1. Configuration Parameters (continued)

	SpiChannelBuffersAllowed	Pre-Compile all Variants	Pre Compile
	SpiDevErrorDetect	Pre-Compile all Variants	Pre Compile
	SpiHwStatusApi	Pre-Compile all Variants	Pre Compile
	SpiInterruptibleSeqAllowed	Pre-Compile all Variants	Pre Compile
	SpiLevelDelivered	Pre-Compile all Variants	Pre Compile
	SpiSupportConcurrentSyncTransmit	Vendor specific	Pre Compile
	SpiVersionInfoApi	Pre-Compile all Variants	Pre Compile
	SpiClockRef	Vendor specific	Pre Compile (4)
	SpiGlobalDmaEnable	Vendor specific	Pre Compile
	SpiSyncTransmitTimeout	Vendor specific	Pre Compile
	SpiOptimizeOneJobSequences	Vendor specific	Pre Compile
	SpiOptimizedSeqNumber	Vendor specific	Pre Compile
	SpiOptimizedChannelsNumber	Vendor specific	Pre Compile
SpiNonAUTOSAR	SpiAllowBigSizeCollections	Vendor specific	Pre Compile
	SpiEnableHWUnitAsyncMode	Vendor specific	Pre Compile
	SpiTSBModeSupport	Vendor Specific	Pre Compile
	SpiITSBModeSupport	Vendor Specific	Pre Compile
	SpiEnableDualClockMode	Vendor specific	Pre Compile
	SpiJobStartNotificationenable	Vendor specific	Pre Compile
	SpiForceDataType	Vendor specific	Pre Compile
	SpiDisableDemReportErrorStatus	Vendor specific	Pre Compile
SpiPhyUnit	SpiPhyUnitMapping	Vendor specific	Pre Compile
	SpiPhyUnitMode	Vendor specific	Post Build
	SpiPhyUnitSync	Vendor specific	Post Build
	SpiPhyUnitClockRef	Vendor specific	Post Build
	SpiPhyUnitAlternateClockRef	Vendor specific	Post Build
	SpiPhyUnitAsyncMethod	Vendor specific	Post Build
	SpiPhyTxDmaChannel	Vendor specific	Post Build
	SpiPhyTxDmaChannelAux	Vendor specific	Post Build
	SpiPhyRxDmaChannel	Vendor specific	Post Build

Chapter 9
Integration Steps

This section gives a brief overview of the steps needed for integrating SPI:

1. Generate the required SPI configurations. For more details refer to the section "Setting up the Plug-ins"
2. Allocate proper memory sections in MemMap.h and linker command file. For more details refer to the section "Memory Allocation"
3. Make sure all include files for compilation are as per the section "Files required for Compilation"
4. Map the ISRs to their vector locations. For more details refer to the section "ISR to configure within OS – dependencies"
5. Compile & build the SPI with all the dependent modules. For more details refer to the sections "Building the Driver" & "Table 5-3" & "Table 5-4" & "Table 5-2"

   Note:MCU shall be initialized with desired global Pre-scalar and system frequency before initializing the SPI driver. PORT shall be initialized with desired signal settings for DSPI.

Chapter 10
ISR Reference

ISR functions exported by the SPI driver.

10.1 Software specification

The following sections contains driver software specifications.

10.1.1 Function Reference

Functions of all functions supported by the driver are as per AUTOSAR SPI Driver software specification Version 4.0 Rev0003 .

10.1.1.1 Function Spi_Dspi_IsrTCF_DSPI_0

This function is the Transfer Complete for DSPI 0. An interrupt will be generated at every frame transmitted.

Details:

Non-AutoSar support function used by interrupt service routine of the transfer complete Rx for DSPI 0

Pre: Pre-compile parameter SPI_LEVEL_DELIVERED shall be LEVEL1 or LEVEL2. Pre-compile parameter DSPI_0_ENABLED shall be STD_ON.

Pre: Pre-compile parameter SPI_LEVEL_DELIVERED shall be LEVEL1 or LEVEL2. Pre-compile parameter DSPI_0_ENABLED shall be STD_ON.

Prototype: `void Spi_Dspi_IsrTCF_DSPI_0(void);`

10.1.1.2 Function Spi_Dspi_IsrTCF_DSPI_1

This function is the Transfer Complete for DSPI 1. An interrupt will be generated at every frame transmitted..

Details:

Non-AutoSar support function used by interrupt service routine of the transfer complete Rx for DSPI 1

Pre: Pre-compile parameter SPI_LEVEL_DELIVERED shall be LEVEL1 or LEVEL2. Pre-compile parameter DSPI_1_ENABLED shall be STD_ON.

Prototype: `void Spi_Dspi_IsrTCF_DSPI_1(void);`

10.1.1.3 Function Spi_Dspi_IsrTCF_DSPI_2

This function is the Transfer Complete for DSPI 2. An interrupt will be generated at every frame transmitted.

Details:

Non-AutoSar support function used by interrupt service routine of the transfer complete Rx for DSPI 2.

Pre: Pre-compile parameter SPI_LEVEL_DELIVERED shall be LEVEL1 or LEVEL2. Pre-compile parameter DSPI_2_ENABLED shall be STD_ON.

Prototype: `void Spi_Dspi_IsrTCF_DSPI_2(void);`

10.1.1.4 Function Spi_Dspi_IsrTCF_DSPI_3

This function is the Transfer Complete for DSPI 3. An interrupt will be generated at every frame transmitted.

Details:

Non-AutoSar support function used by interrupt service routine of the transfer complete Rx for DSPI 3.

Pre: Pre-compile parameter SPI_LEVEL_DELIVERED shall be LEVEL1 or LEVEL2. Pre-compile parameter DSPI_3_ENABLED shall be STD_ON.

Prototype: `void Spi_Dspi_IsrTCF_DSPI_3(void);`

10.1.1.5 Function Spi_Dspi_IsrRxDma_DSPI_0

This function is the DMA Rx notification for the DSPI 0.

Details:

Non-AutoSar support function used by MCL interrupt serive routine for the DMA Rx for DSPI 0

Pre: Pre-compile parameter SPI_LEVEL_DELIVERED shall be LEVEL1 or LEVEL2. Pre-compile parameter DSPI_0_ENABLED shall be STD_ON.

Prototype: `void Spi_Dspi_IsrRxDma_DSPI_0(void);`

10.1.1.6 Function Spi_Dspi_IsrRxDma_DSPI_1

This function is the DMA Rx notification for the DSPI 1.

Details:

Non-AutoSar support function used by MCL interrupt serive routine for the DMA Rx for DSPI 1

Pre: Pre-compile parameter SPI_LEVEL_DELIVERED shall be LEVEL1 or LEVEL2. Pre-compile parameter DSPI_1_ENABLED shall be STD_ON.

Prototype: `void Spi_Dspi_IsrRxDma_DSPI_1(void);`

10.1.1.7 Function Spi_Dspi_IsrRxDma_DSPI_2

This function is the DMA Rx notification for the DSPI 2.

Details:

Non-AutoSar support function used by MCL interrupt serive routine for the DMA Rx for DSPI 2

Pre: Pre-compile parameter SPI_LEVEL_DELIVERED shall be LEVEL1 or LEVEL2. Pre-compile parameter DSPI_2_ENABLED shall be STD_ON.

Prototype: `void Spi_Dspi_IsrRxDma_DSPI_2(void);`

10.1.1.8 Function Spi_Dspi_IsrRxDma_DSPI_3

This function is the DMA Rx notification for the DSPI 3.

Details:

Non-AutoSar support function used by MCL interrupt serive routine for the DMA Rx for DSPI 3

Pre: Pre-compile parameter SPI_LEVEL_DELIVERED shall be LEVEL1 or LEVEL2. Pre-compile parameter DSPI_3_ENABLED shall be STD_ON.

Prototype: `void Spi_Dspi_IsrRxDma_DSPI_3(void);`

Chapter 11
External Assumptions for SPI driver

The section presents requirements that must be complied with when integrating SPI driver into the application.

[SMCAL_CPR_EXT163]

<< If interrupts are locked a centralized function pair to lock and unlock interrupts shall be used. >>

[SPI027]

<< The SPI Handler/Driver's environment shall call the function Spi_ReadIB after a Transmit method call to have relevant data within IB Channel >>

[SPI037]

<< The SPI Handler/Driver's environment shall call the Spi_SetupEB function once for each Channel with EB declared before the SPI Handler/Driver's environment calls a Transmit method on them. >>

[SPI038]

<< The SPI Handler/Driver's environment shall call the function Spi_GetJobResult() to inquire whether the Job transmission has succeeded (SPI_JOB_OK) or failed (SPI_JOB_FAILED). >>

[SPI042]

<< The SPI Handler/Driver's environment shall call the function Spi_GetSequenceResult to inquire whether the full Sequence transmission has succeeded (SPI_SEQ_OK) or failed (SPI_SEQ_FAILED). >>

[SPI048]

<< The callback notifications Spi_JobEndNotification and Spi_SeqEndNotification shall have no parameters and no return value. >>

[SPI052]

<< For the IB Channels the Handler/Driver shall provide the buffering but it is not able to take care of the consistency of the data in the buffer during transmission. The size of the Channel buffer is fixed. >>

[SPI053]

<< For EB Channels the application shall provide the buffering and shall take care of the consistency of the data in the buffer during transmission. >>

[SPI077]

<< To transmit a variable number of data, it is mandatory to call Spi_SetupEB function to store new parameters within SPI Handler/Driver before each Spi_AsyncTransmit function call. >>

[SPI078]

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

[SPI080]

<< When using Interruptible Sequences, the caller must be aware that if the multiple Sequences access the same Channels, the data for these Channels may be overwritten by the highest priority Job accessing each Channel. >>

[SPI084]

<< If different Jobs (and consequently also Sequences) have common Channels, the SPI Handler/Driver's environment shall ensure that read and/or write functions are not called during transmission. Read and write functions can not guarantee the data integrity while Channel data is being transmitted. >>

[SPI085]

<< It is allowed to use the following API calls within the SPI callback notifications:

Spi_ReadIB

Spi_WriteIB

Spi_SetupEB

Spi_GetJobResult

Spi_GetSequenceResult

Spi_GetHWUnitStatus

Spi_Cancel

All other SPI Handler/Driver API calls are not allowed. >>

[SPI121]

<< The SPI Handler/Driver's environment shall configure the SpiInterruptibleSeqAllowed parameter (ON / OFF) in order to select which kind of Sequences the SPI Handler/Driver manages. >>

[SPI173]

<< The SPI Handler/Driver's environment shall call the function Spi_AsyncTransmit after a function call of Spi_SetupEB for EB Channels or a function call of Spi_WriteIB for IB Channels but before the function call Spi_ReadIB. >>

[SPI235]

<< If not applicable, the SPI Handler/Driver module's environment shall pass a NULL pointer to the function Spi_Init. >>

[SPI239]

<< SPI peripherals may depend on the system clock, prescaler(s) and PLL. Thus, changes of the system clock (e.g. PLL on ? PLL off) may also affect the clock settings of the SPI hardware. >>

[SPI244]

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

[SPI257]

<< The SPI Handler/Driver is not able to prevent the overwriting of these 'transmit' buffers by users during transmissions. >>

[SPI265]

<< For implement the call back function other modules are required to provide the routines in the expected manner. The callback notifications Spi_JobEndNotification and Spi_SeqEndNotification as function pointers defined within the initialization data structure (Spi_ConfigType). >>

[SPI280]

<< The buffer provided by the application for the SPI Handler Driver may have a different size. >>

NOTE

This referes in the context of External Buffer

[SPI287]

<< The SPI Handler/Driver's environment shall call this function to inquire whether the specified SPI Hardware microcontroller peripheral is SPI_IDLE or SPI_BUSY. >>

NOTE

This requirement refers to Spi_GetHWUnitStatus()

[SPI291]

<< If width of channel data handled by the hardware inferior to data width han-dled by the user means the data transmitted through the SPI Handler/Driver shall be according to the memory alignment separate the data as two part and send and re-ceive one by one. >>

[SPI298]

<< The operation Spi_Init is Non Re-entrant. >>

[SPI300]

<< The operation Std_ReturnType Spi_DeInit( ) is Non Re-entrant. >>

[SPI325]

<< The operation Spi_GetVersionInfo is Non Re-entrant. >>

[SPI335]

<< The operation Spi_SetAsyncMode is Non Re-entrant. >>

[SPI340]

<< The operation SpiJobEndNotification is Re-entrant. >>

[SPI341]

<< The operation SpiJobEndNotification is Re-entrant. >>

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTest, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. ARM, AMBA, ARM Powered, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and μVision are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. ARM7, ARM9, ARM11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

Document Number IM30SPIASR4.0 Rev0003R2.0.0
Revision 2.1