User's Manual

AUTOSAR MCAL R4.0.3

User's Manual

SPI Driver Component Ver.1.0.6
Embedded User's Manual

Target Device:
RH850/P1x

Renesas Electronics
www.renesas.com

Rev.0.02 Apr 2015

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti- crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority- owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

Abbreviations and Acronyms

Abbreviation / Acronym	Description
ANSI	American National Standards Institute
API	Application Programming Interface
ARXML/arxml	AutosaR eXtensible Mark-up Language
ASIC	Application Specific Integration Circuit
AUTOSAR	AUTomotive Open System Architecture
BSW	Basic SoftWare
CPU	Central Processing Unit
CS	Chip Select
CSIH/CSIG	Enhanced Queued Clocked Serial Interface.
DEM/Dem	Diagnostic Event Manager
DET/Det	Development Error Tracer
DMA	Direct Memory Access
EB	External Buffer
ECU	Electronic Control Unit
EDL	Extended Data Length
EEPROM	Electrically Erasable Programmable Read-Only Memory
FIFO	First In First Out
GNU	GNU's Not Unix
GPT	General Purpose Timer
HW	HardWare
IB	Internal Buffer
Id	Identifier
I/O	Input/Output
ISR	Interrupt Service Routine
KB	Kilo byte
MCAL	Microcontroller Abstraction Layer
MHz	Mega Hertz
MCU	Microcontroller unit
NA	Not Applicable
PLL	Phase Locked Loop
RAM	Random Access Memory
ROM	Read Only Memory
RTE	Run Time Environment
SPI	Serial Peripheral Interface
µs	Micro Seconds

Definitions

Term	Represented by
Sl. No.	Serial Number

Table Of Contents

List Of Figures

List Of Tables

Chapter 1 Introduction

The purpose of this document is to describe the information related to SPI Driver Component for Renesas P1x microcontrollers.

This document shall be used as reference by the users of SPI Driver Component. The system overview of complete AUTOSAR architecture is shown in the below Figure:

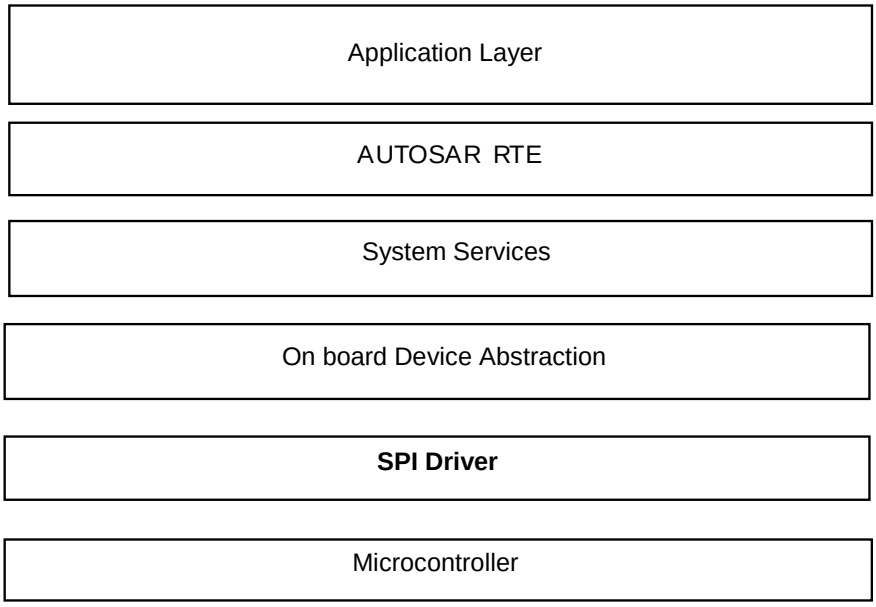

Figure 1-1 System Overview Of AUTOSAR Architecture

The SPI Driver is part of the Microcontroller Abstraction Layer (MCAL), the lowest layer of Basic Software in the AUTOSAR environment.

The Figure in the following page depicts the SPI Driver as part of layered AUTOSAR MCAL Layer:

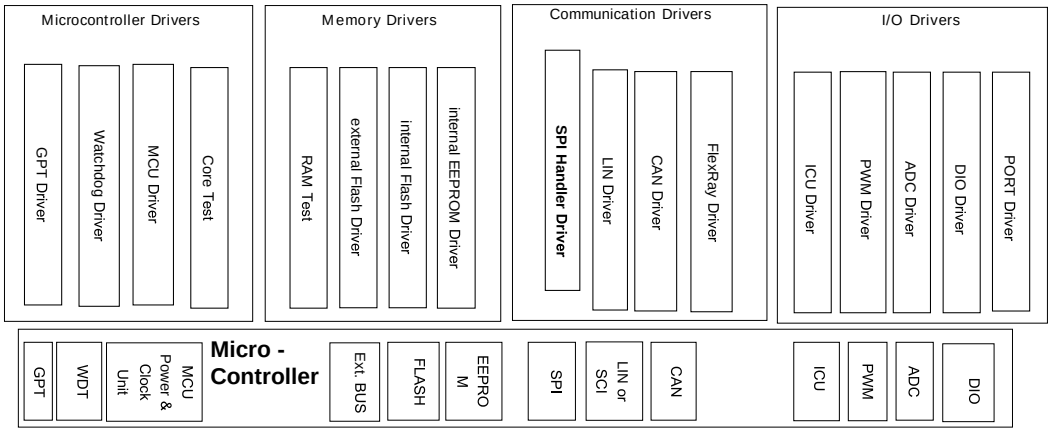

Figure 1-2 System Overview Of The SPI Driver In AUTOSAR MCAL Layer

The SPI Driver Component comprises Embedded software and the Configuration Tool to achieve scalability and configurability.

The SPI Driver component code Generation Tool is a command line tool that accepts ECU configuration description files as input and generates source and header files. The configuration description is an ARXML file that contains information about the configuration for SPI Driver. The tool generates the Spi_PBcfg.c, Spi_Lcfg.c, Spi_Cfg.h and Spi_Cbk.h.

The SPI driver provides services for reading from and writing to devices connected through SPI buses. It provides access to SPI communication to several users (For example, EEPROM, I/O ASICs). It also provides the required mechanism to configure the on-chip SPI peripheral.

1.1. Document Overview

The document has been segmented for easy reference. The table below provides user with an overview of the contents of each section:

Section	Contents
Section 1 (Introduction)	This section provides an introduction and overview of SPI Driver Component.
Section 2 (Reference Documents)	This section lists the documents referred for developing this document.
Section 3 (Integration And Build Process)	This section explains the folder structure, Makefile structure for SPI Driver Component. This section also explains about the Makefile descriptions, Integration of SPI Driver Component with other components, building the SPI Driver Component along with a sample application.
Section 4 (Forethoughts)	This section provides brief information about the SPI Driver Component, the preconditions that should be known to the user before it is used, memory modes, data consistency details, deviation list and Support For Different Interrupt Categories.
Section 5 (Architecture Details)	This section describes the layered architectural details of the SPI Driver Component.
Section 6 (Register Details)	This section describes the register details of SPI Driver Component.
Section 7 (Interaction Between User And SPI Driver Component)	This section describes interaction of the SPI Driver Component with the upper layers.
Section 8 (SPI Driver Component Header And Source File Description)	This section provides information about the SPI Driver Component source files is mentioned. This section also contains the brief note on the tool generated output file.
Section 9 (Generation Tool Guide)	This section provides information on the SPI Driver Component Code Generation Tool.
Section 10 (Application Programming Interface)	This section explains all the APIs provided by the SPI Driver Component.
Section 11 (Development And Production Errors)	This section lists the DET and DEM errors.
Section 12 (Memory Organization)	This section provides the typical memory organization, which must be met for proper functioning of component.
Section 13(P1M Specific information)	This section provides P1M specific information also the information about linker compiler and sample application.
Section 14 (Release Details)	This section provides release details with version name and base version.

Chapter 2 Reference Documents

Sl. No.	Title	Version
1.	Autosar R4.0 AUTOSAR_SWS_SPIHandlerDriver.pdf	3.2.0
2.	AUTOSAR BUGZILLA (http://www.autosar.org/bugzilla) Note: AUTOSAR BUGZILLA is a database, which contains concerns raised against information present in AUTOSAR Specifications.	-
3.	r01uh0436ej0070_rh850p1x.pdf	0.70
4.	Autosar R4.0 AUTOSAR_SWS_CompilerAbstraction.pdf	3.2.0
5.	Autosar R4.0 AUTOSAR_SWS_MemoryMapping.pdf	1.4.0
6.	Autosar R4.0 AUTOSAR_SWS_PlatformTypes.pdf	2.5.0
7.	Autosar R4.0 AUTOSAR_BSW_MakefileInterface.pdf	0.3

Chapter 3 Integration And Build Process

In this section the folder structure of the SPI Driver Component is explained. Description of the Makefiles along with samples is provided in this section.

Remark The details about the C Source and Header files that are generated by the SPI Driver Generation Tool are mentioned in the "AUTOSAR_SPI_Tool_UserManual.pdf".

3.1. SPI Driver Component Makefile

The Makefile provided with the SPI Driver Component consists of the GNU Make compatible script to build the SPI Driver Component in case of any change in the configuration. This can be used in the upper level Makefile (of the application) to link and build the final application executable.

3.1.1. Folder Structure

The files are organized in the following folders:

Remark Trailing slash '\' at the end indicates a folder

X1X\common_platform\modules\spi\src\ Spi_Driver.c
\ Spi.c
\ Spi_Scheduler.c
\Spi_Irq.c
\Spi_Ram.c
\Spi_Version.c

X1X\common_platform\modules\spi\include\Spi_Driver.h
\Spi.h
\Spi_Scheduler.h
\Spi_Irq.h
\Spi_LTTypes.h
\Spi_PBTypes.h
\Spi_Ram.h
\Spi_Version.h
\Spi_Types.h

X1X\P1x\modules\spi\Sample_application\<SubVariant>\make\<Compiler>
\App_SPI_P1M_Sample.mak

X1X\P1x\modules\spi\Sample_application\<SubVariant>\obj\ <compiler>

X1X\common_platform\modules\spi\generator\Spi_X1x.exe

X1X\P1x\common_family\generator
\Sample_Application_P1x.trxml
\P1x_translation.h

X1X\P1x\modules\spi\generator
\R403_SPI_P1x_BSWMDT.arxml

X1X\P1x\modules\spi\user_manual

(User manuals will be available in this folder)

Notes:

1. <Compiler> can be ghs.
2. <SubVariant> can be P1M.
3. <AUTOSAR_version> can be 4.0.3.

Chapter 4 Forethoughts

4.1. General

Following information will aid the user to use the SPI Driver Component software efficiently:

- SPI Driver component does not take care of setting the registers which configure clock, prescaler and PLL.
- SPI Driver component handles only the Master mode.
- SPI Driver component supports full-duplex mode.
- The chip select is implemented using the microcontroller pins and it is configurable.
- The required initialization of the port pins configured for chip select has to be performed by the Port Driver Component.
- The microcontroller pins used for chip select is directly accessed by the SPI Driver component without using the APIs of DIO module.
- Maximum number of channels and sequences configurable is 256 and job is 65536.
- The scope is restricted to post-build with multiple configuration sets.
- The identifiers for channels, jobs and sequences entered by the user should start from 0 and should be continuous.
- The width of the transmitted data unit is configurable and the valid values are 8 bits to 32 bits.
- The number of channels, jobs and sequences should be same across multiple configuration sets.
- The channels, jobs and sequences cannot be deleted or added at post-build time.
- The SPI hardware unit cannot be deleted or added at post–build time. But, the reassignment of the SPI hardware units to different jobs is possible at post-build time.
- The DMA unit cannot be deleted or added at post–build time. But, the reassignment of DMA units to the SPI hardware units is possible at post-build time.
- When the level of scalable functionality is configured as 2, then two SPI buses using separate hardware units are required. In this case, the SPI bus dedicated for synchronous transmission is configurable.
- When the level of scalable functionality is configured as 2, two modes of asynchronous communication using polling or interrupt mechanism are possible. These modes are selectable during execution time.
- When the level of scalable functionality is configured as 1 or 2, If interrupt mechanism is selected during execution time, the transmission and reception will be performed using the on-chip DMA unit only if the DMA mode is enabled through the configuration.
- The LEVEL 2 SPI Handler is specified for microcontrollers that have to provide at least two SPI busses using separated hardware units. Otherwise, using this level of functionality makes no sense.

- When Level Delivered is 0 and 2, the memory mode configured for jobs linked for the synchronous sequence shall be always Direct Access Mode only.
- If user configures 32 bit IB and EB channels and additionally configures DMA in direct access mode there will be a generator error message.
- When the SPI driver is configured in Level 2 (SpiLevelDelivered) and the DMA is also configured (SpiDmaMode), then the asynchronous mode needs to be set for interrupt mode using the API Spi_SetAsyncMode.
- The SPI DMA type is specified by the parameter SPI_DMA_TYPE_USED.

  Note: The DMA will work whenever the DMA access for the LOCAL RAM, which is having PE guard protection is enabled (this can be done by configuring the PE guard registers.)
- Direct Access mode can be effectively used in case of sequence having channels and buffers of significantly different properties.
- Double Buffer mode can be effectively used in case of sequence having more number of jobs, channels and buffers with same hardware properties for continuous transmission of data. For double buffer mode only usage of internal buffers is allowed. FIFO mode can be effectively used at the time of transmit/receive of large amount of data. FIFO mode can also be used in case of sequence having lesser number of jobs and having more channels and buffers.
- In case size of buffers is more than the hardware buffer size i.e. 128 words, an interrupt will occur after every 128 words are transmitted where the hardware buffer will be loaded with the remaining buffers to be transmitted.
- In a particular configurations where CSIH HW units are configured, Spi_Init function must be called before Port_Init function.
- Only if "SpiCsInactive" parameter is set to "true", the PWR bit in CSI hardware will be cleared for that hardware unit, so setting "false" value can lead to unnecessary power consumption.
- When “SpiCsIdleEnforcement” is set to true for the jobs configured for CSIH Hw units, the value configured for "SpiCsInactive" will not have any impact in actual Chip Select behavior".
- The parameter "SpiCsIdleEnforcement" influences the behavior of idle level of the chip select during data transfer and after the transmission of a job.
- When the parameter 'SpiCsIdleEnforcement' is configured as false, the corresponding chip select is deactivated before every channel transmission and stays active after transmission until another job with different CS is transmitted.
- When the parameter 'SpiCsIdleEnforcement' is configured as true, the chip select is deactivated after job transmission. An idle phase of CS is inserted between transmissions of two data buffers. The duration of idle state of the chip select between the channels transmissions will be less than duration of idle state of the chip select between single data of each channel.
- In CSIG,CS is active during the whole job transmission independently of data and is set to inactive state after job is finished.

Table 4-1 Table for Chip Select behavior

Figure	SpiCSInactiveAfterlastdata	SpiCsIdleEnforcement
4-1	FALSE	TRUE
4-2	TRUE	TRUE
4-3	TRUE	FALSE
4-4	FALSE	FALSE

Note: In the below figures, the signal represented in Yellow is the clock signal and the Blue signal is the chip select signal.

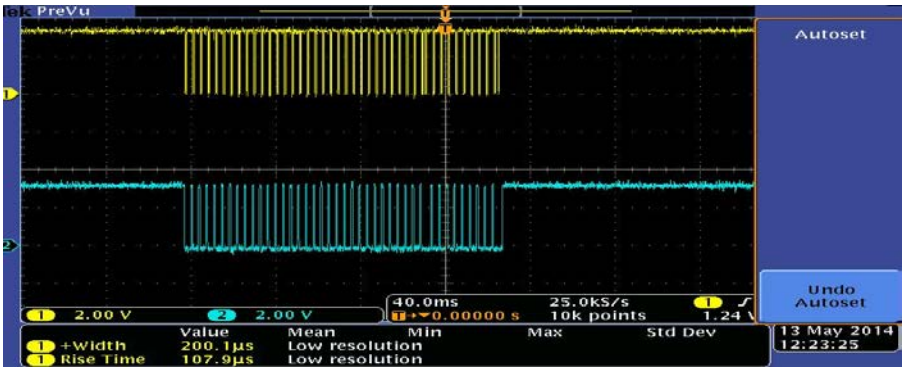

Figure 4-1 Chip select behavior when SpiCSInactiveAfterlastdata is False and SpiCsIdleEnforcement is True

Note: If 'SpiCsIdleEnforcement' is TRUE, Chip select will get deactivated after transmission is over, even if 'SpiCSInactiveAfterlastdata' is configured as FALSE.

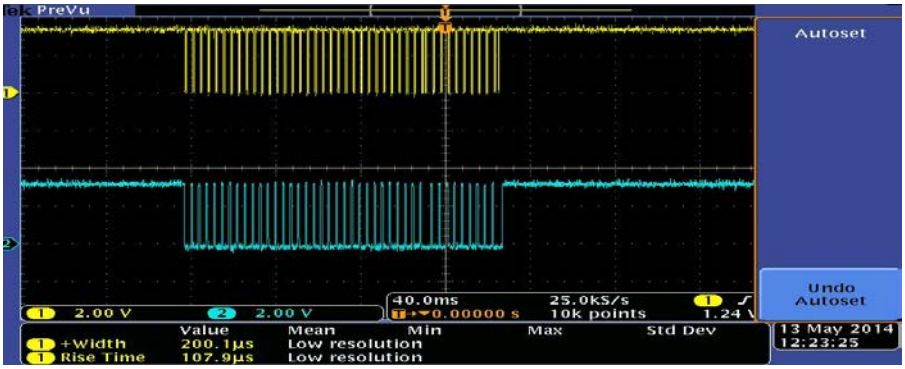

Figure 4-2 Chip select behavior when SpiCSInactiveAfterlastdata is True and SpiCsIdleEnforcement is True

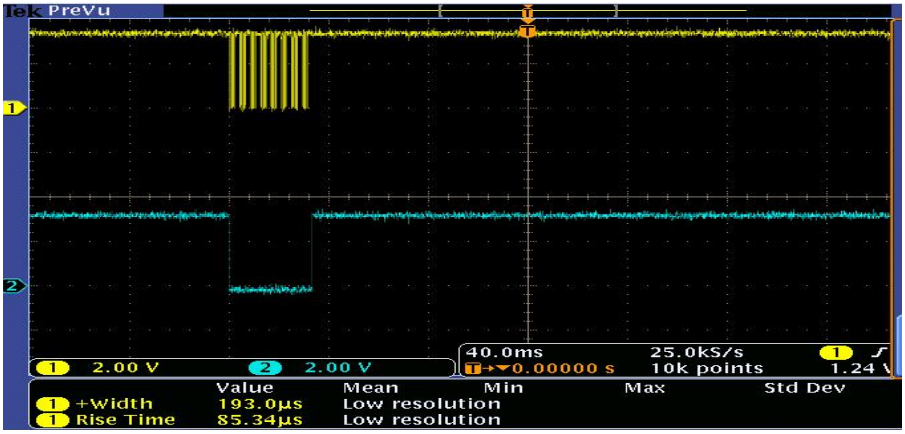

Figure 4-3 Chip select behavior when SpiCSInactiveAfterlastdata is True and SpiCsIdleEnforcement is False

Note:

1. The expected CS behavior may not be observed at high baud rates in case of Asynchronous transmission using Direct Access Mode, due to general limitation of the serial controllers.

2. CS state can be held for Asynchronous transmission by using buffer modes like FIFO.

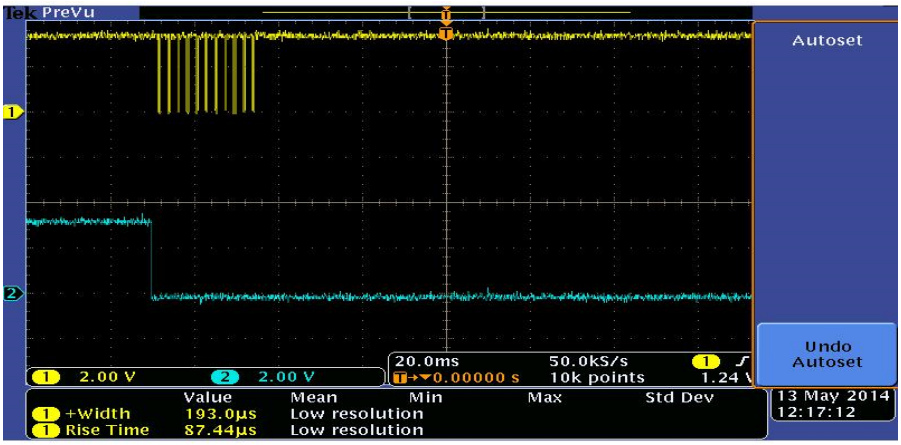

Figure 4-4 Chip select behavior when SpiCSInactiveAfterlastdata is False and SpiCsIdleEnforcement is False

This information is valid only for DIRECT ACCES MODE.

- For availability of Data Consistency Check on the port pins, please refer respective microcontroller user manual.
- Sequences assigned to a hardware channel (CSIHx) which is configured to work with transmit only memory mode can be an interruptible or non-interruptible sequence (specified by the parameter SpiInterruptibleSequence). However, even if the sequence is non-interruptible, it can still be interrupted by CPU-controlled high priority communication functionality. I.e. the parameter SpiInterruptibleSequence is valid only for software interruption.
- Each of the high priority sequences shall refer to a unique chip select line. These lines shall not be referred by any of the low priority sequences too.
- In order to support DEEPSTOP functionality without resetting the microcontroller, the re initialization of the Driver using Spi_Init API is supported. To achieve this functionality the 'SPI_E_ALREADY_INITIALIZED' Det error check is to be suppressed using ‘SpiAlreadyInitDetCheck’ parameter when DET is enabled. When DET is disabled there is no impact of “SpiAlreadyInitDetCheck” parameter.
- In a Hardware channel which has sequences working with transmit only mode and is of high priority, if there is a request for transmission of high priority sequence, then it will interrupt an ongoing sequence with transmit only mode if the sequence is non-interruptible.
- When the sequence is getting transmitted with transmit only mode, if there is a request for high priority sequence, the ongoing sequence will be interrupted after the ongoing job is finished and memory mode will switch from transmit only mode to direct access mode automatically for high priority sequence transmission and after its completion, the interrupted sequence will resume transmission in transmit only mode.
- MCTL1, MCTL2 and CSIHnMRWP0 registers are allowed to be accessed when there is an ongoing communication only when PWR is set.

- Manual transmission is possible only in Direct Access and FIFO modes. However user has to implement his own ISRs for SPI. In case he wants to use Renesas SPI driver transmission in parallel, he has to call Renesas SPI ISRs functions from his custom ISRs (e.g. use different interrupt category mode).

- The file Interrupt_VectorTable.c provided is just a Demo and not all interrupts will be mapped in this file. So the user has to update the Interrupt_VectorTable.c as per his configuration.
- The notifications should be called from user's complex driver ISRs

- High values for parameter 'SpiCsHoldTiming'should not be used with Synchronous Transmit function but if it is used, user should make sure that next consecutive SPI action happens after CS hold time expired.
- The parameter SpiTimeOut generates a scalar value that decides the number of times a loop will be executed while polling. If exceeded the loop breaks reporting a production error.

  This information is valid only for Static Configuration

- The parameter SpiPersistentHWConfiguration decides whether Hardware configuration is static or dynamic. This is applicable for both CSIG and CSIH and both Synchronous and Asynchronous communication and all memory modes.
- If SpiPersistentHWConfiguration is "True", then HW configuration is static (configuration is performed in the function Spi_Init ()function and not during each transmission.

- Static Configuration, allows the user to manually start transmission without invoking SPI module APIs after Spi driver was initialized.

- In Static configuration, all parameters in channel/job/external devices containers linked to a hardware unit should be same. Refer Table 4-2, 4-3 and 4-4 for the list of parameters

Table 4-2 List of parameters in Channel container that are linked to the registers.

Parameter in channel container	Registers linked CSIH-CSIG	
SpiDataWidth	CSIHnCFGx.CSIHnDLSx	CSIHnCFGx0.CSIHnDLS[3:0]
SpiTransferStart	CSIHnCFGx.CSIHnDIRx	CSIHnCFGx0.CSIHnDLS[3:0]

Table 4-3 List of parameters in Job container that are linked to the registers.

Parameter in job container	Registers linked CSIH-CSIG	
SpiPortPinSelect	CSIHnTXOW.CSIHnCSx CSIHnCTL1.CSIHnCSx	-

Table 4-4 List of parameters in External Device container that are linked to the registers.

Parameter in channel container	Registers linked	
	CSIH	CSIG
SpiCsPolarity	CSIHnCTL1.CSIHnCSx	-
SpiCsInactive	CSIHnCTL1.CSIHnCSRI	-
SpiCsIdleEnforcement	CSIHnCFGx.CSIHnIDLx	-
SpiCsIdleTiming	CSIHnCFGx.CSIHnIDx[2:0]	-
SpiCsHoldTiming	CSIHnCFGx.CSIHnHDx[3:0]	-
SpiCsInterDataDelay	CSIHnCFGx.CSIHnINx[3:0]	-
SpiCsSetupTime	CSIHnCFGx.CSIHnSPx[3:0]	-
SpiDataShiftEdge	CSIHnCFGx.CSIHnDAPx	CSIGnCFG0.CSIGnDAP
SpiShiftClockIdleLevel	CSIHnCTL1.CSIHnCKR	CSIGnCTL1.CSIGnCKR
SpiBaudrateConfiguration	CSIHnBRSy.CSIH0BRS[11:0]	CSIGnCTL2.CSIGnBRS
SpiBaudrateRegisterSelect	CSIHnCFGx.CSIHnBRSSx[11:0]	-
SpiInputClockSelect	CSIHnCTL2.CSIHnPRS[2:0]	CSIGnCTL2.CSIGnPRS[2:0]
SpiInterruptDelayMode	CSIHnCTL1.CSIHnSIT	CSIGnCTL1.CSIGnSLIT
SpiParitySelection	CSIHnCFGx.CSIHnPSx[1:0]	CSIGnCFG0.CSIGnPS[1:0]
SpiFifoTimeOut	CSIHnMCTL0.CSIHnTO[4:0]	-
SpiBroadcastingPriority	CSIHnCFGx.CSIHnRCBx	-

4.2. Preconditions

Following preconditions have to be adhered by the user, for proper functioning of the SPI Driver Component:

- The Spi_Lcfg.c, Spi_PBcfg.c, Spi_Cbk.h and Spi_Cfg.h files generated by the SPI Driver Component Code Generation Tool must be compiled and linked along with SPI Driver Component source files.
- The application has to be rebuilt, if there is any change in the Spi_Lcfg.c, Spi_PBcfg.c, Spi_Cbk.h and Spi_Cfg.h files generated by the SPI Driver Component Generation Tool.
- File Spi_PBcfg.c generated for single configuration set or multiple configuration sets using SPI Driver Component Generation Tool can be compiled and linked independently.

- The authorization of the user for calling the software triggering of a hardware reset is not checked in the SPI Driver. This is the responsibility of the upper layer.
- The SPI Driver Component needs to be initialized before accepting any request. The API Spi_Init should be invoked to initialize SPI Driver Component.
- The user should ensure that SPI Driver Component API requests are invoked in the correct and expected sequence and with correct input arguments.
- Input parameters are validated only when the static configuration parameter SPI_DEV_ERROR_DETECT is enabled. Application should ensure that the right parameters are passed while invoking the APIs when SPI_DEV_ERROR_DETECT is disabled.
- A mismatch in the version numbers of header and the source files results in compilation error. User should ensure that the correct versions of the header and the source files are used.
- The ISR functions and the corresponding handler addresses are provided in Table ISR Handler Addresses. User should ensure that Interrupt Vector table configuration is done as per the information provided in the table.
- Within the callback notification functions only following APIs are allowed.

  Spi_ReadIB Spi_WriteIB

  Spi_SetupEB

  Spi_GetJobResult

  Spi_GetSequenceResult

  Spi_GetHWUnitStatus

  Spi_Cancel

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

4.3. User Mode and Supervisor Mode

The below table specifies the APIs which can run in user mode, supervisor mode or both modes:

Table 4-5 User Mode and Supervisory Mode

Sl. No.	API name	Interrupt mode		Polling mode	
		user mode	supervisor mode	user mode	supervisor mode
1.	Spi_Init	-	x	-	x
2.	Spi_DeInit	-	x	-	x
3.	Spi_WriteIB	x	x	x	x
4.	Spi_AsyncTransmit		x	x	x
5.	Spi_ReadIB	x	x	x	x
6.	Spi_SetupEB	x	x	x	x
7.	Spi_GetStatus	x	x	x	x

Sl. No.	API name	Interrupt mode		Polling mode	
8.	Spi_GetJobResult	x	x	x	x
9.	Spi_GetSequenceResult	x	x	x	x
10.	Spi_GetVersionInfo	x	x	x	x
11.	Spi_SyncTransmit	x	x	x	x
12.	Spi_Cancel	-	x	-	x
13.	Spi_SetAsyncMode	x	x	-	x
14.	Spi_MainFunction_Handling	-	x	-	x
15.	Spi_GetHWUnitStatus	x	x	x	x

4.4. Memory modes

The SPI Driver will use different memory modes depending on the HW units selected. If the HW unit configured is CSIG then only direct access mode has to be configured. If the HW unit configured is CSIH then any of the following four modes can be configured.

Table 4-6 HW unit and Memory Mode Selection

HW unit	Memory mode
CSIG0	Direct Access Mode
CSIH(0-3)	Direct Access Mode FIFO Mode Dual Buffer mode Transmit Only Mode

4.5. Data Consistency

To support the re-entrance and interrupt services, the AUTOSAR SPI component will ensure the data consistency while accessing its own RAM storage or hardware registers. The SPI component will use SchM_Enter_Spi_<Exclusive Area> and SchM_Exit_Spi_<Exclusive Area> functions. The SchM_Enter_Spi_<Exclusive Area> function is called before the data needs to be protected and SchM_Exit_Spi_<Exclusive Area> function is called after the data is accessed.

The following exclusive area along with scheduler services is used to provide data integrity for shared resources:

- CHIP_SELECT_PROTECTION
- RAM_DATA_PROTECTION

The functions SchM_Enter_Spi_<Exclusive Area> and SchM_Exit_Spi_<Exclusive Area> can be disabled by disabling the configuration parameter 'Spi_CriticalSectionProtection'. The flowchart will indicate the flow with the pre-compile option 'Spi_CriticalSectionProtection' enabled.

4.6. Deviation List

Table 4-7 SPI Driver Deviation List

Sl. No.	Description	AUTOSAR Bugzilla

Sl. No.	Description	AUTOSAR Bugzilla
1.	The parameter "SpiHwUnitSynchronous" is moved to SpiJob container from SpiChannel container.	48763
2.	The total number of SPI Hardware Units is published as “SPI_MAX_HW_UNIT”.	24328
3.	The parameter “SPI_BAUDRATE” is not used since the value configured for this parameter cannot be mapped directly to the register value. Hence, a parameter ”SpiBaudrateSelection” is used to select input frequency source.	-
4.	The parameter 'SpiTimeClk2Cs' is not used since the value of this parameter is configured as count value. Hence, the parameter 'SpiClk2CsCount' is provided to configure the wait loop count to add delay between clock and chip select.	-
5.	Type of the parameter SpiHwUnit is ENUMERATION-PARAM-DEF with a list of all possible hardware units.	-
6.	The inclusion or deletion of the hardware units will not be possible in the post-build time. But the reassignment of configured HW unit for different jobs is possible.	-
7.	Type of the parameter SpiCs is ENUMERATION-PARAM-DEF with a list of all possible port lines.	-
8.	If the parameter "DataBufferPtr" passed through the API “Spi_ReadIB” is null pointer, then the error SPI_E_PARAM_POINTER will be reported to DET.	-
9.	The channel parameters “SpiChannelType”, “SpiIbNBuffers” and “SpiEbMaxLength” are pre-compile time parameters.	-
10.	A queue will be implemented and maintained if there are more than one sequence is requested for transmission. The length of the queue will be number of configured jobs minus 1.	-
11.	If a sequence is requested for transmission while already one uninterruptible sequence is on-going, the requested sequence will be put on queue.	-

Sl. No.	Description	AUTOSAR Bugzilla
12.	The upper and lower multiplicity of the parameter 'SpiCsIdentifier' is '1' i.e. mandatory and the default value is NULL. The upper and lower multiplicity of the parameter 'SpiEnableCS' is '1' i.e. mandatory and the default value is false.	-
13.	The parameters SpiMaxChannel, SpiMaxJob and SpiMaxSequence in SpiDriverConfiguration is made as mandatory in the Parameter Definition File of SPI Driver Component.	-
14.	Notification related functions and parameters configuration class are changed from Link time to Post Build, vice versa Spi_ Lcfg.c and Spi_Pbcfg.c files structures are updated.	-
15.	The API Spi_GetVersionInfo is implemented as macro without DET error SPI_E_PARAM_POINTER.	-

Chapter 5 Architecture Details

To minimize the effort and to optimize the reuse of developed software on different platforms, the SPI driver is split as High Level Driver and Low Level Driver. The SPI Driver architecture is shown in the following figure:

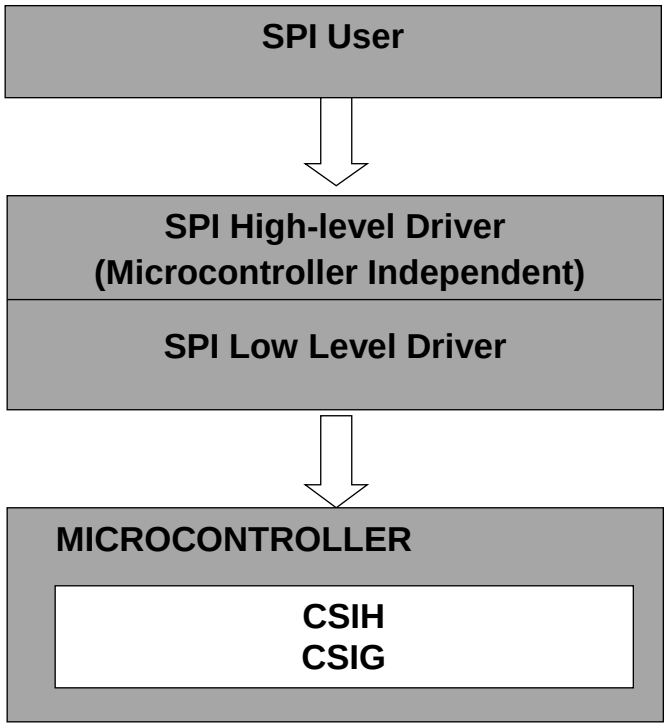

Figure 5-1 SPI Driver Architecture

The High Level Driver exports the AUTOSAR API towards upper modules and it will be designed to allow the compilation for different platforms without or only slight modifications, i.e. that no reference to specific microcontroller features or registers will appear in the High Level Driver. All these references are moved inside a µC specific Low Level Driver. The Low Level Driver interface extends the High Level Driver types and methods in order to adapt it to the specific target microcontroller.

SPI Driver component:

The SPI Driver provides services for reading and writing to devices connected via SPI busses. It provides access to SPI communication to several users like EEPROM, Watchdog, I/O ASICs. It also provides the required mechanism to configure the on chip SPI peripheral.

The SPI Driver component is divided into the following sub modules based on the functionality required:

- Initialization and De-initialization
- Buffer Management
- Communication
- Status information

- Module version information

The basic architecture of the SPI Driver component is illustrated in the following Figure:

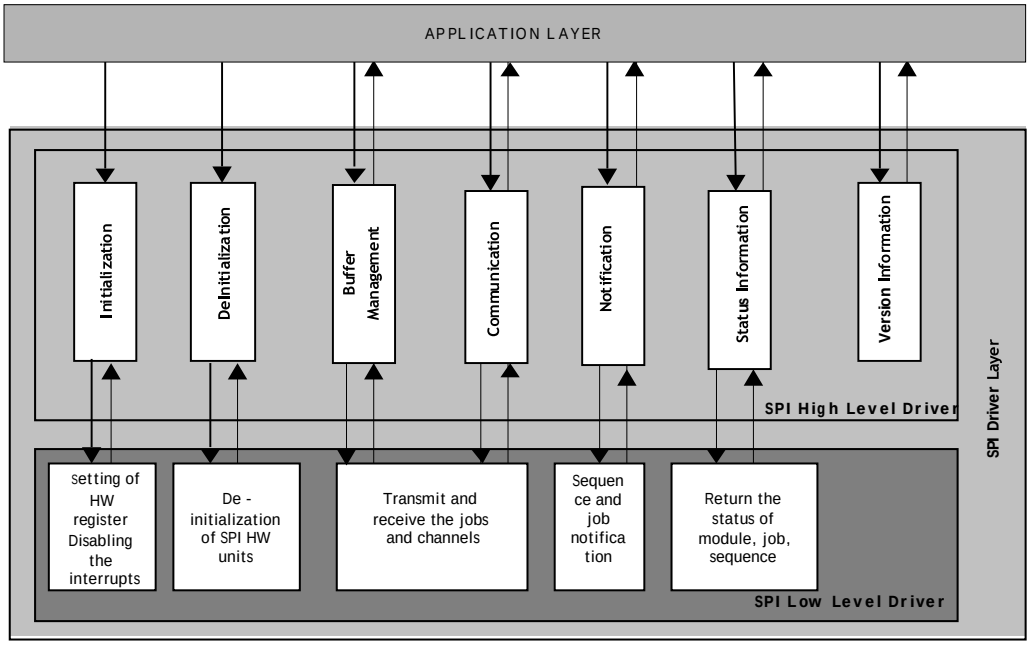

Figure 5-2 Component Overview Of SPI Driver Component

SPI Driver Initialization and De-Initialization module

This module initializes and de-Initializes the SPI driver. It provides the Spi_Init() and Spi_DeInit() APIs. The Spi_Init() API should be invoked before the usage of any other APIs of Watchdog Driver Module.Spi-Init should be called prior to Port_Init. De-initialization function puts all microcontroller SPI peripherals in the same state such as Power On Reset.

Buffer Management

This module provides the services for reading and writing the internal buffers and setting up the external buffer. The type of buffer for each channel is configurable as either internal or external

The APIs related to this module are Spi_WriteIB(), Spi_ReadIB() and Spi_SetupEB().

Communication

This module provides the services for the transmission of data on the SPI bus both synchronously and asynchronously, cancelling the ongoing transmission and setting the asynchronous transfer mode.

The synchronous mode is based on polling mechanism. But for the asynchronous mode, the possible mechanisms are Polling and Interrupt mode. One of these modes is selectable during execution by one of the services provided by this sub-module.

The APIs related to this module are Spi_SyncTransmit(), Spi_AsyncTransmit(), Spi_SetAsyncMode() and Spi_Cancel().

Status Information

This module provides the services for getting the status of the SPI Driver and hardware unit. It also provides the services for getting the result of the specified job and specified sequence.

The APIs related to this module are Spi_GetStatus(), Spi_GetHWUnitStatus(), Spi_GetJobResult() and Spi_GetSequenceResult().

Module Version Information

This module provides APIs for reading module Id, vendor Id and vendor specific version numbers.

The API related to this module is Spi_GetVersionInfo().

Chapter 6 Registers Details

This section describes the register details of SPI Driver Component.

Table 6-1 Register Details

API Name	Registers	Config Parameter	Macro/Variable
Spi_Init	CSIGnCTL0	SpiMemoryModeSelection	SPI_ZERO
	CSIHnCTL0		SPI_ZERO
	DCSTCn	-	SPI_DMA_STR_CLEAR
	DCENn	-	SPI_DMA_DCEN_DISABLE
	DSAn	SpiDma	LpDmaConfig->ulTxRxRegAddress
	DTCTn	SpiTxDmaChannel/ SpiRxDmaChannel	SPI_DMA_8BIT_TX_SETTINGS SPI_DMA_16BIT_TX_SETTINGS SPI_DMA_32BIT_TX_SETTINGS SPI_DMA_8BIT_RX_SETTINGS SPI_DMA_16BIT_RX_SETTINGS SPI_DMA_32BIT_RX_SETTINGS
	DDAn	SpiDma	LpDmaConfig->ulTxRxRegAddress
	DTSn	SpiTxDmaChannel/ SpiRxDmaChannel	SPI_DMA_DISABLE
	DTFRn	SpiTxDmaChannel/ SpiRxDmaChannel	LpDmaConfig->usDmaDtfrRegValue
	CSIGnCTL1	SpiCsInactiveAfterLastData, SpiDataWidth	LunDataAccess1.ulRegData
	CSIHnCTL1		LunDataAccess1.ulRegData
	CSIHTIJC	-	LpIntCntlAddress
	ICCSIGnIR	SpiHwUnitSelection and SpiMemoryModeSelection	Spi_GstHWUnitInfo[LddHWUnit].usRxImrMask
	ICCSIGnIC		Spi_GstHWUnitInfo[LddHWUnit].pTxImrAddress
	ICCSIGnIRE		Spi_GstHWUnitInfo[LddHWUnit].pErrorImrAddress
	ICCSIHnIR		Spi_GstHWUnitInfo[LddHWUnit].usRxImrMask
	ICCSIHnIC		Spi_GstHWUnitInfo[LddHWUnit].pTxImrAddress
	ICCSIHnIJC		LpHWUnitInfo->usTxCancelImrMask
	ICCSIHnIRE		Spi_GstHWUnitInfo[LddHWUnit].pErrorImrAddress
	CSIHnTX0W	-	LunDataAccess1.ulRegData
	SELCSIHDMA	-	SPI_SELCSIHDMA_REG_VAL
Spi_DeInit	CSIGnCTL0	SpiMemoryModeSelection	SPI_ZERO
	CSIHnCTL0		SPI_ZERO
	DCENn	-	SPI_DMA_DCEN_DISABLE
	DTFRRQCn	-	SPI_DMA_DRQ_CLEAR
	DCSTCn	-	SPI_DMA_STR_CLEAR
Spi_WriteIB	CSIHnMRWP0	-	ulRegData
	CSIHnTX0W	-	ulRegData
Spi_AsyncTransmit	CSIHnMCTL0	-	LpJobConfig->usMCtl0Value

API Name	Registers	Config Parameter	Macro/Variable
	CSIGnCFG0	-	LpJobConfig->ulConfigRegValue
	CSIGnCTL0	SpiMemoryModeSelection	SPI_RESET_PWR SPI_SET_DIRECT_ACCESS SPI_SET_MEMORY_ACCESS
	CSIHnCTL0		SPI_RESET_PWR SPI_SET_DIRECT_ACCESS SPI_SET_MEMORY_ACCESS
	CSIGnSTCR0	-	SPI_CLR_STS_FLAGS
	CSIHnSTCR0	-	SPI_CLR_STS_FLAGS
	CSIGnCTL1	SpiCsInactiveAfterLastData, SpiDataWidth	LunDataAccess1.ulRegData LpJobConfig->ulMainCtl1Value SPI_SET_SLIT
	CSIHnCTL1		LunDataAccess1.ulRegData LpJobConfig->ulMainCtl1Value SPI_SET_SLIT
	DCSTCn	-	SPI_DMA_STR_CLEAR
	DCENn	-	SPI_DMA_DCEN_DISABLE
	DTCTn	-	SPI_DMA_FIXED_TX_SETTINGS SPI_DMA_INV_TX_SETTINGS LddNoOfBuffers SPI_DMA_STR_REQ SPI_DMA_ONCE SPI_DMA_FIXED_RX_SETTINGS SPI_DMA_INV_RX_SETTINGS SPI_DMA_ONCE
	DSAn	-	(uint32)LpTxData
	DTFRn	-	(uint32)SPI_ZERO (uint32)(LpDmaConfig->usDmaDtfrRegValue
	DCSTSn	-	SPI_DMA_STR
	DTCn	-	SPI_ONE
	DTFRRQCn	-	SPI_DMA_DRQ_CLEAR
	DCENn	-	SPI_DMA_DCEN_ENABLE
	DDAn	-	(uint32)(&Spi_GddDmaRxData)
	DTFRn	-	SPI_ZERO
	CSIGnCTL2	SpiBaudrateRegisterSelect	LpJobConfig->usCtl2Value
	CSIHnCTL2	SpiFifoTimeOut	LpJobConfig->usCtl2Value
	CSIHnCFG	SpiCsIdleTiming, SpiCsHoldTiming, SpiCsInterDataDelay, SpiCsSetupTime, SpiCsIdleEnforcement	LunDataAccess1.ulRegData
	CSIGnCFG0		LunDataAccess1.ulRegData
	CSIHnCFG0		LunDataAccess1.ulRegData
	CSIHnMCTL1	-	SPI_ZERO
	CSIHnMCTL2	-	LunDataAccess1.ulRegData
	CSIHTX0W	-	LunDataAccess1.ulRegData
	CSIHnCFG	SpiCsIdleTiming, SpiCsHoldTiming, SpiCsInterDataDelay, SpiCsSetupTime, SpiCsIdleEnforcement	LunDataAccess1.ulRegData

API Name	Registers	Config Parameter	Macro/Variable
	CSIGnTX0W	-	LunDataAccess1.ulRegData
	CSIHnBRS[0]	SpiBaudrateConfiguration	LpCsihOsBaseAddr->usCSIHBRS[0]
	CSIHnBRS[1]	-	LpCsihOsBaseAddr->usCSIHBRS[1]
	CSIHnBRS[2]	-	LpCsihOsBaseAddr->usCSIHBRS[2]
	CSIHnBRS[3]	-	LpCsihOsBaseAddr->usCSIHBRS[3]
Spi_ReadIB	CSIHnRX0W	-	LunDataAccess2.ulRegData
	CSIHnRX0H	-	LunDataAccess2.usRegData5[1]
	CSIHnMRWP0	-	LunDataAccess1.ulRegData
	CSIHRX0H	-	LunDataAccess2.usRegData5[0]
Spi_SetupEB	-	-	-
Spi_GetStatus	-	-	-
Spi_GetJobResult	-	-	-
Spi_GetSequenceResult	-	-	-
Spi_SyncTransmit	CSIHnMCTL0	-	-
	CSIGnCTL0	-	LpJobConfig->usMCtl0Value
	CSIHnCTL0	-	SPI_RESET_PWR
	CSIGnCTL0	-	SPI_RESET_PWR
	CSIHnCTL0	-	SPI_SET_DIRECT_ACCESS
	CSIGnCTL0	-	SPI_SET_DIRECT_ACCESS
	CSIHnCTL0	-	SPI_SET_PWR
	CSIGnTX0W	-	SPI_SET_PWR
	CSIHnRX0H	-	LunDataAccess3.ulRegData
	CSIHnRX0H	-	LunDataAccess3.ulRegData
	CSIGnCFG0	-	Spi_GusDataAccess
	CSIGnCFG0	-	LddData
	CSIGnCFG0	-	LpJobConfig->ulConfigRegValue
	CSIGnCTL0	-	LunDataAccess1.ulRegData
	CSIHnCTL0	-	SPI_ZERO
	CSIGnCFG0	-	LddData
	CSIGnCFG0	-	LpJobConfig->ulConfigRegValue
	CSIGnCTL0	-	LunDataAccess1.ulRegData
	CSIHnCTL0	-	SPI_ZERO
	CSIGnSTR0	-	SPI_ZERO
	CSIHnSTR0	-	SPI_HW_BUSY
	CSIGnSTR0	-	SPI_HW_BUSY
	CSIHnSTR0	-	SPI_ZERO
	CSIGnSTCR0	-	SPI_ZERO
	CSIHnSTCR0	-	SPI_CLR_STS_FLAGS
	CSIGnCTL1	-	SPI_CLR_STS_FLAGS
	CSIHnCTL1	SpiCsInactiveAfterLastData, SpiDataWidth	LunDataAccess1.ulRegData
	CSIGnCTL2	SpiBaudrateRegisterSelect	LunDataAccess1.ulRegData
	CSIHnCTL2	SpiFifoTimeOut	LpJobConfig->usCtl2Value

API Name	Registers	Config Parameter	Macro/Variable
	CSIHnTX0W	-	LpJobConfig->usCtl2Value
	CSIHnTX0W	-	LunDataAccess3.ulRegData
	CSIHnCFG	SpiCsIdleTiming, SpiCsHoldTiming, SpiCsInterDataDelay, SpiCsSetupTime, SpiCsIdleEnforcement	LunDataAccess1.ulRegData
	CSIHnCFG		LpJobConfig->ulConfigRegValue
	CSIGnTX0W	-	LunDataAccess1.ulRegData
	CSIGnRX0	-	Spi_GusDataAccess
	CSIGnRX0	-	LddData
	CSIGnRX0	-	LunDataAccess2.usRegData5[1]
	CSIGnRX0	-	LunDataAccess2.usRegData5[0]
	CSIHnBRS[0]	SpiBaudrateConfiguration	LpCsihOsBaseAddr->usCSIHBRS[0]
	CSIHnBRS[1]		LpCsihOsBaseAddr->usCSIHBRS[1]
	CSIHnBRS[2]		LpCsihOsBaseAddr->usCSIHBRS[2]
	CSIHnBRS[3]		LpCsihOsBaseAddr->usCSIHBRS[3]
Spi_GetHWUnitStatus	CSIGnSTR0	-	SPI_CSIG_CSIH_BUSY
	CSIHnSTR0	-	SPI_CSIG_CSIH_BUSY
Spi_Cancel	CSIGnCTL0	-	SPI_ZERO
	CSIHnCTL0	-	SPI_ZERO
	CSIGnCTL0	-	SPI_SET_JOBE
	CSIHnCTL0	-	SPI_SET_JOBE
	CSIHTIJC	-	LpHWUnitInfo->ucTxCancelImrMask
Spi_SetAsyncMode	-	-	-
Spi_MainFunction_Handling	CSIGnCTL0	-	SPI_SET_PWR
	CSIHnCTL0	-	SPI_SET_PWR
	CSIGTIR	-	SPI_CLR_INT_REQ
	CSIGTIC	-	SPI_CLR_INT_REQ
Spi_GetVersionInfo	-	-	-

Chapter 7 Interaction Between The User And SPI Driver Component

The details of the services supported by the SPI Driver Component to the upper layers users and the mapping of the channels to the hardware units is provided in the following sections:

7.1. Services Provided By SPI Driver Component To The User

The SPI Driver Component provides the following functions to upper layer: -

- To provide the required mechanism to configure the on-chip SPI peripheral.
- To initialize and de-initialize the SPI driver.
- To read and write to devices connected through SPI buses.
- To provide the transmission of data on the SPI bus both synchronously and asynchronously.
- To cancel an ongoing transmission.
- To set the asynchronous transfer mode.
- To get the status of the SPI Driver and hardware unit.
- To get the result of the specified job and specified sequence.
- To provide access to SPI communication to several users(for example, EEPROM, I/O ASICs).
- To read the SPI Driver Component version information.

Chapter 8 SPI Driver Component Header And Source File Description

This section explains the SPI Driver Component's source and header files. These files have to be included in the project application while integrating with other modules.

The C header file generated by SPI Driver Generation Tool:

- Spi_Cfg.h
- Spi_Cbk.h

The C source file generated by SPI Driver Generation Tool:

- Spi_PBcfg.c
- Spi_Lcfg.c

The SPI Driver Component C header files:

- Spi_Driver.h
- Spi_PBTypes.h
- Spi_LTTypes.h
- Spi_Ram.h
- Spi.h
- Spi_Irq.h
- Spi_Scheduler.h
- Spi_Version.h
- Spi_Types.h

The SPI Driver Component C source files:

- Spi_Driver.c
- Spi.c
- Spi_Irq.c
- Spi_Ram.c
- Spi_Scheduler.c
- Spi_Version.c

The SPI Driver specific header files:

- Compiler.h
- Compiler_Cfg.h
- MemMap.h
- Platform_Types.h
- rh850_Types.h

The description of the SPI Driver Component files is provided in the table below:

Table 8-1 Description Of The SPI Driver Component Files

File	Details
Spi_Cfg.h	This file is generated by the SPI Driver Component Code Generation Tool for various SPI Driver component pre-compile time parameters. This file contains macro definitions for the configuration elements and exclusive areas for data protection. The macros and the parameters generated will vary with respect to the configuration in the input XML file.
Spi_Cbk.h	This file is generated by the SPI Driver Component Code Generation Tool for provision of function prototype Declarations for SPI callback Notification Functions.
Spi_PBcfg.c	This file contains post-build configuration data. The structures related to channel configuration, job configuration and sequence configuration are provided in this file. Data structures will vary with respect to parameters configured.
Spi_Lcfg.c	This file contains provision of SPI Link time Parameters. The structures related to hardware registers are provided in this file. Data structures will vary with respect to parameters configured.
Spi_Driver.h	This file contains the Function Prototypes that are defined in Spi_Driver.c file.
Spi_PBTypes.h	This file contains the data structure definitions of the channel configuration, job configuration and sequence configuration
Spi_LTTypes.h	This file contains the data structure definitions of CSIG and CSIH hardware registers, Interrupt control registers, DMA hardware registers, Hardware unit information, DMA unit information, storing current status of SPI communication, channel for the link time parameters, function pointer for Callback notification function for Jobs, processing sequence, storing external buffer attributes, Scheduler and DMA Address.
Spi_Ram.h	This file contains the extern declarations for the global variables that are defined in Spi_Ram.c file and the version information of the file.
Spi.h	This file provides extern declarations for all the SPI Driver Component APIs. This file provides service Ids of APIs, DET Error codes and type definitions for SPI Driver initialization structure. This header file shall be included in other modules to use the features of SPI Driver Component.
Spi_Irq.h	This file contains the function prototypes that are defined in Spi_Irq.c file.
Spi_Scheduler.h	This file contains the function prototypes that are defined in Spi_Scheduler.c file.
Spi_Types.h	This file contains the common macro definitions and the data types required internally by the SPI software component.
Spi_Version.h	This file contains the definitions of AUTOSAR version numbers of all modules that are interfaced to SPI Driver.
Spi_Driver.c	This file contains the SPI Low Level Driver code.
Spi.c	This file contains the implementation of all APIs.
Spi_Irq.c	This file contains the ISR functions for SPI Driver Component.
Spi_Ram.c	This file contains the global variables used by SPI Driver Component.
Spi_Scheduler.c	This file contains the SPI Scheduler code. This contains function to schedule the sequences according to the priority of the jobs.
Spi_Version.c	This file contains the code for checking version of all modules that are interfaced to SPI Driver.
Compiler.h	This file Provides compiler specific (non-ANSI) keywords. All mappings of keywords, which are not standardized, and/or compiler specific are placed and organized in this compiler specific header.
Compiler_Cfg.h	This file contains the memory and pointer classes.

File	Details
MemMap.h	This file allows to map variables, constants and code of modules to individual memory sections. Memory mapping can be modified as per ECU specific needs.
Platform_Types.h	This file provides provision for defining platform and compiler dependent types.
rh850_Types.h	This file provides macros to perform supervisor mode (SV) write enabled Register ICxxx and IMR register writing using OR/AND/Direct operation

Chapter 9 Generation Tool Guide

For information on the SPI Driver Component Code Generation Tool, please refer “AUTOSAR_SPI_Tool_UserManual.pdf” document.

Chapter 10 Application Programming Interface

This section explains the Data types and APIs provided by the SPI Driver Component to the Upper layers.

10.1. Imported Types

This section explains the Data types imported by the SPI Driver Component and lists its dependency on other modules.

10.1.1. Standard Types

In this section all types included from the Std_Types.h are listed:

- Std_ReturnType
- Std_VersionInfoType

10.1.2. Other Module Types

In this chapter all types included from the Dem_types.h are listed:

- Dem_EventIdType
- Dem_EventStatusType

10.2. Type Definitions

This section explains the type definitions of SPI Driver Component according to AUTOSAR Specification.

10.2.1. Spi_ConfigType

Name:	Spi_ConfigType	
Type:	Structure	
Range:	Implementation Specific	The contents of the initialization data structure are SPI specific
Description:	This type of the external data structure shall contain the initialization data for the SPI driver/Handler	

10.2.2. Spi_StatusType

Name:	Spi_StatusType	
Type:	Enumeration	
Range:	SPI_UNINIT	The SPI Handler/Driver is not initialized or not usable
	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 type defines a range of specific status for SPI Handler/driver	

10.2.3. Spi_JobResultType

Name:	Spi_JobResultType	
Type:	Enumeration	
Range:	SPI_JOB_OK	The last transmission of the job has been finished successfully
	SPI_JOB_PENDING	The SPI Handler/Driver is performing a SPI Job. The meaning of this status is equal to SPI_BUSY
	SPI_JOB_FAILED	The last transmission of the job has failed
Description:	This type defines a range of specific jobs status for SPI Handler/driver	

10.2.4. Spi_SeqResultType

Name:	Spi_SeqResultType	
Type:	Enumeration	
Range:	SPI_SEQ_OK	The last transmission of the Sequence has been finished successfully
	SPI_SEQ_PENDING	The SPI Handler/Driver is performing a SPI Sequence The meaning of this status is equal to SPI_BUSY
	SPI_SEQ_FAILED	The last transmission of the Sequence has failed
	SPI_SEQ_CANCELLED	The last transmission of the Sequence has been cancelled by user.
Description:	This type defines a range of specific sequences status for SPI Handler/driver	

10.2.5. Spi_DataType

Name:	Spi_DataType	
Type:	uint8,uint16,uint32	
Range:	0 to 255, 0 to 65535, 0 to 4294967296.	This is implementation specific but not all values may be valid within the type This type shall be chosen in order to have the most efficient implementation on a specific microcontroller platform
Description:	Type of application data buffer elements	

10.2.6. Spi_NumberOfDataType

Name:	Spi_NumberOfDataType
Type:	uint16
Range:	0 to 65535
Description:	Type for defining the number of data elements of the type Spi_DataType to send and/or receive by channel

10.2.7. Spi_ChannelType

Name:	Spi_ChannelType
Type:	uint8
Range:	0 to 255
Description:	Specifies the identification(Id) for a channel

10.2.8. Spi_JobType

Name:	Spi_JobType
Type:	uint16
Range:	0 to 65535
Description:	Specifies the identification(Id) for a Job

10.2.9. Spi_SequenceType

Name:	Spi_SequenceType
Type:	uint8
Range:	0 to 255
Description:	Specifies the identification(Id) for a sequence of Jobs

10.2.10. Spi_HWUnitType

Name:	Spi_HWUnitType
Type:	uint8
Range:	0 to 255
Description:	Specifies the identification(Id) for a SPI Hardware microcontroller peripheral(unit)

10.2.11. Spi_AsyncModeType

Name:	Spi_AsyncModeType	
Type:	Enumeration	
Range:	SPI_POLLING_MODE	The asynchronous mechanism is ensured by polling, so interrupts related to SPI busses handled asynchronously are disabled
	SPI_INTERRUPT_MODE	Streaming access mode
Description:	Specifies the asynchronous mechanism mode for SPI busses handled asynchronously in LEVEL2.	

10.3. Function Definitions

Table 10-1 The APIs provided by the SPI Driver Component

Sl. No	API's	API's specific
1.	Spi_Init	-
2.	Spi_DeInit	-
3.	Spi_WriteIB	-
4.	Spi_AsyncTransmit	-
5.	Spi_ReadIB	-
6.	Spi_SetupEB	-
7.	Spi_GetStatus	-
8.	Spi_GetJobResult	-
9.	Spi_GetSequenceResult	-
10.	Spi_GetVersionInfo	-
11.	Spi_SyncTransmit	-
12.	Spi_Cancel	-
13.	Spi_SetAsyncMode	-
14.	Spi_MainFuncnction_Handling	-
15.	Spi_GetHWUnitStatus	-

Chapter 11 Development And Production Errors

In this section the development errors that are reported by the SPI Driver Component are tabulated. The development errors will be reported only when the pre compiler option SpiDevErrorDetect is enabled in the configuration. The production code errors are not supported by SPI Driver Component.

11.1. SPI Driver Component Development Errors

The following table contains the DET errors that are reported by SPI Driver Component. These errors are reported to Development Error Tracer Module when the SPI Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 11-1 DET Errors Of SPI Driver Component

Sl. No.	**1**
Error Code	SPI_E_PARAM_CHANNEL
Related API(s)	Spi_WriteIB, SpiReadIB and Spi_SetupEB
Source of Error	When the API service is invoked with invalid channel Id and if incorrect type of channel (IB or EB) is used with services.
Sl. No.	**2**
Error Code	SPI_E_PARAM_JOB
Related API(s)	Spi_GetJobResult
Source of Error	When the API service is invoked with invalid job Id.
Sl. No.	**3**
Error Code	SPI_E_PARAM_SEQ
Related API(s)	Spi_AsyncTransmit, Spi_GetSequenceResult, Spi_SyncTransmit and Spi_Cancel
Source of Error	When the API service is invoked with invalid sequence Id.
Sl. No.	**4**
Error Code	SPI_E_PARAM_LENGTH
Related API(s)	Spi_SetupEB
Source of Error	When the API service is invoked with length greater than the configured length.
Sl. No.	**5**
Error Code	SPI_E_PARAM_UNIT
Related API(s)	Spi_GetHWUnitStatus
Source of Error	When the API service is invoked with invalid hardware unit Id.
Sl. No.	**6**
Error Code	SPI_E_SEQ_PENDING
Related API(s)	Spi_AsyncTransmit
Source of Error	When the API service is invoked in a wrong sequence.
Sl. No.	**7**
Error Code	SPI_E_SEQ_IN_PROCESS
Related API(s)	Spi_SyncTransmit
Source of Error	When the API service is invoked at wrong time.
Sl. No.	**8**
Error Code	SPI_E_ALREADY_INITIALIZED
Related API(s)	Spi_Init
Source of Error	When the API Spi_Init is invoked when the SPI driver is already initialized.

Sl. No.	9
Error Code	SPI_E_INVALID_DATABASE
Related API(s)	Spi_Init
Source of Error	When the API service is invoked with invalid pointer.
Sl. No.	**10**
Error Code	SPI_E_UNINIT
Related API(s)	Spi_Init, Spi_DeInit, Spi_AsyncTransmit, Spi_Cancel, Spi_GetHWUnitStatus, Spi_GetJobResult, Spi_GetSequenceResult, Spi_WriteIB, Spi_ReadIB, Spi_SetupEB, Spi_SyncTransmit and Spi_SetAsyncMode
Source of Error	When the APIs are invoked without the initialization of SPI Driver Component.
Sl. No.	**11**
Error Code	SPI_E_PARAM_POINTER
Related API(s)	Spi_ReadIB
Source of Error	When the API service is invoked with null pointer. Note: This error code (SPI_E_PARAM_POINTER) is applicable for Autosar R4.0 only.
Sl. No.	**12**
Error Code	SPI_E_PARAM_CONFIG
Related API(s)	Spi_Init
Source of Error	When the API invoked with null config pointer.

11.2. SPI Driver Component Production Errors

In this section the DEM errors identified in the SPI Driver Component are listed. SPI Driver Component reports these errors to DEM by invoking Dem_ReportErrorStatus API. This API is invoked, when the processing of the given API request fails.

Table 11-2 DEM Errors Of SPI Driver Component

Sl. No.	1
Error Code	SPI_E_HARDWARE_ERROR
Related API(s)	Spi_SyncTransmit and Spi_AsyncTransmit
Source of Error	When an overrun occurs when the next reception starts without performing a CPU read of the value of the receive buffer, upon completion of the receive operation.
Sl. No.	**2**
Error Code	SPI_E_DATA_TX_TIMEOUT_FAILURE
Related API(s)	Spi_SyncTransmit
Source of Error	When Hardware data transmit timeout error is detected, This error will be reported to DEM

Chapter 12 Memory Organization

Following picture depicts a typical memory organization, which must be met for proper functioning of SPI Driver Component software.

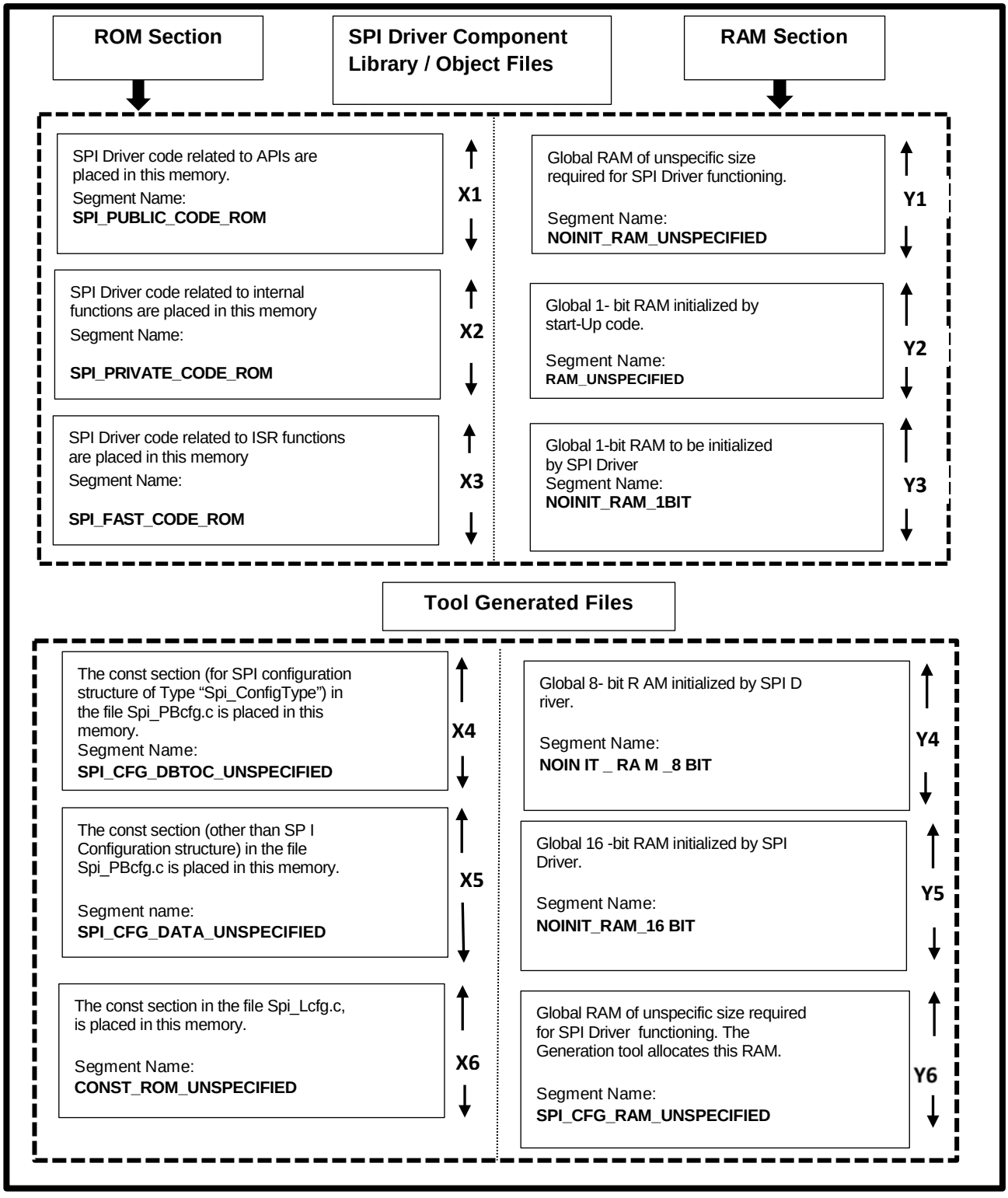

Figure 12-1 SPI Driver Component Driver Organization

ROM Section (X1, X2, X3,X4,X5 and X6):

SPI_PUBLIC_CODE_ROM (X1): API(s) of SPI Driver Component, which can be located in code memory.

SPI_PRIVATE_CODE_ROM (X2): Internal functions of SPI Driver Component code that can be located in code memory.

SPI_FAST_CODE_ROM(X3): SPI Driver code related to ISR functions are placed in this memory Segment Name

SPI_CFG_DBTOC_UNSPECIFIED (X4): This section consists of SPI Driver Component database table of contents generated by the SPI Driver Component Generation Tool. This can be located in code memory.

SPI_CFG_DATA_UNSPECIFIED (X5): This section consists of SPI Driver Component constant configuration structures. This can be located in code memory.

CONST_ROM_UNSPECIFIED (X6): This section consists of SPI Driver Component constant structures used for function pointers in SPI Driver Component. This can be located in code memory.

RAM Section (Y1, Y2, Y3, Y4, Y5 and Y6):

NOINIT_RAM_UNSPECIFIED (Y1): This section consists of the global RAM variables that are used internally by SPI Driver Component. This can be located in data memory.

RAM_UNSPECIFIED (Y2): This section consists of the global RAM variables of 1-bit size that are initialized by start-up code and used internally by SPI Driver Component. This can be located in data memory.

RAM_1BIT (Y3): This section consists of the global RAM variables of 1-bit size that are initialized by start-up code and used internally by SPI Driver Component. The specific sections of respective software components will be merged into this RAM section accordingly.

NOINIT_RAM_8BIT (Y4): This section consists of the global RAM variables of 8-bit size that are used internally by SPI Driver Component. This can be located in data memory.

NOINIT_RAM_16BIT (Y5): This section consists of the global RAM variables of 16-bit size that are used internally by SPI Driver Component. This can be located in data memory.

SPI_CFG_RAM_UNSPECIFIED (Y6): This section consists of the global RAM variables that are generated by SPI Driver Component Generation Tool. This can be located in data memory.

Remark

- X1, X2, Y1, Y2 and Y3 pertain to only SPI Driver Component and do not include memory occupied by Spi_PBcfg.c or Spi_Lcfg.c file generated by SPI Driver Component Generation Tool.

User must ensure that none of the memory areas overlap with each other. Even ‘debug’ information should not overlap.

Chapter 13 P1M Specific Information

P1M supports following devices:

- R7F701304
- R7F701305
- R7F701310
- R7F701311
- R7F701312
- R7F701313
- R7F701314
- R7F701315
- R7F701318
- R7F701319
- R7F701320
- R7F701321
- R7F701322
- R7F701323

13.1. Interaction Between The User And SPI Driver Component

The details of the services supported by the SPI Driver Component to the upper layers users and the mapping of the channels to the hardware units is provided in the following sections:

13.1.1. Translation Header File

The translation header file supports following devices:

- R7F701304
- R7F701305
- R7F701310
- R7F701311
- R7F701312
- R7F701313
- R7F701314
- R7F701315
- R7F701318
- R7F701319
- R7F701320
- R7F701321
- R7F701322
- R7F701323

13.1.2. Parameter Definition File

Parameter definition files support information for P1M

Table 13-1 PDF information for P1M

PDF Files	Devices Supported
R403_SPI_P1M_04_05_12_13_20_21.arxml	701304,701305,701312,701313,701320,701321

R403_SPI_P1M_10_11_14_15_18_19_22_23.arxml	701310,701311,701314,701315,701318,701319,701322,701323

13.1.3. ISR Function

The table below provides the list of handler addresses corresponding to the hardware unit ISR(s) in SPI Driver Component. The user should configure the ISR functions mentioned below.

Table 13-2 Interrupt Handler

Interrupt Source	Name of the ISR Function
INTCSIG0IRE	SPI_CSIG0_TIRE_ISR
	SPI_CSIG0_TIRE_CAT2_ISR
INTCSIG0IR	SPI_CSIG0_TIR_ISR
	SPI_CSIG0_TIR_CAT2_ISR
INTCSIG0IC	SPI_CSIG0_TIC_ISR
	SPI_CSIG0_TIC_CAT2_ISR
INTCSIH0IRE	SPI_CSIH0_TIRE_ISR
	SPI_CSIH0_TIRE_CAT2_ISR
INTCSIH0IR	SPI_CSIH0_TIR_ISR
	SPI_CSIH0_TIR_CAT2_ISR
INTCSIH0IC	SPI_CSIH0_TIC_ISR
	SPI_CSIH0_TIC_CAT2_ISR
INTCSIH0IJC	SPI_CSIH0_TIJC_ISR
	SPI_CSIH0_TIJC_CAT2_ISR
INTCSIH1IRE	SPI_CSIH1_TIRE_ISR
	SPI_CSIH1_TIRE_CAT2_ISR
INTCSIH1IR	SPI_CSIH1_TIR_ISR
	SPI_CSIH1_TIR_CAT2_ISR
INTCSIH1IC	SPI_CSIH1_TIC_ISR
	SPI_CSIH1_TIC_CAT2_ISR
INTCSIH1IJC	SPI_CSIH1_TIJC_ISR
	SPI_CSIH1_TIJC_CAT2_ISR
INTCSIH2IRE	SPI_CSIH2_TIRE_ISR
	SPI_CSIH2_TIRE_CAT2_ISR
INTCSIH2IR	SPI_CSIH2_TIR_ISR
	SPI_CSIH2_TIR_CAT2_ISR
INTCSIH2IC	SPI_CSIH2_TIC_ISR
	SPI_CSIH2_TIC_CAT2_ISR
INTCSIH2IJC	SPI_CSIH2_TIJC_ISR
	SPI_CSIH2_TIJC_CAT2_ISR
INTCSIH3IRE	SPI_CSIH3_TIRE_ISR
	SPI_CSIH3_TIRE_CAT2_ISR
INTCSIH3IR	SPI_CSIH3_TIR_ISR

Interrupt Source	Name of the ISR Function
	SPI_CSIH3_TIR_CAT2_ISR
INTCSIH3IC	SPI_CSIH3_TIC_ISR
	SPI_CSIH3_TIC_CAT2_ISR
INTCSIH3IJC	SPI_CSIH3_TIJC_ISR
	SPI_CSIH3_TIJC_CAT2_ISR

Interrupt Source	Name of the ISR Function
INTDMA[0-7]	SPI_DMA00_ISR
	SPI_DMA00_CAT2_ISR
	SPI_DMA01_ISR
	SPI_DMA01_CAT2_ISR
	SPI_DMA02_ISR
	SPI_DMA02_CAT2_ISR
	SPI_DMA03_ISR
	SPI_DMA03_CAT2_ISR
	SPI_DMA04_ISR
	SPI_DMA04_CAT2_ISR
	SPI_DMA05_ISR
	SPI_DMA05_CAT2_ISR
	SPI_DMA06_ISR
	SPI_DMA06_CAT2_ISR
	SPI_DMA07_ISR
	SPI_DMA07_CAT2_ISR

13.2. Sample Application

The Sample Application is provided as reference to the user to understand the method in which the SPI APIs can be invoked from the application.

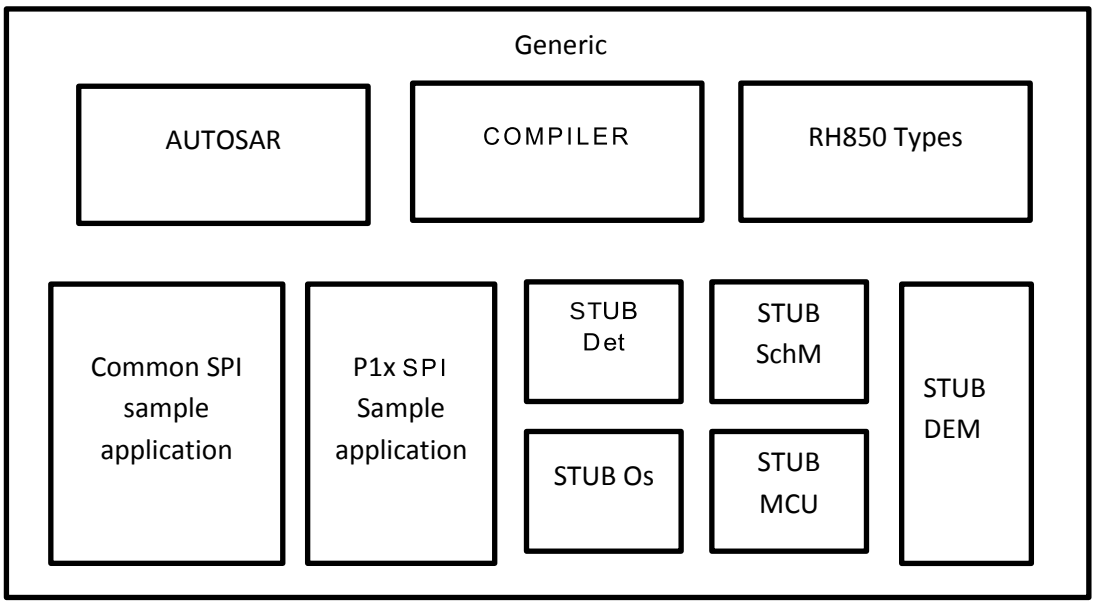

Figure 13-1 Overview Of SPI Driver Sample Application

13.3.1. Sample Application Structure

The Sample Application of the P1M is available in the path

The Sample Application consists of the following folder structure

X1X\P1x\modules\spi\definition\<AUTOSAR_version>\

<SubVariant>\R403_SPI_P1M_04_05_12_13_20_21.arxml
\R403_SPI_P1M_10_11_14_15_18_19_22_23.arxml

X1X\P1x\modules\spi\sample_application\<SubVariant>\<AUTOSAR_version>

\src\Spi_Lcfg.c

\src\Spi_PBcfg.c

\inc\Spi_Cfg.h

\inc\Spi_Cbk.h

/config/App_SPI_P1M_701304_Sample.one
/config/App_SPI_P1M_701304_Sample.arxml
/config/App_SPI_P1M_701304_Sample.html

/config/App_SPI_P1M_701305_Sample.one
/config/App_SPI_P1M_701305_Sample.arxml
/config/App_SPI_P1M_701305_Sample.html

/config/App_SPI_P1M_701310_Sample.one
/config/App_SPI_P1M_701310_Sample.arxml
/config/App_SPI_P1M_701310_Sample.html

/config/App_SPI_P1M_701311_Sample.one
/config/App_SPI_P1M_701311_Sample.arxml
/config/App_SPI_P1M_701311_Sample.html

/config/App_SPI_P1M_701312_Sample.one
/config/App_SPI_P1M_701312_Sample.arxml
/config/App_SPI_P1M_701312_Sample.html

/config/App_SPI_P1M_701313_Sample.one
/config/App_SPI_P1M_701313_Sample.arxml
/config/App_SPI_P1M_701313_Sample.html

/config/App_SPI_P1M_701314_Sample.one
/config/App_SPI_P1M_701314_Sample.arxml
/config/App_SPI_P1M_701314_Sample.html

/config/App_SPI_P1M_701315_Sample.one
/config/App_SPI_P1M_701315_Sample.arxml
/config/App_SPI_P1M_701315_Sample.html

/config/App_SPI_P1M_701318_Sample.one
/config/App_SPI_P1M_701318_Sample.arxml
/config/App_SPI_P1M_701318_Sample.html

/config/App_SPI_P1M_701319_Sample.one
/config/App_SPI_P1M_701319_Sample.arxml
/config/App_SPI_P1M_701319_Sample.html

/config/App_SPI_P1M_701320_Sample.one
/config/App_SPI_P1M_701320_Sample.arxml
/config/App_SPI_P1M_701320_Sample.html

/config/App_SPI_P1M_701321_Sample.one
/config/App_SPI_P1M_701321_Sample.arxml
/config/App_SPI_P1M_701321_Sample.html

/config/App_SPI_P1M_701322_Sample.one
/config/App_SPI_P1M_701322_Sample.arxml
/config/App_SPI_P1M_701322_Sample.html

/config/App_SPI_P1M_701323_Sample.one
/config/App_SPI_P1M_701323_Sample.arxml
/config/App_SPI_P1M_701323_Sample.html

In the Sample Application all the SPI APIs are invoked in the following sequence:

- The API Spi_Init is invoked with a valid database address for the proper initialization of the SPI Driver, all the SPI Driver control registers and RAM variables will get initialized after this API is called.
- The API Spi_GetVersionInfo is invoked to get the version of the SPI Driver module with a variable of Std_VersionInfoType, after the call of this API the passing parameter will get updated with the SPI Driver version details.
- The API Spi_GetHWUnitStatus will return the status of the specified SPI Hardware microcontroller peripheral.
- The API Spi_SyncTransmit will transmit data on the SPI bus synchronously.
- This module will take the passing parameter and set the SPI Driver status to SPI_BUSY. Also it sets the sequence result to SPI_SEQ_PENDING and first job result to SPI_JOB_PENDING and performs the transmission.
- The API Spi_SetAsyncMode will set the asynchronous mechanism mode for SPI busses handled asynchronously.
- The API Spi_MainFunction_Driving is used for Asynchronous transmission of the sequences in polling mode. This service is should be invoked in a scheduler loop if the asynchronous transmission mode is selected as SPI_POLLING_MODE.
- The API Spi_Cancel will cancel the specified on-going sequence transmission without canceling any Job transmission and the SPI Driver will set the sequence result to SPI_SEQ_CANCELLED.
- The API Spi_DeInit is invoked for de-initialization of the all the controls registers and RAM variables.

13.3.2. Building Sample Application

13.3.2.1. Configuration Example

This section contains the typical configuration which is used for measuring

RAM/ROM consumption, stack depth and throughput details

Configuration Details: App_SPI_P1M_701310_Sample.html

13.3.2.2. Debugging The Sample Application

Remark GNU Make utility version 3.81 or above must be installed and available in the path as defined by the environment user variable “GNUMAKE” to complete the build process using the delivered sample files.

- Open a Command window and change the current working directory to ”make” directory present as mentioned in below path:

  “X1X\P1x\common_family\make\<Compiler>”

- Now execute the batch file SampleApp.bat with following parameters

  SampleApp.bat Spi 4.0.3 <Device_name>.

- After this, the tool output files will be generated with the configuration as mentioned in App_SPI_P1M_701310_Sample.html file available in the path:

  “X1X\P1x\modules\spi\sample_application\<SubVariant>\<AUTOSAR_version>\config\App_SPI_P1M_<Device_Name>_Sample.html”

- After this, all the object files, map file and the executable file App_Spi_P1M_Sample.out will be available in the output folder: (“X1X\P1x\modules\spi\sample_application\<SubVariant> \obj\<Compiler>”)

- The executable can be loaded into the debugger and the sample application can be executed.

Remark Executable files with ‘*.out’ extension can be downloaded into the target hardware with the help of Green Hills debugger.

- If any configuration changes (only post-build) are made to the ECU Configuration Description files

  “X1X\P1x\modules\spi\sample_application\<SubVariant> \<AUTOSAR_version>\config\App_SPI_P1M_<Device_Name>_Sample.arxml”

- The database alone can be generated by using the following commands.

  make –f App_SPI_P1M_Sample.mak generate_spi_config

  make –f App_SPI_P1M_Sample.mak App_SPI_P1M_Sample.s37

  After this, a flash able Motorola S-Record file App_SPI_P1M_Sample.s37 is available in the output folder.

  Note: The <Device_name> indicates the device to be compiled, which can be 701304 or 701305 or 701310 or 701314 or 701315 or 701318 or 701319 or 701320 or 701321 or 701322 or 701323

13.3. Memory And Throughput

13.4.1. ROM/RAM Usage

The details of memory usage for the typical configuration, with DET disabled as provided in Section 13.3.2.1 *Configuration Example* are provided in this section.

Table 13-7 ROM/RAM Details without DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes for 701310
1.	ROM	SPI_PUBLIC_CODE_ROM	1412
		SPI_PRIVATE_CODE_ROM	4264
		CONST_ROM_UNSPECIFIED	100
		SPI_CFG_DBTOC_UNSPECIFIED	48
		SPI_CFG_DATA_UNSPECIFIED	164
		SPI_FAST_CODE_ROM	992
2.	RAM	RAM_UNSPECIFIED	4
		NOINIT_RAM_1BIT	5
		NOINIT_RAM_8BIT	5
		NOINIT_RAM_16BIT	10
		NOINIT_RAM_UNSPECIFIED	90
		SPI_CFG_RAM_UNSPECIFIED	0

The details of memory usage for the typical configuration, with DET enabled and all other configurations as provided in13.3.2.1*Configuration Example* are provided in this section.

Table 13-8 ROM/RAM Details with DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes for 701310
1.	ROM	SPI_PUBLIC_CODE_ROM	2432
		SPI_PRIVATE_CODE_ROM	4264
		CONST_ROM_UNSPECIFIED	100
		SPI_CFG_DBTOC_UNSPECIFIED	48
		SPI_CFG_DATA_UNSPECIFIED	164
		SPI_FAST_CODE_ROM	992

2.	RAM	RAM_UNSPECIFIED	4
		NOINIT_RAM_1BIT	5
		NOINIT_RAM_8BIT	5
		NOINIT_RAM_16BIT	10
		NOINIT_RAM_UNSPECIFIED	90
		SPI_CFG_RAM_UNSPECIFIED	0

13.4.2. Stack Depth

The worst-case stack depth for Driver Component is 216 bytes for the typical configuration provided in Section 13.3.2.1*Configuration Example*.

13.4.3. Throughput Details

The throughput details of the APIs for the configuration mentioned in the Section13.3.2.1 *Configuration Example*. The clock frequency used to measure the throughput is 160 MHz for all APIs.

Table 13-9 Throughput Details Of The APIs

Sl. No.	API Name	Throughput in microseconds for 701310	Remarks
1.	Spi_Init	3.690	-
2.	Spi_DeInit	1.710	-
3.	Spi_WriteIB	0.810	-
4.	Spi_AsyncTransmit	8.820	-
5.	Spi_ReadIB	0.720	-
6.	Spi_SetupEB	0.270	-
7.	Spi_GetStatus	0.180	-
8.	Spi_GetJobResult	0.360	-
9.	Spi_GetSequenceResult	0.360	-
10.	Spi_GetVersionInfo	0.360	-
11.	Spi_SyncTransmit	0.360	-
12.	Spi_GetHWUnitStatus	0.180	-
13.	Spi_Cancel	0.810	-
14.	Spi_SetAsyncMode	0.360	SPI_INTERRUPT_ MODE
15.	Spi_SetAsyncMode	0.360	SPI_POLLING_ MODE
16.	Spi_MainFunction_Handling	1.170	-

Chapter 14 Release Details

SPI Driver Software

Version: 1.6.0

Revision History

Sl.No.	Description	Version	Date
1.	Initial Version	1.0.0	25-Oct-2013
2.	Following changes are made. 1. Chapter 2 is updated for referenced documents version. 2. Section 13.1.1 is updated for adding the device names. 3. Section 13.2 is updated for assembler and linker details. 4. Section 13.3 is updated for naming convention change of parameter definition files. 5. Chapter 14 is updated for SPI driver component version information.	1.0.1	28-Jan-2014
3.	Following changes are made. 1. In section 13.4.3,Throughput Details are updated. 2. In Section 13.4.1,ROM/RAM Usage are updated. 3. In Section13.3.1,Sample Application Structure API details are updated. 4. In chapter 5, Architecture Details Spi API are updated. 5. In chapter 14, Release Details Spi software version is updated.	1.0.2	02-May-2014
4.	Following changes are made. 1.Unwanted Device names are removed. 2.In page no 47, header is updated.	1.0.3	12-May-2014
5.	Following changes are made. 1. Chapter 4 is updated for CS logs and note is added regarding general limitation of the serial controllers. 2. Note is added regarding the usage of the parameter 'SpiCsHoldTiming' for synchronous transmission. 3. Name of Table 4-4 and 4-5 is updated. 4. Table 4-3, Table 4-4 and Table 4-5 are updated for Static configuration. 5. Section 4.1, description of parameter 'SpiTimeOut' is updated. 6. In Section 4.1 Note is added regarding extended data size supported by FIFO. 7. Sections 13.4, ROM/RAM and Throughput Details are updated. 8. Section 4.6 Deviation list is updated. 9. Section 13.2.1, 13.2.2 and 13.2.3 are updated for compiler, linker and assembler details. 10. Chapter 14, Release Details are updated. 11. Section 11.2 is updated to delete error code 'SPI_E_SELF_TEST_FAILURE' for Self-Test and SPI_E_READBACK_FAILURE for readback. 12. Chapter 12 Memory Organization is updated to correct section name SPI_START_SEC_CODE_FAST to SPI_FAST_CODE_ROM. 13. Section 13 is updated for device names and to add Parameter Definition files section. 14. Chapter 8 is update to include rh850_types.h file 15. In chapter 4 note is added regarding the DMA access for local RAM area.	1.0.4	27-Oct-2014

6.	Following changes are made. 1. Section 4.1 is updated to correct the notes and spell checks. 2. Revision history points are corrected	1.0.5	19-Nov-2014
7.	Following changes are made: 1.Updated Chapter 2 ‘Reference Documents’ to correct the name and version of device manual. 2.Information regarding Interrupt vector table has been provided in section 4.1 ‘General’. 3.In Chapter 13, ’P1M Specific Information’ P1M 4.0.3 supported devices are updated. 4.Table 13-1 PDF information updated for P1M 4.0.3 supported devices. 5.Section 13.1.1 has been updated to include the translation header file for all P1M 4.0.3 supporting devices. 6.Updated section 13.3.1 ‘Sample Application Structure’ to add all the supported devices for P1M 4.0.3. 7.Updated section 13.3.2 ‘Building the Sample Application’ to add configuration details for the device 701310. 8.Updated section 13.4 ‘Memory and Throughput’ for the device R7F701310. 9.Updated chapter 14 ‘Release Details’ to correct the SPI driver version. 10.Removed section ‘Compiler, Linker and Assembler’ from chapter 13. 11.Updated table 6.1 in Chapter 6 ‘Registers Details’.	1.0.6	29-April-2015

AUTOSAR MCAL R4.0.3 User's Manual
SPI Driver Component Ver.1.0.6
Embedded User's Manual

Publication Date: Rev.0.02, April 29, 2015

Published by: Renesas Electronics Corporation

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.

2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A. Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited

1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada

Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited

Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH Arcadiastrasse 10, 40472 Düsseldorf, Germany Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.

7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China

Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.

Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China

Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited

Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong

Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.

7F, No. 363 Fu Shing North Road Taipei, Taiwan

Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.

1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632

Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.

Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia

Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.

11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.0

AUTOSAR MCAL R4.0.3

User’s Manual

Renesas Electronics Corporation