User's Manual

AUTOSAR MCAL R4.0.3

User's Manual

SPI Driver Component Ver.1.0.12
Embedded User's Manual

Target Device:
RH850/P1x

Renesas Electronics
www.renesas.com

Rev.1.01 Mar 2017

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
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
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
PDF	Parameter Definition File
DIO	Digital Input Output
WDT	Watchdog Timer
RUCG	Renesas Unified Code Generator
µC	Micro controller
XML	eXtensible Mark-up Language
ICU	Input Capture Unit
CAN	Controller Area Network
BUS	BUS Network

PWM	Pulse Width Modulation
PORT	Represents a whole configurable port on a microcontroller device
ADC	Analog to Digital Converter
LIN	Local Interconnect Network

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:

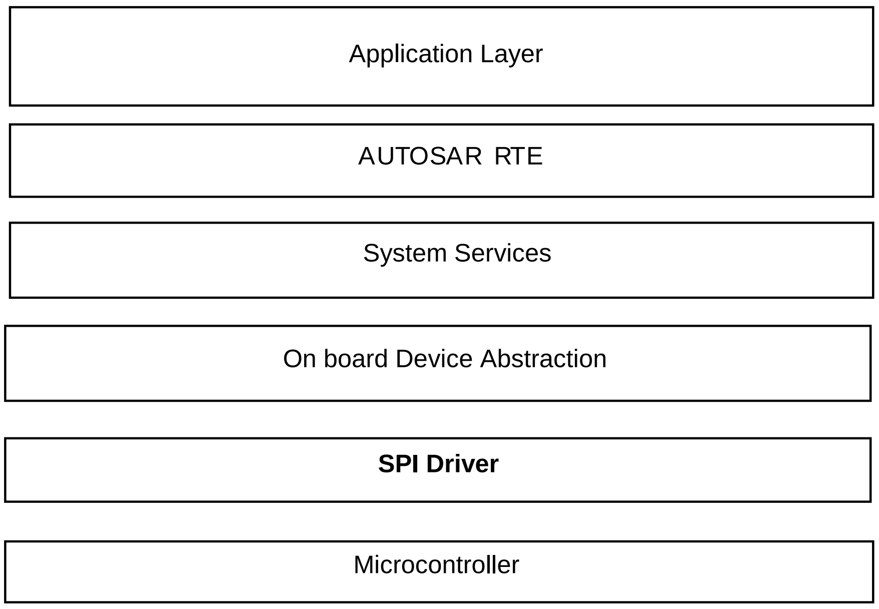


Figure 1-1 System Overview Of AUTOSAR Architecture

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:

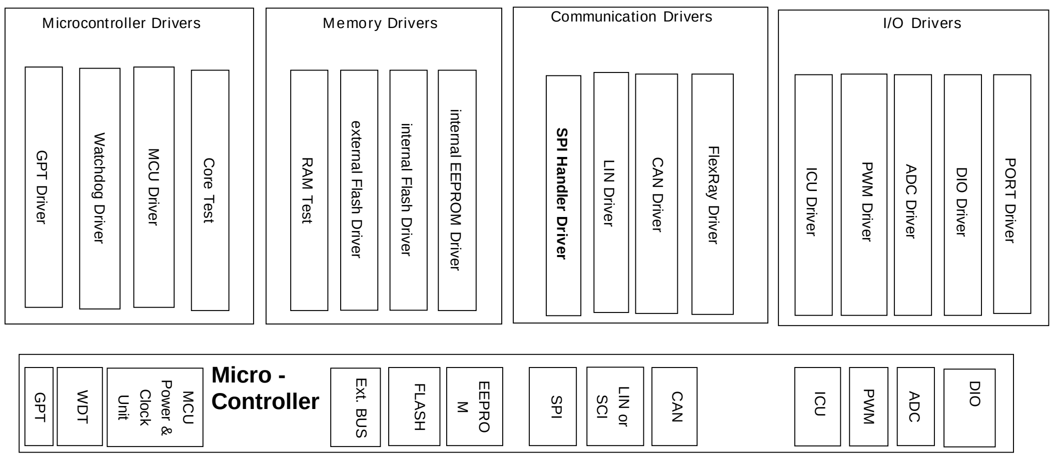


Figure 1-2 System Overview Of The SPI Driver In AUTOSAR MCAL Layer

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 ,DEM errors and hardware 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.	r01uh0436ej0120_rh850p1x.pdf	1.20
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 "R20UT3727EJ0101-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\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.dll

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

tools/RUCG/RUCG.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.
- The SPI Handler/Driver interface configuration is based on Channels, Jobs and Sequences.The Data transmissions will be done according to Channels, Jobs and Sequences configuration parameters.
- 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 channel data received shall be stored in 1 entry deep internal buffers by channel. The SPI Handler/Driver shall not take care of the overwriting of these “receive” buffers by another transmission on the same channel.
- The channel data to be transmitted shall be copied in 1 entry deep internal buffers by channel. The SPI Handler/Driver cannot prevent overwriting of these “transmit” buffers by users during transmissions.
- If different Jobs (and consequently also Sequences) have common Channels, the SPI Handler/Driver’ environment should ensure that read and/or write functions are not called during transmission.
- If a Job contains more than one Channel, all Channels contained have the same Job properties during transmission and are linked together statically.
- 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 1, then the SPI Handler/Driver offers an asynchronous transfer service for SPI buses. An asynchronous transmission means that the user calling the transmission service is not blocked when the transmission is ongoing.

- 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.

- The SPI Handler/Driver is not allowed to suspend a Sequence transmission already started in favour of another Sequence in case of Non-Interruptible Sequences

- 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 "SpiCsInactiveAfterLastData" 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.

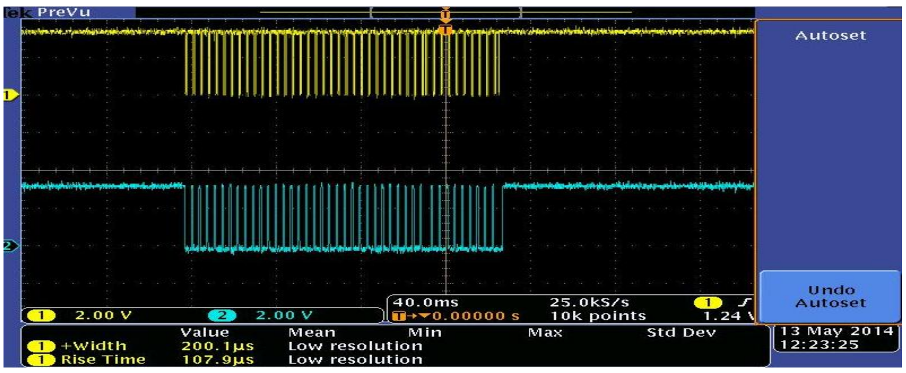
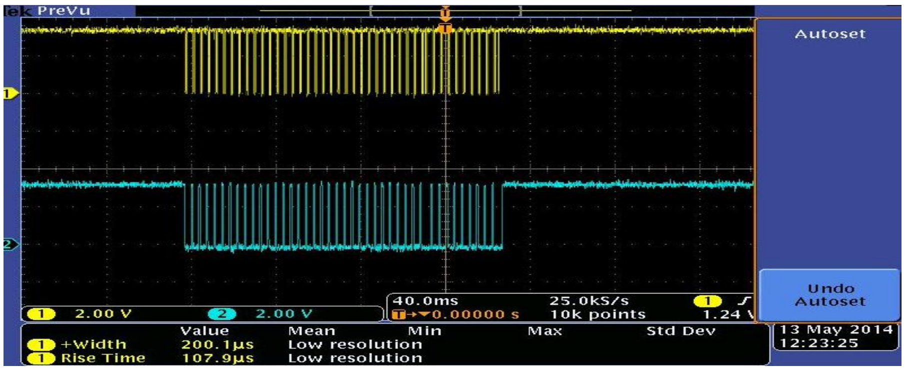


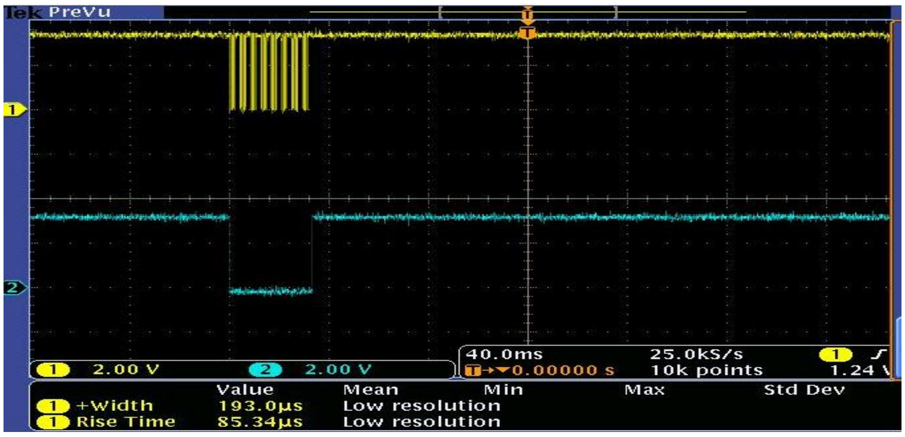
Figure 4-1 Chip select behavior when SpiCSInactiveAfterlastdata is False and SpiCsIdleEnforcement is True

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



```
1  2.00 V        2  2.00 V        40.0ms          25.0kS/s      1  ƒ
                                  T→▼0.00000 s    10k points    1.24 V
              Value      Mean              Min      Max      Std Dev      13 May 2014
1 +Width      200.1µs    Low resolution                                  12:23:25
1 Rise Time   107.9µs    Low resolution
```

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



```
1  2.00 V        2  2.00 V        40.0ms          25.0kS/s      1  ƒ
                                  T→▼0.00000 s    10k points    1.24 V
              Value      Mean              Min      Max      Std Dev
1 +Width      193.0µs    Low resolution
1 Rise Time   85.34µs    Low resolution
```

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.

3. When channel properties are different and SpiCsIdleEnforcement is configured as False, then the corresponding chip select will be deactivated after each channel transmission.

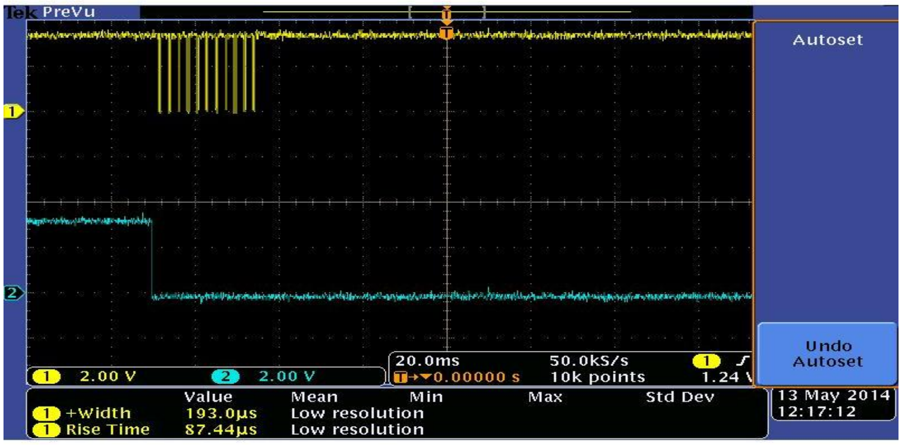


Figure 4-4 Chip select behavior when SpiCSInactiveAfterlastdata is False and SpiCsIdleEnforcement is False

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	-

- Integrator has to ensure that the critical section protection is configured correctly.
- User should invoke Spi_GetErrorInfo before the buffer limit exceeds.
- The user must calculate proper SpiTimeOut value based on the data size configured.
- The failure of self test indicates hardware failure.
- 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' container and fill remaining with zeros.

- 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 accesses to HW registers is possible only in the low level driver layer. The user shall never write or read directly from any register, but shall use the AUTOSAR standard API provided by the MCAL.

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

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

- SPI peripherals may depend on the system clock, prescaler(s) and PLL. Thus, changes of the system clock may also affect the clock settings of the SPI hardware.

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

- Depending on microcontrollers, the SPI peripheral could share registers with other peripherals. In this typical case, the SPI Handler/Driver has a relationship with MCU module for initialising and de-initialising those registers.

- If SpiInternalErrorBufferSize parameter is configured as Zero, Spi_GetErrorInfo feature will be disabled. A Non zero value should be configured to enable this feature.

- Spi Driver status shall be ensured as SPI_IDLE by calling Spi_GetStatus API, before calling Spi_GetErrorInfo API to avoid simultaneous access of Global Error Buffer.

- For Configuring the Parameter SpiTimeOut, User must consider these factors:
  1. Data transmission time strongly depends on the data length and baudrate configured.
  2. The parameter SpiTimeOut should be big enough to cover the worst case scenario in the driver configuration.
  3. MCU clock.
  4. Compiler optimization level.
  5. It is recommended to add additional margin to the timeout based on user experience.

  Example to consider:

  Let’s say, if we are configuring the baud rate as 1 KHz:
  1. For Data length of 16 bit and default data transmission is 8 or 16 bit i.e. 0x10 or 0x5639, the minimum timeout value can be configured as 0xED8.
  2. If user is Configuring Data width selection of 32 bit and default data transmission is 32 bit i.e. 0x5A5A5A5A or 0xFEDCFEDC the maximum

Timeout value can be configured as 0xFFFF which is the worst case needed for the transmission buffer to empty or receiving buffer to get filled In that time.

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.
- Errors checked in the Development Error Detection area are only static configuration checks. No runtime errors are checked here.
- 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.
- 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.
- 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		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	
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.

Sl. No.	API name	Interrupt mode		Polling mode		Known limitation in User Mode
		user mode	supervisor mode	user mode	supervisor 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

Note1: 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 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.

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

Table 4-7 SPI Driver Protected Resources List

API Name	Exclusive Area Type	Protected Resources
Spi_AsyncTransmit	SPI_RAM_DATA_PROTECTION	Global Variable: Spi_GaaSeqCancel,, Spi_GddDriverStatus, Spi_GaaSeqResult, Spi_GucHwUnitStatus, Spi_GddQueueIndex, Spi_GblQueueStatus, Spi_GusAllQueueSts, Spi_GaaSeqQueue, Spi_GddQueueIndex, Spi_GblQueueStatus Spi_GaaJobQueue Spi_GaaJobResult Spi_GaaJobCount Spi_GaaHighPriorityCommRequestAtIdle Spi_GaaHighPriorityCommRequestAtIdle Spi_GaaHighPriorityCommActive Spi_GaaHighPriorityCommRequest Spi_GaaHighPrioritySequence Spi_GucHWFifoBufferSts Spi_GstFifoCurrentCommData HW Registers: IMR and INTC registers

API Name	Exclusive Area Type	Protected Resources
Spi_AsyncTransmit	SPI_CHIP_SELECT_PROTECTION	HW Register: Port PSR Register.
Spi_SyncTransmit	SPI_RAM_DATA_PROTECTION	Global Variables: Spi_GusHwStatus HW Registers: INTC registers
Spi_SyncTransmit	SPI_CHIP_SELECT_PROTECTION	HW Register: Port PSR Register.
Spi_Cancel	SPI_RAM_DATA_PROTECTION	Global Variable: Spi_GaaSeqCancel

Note: The highest measured duration of a critical section is 2.10 micro seconds measured for Spi_AsyncTransmit API.

4.6. Deviation List

Table 4-8 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.	-

Sl. No.	Description	AUTOSAR Bugzilla
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.	-
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.	Memory size measurements (RAM/ROM usage,Stack, Throughput) are not as per requirements.	-

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:

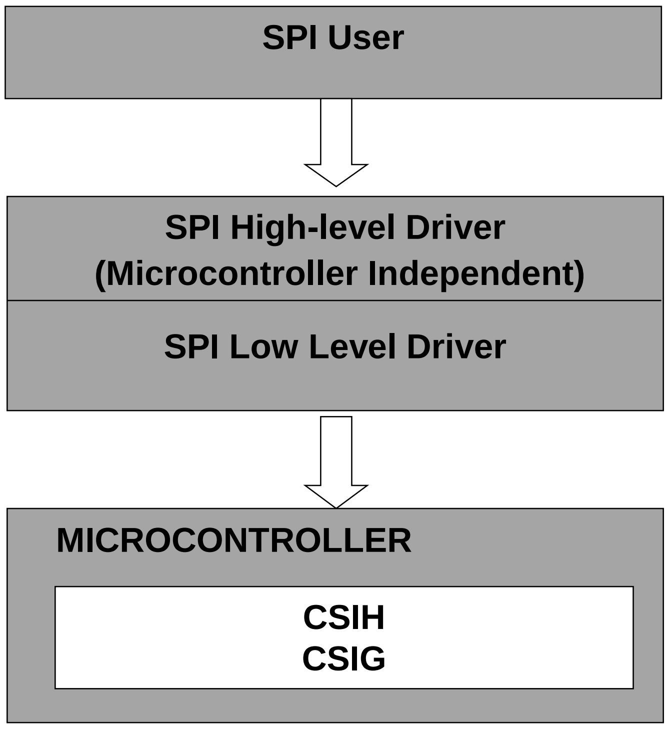


Figure 5-1 SPI Driver Architecture

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
- Communication Error Diagnosis

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

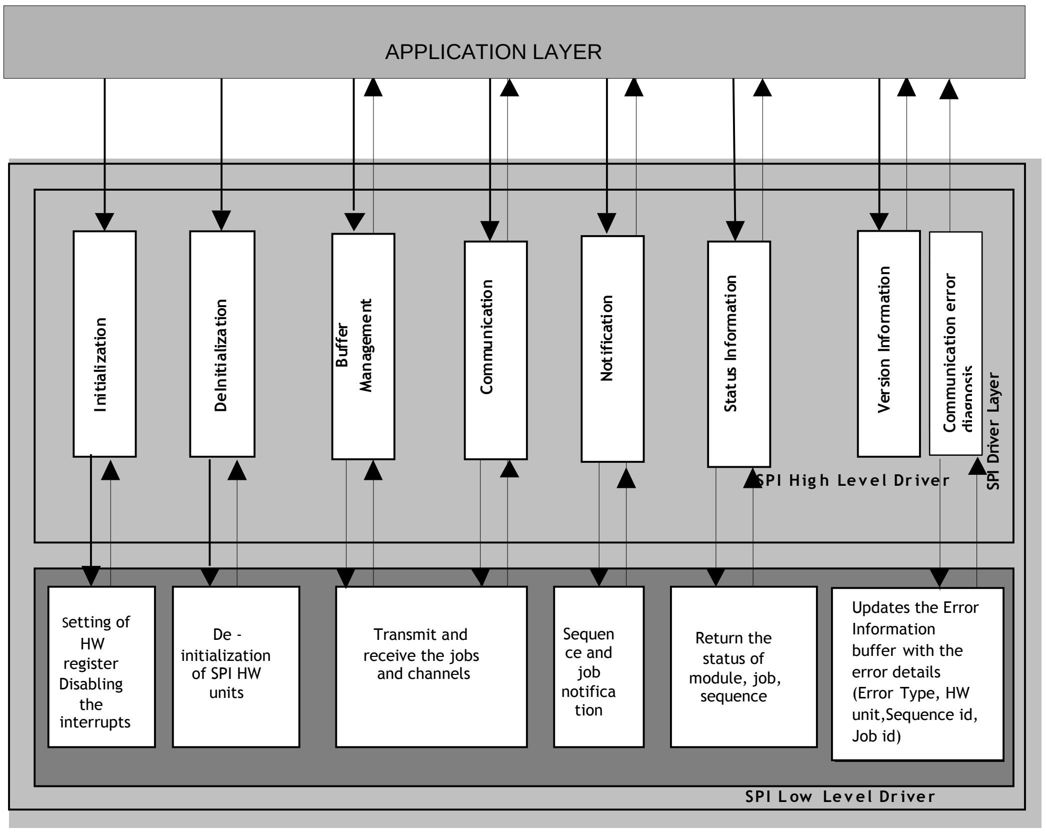

APPLICATION LAYER
Initialization
DeInitialization
Buffer Management
Communication
Notification
Status Information
Version Information
Communication error diagnosis
SPI High Level Driver
SPI Driver Layer
Setting of HW register Disabling the interrupts
De - initialization of SPI HW units
Transmit and receive the jobs and channels
Sequence and job notification
Return the status of module, job, sequence
Updates the Error Information buffer with the error details (Error Type, HW unit,Sequence id, Job id)
SPI Low Level Driver

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().

Communication Error Diagnosis

This module provides the services for collecting the error details when the transmission of data on the SPI bus is failed. A buffer and the size of the buffer shall be passed as arguments to this module. This module provides following detailes of the communication error :
1. Type of the Hardware Error (parity, data consistency, overflow, overrun)
2. HW unit in which error is reported (eg. CSIG0, CSIH3, etc.)
3. Sequence id for which error is reported
4. Job id for which error is reported

These details will be stored in to the passed buffer. This module is implemented for getting error details whenever a hardware error is reported.

The API related to this module is Spi_GetErrorInfo().

There are 2 approaches for using by upper layer.

1. **Polling Method**

The upper layer calls Spi_GetSequenceResult() API and when the return value is SPI_SEQ_FAILED then call to Spi_GetErrorInfo() API can be done to get the detailed information as to why the sequence failed.

2. **Application Callback Function Executed from SPI Error ISR**

The user can be informed each time a SPI error occurred through the DEM error. User can invoke Spi_GetErrorInfo API to get the error details when the DEM is reported.
If call out from DEM is not possible, user can check the error detailed information by calling API Spi_GetErrorInfo after confirmation of failure from Spi_SyncTransmit API.

Note :

- For each error, the error details will be stored in the eror buffer.So it is not a must to invoke Spi_GetErrorInfo every time the error occurs.
- User can decide to call Spi_GetErrorInfo after multiple errors or for each error depending on the application requirement.
- The maximum number of error details that can be hold by the error buffer can be configured by the user.

- To store the Errors generated, error ISR will be invoked in the case Asynchronous transmission only.
- In synchronous transmission case, the internal function used for reporting error in synchronous transmission will be invoked to report the Error.
- All the other steps and approaches are same for both Sync and Async Transmit case.
- The latest communication errors info will be always stored in error buffer and it will not be cleared when it is read.
- Any elements of error buffer will not be changed (not cleared /not shifted) when it is read including partial read.
- The error information from the status register STCR0 is cleared in Error ISR and internal function used for reporting error in synchronous transmission after reading it,to avoid the possibility of reporting multiple errors.
- Whenever an error interrupt occurs, SPI driver will check for all the possible errors one by one and will report separately.
  For Example, if an overflow and parity error is reported simultaneously for a data transmission, corresponding index of the error buffer will contain Overflow Error and other index will store Parity Error.
- No variables used in SPI driver are modified in Spi_GetErrorInfo API.
- Copying the error information from error buffer into the user buffer is done without modifying any Global array.

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	CSIGnCTL0	SpiMemoryModeSelection	W	SPI_ZERO
	CSIHnCTL0		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
	CSIGnCTL1	SpiCsInactiveAfterLastData, SpiDataWidth	W	LunDataAccess1.ulRegData
	CSIHnCTL1		W	LunDataAccess1.ulRegData
	EICn	-	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
	SELCSIHDMA	-	W	SPI_SELECT_CSIH_DMA_REG_VAL
	CSIGnCTL2	SpiInputClockSelect SpiBaudrateConfiguration	W	LpJobConfig->usCtl2Value

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIGCFG0	SpiDataWidth SpiParitySelection SpiTransferStart SpiDataShiftEdge SpiShiftClockIdleLevel	W	LunDataAccess1.ulRegData
	CSIHnSTCR0	-	W	SPI_CSIH_CLR_STS_FLAGS
	CSIGnSTCR0	-	W	SPI_CSIG_CLR_STS_FLAGS
	CSIHnSTR0	-	R	-
	CSIGnSTR0	-	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
	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

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIHnRX0H	-	R	-
	CSIGnRX0	-	R	-
	CSIGnTX0H	-	W	SPI_LOOPBACK_DATA
	CSIHnMCTL1	SpiMemoryModeSelection	W	SPI_CTL_32BIT_REG_VAL
	CSIHnMCTL2	SpiMemoryModeSelection	W	SPI_CTL_32BIT_REG_VAL
	CSIGBCTL0	-	W	SPI_BCTL0_SET_SCE
Spi_DeInit	CSIGnCTL0	SpiMemoryModeSelection	W	SPI_ZERO
	CSIHnCTL0		W	SPI_ZERO
	CSIGnCTL1	-	W	SPI_ZERO
	CSIHnCTL1	-	W	SPI_ZERO
	CSIGnCTL2	-	W	SPI_CTL2_16BIT_REG_DEINIT
	CSIHnCTL2	-	W	SPI_CTL2_16BIT_REG_DEINIT
	CSIGBCTL0	-	W	SPI_CTL_8BIT_REG_MASK
	CSIHnMCTL0	-	W	SPI_MCTL0_16BIT_REG_DEINIT
	CSIHnMCTL1	-	W	SPI_CTL_32BIT_REG_MASK
	CSIHnMCTL2	-	W	SPI_CTL_32BIT_REG_MASK
	CSIGnSTCR0	-	W	SPI_CTL_16BIT_REG_DEIN
	CSIHnSTCR0	-	W	SPI_CTL_16BIT_REG_DEINIT

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIHMRWP0	-	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
	CSIGCFG0	-	W	SPI_CTL_32BIT_REG_MASK
	CSIHCFG0	-	W	SPI_CTL_32BIT_REG_MASK
	DTFRRQCn	-	W	SPI_DMA_DRQ_CLEAR
	DCSTCn	-	W	SPI_DMA_STR_CLEAR
	DTFRRQn	-	R	-
	DCSTn	-	R	-
	DTFRn	-	W	SPI_DMA_DEINIT
	CSIHnMRWP0	-	W	SPI_CTL_32BIT_REG_VAL
	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
	EICn	-	W	SPI_CLR_INT_REQ
	PORTPSRx	SpiPortPinSelect		LpJobConfiguration->ulPortPinMask
Spi_WriteIB	CSIHMCTL0	SpiMemoryModeSelection	W	LusMctlData SPI_TX_ONLY_MODE_SET SPI_DUAL_BUFFER_MODE_SET
	CSIHnMRWP0	-	RW	ulRegData LunDataAccess1.ulRegData
	CSIHnTX0W	-	W	LunDataAccess1.ulRegData

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
Spi_AsyncTransmit	CSIHnMCTL0	-	W	LpJobConfig->usMCtl0Value
	CSIGnCFG0	-	W	LpJobConfig->ulConfigRegValue
	CSIGnCTL0	SpiMemoryModeSelection	W	SPI_RESET_PWR SPI_SET_DIRECT_ACCESS SPI_SET_MEMORY_ACCESS
	CSIHnCTL0		W	SPI_RESET_PWR SPI_SET_DIRECT_ACCESS
	CSIGnSTCR0	-	W	SPI_CLR_STS_FLAGS
	CSIHnSTCR0	-	W	SPI_CLR_STS_FLAGS
	CSIHnSTR0	-	R	-
	CSIGnSTR0	-	R	-
	CSIGnCTL1	SpiCsInactiveAfterLastData, SpiDataWidth	W	LunDataAccess1.ulRegData LpJobConfig->ulMainCtl1Value SPI_SET_SLIT
	CSIHnCTL1		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
	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
	DCENn	-	W	SPI_DMA_DCEN_ENABLE
	DDAn	-	W	(uint32)(&Spi_GddDmaRxData)
	CSIGnCTL2	SpiBaudrateRegisterSelect	W	LpJobConfig->usCtl2Value
	CSIHnCTL2	SpiFifoTimeOut	W	LpJobConfig->usCtl2Value
	CSIHnMCTL2	-	W	LunDataAccess1.ulRegData

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIHnTX0W	-	W	LunDataAccess1.ulRegData, LunDataAccess2.ulRegData, LpDataAccess->ulRegData
	CSIHnTX0H	-	W	LddData, LunDataAccess2.usRegData 5[SPI_ZERO]
	CSIGnTX0H	-	W	LddData, LunDataAccess2.usRegData 5[SPI_ZERO]
	CSIHnCFGx	SpiCsIdleTiming, SpiCsHoldTiming, SpiCsInterDataDelay, SpiCsSetupTime, SpiCsIdleEnforcement	W	LunDataAccess1.ulRegData
	CSIGnTX0W	-	W	LunDataAccess1.ulRegData, LpDataAccess->ulRegData
	CSIHnBRS[0]	SpiBaudrateConfiguration	W	LpCsihOsBaseAddr->usCSIHBRS[0]
	CSIHnBRS[1]	-	W	LpCsihOsBaseAddr->usCSIHBRS[1]
	CSIHnBRS[2]	-	W	LpCsihOsBaseAddr->usCSIHBRS[2]
	CSIHnBRS[3]	-	W	LpCsihOsBaseAddr->usCSIHBRS[3]
	IMRn	SpiHwUnitSelection and SpiMemoryModeSelection	W	LpHWUnitInfo->usRxImrMask, LpHWUnitInfo->usTxImrMask, LpHWUnitInfo->usErrorImrMask, LpHWUnitInfo->usRxImrMask, LpHWUnitInfo->usTxImrMask, LpHWUnitInfo->usTxCancelImrMask, LpHWUnitInfo->usErrorImrMask
	EICn	-	W	SPI_CLR_INT_REQ
	DTFRRQn	-	R	-
	PORTPSRx	SpiPortPinSelect	W	LulPinMskVal & SPI_PORT_REG_MASK, LulPinMskVal
	CSIHnRX0H	-	R	-
	CSIGnRx0	-	R	-
	CSIHnRX0W	-	R	-
Spi_ReadIB	CSIHnRX0W	-	W	LunDataAccess2.ulRegData

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	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
	CSIGnCTL0	-	W	SPI_RESET_PWR SPI_SET_DIRECT_ACCESS LunDataAccess1.ulRegData
	CSIHnCTL0	-	W	SPI_RESET_PWR SPI_SET_DIRECT_ACCESS SPI_SET_PWR SPI_ZERO
	CSIGnTX0W	-	W	LunDataAccess2.ulRegData LunDataAccess1.ulRegData
	CSIHnRX0H	-	RW	LunDataAccess3.ulRegData, Spi_GusSynDataAccess
	CSIGnCFG0	-	RW	Spi_GusAsynDataAccess LddData LpJobConfig->ulConfigRegValue, LunDataAccess1.ulRegData
	CSIGnSTR0	-	R	-
	CSIHnSTR0	-	R	-
	CSIGnSTCR0	-	W	SPI_PE_ERR_CLR, SPI_DCE_ERR_CLR, SPI_OFE_ERR_CLR
	CSIHnSTCR0	-	W	SPI_DCE_ERR_CLR, SPI_PE_ERR_CLR, SPI_OFE_ERR_CLR
	CSIGnCTL1	-	W	LpJobConfig->ulMainCtl1Value, LpMainOsBaseAddr->ulMainCTL1 | SPI_SET_SLIT
	CSIHnCTL1	SpiCsInactiveAfterLastData, SpiDataWidth	W	LunDataAccess1.ulRegData, (LpMainOsBaseAddr->ulMainCTL1 | ~SPI_CSRI_AND_MASK
	CSIGnCTL2	SpiBaudrateRegisterSelect	W	LunDataAccess1.ulRegData, LpJobConfig->usCtl2Value,
	CSIHnCTL2	SpiFifoTimeOut	W	LpJobConfig->usCtl2Value
	CSIHnTX0W	-	W	LpJobConfig->usCtl2Value, LunDataAccess3.ulRegData

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIHnCFG	SpiCsIdleTiming, SpiCsHoldTiming, SpiCsInterDataDelay, SpiCsSetupTime, SpiCsIdleEnforcement	RW	LunDataAccess1.ulRegData LpJobConfig->ulConfigRegValue
	CSIGnRX0	-	R	-
	CSIHnBRS[0]	SpiBaudrateConfiguration	W	LpCsihOsBaseAddr->usCSIHBRS[0], LpJobConfig->usCtl2Value & SPI_CSIH_BRS_MASK
	CSIHnBRS[1]		W	LpCsihOsBaseAddr->usCSIHBRS[1], LpJobConfig->usCtl2Value) & SPI_CSIH_BRS_MASK
	CSIHnBRS[2]		W	LpCsihOsBaseAddr->usCSIHBRS[2], LpJobConfig->usCtl2Value) & SPI_CSIH_BRS_MASK
	CSIHnBRS[3]		W	LpCsihOsBaseAddr->usCSIHBRS[3], LpJobConfig->usCtl2Value) & SPI_CSIH_BRS_MASK
	EICn	-	W	SPI_CLR_INT_REQ
	PORTPSRx	SpiPortPinSelect	W	LulPinMskVal, LulPinMskVal & SPI_PORT_REG_MASK
Spi_GetHWUnitStatus	CSIGnSTR0	-	R	-
	CSIHnSTR0	-	R	-
Spi_Cancel	CSIHnCTL0	-	R/W	SPI_SET_JOBE
	IMRn	-	W	LpHWUnitInfo->ucTxCancelImrMask
	EICn	-	W	SPI_CLR_INT_REQ
Spi_SetAsyncMode	IMRn	SpiHwUnitSelection and SpiMemoryModeSelection	W	Spi_GstHWUnitInfo[LddHWUnit].usRxImrMask, Spi_GstHWUnitInfo[LddHWUnit].usTxImrMask, Spi_GstHWUnitInfo[LddHWUnit].usErrorImrMask, Spi_GstHWUnitInfo[LddHWUnit].usRxImrMask, Spi_GstHWUnitInfo[LddHWUnit].usTxImrMask, Spi_GstHWUnitInfo[LddHWUnit].usTxCancelImrMask Spi_GstHWUnitInfo[LddHWUnit].usErrorImrMask
	EICn	-	w	SPI_CLR_INT_REQ
Spi_MainFunction_Handling	CSIGnCTL0	-	W	SPI_SET_PWR
	CSIHnCTL0	-	W	SPI_SET_PWR

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIGnRX0	-	R	-
	CSIHnRX0H	-	R	-
	CSIGnTX0W	-	W	LunDataAccess1.ulRegData
	CSIHnTX0W	-	W	LunDataAccess1.ulRegData
	CSIGnTX0H	-	W	LddData LunDataAccess2.usRegData5[0]
	CSIHnTX0H	-	W	LddData LunDataAccess2.usRegData5[0]
	CSIHnRX0W	-	R	-
	CSIHnMCTL2	SpiMemoryModeSelection	W	LunDataAccess1.ulRegData
	EICn	-	W	SPI_CLR_INT_REQ
	DCSTCn	-	W	SPI_DMA_STR_CLEAR
	DCSTn	-	R	-
	DCENn	-	W	SPI_DMA_DCEN_DISABLE
	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
	DCENn	-	W	SPI_DMA_DCEN_ENABLE
	DDAn	-	W	(uint32)(&Spi_GddDmaRxData)
	CSIGnSTCR0	-	W	SPI_CLR_STS_FLAGS
	CSIHnSTCR0	-	W	SPI_CLR_STS_FLAGS
	CSIHnSTR0	-	R	-
	CSIGnSTR0	-	R	-
	CSIGnCTL1	SpiCsInactiveAfterLastData, SpiDataWidth	W	LunDataAccess1.ulRegData LpJobConfig->ulMainCtl1Value
	CSIHnCTL1	-	W	LunDataAccess1.ulRegData LpJobConfig->ulMainCtl1Value SPI_SET_SLIT
	CSIGnCTL2	SpiBaudrateRegisterSelect	W	LpJobConfig->usCtl2Value
	CSIHnCTL2	SpiFifoTimeOut	W	LpJobConfig->usCtl2Value
	CSIHnMCTL2	-	W	LunDataAccess1.ulRegData

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIHnCFGx	SpiCsIdleTiming, SpiCsHoldTiming, SpiCsInterDataDelay, SpiCsSetupTime, SpiCsIdleEnforcement	W	LunDataAccess1.ulRegData
	CSIHnBRS[0]	SpiBaudrateConfiguration	W	LpCsihOsBaseAddr->usCSIHBRS[0]
	CSIHnBRS[1]	-	W	LpCsihOsBaseAddr->usCSIHBRS[1]
	CSIHnBRS[2]	-	W	LpCsihOsBaseAddr->usCSIHBRS[2]
	CSIHnBRS[3]	-	W	LpCsihOsBaseAddr->usCSIHBRS[3]
	CSIHnMCTL0	-	W	LpJobConfig->usMCtl0Value
	DTFRRQn	-	R	-
	PORTPSRx	SpiPortPinSelect	W	LulPinMskVal, LulPinMskVal & SPI_PORT_REG_MASK
Spi_GetVersionInfo	-	-	-	-
Spi_GetErrorInfor	-	-	-	-
Spi_SelfTest	CSIGnRX0	-	R	-
	CSIHnRX0H	-	R	-
	CSIGnCTL0	SpiLoopBackSelfTest	W	SPI_SET_DIRECT_ACCESS SPI_ZERO
	CSIHnCTL0	SpiLoopBackSelfTest	W	LpJobConfig->usMCtl0Value SPI_ZERO
	CSIGnCTL1	SpiLoopBackSelfTest	W	SPI_LOOPBACK_ENABLE SPI_ZERO SPI_SET_SLIT
	CSIHnCTL1	SpiLoopBackSelfTest	W	SPI_LOOPBACK_ENABLE SPI_ZERO SPI_SET_SLIT LunDataAccess1.ulRegData
	CSIGnCTL2	SpiLoopBackSelfTest	W	SPI_LOOPBACK_CNTRL2_VALUE SPI_ZERO LpJobConfig->usCtl2Value
	CSIHnCTL2	SpiLoopBackSelfTest	W	SPI_LOOPBACK_CSIH_CNTRL2_VALUE SPI_ZERO ((LpJobConfig->usCtl2Value) & SPI_CSIH_PRE_MASK)

API Name	Registers	Config Parameter	Register Access R/W/RW	Macro/Variable
	CSIGnCFG0	SpiLoopBackSelfTest	W	SPI_LOOPBACK_DLS_SETTING SPI_ZERO LunDataAccess1.ulRegData
	CSIGnSTCR0	SpiLoopBackSelfTest	W	SPI_PE_ERR_CLR SPI_ZERO
	CSIHnSTCR0	SpiLoopBackSelfTest	W	SPI_CSIH_CLR_STS_FLAGS SPI_PE_ERR_CLR
	CSIGnTX0H	SpiLoopBackSelfTest	W	SPI_LOOPBACK_DATA SPI_ZERO
	CSIHnCFG0	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	-
	CSIGnSTR0	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
	IMR	SpiHwUnitSelection and SpiLoopBackSelfTest	W	Spi_GstHWUnitInfo[LddHWUnit].usRxImrMask, Spi_GstHWUnitInfo[LddHWUnit].pTxImrAddress, Spi_GstHWUnitInfo[LddHWUnit].pErrorImrAddress, Spi_GstHWUnitInfo[LddHWUnit].usRxImrMask, LpHWUnitInfo->usTxCancelImrMask
	EICn	-	W	SPI_CLR_INT_REQ

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.
- To copy Hardware Error Details to User Buffer.

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
- 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
- Dem.h

- SchM_Spi.h
- Det.h
- Os.h
- Rte.h
- Std_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
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_RegWrite.h	This file is to have macro definitions for the register write verification.
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.

File	Details
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 thi compiler specific header.
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 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
Dem.h	This file is a stub for DEM Component
Det.h	This file is a stub for DET Component
Os.h	This file is a stub for Os Component
Rte.h	This file is a stub for Rte Component
SchM_Spi.h	This file is a stub for Spi SchM Component
Std_Types.h	This file is a stub for Standard types

Chapter 9 Generation Tool Guide

For information on the SPI Driver Component Code Generation Tool, please refer “R20UT3727EJ0101-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.

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

- Std_ReturnType
- Std_VersionInfoType

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

Following are 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
	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

	Type	Name	Explanation
Element:	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 reported.
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
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_MainFunction_Handling
15.	Spi_GetHWUnitStatus
16.	Spi_SelfTest
17.	Spi_GetErrorInfo

10.3.1 Spi_Init

Name:	Spi_Init		
Prototype:	FUNC (void, SPI_PUBLIC_CODE) Spi_Init (P2CONST(Spi_ConfigType, AUTOMATIC, SPI_APPL_CONST) ConfigPtr)		
Service ID:	0x00		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Pointer to Spi_ConfigType	ConfigPtr	NA
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	void	NA	
Description:	This service performs initialization of the SPI Driver component.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.2 Spi_DeInit

Name:	Spi_DeInit		
Prototype:	FUNC (Std_ReturnType, SPI_PUBLIC_CODE) Spi_DeInit (void)		
Service ID:	0x01		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	NA	NA	NA
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK, E_NOT_OK	
Description:	This service performs De-initialization of the SPI Driver component.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.3 Spi_WriteIB

Name:	Spi_WriteIB		
Prototype:	FUNC (Std_ReturnType, SPI_PUBLIC_CODE) Spi_WriteIB (Spi_ChannelType Channel, P2CONST(Spi_DataType, AUTOMATIC, SPI_APPL_CONST) DataBufferPtr)		
Service ID:	0x02		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Spi_ChannelType	Channel	Min: 0 Max: 255
	Pointer to Spi_DataType	DataBufferPtr	NA
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK, E_NOT_OK	
Description:	This service for writing one or more data to an IB SPI Handler/Driver channel specified by parameter.		
Configuration Dependency:	None		
Preconditions:	The SPI Handler/Driver should have been initialized before this service is called.		

10.3.4 Spi_AsyncTransmit

Name:	Spi_AsyncTransmit		
Prototype:	FUNC (Std_ReturnType, SPI_PUBLIC_CODE) Spi_AsyncTransmit (Spi_SequenceType Sequence)		
Service ID:	0x03		
Sync/Async:	Asynchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Spi_SequenceType	Sequence	Min: 0 Max: 255
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK, E_NOT_OK	
Description:	This service for transmitting data asynchronously		
Configuration Dependency:	None		
Preconditions:	The SPI Handler/Driver should have been initialized before this service is called. This method shall be called after a Spi_SetupEB method for EB Channels or Spi_WriteIB method for IB Channels but before the Spi_ReadIB method.		

10.3.5 Spi_ReadIB

Name:	Spi_ReadIB		
Prototype:	FUNC (Std_ReturnType, SPI_PUBLIC_CODE) Spi_ReadIB (Spi_ChannelType Channel, P2VAR(Spi_DataType, AUTOMATIC, SPI_APPL_DATA) DataBufferPtr)		
Service ID:	0x04		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Spi_ChannelType	Channel	Min: 0 Max: 255
	Pointer to Spi_DataType	DataBufferPtr	NA
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK, E_NOT_OK	
Description:	Service for reading one or more data from an IB SPI Handler/Driver Channel specified by parameter.		

Configuration Dependency:	None
Preconditions:	The SPI Handler/Driver should have been initialized before this service is called. This method shall be called after one Transmit method call to have relevant data within IB Channel.

10.3.6 Spi_SetupEB

Name:	Spi_SetupEB		
Prototype:	FUNC (Std_ReturnType, SPI_PUBLIC_CODE) Spi_SetupEB (Spi_ChannelType Channel, CONST(Spi_DataType, AUTOMATIC, SPI_APPL_DATA) SrcDataBufferPtr P2VAR(Spi_DataType, AUTOMATIC, SPI_APPL_DATA) DesDataBufferPtr Spi_NumberOfDataType Length,)		
Service ID:	0x05		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Pointer to Spi_DataType	SrcDataBufferPtr	NA
	Spi_ChannelType	Channel	Min : 0 MAx: 255
	Spi_NumberOfDataType	Length	Min : 0 MAx: 65535
	Pointer to Spi_DataType	DesDataBufferPtr	NA
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK, E_NOT_OK	
Description:	Service to setup the buffers and the length of data for the EB SPI Handler/Driver Channel specified.		
Configuration Dependency:	None		
Preconditions:	The SPI Handler/Driver should have been initialized before this service is called.		

10.3.7 Spi_GetStatus

Name:	Spi_GetStatus
Prototype:	FUNC (Spi_StatusType, SPI_PUBLIC_CODE) Spi_GetStatus (void)
Service ID:	0x06

Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	NA	NA	NA
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Spi_StatusType	SPI_UNINIT/SPI_IDLE/SPI_BUSY	
Description:	This service shall return the SPI Handler/Driver software module status.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.8 Spi_GetJobResult

Name:	Spi_GetJobResult		
Prototype:	FUNC (Spi_JobResultType, SPI_PUBLIC_CODE) Spi_GetJobResult (Spi_JobType Job)		
Service ID:	0x07		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Spi_JobType	Job	Min: 0 Max: 65535
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Spi_JobResultType	SPI_JOB_OK/SPI_JOB_PENDING/SPI_JOB_FAILED	
Description:	This service shall return the last transmission result of the specified Job.		
Configuration Dependency:	None		
Preconditions:	The SPI Handler/Driver should have been initialized before this service is called.		

10.3.9 Spi_GetSequenceResult

Name:	Spi_GetSequenceResult
Prototype:	FUNC (Spi_SeqResultType, SPI_PUBLIC_CODE) Spi_GetSequenceResult (Spi_SequenceType Sequence)
Service ID:	0x08

Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Spi_SequenceType	Sequence	Min: 0 Max: 255
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Spi_SeqResultType	SPI_SEQ_OK/SPI_SEQ_PENDING/SPI_SEQ_FAILED/ SPI_SEQ_CANCELLED	
Description:	This service shall return the last transmission result of the specified Sequence.		
Configuration Dependency:	None		
Preconditions:	The SPI Handler/Driver should have been initialized before this service is called.		

10.3.10 Spi_SyncTransmit

Name:	Spi_SyncTransmit		
Prototype:	FUNC (Std_ReturnType, SPI_PUBLIC_CODE) Spi_SyncTransmit (Spi_SequenceType Sequence)		
Service ID:	0x0A		
Sync/Async:	Asynchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Spi_SequenceType	Sequence	Min: 0 Max: 255
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK/E_NOT_OK	
Description:	This service is for transmitting data synchronously.		
Configuration Dependency:	None		
Preconditions:	The SPI Handler/Driver should have been initialized before this service is called.		

10.3.11 Spi_GetHWUnitStatus

Name:	Spi_GetHWUnitStatus
Prototype:	FUNC (Spi_StatusType, SPI_PUBLIC_CODE) Spi_GetHWUnitStatus (Spi_HWUnitType HWUnit)
Service ID:	0x0B

Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Spi_HWUnitType	HWUnit	Min: 0 Max: 255
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Spi_StatusType	SPI_UNINIT/SPI_IDLE/SPI_BUSY	
Description:	This service shall return the status of the specified SPI Hardware microcontroller peripheral		
Configuration Dependency:	SpiHwStatusApi should be Enabled		
Preconditions:	The SPI Handler/Driver should have been initialized before this service is called		

10.3.12 Spi_Cancel

Name:	Spi_Cancel		
Prototype:	FUNC (void, SPI_PUBLIC_CODE) Spi_Cancel (Spi_SequenceType Sequence)		
Service ID:	0x0C		
Sync/Async:	Asynchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Spi_SequenceType	Sequence	Min: 0 Max: 255
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	NA	NA	
Description:	This service shall cancel the specified on-going sequence transmission.		
Configuration Dependency:	SpiCancelApi should be Enabled		
Preconditions:	The SPI Handler/Driver should have been initialized before this service is called		

10.3.13 Spi_SetAsyncMode

Name:	Spi_SetAsyncMode
Prototype:	FUNC (Std_ReturnType, SPI_PUBLIC_CODE) Spi_SetAsyncMode (Spi_AsyncModeType Mode)

Service ID:	0x0D		
Sync/Async:	Synchronous		
Reentrancy:	Non Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Spi_AsyncModeType	Mode	SPI_POLLING_MODE / SPI_INTERRUPT_MODE
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK/E_NOT_OK	
Description:	Service to set the asynchronous mechanism mode for SPI buses handled asynchronously.		
Configuration Dependency:	None		
Preconditions:	The SPI Handler/Driver should have been initialized before this service is called		

10.3.14 Spi_GetVersionInfo

Name:	Spi_GetVersionInfo		
Prototype:	FUNC (void, SPI_PUBLIC_CODE) Spi_GetVersionInfo (P2VAR(Std_VersionInfoType, AUTOMATIC, SPI_APPL_DATA) versionInfoPtr)		
Service ID:	0x09		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	NA	NA	NA
Parameters InOut:	NA	NA	NA
Parameters out:	pointer to Std_VersionInfoType	versionInfoPtr	NA
Return Value:	**Type**	**Possible Return Values**	
	NA	NA	
Description:	This service returns the version information of this module. The version information includes: - Module Id - Vendor Id - Vendor specific version numbers		
Configuration Dependency:	SpiVersionInfoApi should be Enabled		
Preconditions:	None		

10.3.15 Spi_MainFunction_Handling

Name:	Spi_MainFunction_Handling		
Prototype:	FUNC(void, SPI_PUBLIC_CODE) Spi_MainFunction_Handling (void)		
Service ID:	0x10		
Sync/Async:	NA		
Reentrancy:	Non Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	NA	NA	NA
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	NA	NA	
Description:	This function is to be invoked in the scheduler loop for asynchronous transmission in polling mode		
Configuration Dependency:	None		
Preconditions:	This function should be invoked only when polling is selected by Spi_SetAsyncMode API		

10.3.16 Spi_SelfTest

Name:	Spi_SelfTest		
Prototype:	FUNC(Spi_ReturnStatus, SPI_PUBLIC_CODE) Spi_SelfTest (Spi_SelfTestType LucTestFeature)		
Service ID:	0x11		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Spi_SelfTestType	LucTestFeature	Min: 0 Max: 255
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Spi_ReturnStatus	SPI_SELFTEST_DRIVERBUSY, SPI_SELFTEST_PASS, SPI_SELFTEST_FAILED, SPI_SELFTEST_INVALID_MODE	
Description:	Function to Execute SPI Self Test		
Configuration Dependency:	None		
Preconditions:	None		

10.3.17 Spi_GetErrorInfo

Name:	Spi_GetErrorInfo		
Prototype:	FUNC(uint8, SPI_PUBLIC_CODE) Spi_GetErrorInfo (P2VAR(Spi_CommErrorType, AUTOMATIC, SPI_CONFIG_DATA) LpUserBuffer, uint8 LucBufferSize)		
Service ID:	0x12		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Pointer to Spi_CommErrorType	LpUserBuffer	NA
	unit8	LucBufferSize	Min: 0 Max: 255
Parameters InOut:	NA	NA	NA
Parameters out:	NA	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	uint8	0 to 255	
Description:	Function to Copy Hardware Error Details to User Buffer		
Configuration Dependency:	None		
Preconditions:	None		

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, Spi_ComErrorISR and Spi_SelfTest
Source of Error	1. Overrun error: When previously received data still resides in the reception register(RX), because it wasn't read, and new data is received. 2. Data Consistency Check error: When data physically sent to the output pin is not identical to the original data that was copied to the shift register. 3. Parity error: When parity check fails during data transmission. Note: When DEM error 'SPI_E_HARDWARE_ERROR' occurs, corresponding sequence result will be updated as failed and sequence will be suspended.
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 a mismatch during write-verify check is detected.

11.3 SPI Driver Hardware Errors

11.3.1 Data Consistency Check

The purpose of the data consistency check is to ensure that the data physically sent to the output pin is identical to the original data that was copied to the shift register.When the data consistency check is active, the data transferred from CSIGnTX0W/CSIGnTX0H or CSIHnTX0W/CSIHnTX0H to the shift register is copied to a separate register. In addition, the physical levels at CSIGTSO/ CSIHTSO are capture and the logical interpretation is written to an own shift register.After completion of the transmission, the data sent is compared with the original transmission data.

11.3.2 Parity Check

Parity is a mean to detect a single bit failure during data transmission. CSIG/CSIH can append a parity bit to the last data bit.The parity bit is checked after reception is complete.When the extended data length (EDL) function is used, a parity bit is added after the last bit of the data.

11.3.3 Overrun

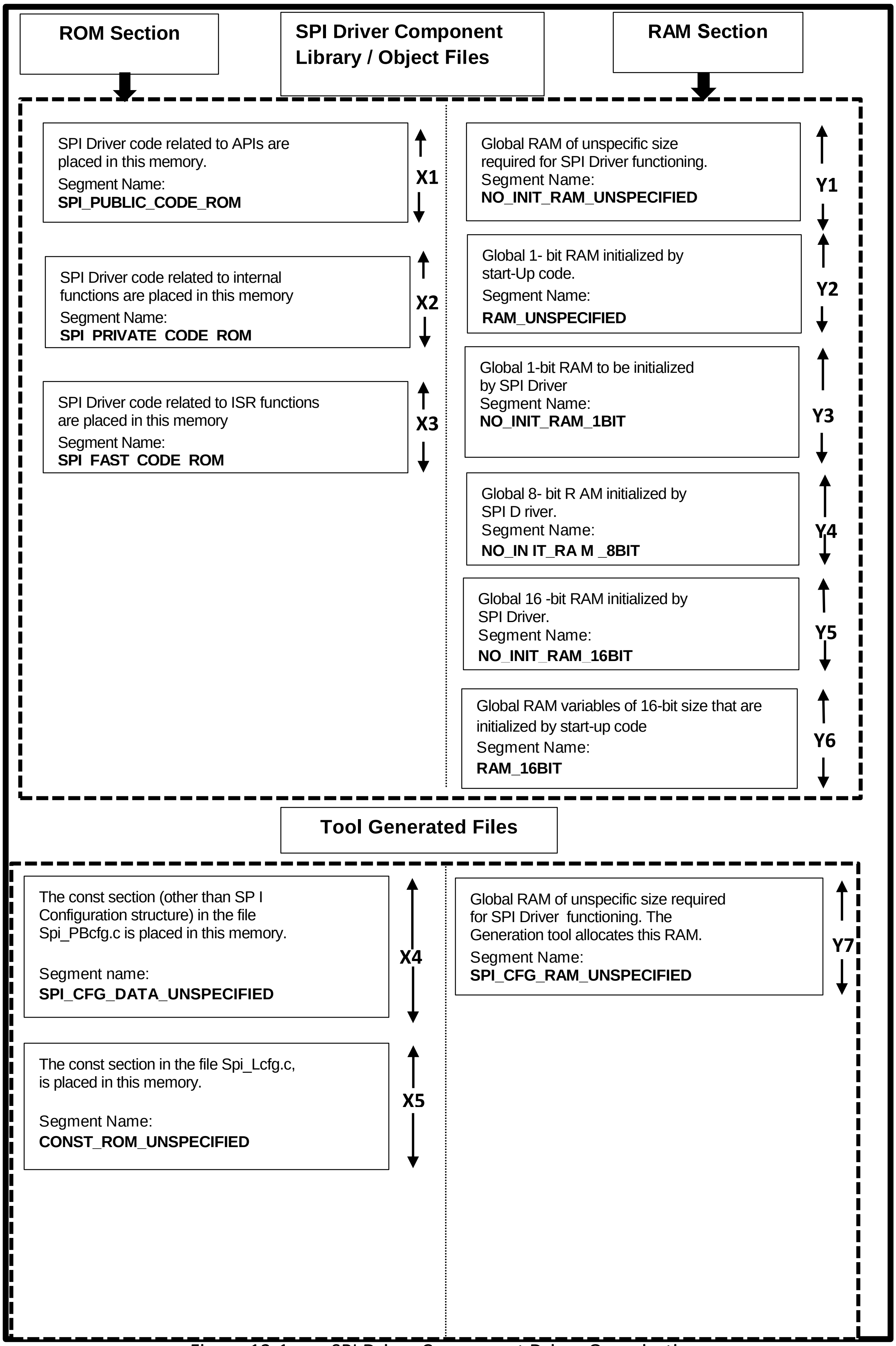
This error occurs when previously received data still resides in the reception register CSIGnRX0/CSIHnRX0, because it wasn’t read, and new data is received.The overrun error is not generated if data reception is disabled.

Note:

In general, If any of the above error is occured,a DEM error 'SPI_E_HARDWARE_ERROR' is reported to DEM .Also corresponding sequence result will be updated as failed and sequence will be suspended.

Chapter 12 Memory Organization

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



SPI Driver code related to APIs are placed in this memory.
Segment Name:
SPI_PUBLIC_CODE_ROM

SPI Driver code related to internal functions are placed in this memory
Segment Name:
SPI PRIVATE CODE ROM

SPI Driver code related to ISR functions are placed in this memory
Segment Name:
SPI FAST CODE ROM

Global RAM of unspecific size required for SPI Driver functioning.
Segment Name:
NO_INIT_RAM_UNSPECIFIED

Global 1- bit RAM initialized by start-Up code.
Segment Name:
RAM_UNSPECIFIED

Global 1-bit RAM to be initialized by SPI Driver
Segment Name:
NO_INIT_RAM_1BIT

Global 8- bit R AM initialized by SPI D river.
Segment Name:
NO_IN IT_RA M _8BIT

Global 16 -bit RAM initialized by SPI Driver.
Segment Name:
NO_INIT_RAM_16BIT

Global RAM variables of 16-bit size that are initialized by start-up code
Segment Name:
RAM_16BIT

The const section (other than SP I Configuration structure) in the file Spi_PBcfg.c is placed in this memory.

Segment name:
SPI_CFG_DATA_UNSPECIFIED

The const section in the file Spi_Lcfg.c, is placed in this memory.

Segment Name:
CONST_ROM_UNSPECIFIED

Global RAM of unspecific size required for SPI Driver functioning. The Generation tool allocates this RAM.
Segment Name:
SPI_CFG_RAM_UNSPECIFIED

Figure 12-1 SPI Driver Component Driver Organization

<u>**ROM Section (X1, X2, X3,X4,X5 and X6):**</u>

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.

<u>**RAM Section (Y1, Y2, Y3, Y4, Y5 and Y6):**</u>

NO_INIT_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.

NO_INIT_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.

NO_INIT_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.

RAM_16BIT (Y6): This section consists of the global RAM variables of 16-bit size that are initialized by start-up code and used internally by SPI software component and other software components. The specific sections of respective software components will be merged into this RAM section accordingly.

SPI_CFG_RAM_UNSPECIFIED (Y7): 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 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
INTDMA[0-15]	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
	SPI_DMA08_ISR
	SPI_DMA08_CAT2_ISR
	SPI_DMA09_ISR
	SPI_DMA09_CAT2_ISR
	SPI_DMA10_ISR
	SPI_DMA10_CAT2_ISR
	SPI_DMA11_ISR
	SPI_DMA11_CAT2_ISR
	SPI_DMA12_ISR
	SPI_DMA12_CAT2_ISR
	SPI_DMA13_ISR
	SPI_DMA13_CAT2_ISR
	SPI_DMA14_ISR
	SPI_DMA14_CAT2_ISR
	SPI_DMA15_ISR
	SPI_DMA15_CAT2_ISR

Note: The functions with “INTERRUPT“ as pilot tag, provides an indication to the compiler that the function following this tag is an interrupt function type. The tag name can vary according to the compiler. User should take care of the tag name with respect to compiler used.

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.

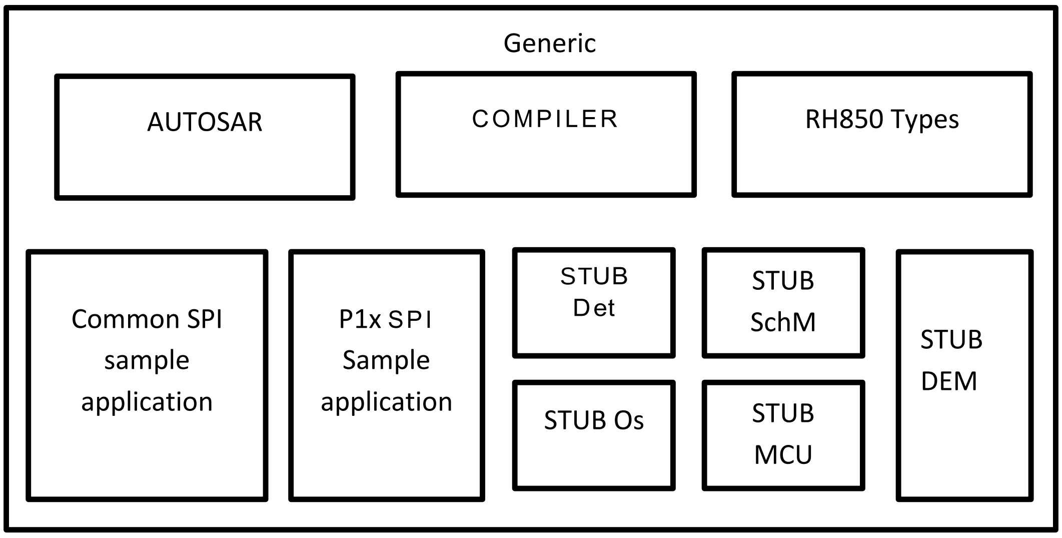


Figure 13-1 Overview Of SPI Driver Sample Application

13.2.1 Sample Application Structure

The Sample Application of the 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.arxml

/config/App_SPI_P1M_701305_Sample.arxml

/config/App_SPI_P1M_701310_Sample.arxml

/config/App_SPI_P1M_701311_Sample.arxml

/config/App_SPI_P1M_701312_Sample.arxml

/config/App_SPI_P1M_701313_Sample.arxml

/config/App_SPI_P1M_701314_Sample.arxml

/config/App_SPI_P1M_701315_Sample.arxml

/config/App_SPI_P1M_701318_Sample.arxml

/config/App_SPI_P1M_701319_Sample.arxml

/config/App_SPI_P1M_701320_Sample.arxml

/config/App_SPI_P1M_701321_Sample.arxml

/config/App_SPI_P1M_701322_Sample.arxml

/config/App_SPI_P1M_701323_Sample.arxml

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_GetErrorInfo copies Hardware Error Details to User Buffer
- 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.

13.2.2 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_P1M_<Device_name>_Sample.arxml

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\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_<Device_Name>_Sample.arxml file available in the path:

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

- 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 701311 or 701312 or 701313 or 701314 or 701315 or 701318 or 701319 or 701320 or 701321 or 701322 or 701323

13.3. Memory And Throughput

13.3.1 ROM/RAM Usage

The details of memory usage for the typical configuration, with DET disabled as provided in Section 13.2.2.1 *Configuration Example* are provided in this section.

Table 13-3 ROM/RAM Details without DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes for 701318
1.	ROM	SPI_PUBLIC_CODE_ROM	730
		SPI_PRIVATE_CODE_ROM	6312
		CONST_ROM_UNSPECIFIED	100
		SPI_CFG_DATA_UNSPECIFIED	212
		SPI_FAST_CODE_ROM	1108
		ROM.RAM_UNSPECIFIED	20
2.	RAM	RAM_UNSPECIFIED	20
		NO_INIT_RAM_1BIT	2
		NO_INIT_RAM_8BIT	0
		NO_INIT_RAM_16BIT	6
		NO_INIT_RAM_UNSPECIFIED	103
		SPI_CFG_RAM_UNSPECIFIED	0

The details of memory usage for the typical configuration, with DET enabled and all other configurations as provided in13.2.2.1 *Configuration Example* are provided in this section.

Table 13-4 ROM/RAM Details with DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes for 701318
1.	ROM	SPI_PUBLIC_CODE_ROM	1672
		SPI_PRIVATE_CODE_ROM	6494
		CONST_ROM_UNSPECIFIED	100
		SPI_CFG_DATA_UNSPECIFIED	212
		SPI_FAST_CODE_ROM	1108
		ROM.RAM_UNSPECIFIED	20

2.	RAM	RAM_UNSPECIFIED	20
		NO_INIT_RAM_1BIT	2
		NO_INIT_RAM_8BIT	0
		NO_INIT_RAM_16BIT	6
		NO_INIT_RAM_UNSPECIFIED	103
		SPI_CFG_RAM_UNSPECIFIED	0

13.3.2 Stack Depth

The worst-case stack depth for Driver Component is 88 bytes for the typical configuration provided in Section 13.2.2.1 *Configuration Example*.

13.3.3 Throughput Details

The throughput details of the APIs for the configuration mentioned in the Section13.2.2.1 *Configuration Example*. The clock frequency used to measure the throughput is 160 MHz for all APIs.

Table 13-5 Throughput Details Of The APIs

Sl. No.	API Name	Throughput in microseconds for 701318	Remarks
1.	Spi_Init	4.000	-
2.	Spi_DeInit	4.550	-
3.	Spi_WriteIB	0.612	-
4.	Spi_AsyncTransmit	11.250	-
5.	Spi_ReadIB	0.437	-
6.	Spi_SetupEB	0.287	-
7.	Spi_GetStatus	0.870	-
8.	Spi_GetJobResult	0.100	-
9.	Spi_GetSequenceResult	0.100	-
10.	Spi_GetVersionInfo	0.150	-
11.	Spi_SyncTransmit	13.950	-
12.	Spi_GetHWUnitStatus	0.362	-
13.	Spi_Cancel	0.662	-
14.	Spi_SetAsyncMode	0.262	SPI_INTERRUPT_ MODE
15.	Spi_SetAsyncMode	2.862	SPI_POLLING_ MODE
16.	Spi_MainFunction_Handling	1.462	-
17.	Spi_SelfTest	2227.500	SPI_LOOP_BACK_SELF_TEST
18.	Spi_SelfTest	57.275	SPI_ECC_SELF_TEST
19.	Spi_GetErrorInfo	0.225	-

Chapter 14 Release Details

SPI Driver Software

Version: 1.6.6

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

Sl.No.	Description	Version	Date
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.	1.0.6	29-April-2015
8.	As per P1x V4.00.05 release following changes are made: 1. Section 4.1 General forethoughts are updated. 2. Section 4.3 User mode/ supervisor mode is updated as per JIRA#ARDAAAE-1426 to add ISR related information. 3. Section 4.6 Deviation list is updated for the memory size measurement mismatches. 4. Section 6 Register details are updated for new added APIs Spi_SelfTest and Spi_GetErrorInfo. 5. Section 10.3 Function definitions is updated for new added APIs Spi_SelfTest and Spi_GetErrorInfo. 6. Section 11.2 Component production errors SPI_E_INT_INCONSISTENT, SPI_E_ECC_SELFTEST_FAILURE, SPI_E_LOOPBACK_SELFTEST_FAILURE and SPI_E_REG_WRITE_VERIFY are added. 7. Section 13.3 Memory and Throughput details are updated. 8. Section 14 S/W driver version is updated. 9. Chapter 11, As per JIRA#ARDAAAE-1419, new development error SPI_E_MAINFUNCTION_HANDLING_INVALIDMODE is added for Spi_MainFunction_Handling API. 10. Section 4.3, as per JIRA#ARDAAAE-1335, "-" is marked for Spi_AsyncTransmit API for interrupt mode in user mode. 11. In section 4.1 As per JIRA#ARDAAAE-1452, Information for 16 bit datawidth selection is added when DMA is configured. 12. Table – 6.1 Register details, 8bit and 32bit settings when DMA is configured are removed. 13. Table 4-5 User Mode and Supervisory Mode is updated.	1.0.7	29-Jan-2016

Sl.No.	Description	Version	Date
9.	Following changes are made: 1. Section 4.1 is updated for adding a note when CsIdleEnforcement is configured as False as per the JIRA ticket #ARDAAAE-1549. 2. Section 4.1 is updated for adding a note about the usage of HW registers. 3. Section 13.4.1 is updated for removing memory section SPI_CFG_DBTOC_UNSPECIFIED as part of ticket ARDAAAE-1672. 4. Software patch version is updated in Chapter4.	1.0.8	07-Apr-2016
10.	Following changes are made: 1. Software patch version is updated in Chapter 14. 2. Section 4.1 is updated for adding the notes as part of requirement analysis. 3. Section 4.3, User mode and Supervisor mode details are updated for Spi_SetAsyncMode. 4. Tables, figures links and numbering is corrected. 5. Stub header files heading is updated and missing header files are added. 6. Section 13.3.1, sample application file structure is updated and Section 13.3.2, Building sample application is updated. 7. Updated Table 6-1 to rename global variable ‘Spi_GusDataAccess’ as ‘Spi_GusSynDataAccess’ or ‘Spi_GusAsynDataAccess’ for synchronous and asynchronous transmission respectively. 8. Updated section 13.3.1 Sample Application Structure to add details about Spi_GetErrorInfo API. 9. Added Spi_GetErrorInfo API in section 11.1 under Related API(s) corresponding to the error SPI_E_UNINIT. 10. Updated 4.1 ‘General’ to add a caution regarding usage of buffers for transmission/reception during DMA operation. 11. Updated Chapter 12 to correct the INIT policy of memory sections from NOINIT to NO_INIT. 12. Chapter 13.1.3 ISR Function “Interrupt Handler” table is updated with note. 13. Updated 4.1 ‘General’ to add the information regarding the number of buffers to be configured in Direct Access or FIFO mode when DMA is configured. 14. Chapter 6, Register access details are updated. 15. Spi_GetErrorInfo details have been added. Chapter 4, 5 and 7 are updated for the same. 16. Section 4.2 Preconditions and Section 4.5 Data Consistency is updated for information about critical section protection. 17. Chapter 6, Register access details are updated. 18. Updated Table 4-1 for information regarding user mode and supervisor mode. 19. Section 3.1.1 is updated to add the header file Spi_RegWrite.h as part of implementing the register write functionality as part of ticket ARDAAAE-1685. 20. Section 4.3, A note is added regarding the critical section usage. 21. Spi_RegWrite.h is added to the folder structure in the section 3.1.1. 22. Chapter 11 is updated for the API details of the DET and DEM errors. 23. Updated Section 10.2 to add details regarding Spi_CommErrorType, Spi_HWErrorsType, Spi_SelfTestType and Spi_ReturnStatus type definitions.	1.0.9	12-Jul-2016

Sl.No.	Description	Version	Date
11.	Following changes are made: 1. Software patch version is updated in Chapter 14. 2. Chapter 13.3 updated for ROM/RAM Usage,Stack Depth and Throughput Details.	1.0.10	28-Oct-2016
12.	Following changes are made: 1. Section 11.2 is updated and 11.3 is added with the hardware errors description details 2. Section 10.3 is updated with the detailed description of the functions 3. Section 3.1.1 is updated with the deletion of the redundant mentioned Driver.h file name 4. Throughput details, RAM/ROM Usage and stack depth values are updated in the section 13.3 5. Section 4.1 is updated with the SpiTimeOut configuring details. 6. The unused segment SPI_CFG_DBTOC_UNSPECIFIED details are removed from the chapter 12. 7. Abbreviations and Acronyms section is updated 8. Chapter 14 is updated with the release details. 9. R-number is updated 10. Notice and Company addresses are updated 11. Copyright information is updated	1.0.11	21-Feb-2017
13.	Following changes are made: 1. Throughput details, RAM/ROM Usage and stack depth values are updated in the section 13.3 2. Software patch version is updated in Chapter 14.	1.0.12	15-Mar-2017

AUTOSAR MCAL R4.0.3 User's Manual
SPI Driver Component Ver.1.0.12
Embedded User's Manual

Publication Date: Rev.1.01, March 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

R20UT3726EJ0101