User's Manual

AUTOSAR MCAL R4.0.3

User's Manual

SPI Driver Component Ver.1.0.2
Embedded User's Manual

Target Device:
RH850/P1x-C

Renesas Electronics
www.renesas.com

Rev.1.00 Feb 2017

Notice

1. 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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other disputes involving 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, including but not limited to, the product data, drawing, chart, program, algorithm, application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics products.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "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 etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (space and undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics 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 ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of failure or malfunction of Renesas Electronics products, 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 by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall 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. You shall not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
11. This document shall not be reprinted, 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.

(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
CSIH/CSIG, CSIG	Enhanced Queued Clocked Serial Interface.
DEM	Diagnostic Event Manager
DET/Det	Development Error Tracer
DIO	Digital Input Output
DMA	Direct Memory Access
EB	External Buffer
ECU	Electronic Control Unit
EDL	Extended Data Length
EEPROM	Electrically Erasable Programmable Read-Only Memory
GNU	GNU's Not Unix
GPT	General Purpose Timer
HW	HardWare
IB	Internal Buffer
Id	Identifier
I/O	Input/Output
ISR	Interrupt Service Routine
MCAL	Microcontroller Abstraction Layer
MHz	Mega Hertz
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-C 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:

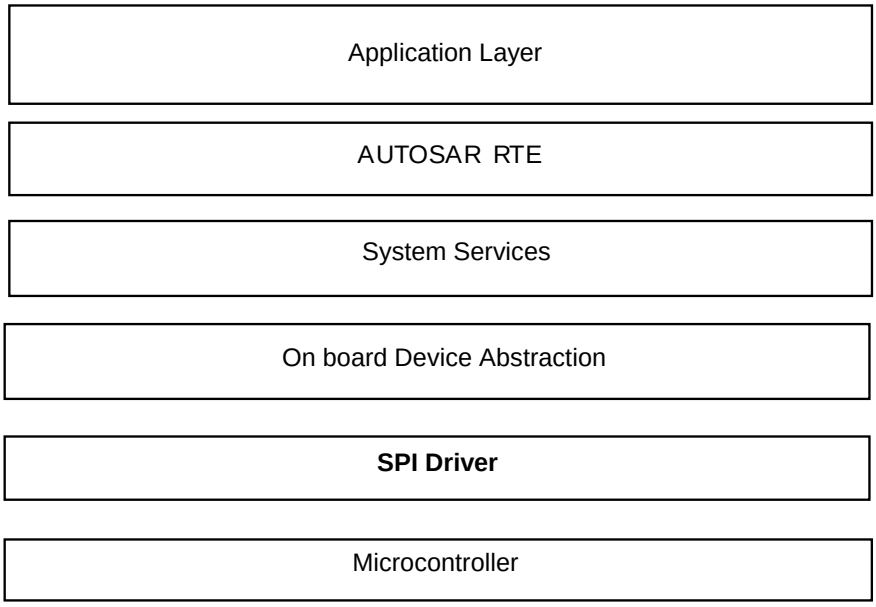

Figure 1-1 System Overview Of AUTOSAR Architecture

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:

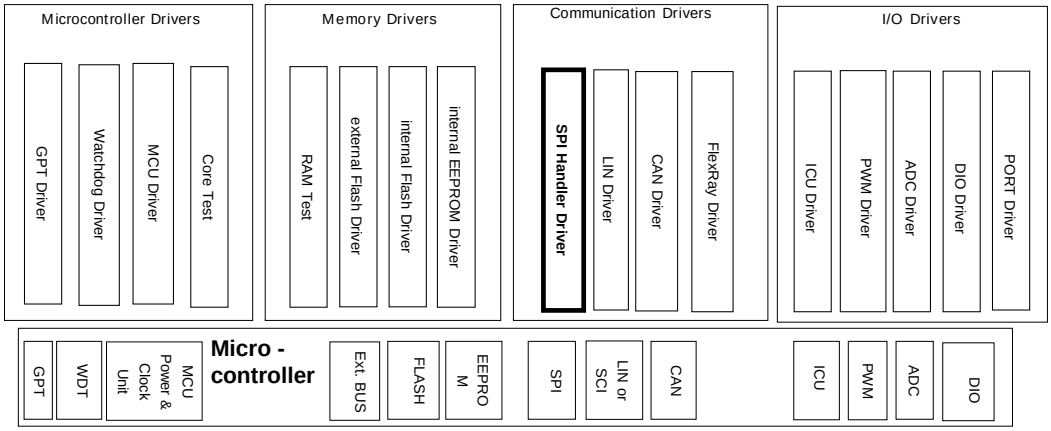

Figure 1-2 System Overview Of The SPI Driver In AUTOSAR MCAL Layer

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_Hardware.c, Spi_Hardware.h, 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(P1X-C Specific information)	This section provides P1x-C specific information like ISR Function, the details of the P1x-C Sample Application and its folder structure and the information about RAM/ROM usage, stack depth and throughput details.
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_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.	r01uh0517ej0070_rh850p1x-c_Open.pdf	Rev.1.00
4.	Specification of Compiler Abstraction (AUTOSAR_SWS_CompilerAbstraction.pdf)	3.2.0
5.	Specification of Memory Mapping (AUTOSAR_SWS_MemoryMapping.pdf)	1.4.0
6.	Specification of Platform Types (AUTOSAR_SWS_PlatformTypes.pdf)	2.5.0

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 "R20UT3660EJ0100-AUTOSAR.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
\Spi_RegWrite.h

X1X\P1x-C\modules\spi\sample_application\<SubVariant>\make\ghs
\App_Spi_P1x-C_Sample.mak
\App_Spi_P1x-C_Sample.ld

X1X\P1x-C\modules\spi\generator
\R403_SPI_P1x-C_BSWMDT.arxml

X1X\P1x-C\modules\spi\user_manual

(User manuals will be available in this folder)

Note: 1. <AUTOSAR_version> should be 4.0.3
2. <SubVariant> can be P1H-C, P1H-CE, P1M-C.

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 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 jobs configurable 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
- 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 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 behaviour 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.

  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 suppressedusing ‘SpiAlreadyInitDetCheck’ parameter when DET is enabled.When DET is disabled there is no impact of “SpiAlreadyInitDetCheck” parameter.
- Hardware high priority sequence mechanism is not supported for P1x-C devices.
- 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()), else it is dynamic.
- SpiTimeOut has been added to have the hold on functions and ongoing process of APIs, SpiTimeOut keeps the track of time and breaks loop if it is exceeds the defined time.

Table 4-1 Registers to be Configured for Static Configuration

CSIH HW Unit
CSIHnCTL0
CSIHnCTL1
CSIHnCTL2
CSIHnCFGx
CSIHnBRSy

Table 4-2 Channel container parameters

Parameter in channel container	Registers linked
SpiDataWidth	CSIHnCFGx.CSIHnDLSx
SpiTransferStart	CSIHnCFGx.CSIHnDIRx

Table 4-3 Job container parameters

Parameter in job container	Registers linked
SpiPortPinSelect	CSIHnTXOW.CSIHnCSx CSIHnCTL1.CSIHnCSx

- Table 4-1 contains the registers that must be configured inside Spi_Init() function.
- All the parameters in channel/job/external devices containers linked to a hardware unit mentioned in Table 4-2 and 4-3 should be same for Static Configuration.
- 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).

- When configuring DMA mode, the number of buffers configured shall be greater than 1 in the case of Direct Access Mode and Fifo Mode.

- The notifications should be called from user's complex driver ISRs.

- When using DMA, 'SpiDataWidthSelection' in 'General' container shall be 'BITS_16', the user shall setup the buffer(EB or IB) in the application as type 'Spi_DataType' for channels that are configured for DMA and fill required data(8 or 16) as configured in 'SpiDataWidth' in 'SpiChannel'.

- The SPI DMA type is specified by the parameter SPI_DMA_TYPE_USED.

- The Buffers used for transmission/reception using DMA shall be initialized and configured in Retention RAM or Global RAM.

  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.)

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_Hardware.c, Spi_Hardware.h, 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_Hardware.c, Spi_Hardware.h,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.

- The user shall configure the exact Module Short Name Spi in configurations when reloading, as specified in config.xml file and the same shall be given in command line.

- 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.

- User have the responsibility to enable or disable the critical protection using the parameter SpiCriticalSectionProtection. By enabling parameter SpiCriticalSectionProtection, Microcontroller HW registers which suffer from concurrent access by multiple tasks are protected.

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 4-4 User Mode and Supervisory Mode

Sl. No.	API name	Interrupt mode		Polling mode		Known limitation in User Mode
		user mode	supervisor mode	user mode	supervisor mode	
1.	Spi_Init	-	x	-	x	The IMR and INTC registers are accessed inside this function. Hence it should not be invoked in User mode.
2.	Spi_DeInit	-	x	-	x	
3.	Spi_WriteIB	x	x	x	x	
4.	Spi_AsyncTransmit	-	x	-	x	The IMR and INTC registers are accessed inside this function. Hence it should not be invoked in User mode.
5.	Spi_ReadIB	x	x	x	x	

Sl. No.	API name	Interrupt mode		Polling mode		Known limitation in User Mode
		user mode	supervisor mode	user mode	supervisor mode	
6.	Spi_SetupEB	x	x	x	x	
7.	Spi_GetStatus	x	x	x	x	
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	The IMR and INTC registers are accessed inside this function. Hence it should not be invoked in User mode.
12.	Spi_Cancel	-	x	-	x	The IMR and INTC registers are accessed inside this function. Hence it should not be invoked in User mode.
13.	Spi_SetAsyncMode	-	x	-	x	The IMR and INTC registers are accessed inside this function. Hence it should not be invoked in User mode.
14.	Spi_MainFunction_Handling	-	-	-	x	The IMR and INTC registers are accessed inside this function. Hence it should not be invoked in User mode
15.	Spi_GetHWUnitStatus	x	x	x	x	
16.	Spi_GetErrorInfo	x	x	x	x	
17.	Spi_SelfTest	-	x	-	x	The IMR and INTC registers are accessed inside this function. Hence it should not be invoked in User mode
18.	All ISRs	-	x	-	-	The IMR and INTC registers are accessed inside this function. Hence it should not be invoked in User mode

Note: Implementation of Critical Section is not dependent on MCAL. Hence Critical Section is not considered to the entries for User mode in the above table.

4.4. Memory modes

The SPI Driver will use different memory modes. The following four modes can be configured.

Table 4-5 HW unit and Memory Mode Selection

HW unit	Memory
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:

- 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.

The information about the API's and the protected resources by the critical section are given in the following table.

Table 4-6 SPI Driver Critical section protection List

API Name	Exclusive Area Type	Protected Resources
Spi_AsyncTransmit	SPI_RAM_DATA_PROTECTION	During communication the status of sequence, job, corresponding hardware unit and communicating data are protected.
Spi_SyncTransmit	SPI_RAM_DATA_PROTECTION	During communication the status of sequence, job, corresponding hardware unit and communicating data are protected.

API Name	Exclusive Area Type	Protected Resources
Spi_Cancel	SPI_RAM_DATA_PROTECTION	During cancelling the status of sequence are protected.

Note: The highest measured duration of a critical section was 1.162 micro seconds measured for **Spi_AsyncTransmit** API.

4.6. Deviation List

Table 4-7 SPI Driver Deviation List

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.	-

Sl. No.	Description	AUTOSAR Bugzilla
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.	-
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	From the file Lcfg.c only notification related structure has been removed.	As per mantis #8421
15	There will be an inactive state in between Chip Select during communication, when channel properties are different.	As per JIRA ARDAAAF-383

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:

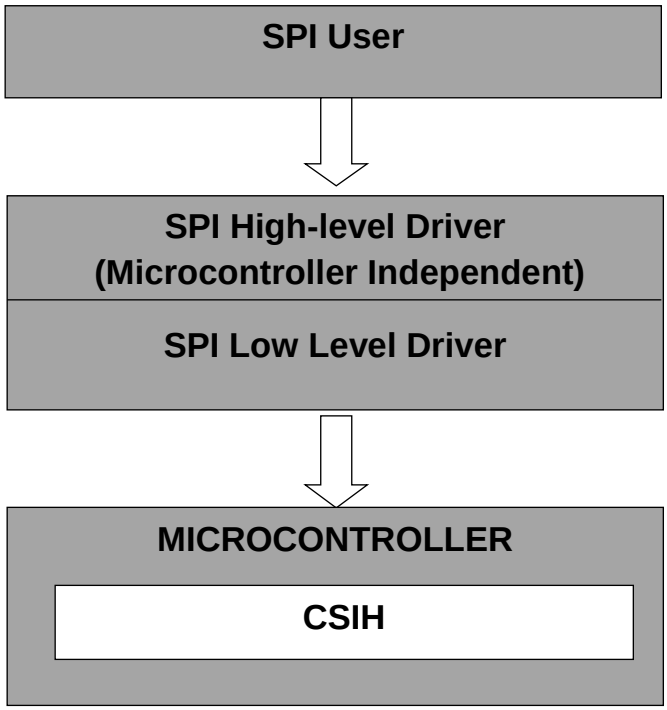

Figure 5-1 SPI Driver Architecture

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:

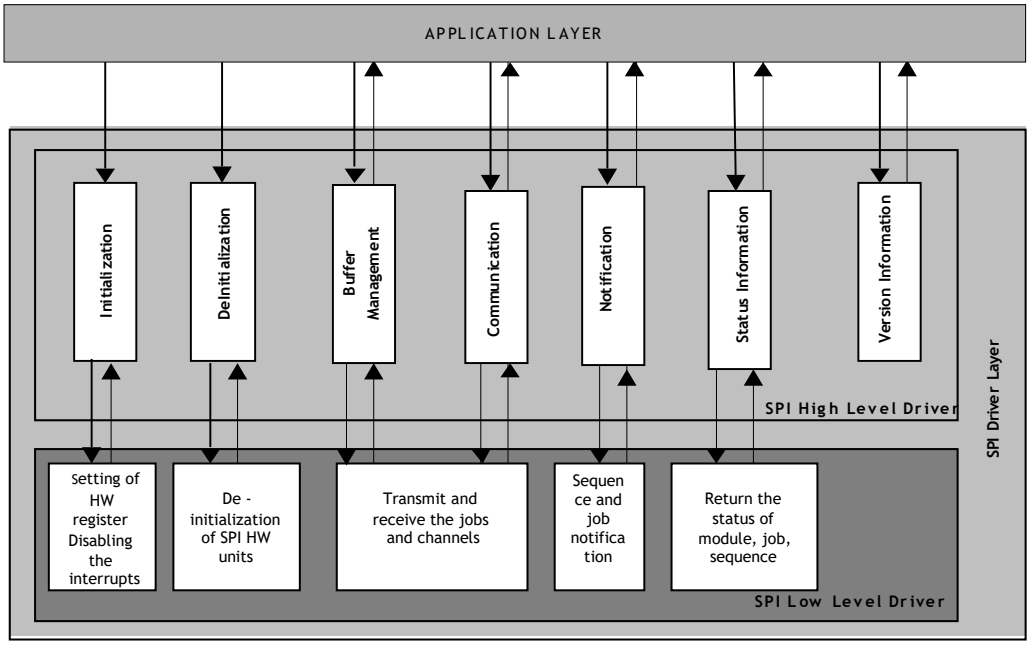

Figure 5-2 Component Overview Of SPI Driver Component

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	Register Access R/W/RW	Macro/Variable
Spi_Init	CSIHnCTL0	SpiMemoryModeSelection	W	SPI_ZERO
	DCSTCn	-	W	SPI_DMA_STR_CLEAR
	DCSTn	-	R	-
	DCENn	-	W	SPI_DMA_DCEN_DISABLE
	DSAn	SpiDma	W	LpDmaConfig->ulTxRxRegAddress
	DTCTn	SpiTxDmaChannel/ SpiRxDmaChannel	W	SPI_DMA_16BIT_TX_SETTINGS SPI_DMA_16BIT_RX_SETTINGS
	DDAn	SpiDma	W	LpDmaConfig->ulTxRxRegAddress
	DTFRn	SpiTxDmaChannel/ SpiRxDmaChannel	W	LpDmaConfig->usDmaDtfrRegValue
	CSIHnCTL1	SpiCsInactiveAfterLastData, SpiDataWidth	W	LunDataAccess1.ulRegData
	ICRn	-	W	SPI_CLR_INT_REQ
	IMRn	SpiHwUnitSelection and SpiMemoryModeSelection	W	Spi_GstHWUnitInfo[LddHWUnit].usRxImrMask, Spi_GstHWUnitInfo[LddHWUnit].pTxImrAddress, Spi_GstHWUnitInfo[LddHWUnit].pErrorImrAddress, Spi_GstHWUnitInfo[LddHWUnit].usRxImrMask, Spi_GstHWUnitInfo[LddHWUnit].pTxImrAddress, LpHWUnitInfo->usTxCancelImrMask, Spi_GstHWUnitInfo[LddHWUnit].pErrorImrAddress
	CSIHnTX0W	-	W	LunDataAccess1.ulRegData
	CSIHnSTCR0	-	W	SPI_CSIH_CLR_STS_FLAGS
	CSIHnSTR0	-	R	-
	CSIHnCTL2	SpiInputClockSelect SpiBaudrateConfiguration	W	LpJobConfig->usCtl2Value & SPI_CSIH_PRE_MASK
	CSIHnMCTL0	SpiMemoryModeSelection	W	LpJobConfig->usMCtl0Value
	CSIHnBRSy	SpiInputClockSelect SpiBaudrateConfiguration	W	(LpJobConfigCSConfig->usCtl2Value) & SPI_CSIH_BRS_MASK

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIHnCFGx	SpiDataWidth SpiParitySelection SpiTransferStart SpiDataShiftEdge SpiShiftClockIdleLevel	W	LunDataAccess1.ulRegData
	ECCCSIHnCTL	SpiECCSelfTest	R/W	SET_EC1EDIC_EC2EDIC ECC_CTL_ECEMF_SET ECC_CTL_ECER1F_ECER2F_CLEAR CTL_ERRCLR_FLAG CTL_2BIT_ERRCLR_FLAG CTL_1BIT_ERR_FLAG
	ECCCSIHnTMC	SpiECCSelfTest	W	SET_TMC_BITS SET_TEST_DISABLE
	ECCCSIHnTRC	SpiECCSelfTest	W	TRC_ERDB_INITIALIZE
	ECCCSIHnTED	SpiECCSelfTest	R/W	RAM_INITIALIZE, ALL_ZERO_PATTERN, ALL_ONE_PATTERN, TWO_BIT_PATTERN
	CSIHnRX0H	-	R	-
	CSIHnMCTL1	SpiMemoryModeSelection	W	SPI_CTL_32BIT_REG_VAL
	CSIHnMCTL2	SpiMemoryModeSelection	W	SPI_CTL_32BIT_REG_VAL
	CSIHnMRWP0	-	RW	LunDataAccess1.ulRegData
Spi_DeInit	CSIHnCTL0	SpiMemoryModeSelection	W	SPI_ZERO
	CSIHnCTL1	-	W	SPI_ZERO
	CSIHnCTL2	-	W	SPI_CTL2_16BIT_REG_DEINIT
	CSIHnMCTL0	-	W	SPI_MCTL0_16BIT_REG_DEINIT
	CSIHnMCTL1	-	W	SPI_CTL_32BIT_REG_MASK
	CSIHnMCTL2	-	W	SPI_CTL_32BIT_REG_MASK
	CSIHnSTCR0	-	W	SPI_CTL_16BIT_REG_DEINIT
	CSIHnMRWP0	-	W	SPI_CTL_32BIT_REG_MASK
	CSIHnBRSy	-	W	SPI_CTL_16BIT_REG_DEINIT
	DSAn	-	W	SPI_DMA_DEINIT
	DDAn	-	W	SPI_DMA_DEINIT
	DCENn	-	W	SPI_DMA_DCEN_DISABLE
	DTCTn	-	W	SPI_DMA_DEINIT
	DTFRRQCn	-	W	SPI_DMA_DRQ_CLEAR
	DCSTCn	-	W	SPI_DMA_STR_CLEAR
	DTFRRQn	-	R	-

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	DCSTn	-	R	-
	DTFRn	-	W	SPI_DMA_DEINIT
	CSIHnCFGx		W	SPI_CTL_32BIT_REG_VAL
	IMRn	SpiHwUnitSelection and SpiMemoryModeSelection	W	Spi_GstHWUnitInfo[LddHWUnit].usRxImrMask, Spi_GstHWUnitInfo[LddHWUnit].pTxImrAddress, Spi_GstHWUnitInfo[LddHWUnit].pErrorImrAddress, Spi_GstHWUnitInfo[LddHWUnit].usRxImrMask, Spi_GstHWUnitInfo[LddHWUnit].pTxImrAddress, LpHWUnitInfo->usTxCancelImrMask, Spi_GstHWUnitInfo[LddHWUnit].pErrorImrAddress
	ICRn	-	W	SPI_CLR_INT_REQ
	CSIHnSTR0	-	R	-
Spi_WriteIB	CSIHMCTL0	SpiMemoryModeSelection	W	LusMctlData SPI_TX_ONLY_MODE_SET SPI_DUAL_BUFFER_MODE_SET
	CSIHnMRWP0	-	RW	LunDataAccess1.ulRegData
	CSIHnTX0W	-	W	LunDataAccess1.ulRegData
Spi_AsyncTransmit	CSIHnMCTL0	-	W	LpJobConfig->usMCtl0Value
	CSIHnCTL0	SpiMemoryModeSelection	W W	SPI_RESET_PWR SPI_SET_DIRECT_ACCESS SPI_SET_MEMORY_ACCESS
	CSIHnSTCR0	-	W	SPI_CLR_STS_FLAGS
	CSIHnSTR0	-	R	-
	CSIHnCTL1	SpiCsInactiveAfterLastData, SpiDataWidth	W	LunDataAccess1.ulRegData LpJobConfig->ulMainCtl1Value SPI_SET_SLIT
	DCSTCn	-	W	SPI_DMA_STR_CLEAR
	DCSTn	-	R	-
	DCENn	-	W	SPI_DMA_DCEN_DISABLE
	DTCTn	-	W	SPI_DMA_FIXED_TX_SETTINGS SPI_DMA_INV_TX_SETTINGS LddNoOfBuffers SPI_DMA_STR_REQ SPI_DMA_ONCE SPI_DMA_FIXED_RX_SETTINGS

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	DSAn	-	W	(uint32)LpTxData
	DTFRn	-	W	(uint32)SPI_ZERO (uint32)(LpDmaConfig->usDmaDtfrRegValue
	DCSTSn	-	W	SPI_DMA_STR
	DTCn	-	W	SPI_ONE
	DTFRRQCn	-	W	SPI_DMA_DRQ_CLEAR
	DDAn	-	W	(uint32)(&Spi_GddDmaRxData)
	CSIHnCTL2	SpiBaudrateRegisterSelect	W	LpJobConfig->usCtl2Value
	CSIHnMCTL2	-	W	LunDataAccess1.ulRegData
	CSIHnTX0W	-	W	LunDataAccess1.ulRegData, LunDataAccess2.ulRegData, LpDataAccess->ulRegData
	CSIHnTX0H	-	W	LddData, LunDataAccess2.usRegData5[SPI_ZERO]
	CSIHnCFGx	SpiCsIdleTiming, SpiCsHoldTiming, SpiCsInterDataDelay, SpiCsSetupTime, SpiCsIdleEnforcement	W	LunDataAccess1.ulRegData
	CSIHnBRSy	SpiBaudrateConfiguration	W	Csih_BaseAddress[LddHWUnit]->BRSy
	IMRn	SpiHwUnitSelection and SpiMemoryModeSelection	W	Spi_GstHWUnitInfo[LddHWUnit].ulRxImrMask, Spi_GstHWUnitInfo[LddHWUnit].ulTxImrMask, Spi_GstHWUnitInfo[LddHWUnit].ulErrorImrMask, Spi_GstHWUnitInfo[LddHWUnit].ulTxCancelImrMask
	ICRn	-	W	SPI_CLR_INT_REQ
	DTFRRQn	-	R	-
	CSIHnRX0H	-	R	-
	CSIHnRX0W	-	R	-
Spi_ReadIB	CSIHnRX0W	-	W	LunDataAccess2.ulRegData
	CSIHnRX0H	-	W	LunDataAccess2.usRegData5[SPI_ONE], LunDataAccess2.usRegData5[SPI_ZERO]
	CSIHnMRWP0	-	RW	LunDataAccess1.ulRegData
Spi_SetupEB	-	-	-	-
Spi_GetStatus	-	-	-	-
Spi_GetJobResult	-	-	-	-
Spi_GetSequenceResult	-	-	-	-
Spi_SyncTransmit	CSIHnMCTL0	-	W	LpJobConfig->usMCtl0Value

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIHnCTL0	-	W W	SPI_RESET_PWR SPI_SET_DIRECT_ACCESS SPI_SET_PWR SPI_ZERO
	CSIHnRX0H	-	RW	LunDataAccess3.ulRegData, Spi_GusSynDataAccess
	CSIHnSTR0	-	R	-
	CSIHnSTCR0	-	W	SPI_DCE_ERR_CLR, SPI_PE_ERR_CLR, SPI_OFE_ERR_CLR
	CSIHnCTL1	SpiCsInactiveAfterLastData, SpiDataWidth	W	LunDataAccess1.ulRegData, (LpMainOsBaseAddr->ulMainCTL1 | ~SPI_CSRI_AND_MASK
	CSIHnCTL2	SpiBaudrateRegisterSelect	W	LunDataAccess1.ulRegData, LpJobConfig->usCtl2Value
	CSIHnTX0W	-	W	LpJobConfig->usCtl2Value, LunDataAccess3.ulRegData
	CSIHnBRSy	SpiBaudrateConfiguration	W	Csih_BaseAddress[LddHWUnit]->BRSy , LpJobConfig->usCtl2Value & SPI_CSIH_BRS_MASK
	ICRn	-	W	SPI_CLR_INT_REQ
	CSIHnCFGx	SpiCsIdleTiming, SpiCsHoldTiming, SpiCsInterDataDelay, SpiCsSetupTime, SpiCsIdleEnforcement	W	LunDataAccess1.ulRegData
Spi_GetHWUnitStatus	CSIHnSTR0	-	R	-
Spi_Cancel	CSIHnCTL0	-	R/W	SPI_SET_JOBE
	IMRn	-	W	Spi_GstHWUnitInfo[LddHWUnit].ulTxCancelImrMask
	ICRn	-	W	SPI_CLR_INT_REQ
Spi_SetAsyncMode	IMRn	SpiHwUnitSelection and SpiMemoryModeSelection	W	Spi_GstHWUnitInfo[LddHWUnit].ulRxImrMask, Spi_GstHWUnitInfo[LddHWUnit].ulTxImrMask, Spi_GstHWUnitInfo[LddHWUnit].ulErrorImrMask, Spi_GstHWUnitInfo[LddHWUnit].ulTxCancelImrMask
	ICRn	-	w	SPI_CLR_INT_REQ
Spi_MainFunction_Handling	CSIHnCTL0	-	W	SPI_SET_PWR
	CSIHnRX0H	-	R	-
	CSIHnTX0W	-	W	LunDataAccess1.ulRegData
	CSIHnTX0H	-	W	LddData LunDataAccess2.usRegData5[0]

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIHnRX0W	-	R	-
	CSIHnMCTL2	SpiMemoryModeSelection	W	LunDataAccess1.ulRegData
	ICRn	-	W	SPI_CLR_INT_REQ
	DCSTCn	-	W	SPI_DMA_STR_CLEAR
	DCSTn	-	R	-
	DCENn	-	W	SPI_DMA_DCEN_DISABLE SPI_DMA_DCEN_ENABLE
	DTCTn	-	W	SPI_DMA_FIXED_TX_SETTINGS
	DSAn	-	W	(uint32)LpTxData
	DTFRn	-	W	(uint32)SPI_ZERO (uint32)(LpDmaConfig->usDmaDtfrRegValue
	DCSTSn	-	W	SPI_DMA_STR
	DTCn	-	W	SPI_ONE
	DTFRRQCn	-	W	SPI_DMA_DRQ_CLEAR
	DDAn	-	W	(uint32)(&Spi_GddDmaRxData)
	CSIHnSTCR0	-	W	SPI_CLR_STS_FLAGS
	CSIHnSTR0	-	R	-
	CSIHnCTL1	SpiCsInactiveAfterLastData, SpiDataWidth	W	LunDataAccess1.ulRegData LpJobConfig->ulMainCtl1Value SPI_SET_SLIT
	CSIHnCTL2	SpiBaudrateRegisterSelect	W	LpJobConfig->usCtl2Value
	IMRn	SpiHwUnitSelection and SpiMemoryModeSelection	W	Spi_GstHWUnitInfo[LddHWUnit].ulRxImrMask, Spi_GstHWUnitInfo[LddHWUnit].ulTxImrMask, Spi_GstHWUnitInfo[LddHWUnit].ulErrorImrMask, Spi_GstHWUnitInfo[LddHWUnit].ulTxCancelImrMask
	CSIHnCFGx	SpiCsIdleTiming, SpiCsHoldTiming, SpiCsInterDataDelay, SpiCsSetupTime, SpiCsIdleEnforcement	W	LunDataAccess1.ulRegData
	CSIHnBRSy	SpiBaudrateConfiguration	W	Csih_BaseAddress[LddHWUnit]->BRSy
	CSIHnMCTL0	-	W	LpJobConfig->usMCtl0Value
	DTFRRQn	-	R	-
Spi_GetVersionInfo	-	-	-	-
Spi_GetErrorInfo	-	-	-	-
Spi_SelfTest	CSIHnRX0H	-	R	-

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIHnCTL0	SpiLoopBackSelfTest	W	SPI_SET_DIRECT_ACCESS SPI_ZERO
	CSIHnCTL1	SpiLoopBackSelfTest	W	SPI_LOOPBACK_ENABLE SPI_ZERO SPI_SET_SLIT LunDataAccess1.ulRegData
	CSIHnCTL2	SpiLoopBackSelfTest	W	SPI_LOOPBACK_CSIH_CNTRL2_VALUE SPI_ZERO ((LpJobConfig->usCtl2Value) & SPI_CSIH_PRE_MASK)
	CSIHnSTCR0	SpiLoopBackSelfTest	W	SPI_CSIH_CLR_STS_FLAGS SPI_PE_ERR_CLR SPI_ZERO
	CSIHnCFGx	SpiLoopBackSelfTest	W	SPI_LOOPBACK_DLS_SETTING SPI_ZERO LunDataAccess1.ulRegData
	CSIHnBRSy	SpiLoopBackSelfTest	W	SPI_LOOPBACK_CSIH_BRS0_VALUE SPI_ZERO ((LpJobConfigCSConfig- >usCtl2Value) & SPI_CSIH_BRS_MASK)
	CSIHnTX0W	SpiLoopBackSelfTest	W	SPI_LOOPBACK_DATA SPI_ZERO
	CSIHnSTR0	SpiLoopBackSelfTest	R	-
	ECCCSIHnCTL	SpiECCSelfTest	R/W	SET_EC1EDIC_EC2EDIC ECC_CTL_ECEMF_SET ECC_CTL_ECER1F_ECER2F_CLEAR CTL_ERRCLR_FLAG CTL_2BIT_ERRCLR_FLAG CTL_1BIT_ERR_FLAG
	ECCCSIHnTMC	SpiECCSelfTest	W	SET_TMC_BITS SET_TEST_DISABLE
	ECCCSIHnTRC	SpiECCSelfTest	W	TRC_ERDB_INITIALIZE
	ECCCSIHnTED	SpiECCSelfTest	R/W	RAM_INITIALIZE, ALL_ZERO_PATTERN, ALL_ONE_PATTERN, TWO_BIT_PATTERN
	IMRn	SpiHwUnitSelection and SpiLoopBackSelfTest	W	Spi_GstHWUnitInfo[LddHWUnit].usRxImrMask, Spi_GstHWUnitInfo[LddHWUnit].pTxImrAddress, Spi_GstHWUnitInfo[LddHWUnit].pErrorImrAddress, Spi_GstHWUnitInfo[LddHWUnit].usRxImrMask, LpHWUnitInfo->usTxCancelImrMask
	ICRn	-	W	SPI_CLR_INT_REQ

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIHnMCTL0	SpiMemoryModeSelection	W	LpJobConfig->usMCtl0Value

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
- Spi_Hardware.h

The C source file generated by SPI Driver Generation Tool:

- Spi_PBcfg.c
- Spi_Lcfg.c
- Spi_Hardware.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
- Spi_RegWrite.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 Stub C header files:

- Compiler.h
- Compiler_Cfg.h
- MemMap.h
- Platform_Types.h
- rh850_Types.h
- Det.h
- Rte.h
- SchM.h
- SchM_Spi.h
- Dem.h
- Dem_cfg.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
Spi_Hardware.h	This file contains the #define macros for the hardware registers to be used by the driver.
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_Hardware.c	This file contains the reference objects for the structures of hardware register which is defined in device header file.
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 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.

File	Details
Compiler_Cfg.h	This file contains the memory and pointer classes.
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
Platform_Types.h	This file provides provision for defining platform and compiler dependent types.
Spi_RegWrite.h	This file contains macro for register write verify check.
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.
Det.h	This file is a stub for DET Component.
Rte.h	This file is a stub for Rte Component.
SchM.h	This file is a stub for Schm Component.
SchM_Spi.h	Header file information for Schm application.
Dem.h	This file is a stub for DEM component.
Dem_cfg.h	This file contains the stub values for Dem_Cfg.h.

Chapter 9 Generation Tool Guide

For information on the SPI Driver Component Code Generation Tool, please refer “R20UT3660EJ0100-AUTOSAR.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(IdId) 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
	SPI_INTERRUPT_MODE	Streaming access mode
Description:	Specifies the asynchronous mechanism mode for SPI busses handled asynchronously in LEVEL2.	

Following are the internal type definitions used by the SPI Driver module.

10.2.12. Spi_CommErrorType

Name:	Spi_CommErrorType		
Type:	Structure		
Element:	**Type**	**Name**	**Explanation**
	Spi_HWErrorsType	ErrorType	This is the type of the hardware error.
	Spi_HWUnitType	HwUnit	This is the hardware unit in which error is reported.
	Spi_SequenceType	SeqID	This is the sequence id for which error is reported.
	Spi_JobType	JobID	This is the job id for which error is
Description:	This type is used to provide the details regarding the type of hardware errors, hardware unit, sequence and job in which the errors were reported.		

10.2.13. Spi_HWErrorsType

Name:	Spi_HWErrorsType	
Type:	Enumeration	
Range:	SPI_NO_ERROR	No hardware error has occured.
	SPI_OVERRUN_ERROR	Over Run Error has occured.
	SPI_PARITY_ERROR	Parity Error has occured.
	SPI_DATA_CONSISTENCY_ERROR	Data Consistency Error has occured
	SPI_OVERFLOW_ERROR	Over Flow Error has occured
	SPI_ECC_1BIT_ERROR	1 Bit ECC Error has occured
Description:	This type defines different types of hardware errors in SPI driver.	

10.2.14. Spi_SelfTestType

Name:	Spi_SelfTestType
Type:	uint8
Range:	0 to 255
Description:	Specifies the type for self-test functionality.

10.2.15. Spi_ReturnStatus

Name:	Spi_ReturnStatus	
Type:	Enumeration	
Range:	SPI_SELFTEST_INVALID_MODE	When invalid argument other than LoopBack_Init/ LoopBack_Init_RunTime/ ECC_Init_RunTime/ ECC_Init are
	SPI_SELFTEST_DRIVERBUSY	When SelfTest API is invoked during any active transmission, i.e when driver is busy.
Range:	SPI_SELFTEST_PASS	SelfTest functionality is successful.
	SPI_SELFTEST_FAILED	SelfTest functionality is failed.
Description:	This type defines the return status of the self-test functionality.	

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	-
16	Spi_GetErrorInfo	-
17	Spi_SelfTest	-

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, Spi_SelfTest
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_DeInit, Spi_AsyncTransmit, Spi_Cancel, Spi_GetStatus, Spi_GetHWUnitStatus, Spi_GetJobResult, Spi_GetSequenceResult, Spi_WriteIB, Spi_ReadIB, Spi_SetupEB, Spi_SyncTransmit, Spi_SetAsyncMode, Spi_MainFunction_Handling and Spi_GetErrorInfo.
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 and Spi_GetVersionInfo.
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.
Sl. No.	**13**
Error Code	SPI_E_MAINFUNCTION_HANDLING_INVALIDMODE
Related API(s)	Spi_MainFunction_Handling
Source of Error	When the API invoked in SPI_INTERRUPT_MODE.

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_Init , Spi_SyncTransmit, Spi_MainFunction_Handling and Spi_SelfTest.
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, Spi_Init and Spi_SelfTest.
Source of Error	When Hardware data transmit timeout error is detected, This error will be reported to DEM
Sl. No.	**3**
Error Code	SPI_E_INT_INCONSISTENT
Related API(s)	All ISRs
Source of Error	DemEventParameter which shall be issued when Interrupt consistency error was detected.
Sl. No.	**4**
Error Code	SPI_E_ECC_SELFTEST_FAILURE
Related API(s)	Spi_Init and Spi_SelfTest
Source of Error	DemEventParameter which shall be issued when Ecc selft test error was detected.
Sl. No.	**5**
Error Code	SPI_E_LOOPBACK_SELFTEST_FAILURE
Related API(s)	Spi_Init and Spi_SelfTest
Source of Error	DemEventParameter which shall be issued when loop back self-test error was detected.
Sl. No.	**6**
Error Code	SPI_E_REG_WRITE_VERIFY
Related API(s)	All APIs accessing the registers
Source of Error	DemEventParameter which shall be issued when loop back self-test error was detected.

Chapter 12 Memory Organization

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

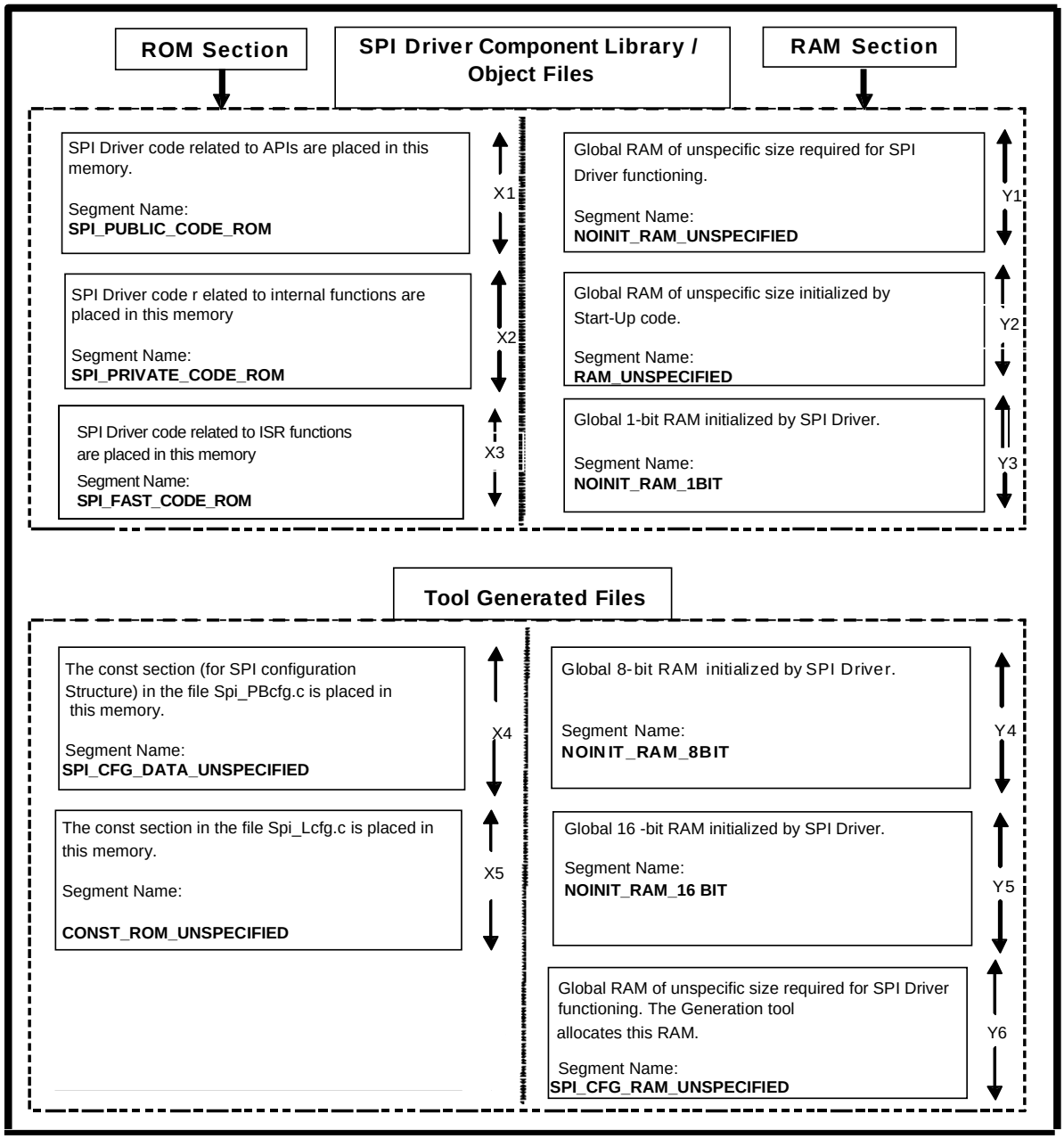

Figure 12-1 SPI Driver Component Driver Organization

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_DATA_UNSPECIFIED (X4): This section consists of SPI Driver Component constant configuration structures. This can be located in code memory.

CONST_ROM_UNSPECIFIED (X5): 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.

NOINIT_RAM_1BIT (Y3): This section consists of the global RAM variables of 1-bit size that are 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, Y3, Y4, Y5, Y6 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 P1x-C Specific Information

P1X-C supports following devices:

- R7F701370A(CPU1(PE1)), R7F701371(CPU1(PE1)), R7F701372(CPU1(PE1)), R7F701373, R7F701374.

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. 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 13-1 Interrupt Vector Table

Interrupt Source	Name of the ISR Function
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

Interrupt Source	Name of the ISR Function
INTCSIH3IR	SPI_CSIH3_TIR_ISR
	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
INTDMA00	SPI_DMA00_ISR
	SPI_DMA00_CAT2_ISR
INTDMA01	SPI_DMA01_ISR
	SPI_DMA01_CAT2_ISR
INTDMA02	SPI_DMA02_ISR
	SPI_DMA02_CAT2_ISR
INTDMA03	SPI_DMA03_ISR
	SPI_DMA03_CAT2_ISR
INTDMA04	SPI_DMA04_ISR
	SPI_DMA04_CAT2_ISR
INTDMA05	SPI_DMA05_ISR
	SPI_DMA05_CAT2_ISR
INTDMA06	SPI_DMA06_ISR
	SPI_DMA06_CAT2_ISR
INTDMA07	SPI_DMA07_ISR
	SPI_DMA07_CAT2_ISR
INTDMA08	SPI_DMA08_ISR
	SPI_DMA08_CAT2_ISR
INTDMA09	SPI_DMA09_ISR
	SPI_DMA09_CAT2_ISR
INTDMA10	SPI_DMA10_ISR
	SPI_DMA10_CAT2_ISR
INTDMA11	SPI_DMA11_ISR
	SPI_DMA11_CAT2_ISR
INTDMA12	SPI_DMA12_ISR
	SPI_DMA12_CAT2_ISR
INTDMA13	SPI_DMA13_ISR
	SPI_DMA13_CAT2_ISR
INTDMA14	SPI_DMA14_ISR
	SPI_DMA14_CAT2_ISR
INTDMA15	SPI_DMA15_ISR
	SPI_DMA15_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.

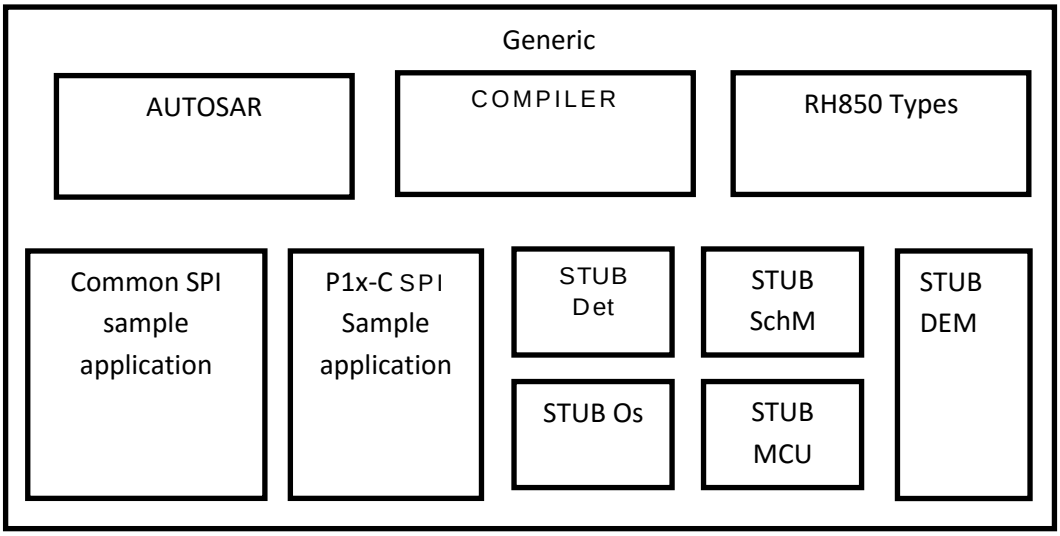

Figure 13-1 Overview Of SPI Driver Sample Application

13.2.1. Sample Application Structure

The Sample Application of the P1X-C is available in the path

X1X\P1x-C\modules\spi\sample_application

The Sample Application consists of the following folder structure

X1X\P1x-C\modules\spi\definition\<AUTOSAR_version>\common\
R403_SPI_P1x-C.arxml

X1X\P1x-C
\modules\spi\sample_application\<SubVariant>\<AUTOSAR_version>
\src\Spi_Lcfg.c
\src\Spi_PBcfg.c
\src\Spi_Hardware.c
\inc\Spi_Cfg.h
\inc\Spi_Cbk.h
\inc\Spi_ Hardware.h
\config\App_SPI_P1x-C_<Device_Name>_Sample.arxml

Note For P1x-C <Device_Name> can be 701370A, 701371, 701372, 701373, 701374.

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 passed 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 passed 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.
- The API Spi_GetErrorInfo copies Hardware Error Details to User Buffer.

13.2.2. Building Sample Application

13.2.2.1. 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_<SubVariant>_<Device_Name>_Sample.arxml.

Note For P1x-C <Device_Name> can be 701370A, 701371, 701372, 701373, 701374.

13.2.2.2. 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-C\common_family\make\ghs\<Compiler>”
- Now execute the batch file SampleApp.bat with following parameters SampleApp.bat Spi 4.0.3 <Device_name>
- After this, all the object files, map file and the executable file App_Spi_P1x-C_Sample.out will be available in the output folder:
  (“X1X\P1x-C\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-C\modules\spi\sample_application\<SubVariant>\<AUTOSAR_version>\config\App_SPI_P1X-C_701372_Sample.arxml"

- The database alone can be generated by using the following commands.
  make –f App_SPI_P1x-C_Sample.mak generate_spi_config

  make –f App_SPI_P1x-C_Sample.mak App_SPI_P1x-C_Sample.s37

- After this, a flash able Motorola S-Record file App_SPI_P1x-C_Sample.s37 is available in the output folder.

Note: The <Device_name> indicates the device to be compiled, which can be 701370A, 701371, 701372, 701373, 701374 and <SubVariant> can be P1H-C, P1H-CE, P1M-C.

13.3. Memory And Throughput

Typical Configuration

- DET OFF
- DMA disabled
- All other Pre-Compile Settings ON
- 2 16bit SPI channels
  - with external buffers
  - with internal buffers
- 2 SPI jobs
  - CSIH in direct access mode
- 2 external devices configured
- SpiLevelDelivered configured as 2

13.3.1. ROM/RAM Usage

The details of memory usage for the typical configuration, with DET disabled as provided in Table 13-2

Table 13-2 ROM/RAM Details Without DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes
1	ROM	DEFAULT_CODE_ROM	7782
		CONST_ROM_UNSPECIFIED	456
		CONST_ROM_32BIT	32

Sl. No.	ROM/RAM	Segment Name	Size in bytes
2	RAM	NOINIT_RAM_UNSPECIFIED	156
		RAM_UNSPECIFIED	2
		NOINIT_RAM_1BIT	8
		NOINIT_RAM_8BIT	9
		NOINIT_RAM_16BIT	22
		RAM_8BIT	2

The details of memory usage for the typical configuration, with DET enabled and all other configurations as provided in Table 13-3.

Table 13-3 ROM/RAM Details With DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes
1	ROM	DEFAULT_CODE_ROM	8856
		CONST_ROM_UNSPECIFIED	456
		CONST_ROM_32BIT	32
2	RAM	NOINIT_RAM_UNSPECIFIED	156
		RAM_UNSPECIFIED	2
		NOINIT_RAM_1BIT	8
		NOINIT_RAM_8BIT	9
		NOINIT_RAM_16BIT	22
		RAM_8BIT	2

13.3.2. Stack Depth

The worst-case stack depth for Driver Component is 188 bytes for the typical configuration provided in Section 13.2.2.1.

13.3.3. Throughput Details

The throughput details of the APIs for the configuration mentioned in the Section13.2.2.1 *Configuration* are provided in this section. The clock frequency used to measure the throughput is 240 MHz for all APIs.

Table 13-4 Throughput Details Of The APIs

Sl. No.	API Name	Throughput in microseconds	Remarks
1	Spi_Init	2.137	-
2	Spi_DeInit	2.500	-
3	Spi_WriteIB	0.387	-
4	Spi_AsyncTransmit	5.325	-
5	Spi_ReadIB	0. 250	-
6	Spi_SetupEB	0.200	-
7	Spi_GetStatus	0.620	-
8	Spi_GetJobResult	0. 620	-
9	Spi_GetSequenceResult	0. 620	-
10	Spi_GetVersionInfo	0.100	-
11	Spi_SyncTransmit	8.400	-
12	Spi_GetHWUnitStatus	0.187	-
13	Spi_Cancel	0.275	-
14	Spi_SetAsyncMode	1.437	SPI_POLLING_MODE
15	Spi_SetAsyncMode	0.175	SPI_INTERRUPT_ MODE
16	Spi_MainFunction_Handling	0.850	-
17	Spi_SelfTest	649.850	SPI_LOOP_BACK_SELF _TEST
18	Spi_SelfTest	32.150	SPI_ECC_SELF_TEST
19	Spi_GetErrorInfo	0.125	-

Chapter 14 Release Details

SPI Driver Software

Version: 2.0.0

Revision History

Sl.No.	Description	Version	Date
1.	Initial Version	1.0.0	05-Aug-2015
2	Following changes are made: 1. Table 4-4 User Mode and Supervisory Mode is updated. 2. In section 4, Information for 16 bit datawidth selection is added when DMA is configured. 3. Table 6-1 Register details, 8bit and 32bit settings when DMA is configured are removed. 4. In section 4.6, Information for the limitation for CS added. 5. In section 4.2, Note about the user Configuration of Module Short Name was added. 6. In section 11.1, new development error SPI_E_MAINFUNCTION_HANDLING_INVALIDMODE is added for Spi_MainFunction_Handling API.	1.0.1	28-Mar-2016
3	Following changes are made: 1. Removed Section 13.2, Compiler, Linker and Assembler. 2. In section 4.3, Note about entries for User mode dependency of Critical Section added. 3. In section 4.5, Critical section details are updated by adding Table 4-6. 4. In section 4.1, Note added regarding the DMA access for local RAM area. 5. In section 12, Memory Organization is updated by adding information about SPI_START_SEC_CODE_FAST. 6. Section 6, Register access details are updated. 7. Updated section 13.2.1 Sample Application Structure to add details about Spi_GetErrorInfo API. 8. Added Spi_GetErrorInfo API in section 11.1 under Related API(s) corresponding to the error SPI_E_UNINIT. 9. Section 3 updated R number in remarks. 10. Folder Structure updated in the section 3.1.1. 11. Table 4-4 User mode and Supervisory mode is updated. 12. Section 8 updated for file information. 13. Section 9 updated for R number. 14. Table 10-1 updated with API name. 15. Memory, Throughput and stack depth Details are updated in section 13.3. 16. Release details updated in section 14. 17. Chapter 13, Added Processor name along with Device variants. 18. Figure 12-1 SPI Driver Component Driver Organization has been updated in Chapter 12. 19. Removed traces of .one and .html from the section 13.2 Sample Application.	1.0.2	15-Feb-2017

AUTOSAR MCAL R4.0.3 User's Manual
SPI Driver Component Ver.1.0.2
Embedded User's Manual

Publication Date: Rev.1.00, Feb 15, 2017

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.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

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: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, 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 India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

AUTOSAR MCAL R4.0.3

User's Manual

Renesas Electronics Corporation

R20UT3659EJ0100