RX Family

SCI Module Using Firmware Integration Technology

Introduction

This application note describes the serial communications interface (SCI) module and the infrared data association (IrDA) interface which uses Firmware Integration Technology (FIT). This module uses SCI to provide Asynchronous, Synchronous, and SPI (SSPI) support for all channels of the SCI peripheral and Infrared data communication support for the IrDA peripheral. In this document, this module is referred to as the SCI FIT module.

Target Devices

- RX110, RX111, RX113 Groups
- RX130 Group
- RX140 Group
- RX13T Group
- RX230, RX231 Groups
- RX23T Group
- RX23W Group
- RX23E-A Group
- RX24T Group
- RX24U Group
- RX64M Group
- RX65N, RX651 Groups
- RX66T Group
- RX66N Group
- RX671 Group
- RX71M Group
- RX72T Group
- RX72M Group
- RX72N Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Target Compilers

Renesas Electronics C/C++ Compiler Package for RX Family

GCC for Renesas RX

IAR C/C++ Compiler for Renesas RX

For details of the confirmed operation contents of each compiler, refer to “6.1 Confirmed Operation Environment".

Contents

1. Overview

1.1 SCI FIT Module

The SCI FIT module can be used by being implemented in a project as an API. See section 2.13, Adding the FIT Module to Your Project for details on methods to implement this FIT module into a project.

1.2 Overview of the SCI FIT Module

The SCI FIT module supports the following SCI peripheral functions depending on the RX MCU Groups.

Table 1.1 SCI Peripheral Functions Supported by MCU Groups

	SCIc	SCId	SCIe	SCIf	SCIg	SCIh	SCIi	SCIj	SCIk	SCIm
RX110			✓	✓						
RX111			✓	✓						
RX113			✓	✓						
RX130					✓	✓				
RX140					✓	✓			✓	
RX13T					✓	✓				
RX230					✓	✓				
RX231					✓	✓				
RX23T					✓					
RX23W					✓	✓				
RX23E-A					✓	✓				
RX24T					✓					
RX24U					✓					
RX64M					✓	✓				
RX65N					✓	✓	✓			
RX66T						✓	✓	✓		
RX66N						✓	✓	✓		
RX71M					✓	✓				
RX72T						✓	✓	✓		
RX72M						✓	✓	✓		
RX72N						✓	✓	✓		
RX671						✓			✓	✓

It is recommended that you review the Serial Communications Interface chapter in the Hardware User's Manual for your specific RX Family MCU for full details on this peripheral circuit. All basic UART, Master SPI, Master Synchronous, and IrDA interface (1) mode functionality is supported by this driver. Additionally, the driver supports the following features in Asynchronous mode:

- noise cancellation
- outputting baud clock on the SCK pin
- one-way flow control of either CTS or RTS

In IrDA interface mode, the module generates IrDA communication waveforms, and transmit and receives data via infrared light using the infrared data association (IrDA) interface and serial communications interface (SCI) base on the IrDA (Infrared Data Association) standard 1.0.

The module can be used in cooperation with DMAC or DTC.

Note:

1. The IrDA interface mode is only supported for RX113, RX23W, RX230, RX231 devices.

Features not supported by this driver are:

- extended mode (channel 12)
- multiprocessor mode (all channels)
- event linking

Handling of Channels

This is a multi-channel driver, and it supports all channels present on the peripheral. Specific channels can be excluded via compile-time defines to reduce driver RAM usage and code size if desired. These defines are specified in “r_sci_rx_config.h”.

An individual channel is initialized in the application by calling R_SCI_Open(). This function applies power to the peripheral and initializes settings particular to the specified mode. A handle is returned from this function to uniquely identify the channel. The handle references an internal driver structure that maintains pointers to the channel’s register set, buffers, and other critical information. It is also used as an argument for the other API functions.

Interrupts, and Transmission and Reception

Interrupts supported by this driver are TXI, TEI, RXI, and ERI. For Asynchronous mode, circular buffers are used to queue incoming as well as outgoing data. The size of these buffers can also be set on compilation.

The TXI and TEI interrupts are only used in Asynchronous mode. The TXI interrupt occurs when a byte in the TDR register has been shifted into the TSR register. During this interrupt, the next byte in the transmit circular buffer is placed into the TDR register to be ready for transmit. If a callback function is provided in the R_SCI_Open() call, it is called here with a TEI event passed to it. Support for TEI interrupts may be removed from the driver via a setting in “r_sci_rx_config.h”.

The RXI interrupt occurs each time the RDR register has shifted in a byte. In Asynchronous mode, this byte is loaded into the receive circular buffer during the interrupt for access later via an R_SCI_Receive() call at the application level. If a callback function is provided, it is called with a receive event. If the receive queue is full, it is called with a queue full event while the last received byte is not stored. In SSPI and Synchronous modes, the shifted-in byte is loaded directly into the receive buffer specified from the last R_SCI_Receive() or R_SCI_SendReceive() call. The data received before R_SCI_Receive() or R_SCI_SendReceive() call is ignored. With SSPI and Synchronous modes, data is transmitted and received in the RXI interrupt handler. The number of data remaining to be transferred or received can be checked with the value of the transmit counter (tx_cnt) and received counter (rx_cnt) in the handle set for the fourth parameter of the R_SCI_Open function. Refer to 2.10, Parameters for details.

Error Detection

The ERI interrupt occurs when a framing, overrun, or parity error is detected by the receive device. If a callback function is provided, the interrupt determines which error occurred and notifies the application of the event. Refer to 2.12, Callback Function for details.
This FIT module clears the error flag in the ERI interrupt handler regardless of the callback function provided or not. If the FIFO function is enabled, the callback function is called before the error flag is cleared. So, the data where the error occurred can be determined by reading the FRDR register for the number of data received. Refer to 2.12 Callback Function for details.

Note when using SCI with DTC/DMAC support:

- When using SCI with DTC: Change Heap size of BSP (‘BSP_CFG_HEAP_BYTES’) to 0x1000 instead of default value is 0x400.
- RX23W, RX23E-A, RX230, RX231 have only 4 DMAC channels, when using DMAC with SCI, please choose DMAC channel from 0 to 3 (SCI_CFG_CHn_TX_DMACA_CH_NUM and SCI_CFG_CHn_RX_DMACA_CH_NUM).

- Some command of Control function are not supported in this mode (SCI_CMD_EN_CTS_IN when using DMAC channel from 4 to 7).
- Need to import DTC/DMAC FIT module, and initiate for DMAC, open for DTC before using SCI functions with DTC/DMAC support.
- Configuration for a channel SCI must be same for TX and RX (Example: if DTC is the data transfer method of SCI1 TX, DTC must be the data transfer method of SCI1 RX for both SYNC and ASYNC mode).
- Chosen DMAC channel must be different between TX and RX of same SCI channel, between SCI channel and SCI channel (Example: SCI0/TX use DMAC0 -> SCI0/RX can't use DMAC0, must use another channel DMAC such as DMAC1; SCI1/TX can't use DMAC0 or DMAC1 -> it must use different DMAC channel such as SCI1/TX use DMAC2, SCI1/RX use DMAC3).
- When using DTC/DMAC, SCI FIT does not use BYTEQ to send/receive data.
- When using DTC/DMAC, SCI FIT does not support circular buffer.

Note when using IrDA interface mode:

- The standard prescribes that the minimum high-level pulse width should be 1.41 µs and the maximum high-level pulse width should be the bit period × (3/16 + 2.5%) or (the bit period ×3/16) + 1.08 µs.
- The default value of IrDA output pulse width is SCI_IRDA_OUT_WIDTH_3_16 (the bit period × 3/16). Choose the output width, the corresponding operating frequencies and bit rates of this module when setting the pulse width shorter than the default value.

1.3 API Overview

Table 1.2 lists the API functions included in this module.

Table 1.2 API Functions

Function Name	Description
R_SCI_Open()	Applies power to the SCI channel, initializes the associated registers, enables interrupts, and provides the channel handle for use with other API functions. Specifies the callback function which is called when a receive error or other interrupt events occur.
R_SCI_Close()	Removes power to the SCI channel and disables the associated interrupts.
R_SCI_Send()	Initiates transmit if transmitter is not in use.
R_SCI_Receive()	For Asynchronous mode, fetches data from a queue which is filled by RXI interrupts. For Synchronous and SSPI modes, initiates dummy data transmission and reception if transceiver is not in use.
R_SCI_SendReceive()	For Synchronous and SSPI modes only. Transmits and receives data simultaneously if the transceiver is not in use.
R_SCI_Control()	Handles special hardware or software operations for the SCI channel.
R_SCI_GetVersion()	Returns at runtime the driver version number.

1.4 Limitations

None.

1.5 Using the FIT SCI module

1.5.1 Using FIT SCI module in C++ project

For C++ project, add FIT SCI module interface header file within extern “C”{}:

```
Extern “C”
{
  #include “r_smc_entry.h”
  #include “r_sci_rx_if.h”
}
```

2. API Information

This FIT module has been confirmed to operate under the following conditions.

2.1 Hardware Requirements

The MCU used must support the following functions:

- SCI
- GPIO
- DMAC/DTC (if want to use DMAC/DTC data transfer features)
- IrDA

2.2 Software Requirements

This driver is dependent upon the following FIT module:

- Renesas Board Support Package (r_bsp) v5.20 or higher
- r_byteq (Asynchronous mode only)

2.3 Limitations

2.3.1 RAM Location Limitations

In FIT, if a value equivalent to NULL is set as the pointer argument of an API function, error might be returned due to parameter check. Therefore, do not pass a NULL equivalent value as pointer argument to an API function.

The NULL value is defined as 0 because of the library function specifications. Therefore, the above phenomenon would occur when the variable or function passed to the API function pointer argument is located at the start address of RAM (address 0x0). In this case, change the section settings or prepare a dummy variable at the top of the RAM so that the variable or function passed to the API function pointer argument is not located at address 0x0.

In the case of the CCRX project (e2 studio V7.5.0), the RAM start address is set as 0x4 to prevent the variable from being located at address 0x0. In the case of the GCC project (e2 studio V7.5.0) and IAR project (EWRX V4.12.1), the start address of RAM is 0x0, so the above measures are necessary.

The default settings of the section may be changed due to the IDE version upgrade. Please check the section settings when using the latest IDE.

2.4 Supported Toolchain

This driver has been confirmed to work with the toolchain listed in 6.1, Confirmed Operation Environment.

2.5 Interrupt Vector

The RXIn and ERIn interrupt is enabled by executing the R_SCI_Open function (for asynchronous mode).

For SSPI and synchronous modes, interrupts TXIn and TEIn are not used in these mode.

Table 2.1 lists the interrupt vector used in the SCI FIT Module.

Table 2.1 Interrupt Vector Used in the SCI FIT Module

Device	Interrupt Vector
RX110, RX111, RX113, RX130, RX140, RX13T, RX230, RX231, RX23T, RX23W, RX23E-A, RX24T, RX24U (1)	ERI2 interrupt (vector no.: 186) RXI2 interrupt (vector no.: 187) TXI2 interrupt (vector no.: 188) TEI2 interrupt (vector no.: 189) ERI3 interrupt (vector no.: 190) RXI3 interrupt (vector no.: 191) TXI3 interrupt (vector no.: 192) TEI3 interrupt (vector no.: 193) ERI4 interrupt (vector no.: 194) RXI4 interrupt (vector no.: 195) TXI4 interrupt (vector no.: 196) TEI4 interrupt (vector no.: 197) ERI7 interrupt (vector no.: 206) RXI7 interrupt (vector no.: 207) TXI7 interrupt (vector no.: 208) TEI7 interrupt (vector no.: 209) ERI10 interrupt (vector no.: 210) RXI10 interrupt (vector no.: 211) TXI10 interrupt (vector no.: 212) TEI10 interrupt (vector no.: 213) ERI0 interrupt (vector no.: 214) RXI0 interrupt (vector no.: 215) TXI0 interrupt (vector no.: 216) TEI0 interrupt (vector no.: 217) ERI1 interrupt (vector no.: 218) RXI1 interrupt (vector no.: 219) TXI1 interrupt (vector no.: 220) TEI1 interrupt (vector no.: 221) ERI5 interrupt (vector no.: 222) RXI5 interrupt (vector no.: 223) TXI5 interrupt (vector no.: 224) TEI5 interrupt (vector no.: 225) ERI6 interrupt (vector no.: 226) RXI6 interrupt (vector no.: 227) TXI6 interrupt (vector no.: 228) TEI6 interrupt (vector no.: 229)

Note 1. Available interrupt vectors vary depending on the MCU used and the number of pins on the MCU.

Device	Interrupt Vector
RX110, RX111, RX113, RX130, RX13T, RX230, RX231, RX23T, RX23W, RX23E-A RX24T, RX24U (1)	ERI8 interrupt (vector no.: 230) RXI8 interrupt (vector no.: 231) TXI8 interrupt (vector no.: 232) TEI8 interrupt (vector no.: 233) ERI9 interrupt (vector no.: 234) RXI9 interrupt (vector no.: 235) TXI9 interrupt (vector no.: 236) TEI9 interrupt (vector no.: 237) ERI12 interrupt (vector no.: 238) RXI12 interrupt (vector no.: 239) TXI12 interrupt (vector no.: 240) TEI12 interrupt (vector no.: 241) ERI11 interrupt (vector no.: 250) RXI11 interrupt (vector no.: 251) TXI11 interrupt (vector no.: 252) TEI11 interrupt (vector no.: 253)
RX64M, RX71M	RXI0 interrupt (vector no.: 58) TXI0 interrupt (vector no.: 59) RXI1 interrupt (vector no.: 60) TXI1 interrupt (vector no.: 61) RXI2 interrupt (vector no.: 62) TXI2 interrupt (vector no.: 63) RXI3 interrupt (vector no.: 80) TXI3 interrupt (vector no.: 81) RXI4 interrupt (vector no.: 82) TXI4 interrupt (vector no.: 83) RXI5 interrupt (vector no.: 84) TXI5 interrupt (vector no.: 85) RXI6 interrupt (vector no.: 86) TXI6 interrupt (vector no.: 87) RXI7 interrupt (vector no.: 98) TXI7 interrupt (vector no.: 99) RXI12 interrupt (vector no.: 116) TXI12 interrupt (vector no.: 117)

Device	Interrupt Vector
RX64M, RX71M	GROUPBL0 interrupt (vector no.: 110) TEI0 interrupt (group interrupt source no.: 0) ERI0 interrupt (group interrupt source no.: 1) TEI1 interrupt (group interrupt source no.: 2) ERI1 interrupt (group interrupt source no.: 3) TEI2 interrupt (group interrupt source no.: 4) ERI2 interrupt (group interrupt source no.: 5) TEI3 interrupt (group interrupt source no.: 6) ERI3 interrupt (group interrupt source no.: 7) TEI4 interrupt (group interrupt source no.: 8) ERI4 interrupt (group interrupt source no.: 9) TEI5 interrupt (group interrupt source no.: 10) ERI5 interrupt (group interrupt source no.: 11) TEI6 interrupt (group interrupt source no.: 12) ERI6 interrupt (group interrupt source no.: 13) TEI7 interrupt (group interrupt source no.: 14) ERI7 interrupt (group interrupt source no.: 15) TEI12 interrupt (group interrupt source no.: 16) ERI12 interrupt (group interrupt source no.: 17)
RX65N, RX671	RXI0 interrupt (vector no.: 58) TXI0 interrupt (vector no.: 59) RXI1 interrupt (vector no.: 60) TXI1 interrupt (vector no.: 61) RXI2 interrupt (vector no.: 62) TXI2 interrupt (vector no.: 63) RXI3 interrupt (vector no.: 80) TXI3 interrupt (vector no.: 81) RXI4 interrupt (vector no.: 82) TXI4 interrupt (vector no.: 83) RXI5 interrupt (vector no.: 84) TXI5 interrupt (vector no.: 85) RXI6 interrupt (vector no.: 86) TXI6 interrupt (vector no.: 87) RXI7 interrupt (vector no.: 98) TXI7 interrupt (vector no.: 99) RXI8 interrupt (vector no.: 100) TXI8 interrupt (vector no.: 101) RXI9 interrupt (vector no.: 102) TXI9 interrupt (vector no.: 103) RXI10 interrupt (vector no.: 104) TXI10 interrupt (vector no.: 105) RXI11 interrupt (vector no.: 114) TXI11 interrupt (vector no.: 115) RXI12 interrupt (vector no.: 116) TXI12 interrupt (vector no.: 117)

Device	Interrupt Vector
RX65N,RX671	GROUPBL0 interrupt (vector no.: 110) TEI0 interrupt (group interrupt source no.: 0) ERI0 interrupt (group interrupt source no.: 1) TEI1 interrupt (group interrupt source no.: 2) ERI1 interrupt (group interrupt source no.: 3) TEI2 interrupt (group interrupt source no.: 4) ERI2 interrupt (group interrupt source no.: 5) TEI3 interrupt (group interrupt source no.: 6) ERI3 interrupt (group interrupt source no.: 7) TEI4 interrupt (group interrupt source no.: 8) ERI4 interrupt (group interrupt source no.: 9) TEI5 interrupt (group interrupt source no.: 10) ERI5 interrupt (group interrupt source no.: 11) TEI6 interrupt (group interrupt source no.: 12) ERI6 interrupt (group interrupt source no.: 13) TEI7 interrupt (group interrupt source no.: 14) ERI7 interrupt (group interrupt source no.: 15) TEI12 interrupt (group interrupt source no.: 16) ERI12 interrupt (group interrupt source no.: 17) GROUPBL1 interrupt (vector no.: 111) TEI8 interrupt (group interrupt source no.: 24) ERI8 interrupt (group interrupt source no.: 25) TEI9 interrupt (group interrupt source no.: 26) ERI9 interrupt (group interrupt source no.: 27) GROUPAL0 interrupt (vector no.: 112) TEI10 interrupt (group interrupt source no.: 8) ERI10 interrupt (group interrupt source no.: 9) TEI11 interrupt (group interrupt source no.: 12) ERI11 interrupt (group interrupt source no.: 13)

Device	Interrupt Vector
RX66T, RX72T	RXI1 interrupt (vector no.: 60) TXI1 interrupt (vector no.: 61) RXI5 interrupt (vector no.: 84) TXI5 interrupt (vector no.: 85) RXI6 interrupt (vector no.: 86) TXI6 interrupt (vector no.: 87) RXI8 interrupt (vector no.: 100) TXI8 interrupt (vector no.: 101) RXI9 interrupt (vector no.: 102) TXI9 interrupt (vector no.: 103) RXI11 interrupt (vector no.: 114) TXI11 interrupt (vector no.: 115) RXI12 interrupt (vector no.: 116) TXI12 interrupt (vector no.: 117) GROUPBL0 interrupt (vector no.: 110) • TEI1 interrupt (group interrupt source no.: 2) • ERI1 interrupt (group interrupt source no.: 3) • TEI5 interrupt (group interrupt source no.: 10) • ERI5 interrupt (group interrupt source no.: 11) • TEI6 interrupt (group interrupt source no.: 12) • ERI6 interrupt (group interrupt source no.: 13) • TEI12 interrupt (group interrupt source no.: 16) • ERI12 interrupt (group interrupt source no.: 17) GROUPBL1 interrupt (vector no.: 111) • TEI8 interrupt (group interrupt source no.: 24) • ERI8 interrupt (group interrupt source no.: 25) • TEI9 interrupt (group interrupt source no.: 26) • ERI9 interrupt (group interrupt source no.: 27) GROUPAL0 interrupt (vector no.: 112) • TEI11 interrupt (group interrupt source no.: 12) • ERI11 interrupt (group interrupt source no.: 13)
RX72M, RX72N, RX66N	RXI0 interrupt (vector no.: 58) TXI0 interrupt (vector no.: 59) RXI1 interrupt (vector no.: 60) TXI1 interrupt (vector no.: 61) RXI2 interrupt (vector no.: 62) TXI2 interrupt (vector no.: 63) RXI3 interrupt (vector no.: 80) TXI3 interrupt (vector no.: 81) RXI4 interrupt (vector no.: 82) TXI4 interrupt (vector no.: 83) RXI5 interrupt (vector no.: 84) TXI5 interrupt (vector no.: 85) RXI6 interrupt (vector no.: 86) TXI6 interrupt (vector no.: 87) RXI7 interrupt (vector no.: 98) TXI7 interrupt (vector no.: 99) RXI8 interrupt (vector no.: 100) TXI8 interrupt (vector no.: 101) RXI9 interrupt (vector no.: 102) TXI9 interrupt (vector no.: 103) RXI10 interrupt (vector no.: 104)

	TXI10 interrupt (vector no.: 105) RXI11 interrupt (vector no.: 114) TXI11 interrupt (vector no.: 115) RXI12 interrupt (vector no.: 116) TXI12 interrupt (vector no.: 117)

Device	Interrupt Vector
RX72M, RX72N, RX66N	GROUPBL0 interrupt (vector no.: 110) TEI0 interrupt (group interrupt source no.: 0) ERI0 interrupt (group interrupt source no.: 1) TEI1 interrupt (group interrupt source no.: 2) ERI1 interrupt (group interrupt source no.: 3) TEI2 interrupt (group interrupt source no.: 4) ERI2 interrupt (group interrupt source no.: 5) TEI3 interrupt (group interrupt source no.: 6) ERI3 interrupt (group interrupt source no.: 7) TEI4 interrupt (group interrupt source no.: 8) ERI4 interrupt (group interrupt source no.: 9) TEI5 interrupt (group interrupt source no.: 10) ERI5 interrupt (group interrupt source no.: 11) TEI6 interrupt (group interrupt source no.: 12) ERI6 interrupt (group interrupt source no.: 13) TEI12 interrupt (group interrupt source no.: 16) ERI12 interrupt (group interrupt source no.: 17) GROUPAL0 interrupt (vector no.: 112) TEI7 interrupt (group interrupt source no.: 22) ERI7 interrupt (group interrupt source no.: 23) TEI8 interrupt (group interrupt source no.: 0) ERI8 interrupt (group interrupt source no.: 1) TEI9 interrupt (group interrupt source no.: 4) ERI9 interrupt (group interrupt source no.: 5) TEI10 interrupt (group interrupt source no.: 8) ERI10 interrupt (group interrupt source no.: 9) TEI11 interrupt (group interrupt source no.: 12) ERI11 interrupt (group interrupt source no.: 13)

2.6 Header Files

All API calls and their supporting interface definitions are located in r_sci_rx_if.h.

2.7 Integer Types

This project uses ANSI C99. These types are defined in stdint.h.

2.8 Configuration Overview

The configuration option settings of this module are located in r_sci_rx_config.h. The option names and setting values are listed in the table below:

Configuration options in r_sci_rx_config.h (1/2)	
SCI_CFG_PARAM_CHECKING_ENABLE 1	1: Parameter checking is included in the build. 0: Parameter checking is omitted from the build. Setting this #define to BSP_CFG_PARAM_CHECKING_ENABLE utilizes the system default setting.
SCI_CFG_ASYNC_INCLUDED 1 SCI_CFG_SYNC_INCLUDED 0 SCI_CFG_SSPI_INCLUDED 0 SCI_CFG_IRDA_INCLUDED 0	These #defines are used to include code specific to their mode of operation. A value of 1 means that the supporting code will be included. Use a value of 0 for unused modes to reduce overall code size.
SCI_CFG_USE_CIRCULAR_BUFFER 0	This #define is used only with Asynchronous mode. A value of 1 means that the circular buffer will be used. Use a value of 0 if circular buffer is not used. Note: Set BSP_CFG_RUN_IN_USER_MODE = 0 and BYTEQ_CFG_PROTECT_QUEUE = 1 when setting SCI_CFG_USE_CIRCULAR_BUFFER = 1. If not, build error would occur.
SCI_CFG_DUMMY_TX_BYTE 0xFF	This #define is used only with SSPI and Synchronous mode. It is the value of dummy data which is clocked out for each byte clocked in during the R_SCI_Receive() function call.
SCI_CFG_CH0_INCLUDED 0 SCI_CFG_CH1_INCLUDED 1 SCI_CFG_CH2_INCLUDED 0 SCI_CFG_CH3_INCLUDED 0 SCI_CFG_CH4_INCLUDED 0 SCI_CFG_CH5_INCLUDED 0 SCI_CFG_CH6_INCLUDED 0 SCI_CFG_CH7_INCLUDED 0 SCI_CFG_CH8_INCLUDED 0 SCI_CFG_CH9_INCLUDED 0 SCI_CFG_CH10_INCLUDED 0 SCI_CFG_CH11_INCLUDED 0 SCI_CFG_CH12_INCLUDED 0	Each channel has associated with it transmit and receive buffers, counters, interrupts, and other program and RAM resources. Setting a #define to 1 allocates resources for that channel. **Note that only CH1 is enabled by default. Be sure to enable the channels you will be using in the config file.**
SCI_CFG_CH0_TX_BUFSIZ 80 SCI_CFG_CH1_TX_BUFSIZ 80 SCI_CFG_CH2_TX_BUFSIZ 80 SCI_CFG_CH3_TX_BUFSIZ 80 SCI_CFG_CH4_TX_BUFSIZ 80 SCI_CFG_CH5_TX_BUFSIZ 80 SCI_CFG_CH6_TX_BUFSIZ 80 SCI_CFG_CH7_TX_BUFSIZ 80 SCI_CFG_CH8_TX_BUFSIZ 80 SCI_CFG_CH9_TX_BUFSIZ 80 SCI_CFG_CH10_TX_BUFSIZ 80 SCI_CFG_CH11_TX_BUFSIZ 80 SCI_CFG_CH12_TX_BUFSIZ 80	These #defines specify the size of the buffer to be used in Asynchronous mode for the transmit queue on each channel. If the corresponding SCI_CFG_CHn_INCLUDED is set to 0, or SCI_CFG_ASYNC_INCLUDED is set to 0, the buffer is not allocated.

Configuration options in r_sci_rx_config.h (2/2)	
SCI_CFG_CH0_RX_BUFSIZ 80 SCI_CFG_CH1_RX_BUFSIZ 80 SCI_CFG_CH2_RX_BUFSIZ 80 SCI_CFG_CH3_RX_BUFSIZ 80 SCI_CFG_CH4_RX_BUFSIZ 80 SCI_CFG_CH5_RX_BUFSIZ 80 SCI_CFG_CH6_RX_BUFSIZ 80 SCI_CFG_CH7_RX_BUFSIZ 80 SCI_CFG_CH8_RX_BUFSIZ 80 SCI_CFG_CH9_RX_BUFSIZ 80 SCI_CFG_CH10_RX_BUFSIZ 80 SCI_CFG_CH11_RX_BUFSIZ 80 SCI_CFG_CH12_RX_BUFSIZ 80	These #defines specify the size of the buffer to be used in Asynchronous mode for the receive queue on each channel. If the corresponding SCI_CFG_CHn_INCLUDED is set to 0, or SCI_CFG_ASYNC_INCLUDED is set to 0, the buffer is not allocated.
SCI_CFG_TEI_INCLUDED 0	Setting this #define to 1 causes the Transmit Buffer Empty interrupt code to be included. This interrupt occurs when the last bit of the last byte of data has been sent. The interrupt calls the user's callback function (specified in R_SCI_Open()) and passes it an SCI_EVT_TEI event.
SCI_CFG_RXERR_PRIORITY 3	RX63N/631 ONLY. This sets the Group12 receiver error interrupt priority level. 1 is the lowest priority and 15 is the highest. This interrupt handles overrun, framing, and parity errors for all channels.
SCI_CFG_ERI_TEI_PRIORITY 3	RX64M/RX71M/RX65N/RX72M/RX72N/RX66N/RX671 ONLY. This sets the receiver error interrupt (ERI) and transmit end interrupt (TEI) priority level. 1 is the lowest priority and 15 is the highest. The ERI interrupt handles overrun, framing, and parity errors for all channels. The TEI interrupt indicates when the last bit has been transmitted and the transmitter is idle (Asynchronous mode).
SCI_CFG_CH7_FIFO_INCLUDED 0 SCI_CFG_CH8_FIFO_INCLUDED 0 SCI_CFG_CH9_FIFO_INCLUDED 0 SCI_CFG_CH10_FIFO_INCLUDED 0 SCI_CFG_CH11_FIFO_INCLUDED 0	ONLY MCUs which has the SCI module (SCIi) with FIFO function. 1: Processing regarding the FIFO function is included in the build 0: processing regarding the FIFO function is omitted from the build
SCI_CFG_CH7_TX_FIFO_THRESH 8 SCI_CFG_CH8_TX_FIFO_THRESH 8 SCI_CFG_CH9_TX_FIFO_THRESH 8 SCI_CFG_CH10_TX_FIFO_THRESH 8 SCI_CFG_CH11_TX_FIFO_THRESH 8	ONLY MCUs which has the SCI module (SCIi) with FIFO function. When the SCI operating mode is clock synchronous mode or simple SPI mode, set the values same as the receive FIFO threshold value. 0 to 15: Specifies the threshold value of the transmit FIFO.
SCI_CFG_CH7_RX_FIFO_THRESH 8 SCI_CFG_CH8_RX_FIFO_THRESH 8 SCI_CFG_CH9_RX_FIFO_THRESH 8 SCI_CFG_CH10_RX_FIFO_THRESH 8 SCI_CFG_CH11_RX_FIFO_THRESH 8	ONLY MCUs which has the SCI module (SCIi) with FIFO function. 1 to 15: Specifies the threshold value of the receive FIFO.

SCI_CFG_CH0_DATA_MATCH_INCLUDED 0 SCI_CFG_CH1_DATA_MATCH_INCLUDED 0 SCI_CFG_CH2_DATA_MATCH_INCLUDED 0 SCI_CFG_CH3_DATA_MATCH_INCLUDED 0 SCI_CFG_CH4_DATA_MATCH_INCLUDED 0 SCI_CFG_CH5_DATA_MATCH_INCLUDED 0 SCI_CFG_CH6_DATA_MATCH_INCLUDED 0 SCI_CFG_CH7_DATA_MATCH_INCLUDED 0 SCI_CFG_CH8_DATA_MATCH_INCLUDED 0 SCI_CFG_CH9_DATA_MATCH_INCLUDED 0 SCI_CFG_CH10_DATA_MATCH_INCLUDED 0 SCI_CFG_CH11_DATA_MATCH_INCLUDED 0	RX65N/RX66T/RX72T/RX72M/RX72N/RX66N/RX671 ONLY. It has the SCI module (SCIi, SCIj or SCIm, SCIk) with Data match function. 1: Processing regarding the data match function is included in the build 0: processing regarding the data match function is omitted from the build
SCI_CFG_CH0_TX_DTC_DMACA_ENABLE 0 SCI_CFG_CH1_TX_DTC_DMACA_ENABLE 0 SCI_CFG_CH2_TX_DTC_DMACA_ENABLE 0 SCI_CFG_CH3_TX_DTC_DMACA_ENABLE 0 SCI_CFG_CH4_TX_DTC_DMACA_ENABLE 0 SCI_CFG_CH5_TX_DTC_DMACA_ENABLE 0 SCI_CFG_CH6_TX_DTC_DMACA_ENABLE 0 SCI_CFG_CH7_TX_DTC_DMACA_ENABLE 0 SCI_CFG_CH8_TX_DTC_DMACA_ENABLE 0 SCI_CFG_CH9_TX_DTC_DMACA_ENABLE 0 SCI_CFG_CH10_TX_DTC_DMACA_ENABLE 0 SCI_CFG_CH11_TX_DTC_DMACA_ENABLE 0 SCI_CFG_CH12_TX_DTC_DMACA_ENABLE 0	Choose method to transfer data for SCI channel 0 : Using CPU to transfer data 1 : Using DTC to transfer data 2 : Using DMAC to transfer data
SCI_CFG_CH0_RX_DTC_DMACA_ENABLE 0 SCI_CFG_CH1_RX_DTC_DMACA_ENABLE 0 SCI_CFG_CH2_RX_DTC_DMACA_ENABLE 0 SCI_CFG_CH3_RX_DTC_DMACA_ENABLE 0 SCI_CFG_CH4_RX_DTC_DMACA_ENABLE 0 SCI_CFG_CH5_RX_DTC_DMACA_ENABLE 0 SCI_CFG_CH6_RX_DTC_DMACA_ENABLE 0 SCI_CFG_CH7_RX_DTC_DMACA_ENABLE 0 SCI_CFG_CH8_RX_DTC_DMACA_ENABLE 0 SCI_CFG_CH9_RX_DTC_DMACA_ENABLE 0 SCI_CFG_CH10_RX_DTC_DMACA_ENABLE 0 SCI_CFG_CH11_RX_DTC_DMACA_ENABLE 0 SCI_CFG_CH12_RX_DTC_DMACA_ENABLE 0	Choose method to transfer data for SCI channel 0 : Using CPU to transfer data 1 : Using DTC to transfer data 2 : Using DMAC to transfer data
SCI_CFG_CH0_TX_DMACA_CH_NUM 0 SCI_CFG_CH1_TX_DMACA_CH_NUM 0 SCI_CFG_CH2_TX_DMACA_CH_NUM 0 SCI_CFG_CH3_TX_DMACA_CH_NUM 0 SCI_CFG_CH4_TX_DMACA_CH_NUM 0 SCI_CFG_CH5_TX_DMACA_CH_NUM 0 SCI_CFG_CH6_TX_DMACA_CH_NUM 0 SCI_CFG_CH7_TX_DMACA_CH_NUM 0 SCI_CFG_CH8_TX_DMACA_CH_NUM 0 SCI_CFG_CH9_TX_DMACA_CH_NUM 0 SCI_CFG_CH10_TX_DMACA_CH_NUM 0 SCI_CFG_CH11_TX_DMACA_CH_NUM 0 SCI_CFG_CH12_TX_DMACA_CH_NUM 0	ONLY MCUs which has the DMAC function (Refer 1.2 Section) 0 to 7: Specifies DMAC channel for SCI TX

SCI_CFG_CH0_RX_DMACA_CH_NUM 0 SCI_CFG_CH1_RX_DMACA_CH_NUM 0 SCI_CFG_CH2_RX_DMACA_CH_NUM 0 SCI_CFG_CH3_RX_DMACA_CH_NUM 0 SCI_CFG_CH4_RX_DMACA_CH_NUM 0 SCI_CFG_CH5_RX_DMACA_CH_NUM 0 SCI_CFG_CH6_RX_DMACA_CH_NUM 0 SCI_CFG_CH7_RX_DMACA_CH_NUM 0 SCI_CFG_CH8_RX_DMACA_CH_NUM 0 SCI_CFG_CH9_RX_DMACA_CH_NUM 0 SCI_CFG_CH10_RX_DMACA_CH_NUM 0 SCI_CFG_CH11_RX_DMACA_CH_NUM 0 SCI_CFG_CH12_RX_DMACA_CH_NUM 0	ONLY MCUs which has the DMAC function (Refer 1.2 Section) 0 to 7: Specifies DMAC channel for SCI RX
SCI_CFG_CH0_TX_SIGNAL_TRANSITION_TIMING_INCLUDED…..0 SCI_CFG_CH1_TX_SIGNAL_TRANSITION_TIMING_INCLUDED…..0 SCI_CFG_CH2_TX_SIGNAL_TRANSITION_TIMING_INCLUDED…..0 SCI_CFG_CH3_TX_SIGNAL_TRANSITION_TIMING_INCLUDED….. 0 SCI_CFG_CH4_TX_SIGNAL_TRANSITION_TIMING_INCLUDED….0 SCI_CFG_CH5_TX_SIGNAL_TRANSITION_TIMING_INCLUDED…. 0 SCI_CFG_CH6_TX_SIGNAL_TRANSITION_TIMING_INCLUDED…. 0 SCI_CFG_CH7_TX_SIGNAL_TRANSITION_TIMING_INCLUDED…. 0 SCI_CFG_CH8_TX_SIGNAL_TRANSITION_TIMING_INCLUDED…. 0 SCI_CFG_CH9_TX_SIGNAL_TRANSITION_TIMING_INCLUDED…. 0 SCI_CFG_CH10_TX_SIGNAL_TRANSITION_TIMING_INCLUDED…. 0 SCI_CFG_CH11_TX_SIGNAL_TRANSITION_TIMING_INCLUDED…. 0	ONLY MCUs which has the SCI module with transition timing function. 1: Processing regarding the transition timing function is included in the build. 0: processing regarding the transition timing function is omitted from the build.

SCI_CFG_CH0_RX_DATA_SAMPLING_TIMING_INCLUDED....0 SCI_CFG_CH1_RX_DATA_SAMPLING_TIMING_INCLUDED….0 SCI_CFG_CH2_RX_DATA_SAMPLING_TIMING_INCLUDED….0 SCI_CFG_CH3_RX_DATA_SAMPLING_TIMING_INCLUDED….0 SCI_CFG_CH4_RX_DATA_SAMPLING_TIMING_INCLUDED….0 SCI_CFG_CH5_RX_DATA_SAMPLING_TIMING_INCLUDED….0 SCI_CFG_CH6_RX_DATA_SAMPLING_TIMING_INCLUDED….0 SCI_CFG_CH7_RX_DATA_SAMPLING_TIMING_INCLUDED….0 SCI_CFG_CH8_RX_DATA_SAMPLING_TIMING_INCLUDED….0) SCI_CFG_CH9_RX_DATA_SAMPLING_TIMING_INCLUDED….0 SCI_CFG_CH10_RX_DATA_SAMPLING_TIMING_INCLUDED….0 SCI_CFG_CH11_RX_DATA_SAMPLING_TIMING_INCLUDED….0	ONLY MCUs which has the SCI module with data sampling function. 1: Processing regarding the data sampling function is included in the build. 0: processing regarding the data sampling function is omitted from the build.
SCI_CFG_CH5_IRDA_INCLUDED 0	RX113/RX23W/ RX230/RX231 ONLY. It has the SCI module with IrDA data communication waveform. 1: Processing regarding the IrDA function is included in the build 0: processing regarding the IrDA function is omitted from the build
SCI_CFG_CH5_IRDA_IRTXD_INACTIVE_LEVEL 1	Indicates the level of the selected IRTXD pin inactive state. - When this is set to 0, the selected IRTXD pin outputs low. - When this is set to 1, the selected IRTXD pin outputs high.
SCI_CFG_CH5_IRDA_IRRXD_INACTIVE_LEVEL 1	Indicates the level of the selected IRRXD pin inactive state. - When this is set to 0, the selected IRRXD pin outputs low. - When this is set to 1, the selected IRRXD pin outputs high.

2.9 Code Size

Typical code sizes associated with this module are listed below.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.8, Configuration Overview. The table lists reference values when the C compiler's compile options are set to their default values, as described in 2.4, Supported Toolchain. The compile option default values are optimization level: 2, optimization type: for size, and data endianness: little-endian. The code size varies depending on the C compiler version and compile options.

ROM and RAM minimum sizes (bytes) (1/4)					
Device	Category		Memory usage		Remarks
			Renesas Compiler		
			With Parameter Checking	Without Parameter Checking	
RX130	Asynchronous mode	ROM	4116 bytes	3774 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	3845 bytes	3441 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5143 bytes	4657 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Maximum stack usage		100 bytes		
RX13T	Asynchronous mode	ROM	2917 bytes	2664 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2647 bytes	2341 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	3946 bytes	3594 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Asynchronous mode + DTC	ROM	3591 bytes	3258 bytes	1 channel used
		RAM	446 bytes	446 bytes	1 channel used
	Clock synchronous mode + DTC	ROM	3434 bytes	3045 bytes	1 channel used
		RAM	290 bytes	290 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI) + DTC	ROM	4907 bytes	4432 bytes	Total 2 channels used
		RAM	706 bytes	706 bytes	Total 2 channels used
	Maximum stack usage		160 bytes		

RX231	Asynchronous mode	ROM	3496 bytes	2573 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2704 bytes	2231 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	IrDA interface mode	ROM	2768 bytes	2402 bytes	1 channel used
		RAM	196 bytes	196 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	4067 bytes	3498 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Maximum stack usage		72 bytes		

ROM and RAM minimum sizes (bytes) (2/4)					
Device	Category		Memory usage		Remarks
			Renesas Compiler		
			With Parameter Checking	Without Parameter Checking	
RX23W	Asynchronous mode	ROM	2892 bytes	2559 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2600 bytes	2217 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	IrDA interface mode	ROM	2664 bytes	2388 bytes	1 channel used
		RAM	196 bytes	196 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	4003 bytes	3484 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Maximum stack usage		72 bytes		
RX23E-A	Asynchronous mode	ROM	2725 bytes	2400 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Asynchronous mode + Circular buffer	ROM	2724 bytes	2412 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2468 bytes	2094 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	3751 bytes	3282 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Maximum stack usage		72 bytes		

ROM and RAM minimum sizes (bytes) (3/4)					
Device	Communication method		Memory usage		Remarks
			Renesas Compiler		
			With Parameter Checking	Without Parameter Checking	
RX64M	Asynchronous mode	ROM	2861 bytes	2500 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2598 bytes	2185 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	3894 bytes	3389 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Maximum stack usage		80 bytes		
RX65N	Asynchronous mode	ROM	2852 bytes	2488 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2586 bytes	2173 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	3885 bytes	3377 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Asynchronous mode + DTC	ROM	3642 bytes	3280 bytes	1 channel used
		RAM	446 bytes	446 bytes	1 channel used
	Maximum stack usage		180 bytes		
	FIFO mode + Asynchronous mode	ROM	3758 bytes	3348 bytes	1 channel used
		RAM	200 bytes	200 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	3714 bytes	3223 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	5306 bytes	4723 bytes	Total 2 channels used
		RAM	408 bytes	408 bytes	Total 2 channels used

	FIFO mode + Asynchronous mode + Clock synchronous mode + DMAC	ROM	8865 bytes	8300 bytes	Total 2 channels used
		RAM	530 bytes	530 bytes	Total 2 channels used
	Maximum stack usage		204 bytes		

ROM and RAM minimum sizes (bytes) (4/4)					
Device	Communication method		Memory usage		Remarks
			Renesas Compiler		
			With Parameter Checking	Without Parameter Checking	
RX66T	Asynchronous mode	ROM	2845 bytes	2481 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2579 bytes	2166 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	3768 bytes	3260 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Maximum stack usage		80 bytes		
	FIFO mode + Asynchronous mode	ROM	3748 bytes	3338 bytes	1 channel used
		RAM	200 bytes	200 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	3705 bytes	3214 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	5143 bytes	4560 bytes	Total 2 channels used
		RAM	364 bytes	364 bytes	Total 2 channels used
	Maximum stack usage		80 bytes		
RX72T	Asynchronous mode	ROM	2845 bytes	2481 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2579 bytes	2166 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	3732 bytes	3224 bytes	Total 2 channels used
		RAM	356 bytes	356 bytes	Total 2 channels used
	Maximum stack usage		80 bytes		
	FIFO mode + Asynchronous mode	ROM	3748 bytes	3338 bytes	1 channel used
		RAM	200 bytes	200 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	3705 bytes	3214 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used

	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	5166 bytes	4583 bytes	Total 2 channels used
		RAM	364 bytes	364 bytes	Total 2 channels used
	Maximum stack usage		80 bytes		
RX72M	Asynchronous mode	ROM	2866 bytes	2502 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2600 bytes	2187 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	3899 bytes	3391 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Asynchronous mode + DTC	ROM	3656 bytes	3292 bytes	1 channel used
		RAM	446 bytes	446 bytes	1 channel used
	Maximum stack usage		180 bytes		
	FIFO mode + Asynchronous mode	ROM	3769 bytes	3359 bytes	1 channel used
		RAM	227 bytes	227 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	3726 bytes	3235 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5318 bytes	4735 bytes	Total 2 channels used
		RAM	408 bytes	408 bytes	Total 2 channels used
	FIFO mode + Asynchronous mode + Clock synchronous mode + DMAC	ROM	8877 bytes	8312 bytes	Total 2 channels used
		RAM	530 bytes	530 bytes	Total 2 channels used
	Maximum stack usage		204 bytes		
RX72N	Asynchronous mode	ROM	2922 bytes	2558 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2657 bytes	2244 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	3956 bytes	3448 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used

	Maximum stack usage		88 bytes		
	FIFO mode + Asynchronous mode	ROM	3825 bytes	3415 bytes	1 channel used
		RAM	200 bytes	200 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	3769 bytes	3278 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	5364 bytes	4781 bytes	Total 2 channels used
		RAM	408 bytes	408 bytes	Total 2 channels used
	Maximum stack usage		100 bytes		
RX66N	Asynchronous mode	ROM	2922 bytes	2502 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2657 bytes	2244 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	3956 bytes	3448 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Maximum stack usage		92 bytes		
	FIFO mode + Asynchronous mode	ROM	3825 bytes	3415 bytes	1 channel used
		RAM	200 bytes	200 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	3769 bytes	3278 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5364 bytes	4781 bytes	Total 2 channels used
		RAM	408 bytes	408 bytes	Total 2 channels used
	Maximum stack usage		100 bytes		
RX671	Asynchronous mode	ROM	3226 bytes	3970 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Asynchronous mode + Circular buffer	ROM	3325 bytes	3982 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2858 bytes	2458 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	4391 bytes	3891 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Maximum stack usage		72 bytes		

	FIFO mode + Asynchronous mode	ROM	4263 bytes	3860 bytes	1 channel used
		RAM	200 bytes	200 bytes	1 channel used
	FIFO mode + Asynchronous mode + Circular buffer	ROM	4250 bytes	3860 bytes	1 channel used
		RAM	200 bytes	200 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	3943 bytes	3476 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	5817 bytes	5248 bytes	Total 2 channels used
		RAM	408 bytes	408 bytes	Total 2 channels used
	Maximum stack usage		72 bytes		
RX140	Asynchronous mode	ROM	3080 bytes	2751 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Asynchronous mode + Circular buffer	ROM	3087 bytes	2757 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	2614 bytes	2240 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	4212 bytes	3738 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Maximum stack usage		72 bytes		

ROM and RAM minimum sizes (bytes) (1/3)					
Device	Category		Memory usage		Remarks
			GCC		
			With Parameter Checking	Without Parameter Checking	
RX130	Asynchronous mode	ROM	6960 bytes	6400 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Clock synchronous mode	ROM	6612 bytes	5988 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	8836 bytes	8020 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		
RX13T	Asynchronous mode	ROM	7400 bytes	6776 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	6996 bytes	6484 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	9376 bytes	8584 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Asynchronous mode + DTC	ROM	8748 bytes	8140 bytes	1 channel used
		RAM	448 bytes	448 bytes	1 channel used
	Clock synchronous mode + DTC	ROM	8552 bytes	7872 bytes	1 channel used
		RAM	294 bytes	294 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI) + DTC	ROM	11368 bytes	10464 bytes	Total 2 channels used
		RAM	708 bytes	708 bytes	Total 2 channels used
	Maximum stack usage		-		
RX231	Asynchronous mode	ROM	5568 bytes	4968 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used

	Clock synchronous mode	ROM	5116 bytes	4428 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	IrDA interface mode	ROM	5748 bytes	5244 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	7724 bytes	6812 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		
RX23E-A	Asynchronous mode	ROM	5456 bytes	4856 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Asynchronous mode + Circular buffer	ROM	5440 bytes	4824 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Clock synchronous mode	ROM	5012 bytes	4324 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	7724 bytes	6820 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		

ROM and RAM minimum sizes (bytes) (2/3)					
Device	Communication method		Memory usage		Remarks
			GCC		
			With Parameter Checking	Without Parameter Checking	
RX64M	Asynchronous mode	ROM	5048 bytes	4432 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Clock synchronous mode	ROM	4708 bytes	4044 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	6964 bytes	6100 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		
RX65N	Asynchronous mode	ROM	5056 bytes	4424 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Clock synchronous mode	ROM	4700 bytes	4036 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	6964 bytes	6092 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		
	FIFO mode + Asynchronous mode	ROM	6824 bytes	6112 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	6980 bytes	6164 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	9732 bytes	8740 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		

ROM and RAM minimum sizes (bytes) (3/3)					
Device	Communication method		Memory usage		Remarks
			GCC		
			With Parameter Checking	Without Parameter Checking	
RX66T	Asynchronous mode	ROM	5056 bytes	4424 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Clock synchronous mode	ROM	4700 bytes	4036 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	6964 bytes	6092 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		
	FIFO mode + Asynchronous mode	ROM	6824 bytes	6112 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	6980 bytes	6164 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	9572 bytes	8580 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		
RX72T	Asynchronous mode	ROM	5056 bytes	4424 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Clock synchronous mode	ROM	4700 bytes	4036 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	6964 bytes	6092 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		
	FIFO mode + Asynchronous mode	ROM	6824 bytes	6112 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel

					used
	FIFO mode + Clock synchronous mode	ROM	6996 bytes	6164 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	9732 bytes	8740bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		
RX72M	Asynchronous mode	ROM	5520 bytes	4848 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Clock synchronous mode	ROM	5124 bytes	4388 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	7620 bytes	6636 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		
	FIFO mode + Asynchronous mode	ROM	7400 bytes	6616 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	7564 bytes	6692 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	10620 bytes	9524 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		
RX72N	Asynchronous mode	ROM	5576 bytes	4896 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	5264 bytes	4436 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	7684 bytes	6692 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels

					used
	Maximum stack usage		-		
	FIFO mode + Asynchronous mode	ROM	7456 bytes	6664 bytes	1 channel used
		RAM	200 bytes	200 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	7604 bytes	6732 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	10652 bytes	9548 bytes	Total 2 channels used
		RAM	408 bytes	408 bytes	Total 2 channels used
	Maximum stack usage		-		
RX66N	Asynchronous mode	ROM	5576 bytes	4896 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	5164 bytes	4436 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	7684 bytes	6692 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Maximum stack usage		-		-
	FIFO mode + Asynchronous mode	ROM	7456 bytes	6664 bytes	1 channel used
		RAM	200 bytes	200 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	7604 bytes	6732 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	10652 bytes	9548 bytes	Total 2 channels used
		RAM	408 bytes	408 bytes	Total 2 channels used
	Maximum stack usage		-		

RX671	Asynchronous mode	ROM	6732 bytes	6052bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Asynchronous mode + Circular buffer	ROM	6708 bytes	6020 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Clock synchronous mode	ROM	5660 bytes	4600 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	8912 bytes	7952 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		
	FIFO mode + Asynchronous mode	ROM	8748 bytes	7956 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	FIFO mode + Asynchronous mode + Circular buffer	ROM	8700 bytes	7900 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	8410 bytes	7236 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	12048 bytes	10952 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		

RX140	Asynchronous mode	ROM	6368 bytes	5376bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Asynchronous mode + Circular buffer	ROM	6200 bytes	5432 bytes	1 channel used
		RAM	160 bytes	160 bytes	1 channel used
	Clock synchronous mode	ROM	5136 bytes	4072 bytes	1 channel used
		RAM	0 bytes	0 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	8480 bytes	7200 bytes	Total 2 channels used
		RAM	320 bytes	320 bytes	Total 2 channels used
	Maximum stack usage		-		

ROM and RAM minimum sizes (bytes) (1/3)					
Device	Category		Memory usage		Remarks
			IAR Compiler		
			With Parameter Checking	Without Parameter Checking	
RX130	Asynchronous mode	ROM	4431 bytes	3847 bytes	1 channel used
		RAM	576 bytes	576 bytes	1 channel used
	Clock synchronous mode	ROM	3791 bytes	3207 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5797 bytes	4989 bytes	Total 2 channels used
		RAM	776 bytes	776 bytes	Total 2 channels used
	Maximum stack usage		180 bytes		
RX13T	Asynchronous mode	ROM	4233 bytes	3671 bytes	1 channel used
		RAM	577 bytes	541 bytes	1 channel used
	Clock synchronous mode	ROM	3585 bytes	3025 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5587 bytes	4801 bytes	Total 2 channels used
		RAM	777 bytes	777 bytes	Total 2 channels used
	Asynchronous mode + DTC	ROM	6259 bytes	5592 bytes	1 channel used
		RAM	760 bytes	760 bytes	1 channel used
	Clock synchronous mode + DTC	ROM	5788 bytes	5120 bytes	1 channel used
		RAM	219 bytes	219 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI) + DTC	ROM	7944 bytes	7050 bytes	Total 2 channels used
		RAM	1020 bytes	1020 bytes	Total 2 channels used
	Maximum stack usage		160 bytes		
RX231	Asynchronous mode	ROM	4392 bytes	3802 bytes	1 channel used
		RAM	577 bytes	577 bytes	1 channel used

	Clock synchronous mode	ROM	3737 bytes	3153 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	IrDA interface mode	ROM	4475 bytes	3945 bytes	1 channel used
		RAM	581 bytes	581 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5804 bytes	4990 bytes	Total 2 channels used
		RAM	777 bytes	777 bytes	Total 2 channels used
	Maximum stack usage		180 bytes		
RX23E-A	Asynchronous mode	ROM	4005 bytes	3509 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Asynchronous mode + Circular buffer	ROM	4028 bytes	3524 bytes	1 channel used
		RAM	192 bytes	192 bytes	1 channel used
	Clock synchronous mode	ROM	3677 bytes	3110 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5371 bytes	4651 bytes	Total 2 channels used
		RAM	392 bytes	392 bytes	Total 2 channels used
	Maximum stack usage		148 bytes		

ROM and RAM minimum sizes (bytes) (2/3)					
Device	Communication method		Memory usage		Remarks
			IAR Compiler		
			With Parameter Checking	Without Parameter Checking	
RX64M	Asynchronous mode	ROM	4566 bytes	3962 bytes	1 channel used
		RAM	577 bytes	577 bytes	1 channel used
	Clock synchronous mode	ROM	3935 bytes	3333 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5940 bytes	5112 bytes	Total 2 channels used
		RAM	777 bytes	777 bytes	Total 2 channels used
	Maximum stack usage		204 bytes		
RX65N	Asynchronous mode	ROM	4565 bytes	3962 bytes	1 channel used
		RAM	577 bytes	577 bytes	1 channel used
	Clock synchronous mode	ROM	3924 bytes	3329 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5935 bytes	5108 bytes	Total 2 channels used
		RAM	777 bytes	777 bytes	Total 2 channels used
	Maximum stack usage		204 bytes		
	FIFO mode + Asynchronous mode	ROM	5872 bytes	5172 bytes	1 channel used
		RAM	585 bytes	585 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	5577 bytes	4875 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	7960 bytes	7026 bytes	Total 2 channels used
		RAM	793 bytes	793 bytes	Total 2 channels used
	Maximum stack usage		240 bytes		

ROM and RAM minimum sizes (bytes) (3/3)					
Device	Communication method		Memory usage		Remarks
			IAR Compiler		
			With Parameter Checking	Without Parameter Checking	
RX66T	Asynchronous mode	ROM	4562 bytes	3961 bytes	1 channel used
		RAM	577 bytes	577 bytes	1 channel used
	Clock synchronous mode	ROM	3925 bytes	3332 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5815 bytes	4990 bytes	Total 2 channels used
		RAM	741 bytes	741 bytes	Total 2 channels used
	Maximum stack usage		204 bytes		
	FIFO mode + Asynchronous mode	ROM	5869 bytes	5171 bytes	1 channel used
		RAM	585 bytes	585 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	5578 bytes	4878 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	7837 bytes	6905 bytes	Total 2 channels used
		RAM	749 bytes	749 bytes	Total 2 channels used
	Maximum stack usage		240 bytes		
RX72T	Asynchronous mode	ROM	4567 bytes	3962 bytes	1 channel used
		RAM	577 bytes	577 bytes	1 channel used
	Clock synchronous mode	ROM	3926 bytes	3329 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5940 bytes	5111 bytes	Total 2 channels used
		RAM	777 bytes	777 bytes	Total 2 channels used
	Maximum stack usage		204 bytes		
	FIFO mode + Asynchronous mode	ROM	5893 bytes	5191 bytes	1 channel used
		RAM	585 bytes	585 bytes	1 channel used
	FIFO mode + Clock synchronous	ROM	5579 bytes	4875 bytes	1 channel used

	mode	RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	7965 bytes	7029 bytes	Total 2 channels used
		RAM	793 bytes	793 bytes	Total 2 channels used
	Maximum stack usage		240 bytes		
RX72M	Asynchronous mode	ROM	4482 bytes	3854 bytes	1 channel used
		RAM	577 bytes	577 bytes	1 channel used
	Clock synchronous mode	ROM	3797 bytes	3192 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5891 bytes	5042 bytes	Total 2 channels used
		RAM	777 bytes	777 bytes	Total 2 channels used
	Maximum stack usage		264 bytes		
	FIFO mode + Asynchronous mode	ROM	5777 bytes	5050 bytes	1 channel used
		RAM	585 bytes	585 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	5438 bytes	4723 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	7911 bytes	6952 bytes	Total 2 channels used
		RAM	793 bytes	793 bytes	Total 2 channels used
	Maximum stack usage		288 bytes		
RX72N	Asynchronous mode	ROM	4441 bytes	3842 bytes	1 channel used
		RAM	577 bytes	577 bytes	1 channel used
	Clock synchronous mode	ROM	3800 bytes	3213 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5734 bytes	4911 bytes	Total 2 channels used
		RAM	581 bytes	581 bytes	Total 2 channels used
	Maximum stack usage		148 bytes		
	FIFO mode + Asynchronous mode	ROM	5722 bytes	5026 bytes	1 channel used
		RAM	585 bytes	585 bytes	1 channel used

	FIFO mode + Clock synchronous mode	ROM	5411 bytes	4709 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	7794 bytes	6864 bytes	Total 2 channels used
		RAM	793 bytes	793 bytes	Total 2 channels used
	Maximum stack usage		192 bytes		
RX66N	Asynchronous mode	ROM	4441 bytes	3838 bytes	1 channel used
		RAM	577 bytes	577 bytes	1 channel used
	Clock synchronous mode	ROM	3808 bytes	3209 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	5815 bytes	4988 bytes	Total 2 channels used
		RAM	777 bytes	777 bytes	Total 2 channels used
	Maximum stack usage		148 bytes		
	FIFO mode + Asynchronous mode	ROM	5722 bytes	5031 bytes	1 channel used
		RAM	585 bytes	585 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	5411 bytes	4713 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	7798 bytes	6864 bytes	Total 2 channels used
		RAM	793 bytes	793 bytes	Total 2 channels used
	Maximum stack usage		192 bytes		
RX671	Asynchronous mode	ROM	4935 bytes	4311 bytes	1 channel used
		RAM	577 bytes	577 bytes	1 channel used
	Asynchronous mode + Circular buffer	ROM	4958 bytes	4326 bytes	1 channel used
		RAM	577 bytes	577 bytes	1 channel used
	Clock synchronous mode	ROM	3950 bytes	3337 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	6271 bytes	5489 bytes	Total 2 channels used
		RAM	777 bytes	777 bytes	Total 2 channels used
	Maximum stack usage		152 bytes		

	FIFO mode + Asynchronous mode	ROM	6309 bytes	5592 bytes	1 channel used
		RAM	585 bytes	585 bytes	1 channel used
	FIFO mode + Asynchronous mode + Circular buffer	ROM	6318 bytes	5599 bytes	1 channel used
		RAM	585 bytes	585 bytes	1 channel used
	FIFO mode + Clock synchronous mode	ROM	5487 bytes	4839 bytes	1 channel used
		RAM	44 bytes	44 bytes	1 channel used
	FIFO mode + Asynchronous mode + Clock synchronous mode	ROM	8365 bytes	7461 bytes	Total 2 channels used
		RAM	793 bytes	793 bytes	Total 2 channels used
	Maximum stack usage		196 bytes		
RX140	Asynchronous mode	ROM	4740 bytes	4150 bytes	1 channel used
		RAM	577 bytes	577 bytes	1 channel used
	Asynchronous mode + Circular buffer	ROM	4835 bytes	4231 bytes	1 channel used
		RAM	577 bytes	577 bytes	1 channel used
	Clock synchronous mode	ROM	3811 bytes	3235 bytes	1 channel used
		RAM	36 bytes	36 bytes	1 channel used
	Asynchronous mode + Clock synchronous mode (or simple SPI)	ROM	6228 bytes	5410 bytes	Total 2 channels used
		RAM	777 bytes	777 bytes	Total 2 channels used
	Maximum stack usage		148 bytes		

RAM requirements vary based on the number of channels configured. Each channel has associated data structures in RAM. In addition, for Asynchronous mode, each Async channel will have a Transmit queue and a Receive queue. The buffers for these queues each have a minimum size of 2 bytes, or a total of 4 bytes per channel. Since the queue buffer sizes are user configurable, the RAM requirement will be increased or decreased directly by the amount allocated for buffers.

The formula for calculating Async mode RAM requirements is:

Number of channels used (1 to 12) × (Data structure per channel (32 bytes)
+ Transmit queue buffer size (size specified by SCI_CFG_CHn_TX_BUFSIZ)
+ Receive queue buffer size (size specified by SCI_CFG_CHn_RX_BUFSIZ))

* For FIFO mode, the data structure per channel is 36 bytes.

The Sync and SPI mode RAM requirements are number of channels × data structure per channel (fixed at 36 bytes, for FIFO mode, fixed at 40 bytes).

The ROM requirements vary based on the number of channels configured for use. The exact amount varies depending on the combination of channels selected and the effects of compiler code optimization.

2.10 Parameters

This section describes the parameter structure used by the API functions in this module. The structure is located in r_sci_rx_if.h as are the prototype declarations of API functions.

Structure for Managing Channels

This structure is to store management information required to control SCI channels. The contents of the structure vary depending on settings of the configuration option and the device used. Though the user does not need to care for the contents of the structure, if clock synchronous mode/SSPI mode is used, the number of data to be processed can be checked with tx_cnt or rx_cnt.

The following shows an example of the structure for RX65N:

```
typedef struct st_sci_ch_ctrl       // Channel management structure
{
sci_ch_rom_t const *rom;      // Start address of the SCI register for the
channel
#if (SCI_CFG_IRDA_INCLUDED)
    sci_irda_ch_port_rom_t const *port_rom;     // Port setting values for pins
IRTXD and IRRXD
#endif
sci_mode_t mode;        // SCI operating mode currently set for the channel
uint32_t baud_rate;     // Baud rate currently set for the channel
void (*callback)(void *p_args);     // Address of the callback function
union
{
#if (SCI_CFG_ASYNC_INCLUDED)
byteq_hdl_t que;        // Transmit byte queue (asynchronous mode)
#endif
uint8_t *buf;     // Start address of the transmit buffer
//(clock synchronous/SSPI mode)
} u_tx_data;
union
{
#if (SCI_CFG_ASYNC_INCLUDED)
byteq_hdl_t que;        // Receive byte queue (asynchronous mode)
#endif
uint8_t *buf;           // Start address of the receive buffer
                        //(synchronous/SSPI mode)
} u_rx_data;
bool tx_idle;           // Transmission idle state (idle state/transmitting)
#if (SCI_CFG_SSPI_INCLUDED || SCI_CFG_SYNC_INCLUDED)
bool save_rx_data;            // Receive data storage (enable/disable)
uint16_t tx_cnt;        // Transmit counter
uint16_t rx_cnt;        // Receive counter
bool tx_dummy;          // Transmit dummy data (enable/disable)
#endif
uint32_t pclk_speed;    // Operating frequency of the peripheral module clock
#if SCI_CFG_FIFO_INCLUDED
uint8_t fifo_ctrl;            // FIFO function (enable/disable)
uint8_t rx_dflt_thresh;       // Recive FIFO threshold value (default)
uint8_t rx_curr_thresh;       // Recive FIFO threshold value (current)
uint8_t tx_dflt_thresh;       // Transmit FIFO threshold value (default)
uint8_t tx_curr_thresh;       // Transmit FIFO threshold value (current)
#endif
#if ((TX_DTC_DMACA_ENABLE || RX_DTC_DMACA_ENABLE))
    uint8_t                         qindex_app_tx;
    uint8_t                         qindex_int_tx;
    uint8_t                         qindex_app_rx;
    uint8_t                         qindex_int_rx;
    sci_fifo_ctrl_t                queue[2];
#endif
```

```
} sci_ch_ctrl_t;
```

2.11 Return Values

This section describes return values of API functions. This enumeration is located in r_sci_rx_if.h as are the prototype declarations of API functions.

```
typedef enum e_sci_err      // SCI API error codes
{
    SCI_SUCCESS=0,
    SCI_ERR_BAD_CHAN,       // Non-existent channel number
    SCI_ERR_OMITTED_CHAN,   // SCI_CHx_INCLUDED is 0 in config.h
    SCI_ERR_CH_NOT_CLOSED,  // Channel still running in another mode
    SCI_ERR_BAD_MODE,       // Unsupported or incorrect mode for channel
    SCI_ERR_INVALID_ARG,    // Argument is not valid for parameter
    SCI_ERR_NULL_PTR,       // Received null ptr; missing required argument
    SCI_ERR_XCVR_BUSY,      // Cannot start data transfer; transceiver busy

    // Asynchronous/Infrared mode only
    SCI_ERR_QUEUE_UNAVAILABLE,  // Cannot open tx or rx queue or both
    SCI_ERR_INSUFFICIENT_SPACE, // Not enough space in transmit queue
    SCI_ERR_INSUFFICIENT_DATA,  // Not enough data in receive queue

    // Synchronous/SSPI modes only
    SCI_ERR_XFER_NOT_DONE,   // Data transfer still in progress
    SCI_ERR_DTC,
    SCI_ERR_DMACA,
    SCI_ERR_DTC_DMACA
} sci_err_t;
```

2.12 Callback Function

In this module, the callback function specified by the user is called when the RXIn, ERIn interrupt occurs.

The callback function is specified by storing the address of the user function in the “void (* const p_callback)(void *p_args)” structure member (see 2.10, Parameters). When the callback function is called, the variable which stores the constant is passed as the argument.

The argument is passed as void type. Thus the argument of the callback function is cast to a void pointer. See examples below as reference.

When using a value in the callback function, type cast the value.

The following shows an example template for the callback function in asynchronous mode.

```
void MyCallback(void *p_args)
{
sci_cb_args_t *args;
args = (sci_cb_args_t *)p_args;
if (args->event == SCI_EVT_RX_CHAR)
{
//from RXI interrupt; character placed in queue is in args->byte
nop();
}
else if (args->event == SCI_EVT_RX_CHAR_MATCH)
{
//from RXI interrupt, received data match comparison data
//character placed in queue is in args->byte
nop();
}

#if SCI_CFG_TEI_INCLUDED
else if (args->event == SCI_EVT_TEI)
{
// from TEI interrupt; transmitter is idle
```

```
// possibly disable external transceiver here
nop();
}
#endif
else if (args->event == SCI_EVT_RXBUF_OVFL)
{
// from RXI interrupt; receive queue is full
// unsaved char is in args->byte
// will need to increase buffer size or reduce baud rate
nop();
}
else if (args->event == SCI_EVT_OVFL_ERR)
{
// from ERI/Group12 interrupt; receiver overflow error occurred
// error char is in args->byte
// error condition is cleared in ERI routine
nop();
}
else if (args->event == SCI_EVT_FRAMING_ERR)
{
// from ERI/Group12 interrupt; receiver framing error occurred
// error char is in args->byte; if = 0, received BREAK condition
// error condition is cleared in ERI routine
nop();
}
else if (args->event == SCI_EVT_PARITY_ERR)
{
// from ERI/Group12 interrupt; receiver parity error occurred
// error char is in args->byte
// error condition is cleared in ERI routine
nop();
}
else if ( args->event == SCI_EVT_RX_DONE)
{
// Receive full data when SCI supported by DTC/DMAC
      nop();
}
}
```

The following shows an example template for the callback function in SSPI mode.

```
void sspiCallback(void *p_args)
{
sci_cb_args_t *args;
args = (sci_cb_args_t *)p_args;
if (args->event == SCI_EVT_XFER_DONE)
{
// data transfer completed
nop();
}
else if (args->event == SCI_EVT_XFER_ABORTED)
{
// data transfer aborted
nop();
}
else if (args->event == SCI_EVT_OVFL_ERR)
{
// from ERI or Group12 (RX63x) interrupt; receiver overflow error occurred
// error char is in args->byte
// error condition is cleared in ERI/Group12 interrupt routine
nop();
```

```
}
else if ( args->event == SCI_EVT_RX_SYNC_DONE)
{
// Receive full data when SCI supported by DTC/DMAC
      nop();
}
}
```

The following shows an example template for the callback function in Infrared communication mode.

```
void irdaCallback(void *p_args)
{
      sci_cb_args_t *args;
      args = (sci_cb_args_t *)p_args;
      if (SCI_EVT_RX_CHAR == args->event)
      {
            // from RXI interrupt; character placed in queue is in args->byte
            nop();
      }
#if SCI_CFG_TEI_INCLUDED
      else if (SCI_EVT_TEI == args->event)
      {
            // from TEI interrupt; transmitter is idle
            // possibly disable external transceiver here
            nop();
      }
#endif
      else if (SCI_EVT_RXBUF_OVFL == args->event)
      {
            // from RXI interrupt; receive queue is full
            // unsaved char is in args->byte
            // will need to increase buffer size or reduce baud rate
            nop();
      }

      else if (SCI_EVT_OVFL_ERR == args->event)
      {
            // from ERI/Group12 interrupt; receiver overflow error occurred
            // error char is in args->byte
            // error condition is cleared in ERI routine
            nop();
      }
      else if (SCI_EVT_FRAMING_ERR == args->event)
      {
            // from ERI/Group12 interrupt; receiver framing error occurred
            // error char is in args->byte; if = 0, received BREAK condition
            // error condition is cleared in ERI routine
            nop();
      }
}
```

This FIT module calls the callback function specified by the user when a receive error interrupt occurs, when 1-byte data is received in asynchronous mode, when transmissions/receptions for the specified number of bytes have been completed in clock synchronous or SSPI mode, and when a transmit end interrupt occurs.

Note that if the FIFO function is enabled in asynchronous mode, the callback function is executed when receptions for the maximum number of times specified with SCI_CFG_CHn_RX_FIFO_THRESH have been completed or 15 etu (1) has elapsed from the stop bit of the last received data.

The callback function is set by specifying the address of the callback function to the fourth parameter of R_SCI_Open(). When the callback function is called, the following parameters are set.

```
typedef struct st_sci_cb_args       // Arguments of the callback function
{
sci_hdl_t hdl;          // Handle upon an event occurrence
sci_cb_evt_t event;     // Event which triggered the event occurred
uint8_t byte;           // Receive data upon an event occurrence
uint8_t num;                // Receive data size (valid only when FIFO is
used)
} sci_cb_args_t;

typedef enum e_sci_cb_evt           // Event for the callback function
{
// Events for asynchronous and infrared communication mode
SCI_EVT_TEI,                // TEI interrupt occurred.
SCI_EVT_RX_CHAR,        // Character received; Have placed in the queue.
SCI_EVT_RXBUF_OVFL,     // Receive queue full; No more data can be stored.
SCI_EVT_FRAMING_ERR,    // Framing error occurred in the receiver.
// Events for asynchronous mode
SCI_EVT_PARITY_ERR,     // Parity error occurred in the receiver.
SCI_EVT_RX_CHAR_MATCH   // Received data match; already place in the queue.
// Events for SSPI/clock synchronous mode
SCI_EVT_XFER_DONE,          // Transfer completed.
SCI_EVT_XFER_ABORTED,   // Transfer canceled.
// Common event
SCI_EVT_OVFL_ERR,           // Overrun error occurred in receive device
/* Receive Sync Done */
SCI_EVT_RX_SYNC_DONE,
/* Receive Async Done */
SCI_EVT_RX_DONE
} sci_cb_evt_t;
```

Since the argument is passed as a void pointer, arguments of the callback function must be the pointer variable of type void, for example, when using the argument value within the callback function, it must be type-casted.

Note 1. etu (Elementary Time Unit): 1-bit transfer period

When the following events occur, a received data stored in the argument of the callback function becomes undefined value:

- SCI_EVT_TEI
- SCI_EVT_XFER_DONE
- SCI_EVT_XFER_ABORTED
- SCI_EVT_OVFL_ERR (when FIFO function enabled)
- SCI_EVT_PARITY_ERR (when FIFO function enabled)
- SCI_EVT_FRAMING_ERR (when FIFO function enabled)

2.13 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio
By using the FIT Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

2.14 “for”, “while” and “do while” statements

In this module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

The following shows example of description.

```
while statement example :
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}

for statement example :
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}

do while statement example :
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /* WAIT_LOOP */
```

3. API Functions

R_SCI_Open()

This function applies power to the SCI channel, initializes the associated registers, enables interrupts, and provides the channel handle for use with other API functions. This function must be called before calling any other API functions.

Format

```
sci_err_t       R_SCI_Open (
                uint8_t  const          chan,
                        sci_mode_t  const    mode,
                sci_cfg_t * const       p_cfg,
                void                    (* const p_callback)(void *p_args),
                sci_hdl_t * const       p_hdl
)
```

Parameters

uint8_t const chan
Channel to initialize.

sci_mode_t const mode
Operational mode (see enumeration below)

*sci_cfg_t * const p_cfg*
Pointer to configuration union, structure elements (see below) are specific to mode

p_callback
Pointer to function called from interrupt when an RXI or receiver error is detected or for transmit end (TEI) condition
Refer to 2.12, Callback Function for details.

*sci_hdl_t * const p_hdl*
Pointer to a handle for channel (value set here)
Confirm the return value from R_SCI_Open is “SCI_SUCCESS” and then set the first parameter for the other APIs except R_SCI_GetVersion(). Refer to 2.10, Parameters.

The following SCI modes are currently supported by this driver module. The mode specified determines the union structure element used for the p_cfg parameter.

```
typedef enum e_sci_mode     // SCI operational modes
{
    SCI_MODE_OFF=0,         // channel not in use
    SCI_MODE_ASYNC,         // Asynchronous
    SCI_MODE_SSPI,          // Simple SPI
    SCI_MODE_SYNC,          // Synchronous
    SCI_MODE_IRDA,          // Infrared data communication
    SCI_MODE_MAX            // End of modes currently supported
} sci_mode_t;
```

#defines shown on the next page indicate configurable options for Asynchronous mode used in its configuration structure. These values correspond to bit definitions in the SMR register and specify the data length, the parity function, and the STOP bit. The BRR register and the SEMR register are set using the clock source (8x/16x of the internal/external clock) specified with clk_src of the sci_uart_t structure and the bit rate specified with baud_rate of the sci_uart_t structure. Please note this does not guarantee the

specified bit rate (there may be some errors depending on the setting). In addition, when using the channel 10 and 11 in the Synchronous mode or SSPI mode with the FIFO feature, you will not be able to set high-speed bit rate than PCLKA/8. (For example, if PCLKA is 120 MHz, it is possible to set the bit rate of equal to or less than 15 Mbps.)

The following shows the union for p_cfg:

```
typedef union
{
    sci_uart_t      async;
    sci_sync_sspi_t sync;
    sci_sync_sspi_t sspi;
    sci_irda_t      irda;
} sci_cfg_t;
```

The following shows the structure used for settings in Asynchronous mode:

```
typedef struct st_sci_uart
{
    uint32_t    baud_rate;      // ie 9600, 19200, 115200 (valid for internal
clock)
    uint8_t     clk_src;        // use SCI_CLK_INT/EXT8/EXT16
    uint8_t     data_size;      // use SCI_DATA_nBIT
    uint8_t     parity_en;      // use SCI_PARITY_ON/OFF
    uint8_t     parity_type;    // use SCI_ODD/EVEN_PARITY
    uint8_t     stop_bits;      // use SCI_STOPBITS_1/2
    uint8_t     int_priority;   // txi, tei, rxi, eri INT priority; 1=low,
15=high
} sci_uart_t;
```

The following shows the definitions of the structure (sci_uart_t) members used in Asynchronous mode:

```
/* Definitions for the sck_src member. */
#define SCI_CLK_INT         0x00  // use internal clock for baud rate generation
#define SCI_CLK_EXT_8X      0x03  // use external clock 8x baud rate
#define SCI_CLK_EXT_16X     0x02  // use external clock 16x baud rate

/* Definitions for the data_size member. */
#define SCI_DATA_7BIT       0x40    // 7-bit length
#define SCI_DATA_8BIT       0x00    // 8-bit length

/* Definitions for the parity_en member. */
#define SCI_PARITY_ON       0x20    // Parity ON
#define SCI_PARITY_OFF      0x00    // Parity OFF

/* Definitions for the parity_type member. */
#define SCI_ODD_PARITY      0x10    // Odd parity
#define SCI_EVEN_PARITY     0x00    // Even parity

/* Definitions for the stop_bits member.
#define SCI_STOPBITS_2      0x08    // 2-stop bit
#define SCI_STOPBITS_1      0x00    // 1-stop bit
```

The following shows the structure used for settings in SSPI and Synchronous modes:

```
typedef struct st_sci_sync_sspi
{
    sci_spi_mode_t  spi_mode;       // clock polarity and phase; unused for sync
    uint32_t        bit_rate;       // ie 1000000 for 1Mbps
```

```
    bool            msb_first;
    bool            invert_data;
    uint8_t         int_priority;   // rxi,eri interrupt priority; 1=low,
15=high
} sci_sync_sspi_t;
```

The following shows the enumeration used for spi_mode of the sci_sync_sspi_t structure in SSPI or Synchronous mode:

```
typedef enum e_sci_spi_mode
{
    SCI_SPI_MODE_OFF = 1, // Used in synchronous mode
    SCI_SPI_MODE_0 = 0x80,// SPMR Register CKPH=1, CKPOL=0
                          // Mode 0: 00 CPOL=0 resting lo, CPHA=0 leading
edge/rising
    SCI_SPI_MODE_1 = 0x40,// SPMR Register CKPH=0, CKPOL=1
                          // Mode 1: 01 CPOL=0 resting lo, CPHA=1 trailing
edge/falling
    SCI_SPI_MODE_2 = 0xC0,// SPMR Register CKPH=1, CKPOL=1
                          // Mode 2: 10 CPOL=1 resting hi, CPHA=0 leading
edge/falling
    SCI_SPI_MODE_3 = 0x00 // SPMR Register CKPH=0, CKPOL=0
                          // Mode 3: 11 CPOL=1 resting hi, CPHA=1 trailing
edge/rising
} sci_spi_mode_t;
```

The following shows the structure used for settings in Infrared communication mode:

```
typedef struct st_sci_irda
{
    uint32_t    baud_rate;      // ie 9600, 19200, 115200 (valid for internal
clock)
    uint8_t     clk_out_width;  // Setting value for the high pulse output width
of the IrDA IRTXD pin
    uint8_t     int_priority;   // txi, tei, rxi, eri INT priority; 1=low,
15=high
} sci_irda_t;
```

The following shows the definitions of the structure (sci_irda_t) members used in Infrared communication mode:

```
/* Definitions for the clk_out_width member. */
#define SCI_IRDA_OUT_WIDTH_3_16  (0x00U)
#define SCI_IRDA_OUT_WIDTH_2     (0x01U)
#define SCI_IRDA_OUT_WIDTH_4     (0x02U)
#define SCI_IRDA_OUT_WIDTH_8     (0x03U)
#define SCI_IRDA_OUT_WIDTH_16    (0x04U)
#define SCI_IRDA_OUT_WIDTH_32    (0x05U)
#define SCI_IRDA_OUT_WIDTH_64    (0x06U)
#define SCI_IRDA_OUT_WIDTH_128   (0x07U)
```

Return Values

[SCI_SUCCESS]	*/* Successful; channel initialized */*
[SCI_ERR_BAD_CHAN]	*/* Channel number is invalid for part*/*
[SCI_ERR_OMITTED_CHAN]	*/* Corresponding SCI_CHx_INCLUDED is invalid (0) */*
[SCI_ERR_CH_NOT_CLOSED]	*/* Channel currently in operation; Perform R_SCI_Close() first*/*
[SCI_ERR_BAD_MODE]	*/* Mode specified not currently supported*/*
[SCI_ERR_NULL_PTR]	*/* p_cfg pointer is NULL*/*
[SCI_ERR_INVALID_ARG]	*/* An element of the p_cfg structure contains an invalid value. */*

[SCI_ERR_QUEUE_UNAVAILABLE] */* Cannot open transmit or receive queue or both (Asynchronous mode) */*

Properties
Prototyped in file “r_sci_rx_if.h”

Description
Initializes an SCI channel for a particular mode and provides a Handle in **p_hdl* for use with other API functions. RXI and ERI interrupts are enabled in all modes. TXI is enabled in Asynchronous mode.

Example: Asynchronous Mode
```
    sci_cfg_t   config;
    sci_hdl_t   Console;
    sci_err_t   err;

    config.async.baud_rate = 115200;
    config.async.clk_src = SCI_CLK_INT;
    config.async.data_size = SCI_DATA_8BIT;
    config.async.parity_en = SCI_PARITY_OFF;
    config.async.parity_type = SCI_EVEN_PARITY;   // ignored because parity is
disabled
    config.async.stop_bits = SCI_STOPBITS_1;
    config.async.int_priority = 2;                // 1=lowest, 15=highest

    err = R_SCI_Open(SCI_CH1, SCI_MODE_ASYNC, &config, MyCallback, &Console);
```

Example: SSPI Mode
```
    sci_cfg_t   config;
    sci_hdl_t   sspiHandle;
    sci_err_t   err;

    config.sspi.spi_mode = SCI_SPI_MODE_0;
    config.sspi.bit_rate = 1000000;           // 1 Mbps
    config.sspi.msb_first = true;
    config.sspi.invert_data = false;
    config.sspi.int_priority = 4;
    err = R_SCI_Open(SCI_CH12, SCI_MODE_SSPI, &config, sspiCallback,
&sspiHandle);
```

Example: Synchronous Mode
```
    sci_cfg_t   config;
    sci_hdl_t   syncHandle;
    sci_err_t   err;

    config.sync.spi_mode = SCI_SPI_MODE_OFF;
    config.sync.bit_rate = 1000000;           // 1 Mbps
    config.sync.msb_first = true;
    config.sync.invert_data = false;
    config.sync.int_priority = 4;
    err = R_SCI_Open(SCI_CH12, SCI_MODE_SYNC, &config, syncCallback,
&syncHandle);
```

Example: Infrared Data Communication Mode
```
    sci_cfg_t   config;
    sci_hdl_t   Console;
    sci_err_t   err;

    config.irda.baud_rate = 115200;
    config.irda.clk_src = SCI_IRDA_OUT_WIDTH_3_16;
```

```
    config.irda.int_priority = 2;                // 1=lowest, 15=highest

    err = R_SCI_Open(SCI_CH5, SCI_MODE_IRDA, &config, irdaCallback, &Console);
```

Special Notes:
The driver calculates the optimum values for BRR, SEMR.ABCS, and SMR.CKS using BSP_PCLKA_HZ and BSP_PCLKB_HZ as defined in mcu_info.h of the board support package. This however does not guarantee a low bit error rate for all peripheral clock/baud rate combinations.

If an external clock is used in Asynchronous mode, the pin direction must be selected before calling the R_SCI_Open() function, and the pin function and mode must be selected after calling the R_SCI_Open() function. The following is an example initialization for RX111 channel 1:

```
Before the R_SCI_Open() function call

    PORT1.PDR.BIT.B7 = 0;       // set SCK pin direction to input (dflt)

After the R_SCI_Open() function call

MPC.P17PFS.BYTE = 0x0A;     // Pin Func Select P17 SCK1
    PORT1.PMR.BIT.B7 = 1;       // set SCK pin mode to peripheral
```

For settings of the pins used for communications, the pin directions and their outputs must be selected before calling the R_SCI_Open() function, and the pin functions and modes must be selected after calling the R_SCI_Open() function.
An example for initializing channel 6 for SSPI on the RX64M is as follows:

```
Before the R_SCI_Open() function call

    PORT0.PODR.BIT.B2 = 0;      // set line low
    PORT0.PODR.BIT.B0 = 0;      // set line low
    PORT0.PDR.BIT.B2 = 1;       // set clock pin direction to output
    PORT0.PDR.BIT.B0 = 1;       // set MOSI pin direction to output
    PORT0.PDR.BIT.B1 = 0;       // set MISO pin direction to input

After the R_SCI_Open() function call

    MPC.P00PFS.BYTE = 0x0A;     // Pin Func Select P00 MOSI
    MPC.P01PFS.BYTE = 0x0A;     // Pin Func Select P01 MISO
    MPC.P02PFS.BYTE = 0x0A;     // Pin Func Select P02 SCK
    PORT0.PMR.BIT.B0 = 1;       // set MOSI pin mode to peripheral
    PORT0.PMR.BIT.B1 = 1;       // set MISO pin mode to peripheral
    PORT0.PMR.BIT.B2 = 1;       // set clock pin mode to peripheral
```

When using Asynchronous mode, two byte queues are used for one channel. Adjust the number of byte queues as necessary. Refer to the application note "BYTEQ Module Using Firmware Integration Technology (R01AN1683)" for details.

R_SCI_Close()

This function removes power from the SCI channel and disables the associated interrupts.

Format

```
sci_err_t       R_SCI_Close (
                sci_hdl_t const   hdl
)
```

Parameters

sci_hdl_t const hdl

Handle for channel
Set *hdl* when R_SCI_Open() is successfully processed.

Return Values

[SCI_SUCCESS] */* Successful; channel closed */*
[SCI_ERR_NULL_PTR] */* hdl is NULL */*

Properties

Prototyped in file “r_sci_rx_if.h”

Description

Disables the SCI channel designated by the handle and enters module-stop state.

Example

```
sci_hdl_t   Console;
  ...
err = R_SCI_Open(SCI_CH1, SCI_MODE_ASYNC, &config, MyCallback, &Console);
  ...
err = R_SCI_Close(Console);
```

Special Notes:

This function will abort any transmission or reception that may be in progress.

R_SCI_Send()

Initiates transmit if transmitter is not in use. Queues data for later transmit when in Asynchronous mode.

Format

```
sci_err_t       R_SCI_Send (
                sci_hdl_t const   hdl,
                uint8_t           *p_src,
                uint16_t const    length
)
```

Parameters

sci_hdl_t const hdl
Handle for channel
Set *hdl* when R_SCI_Open() is successfully processed.

uint8_t p_src*
Pointer to data to transmit

uint16_t const length
Number of bytes to send

Return Values

[SCI_SUCCESS]	*/* Transmit initiated or loaded into queue (Asynchronous) */*
[SCI_ERR_NULL_PTR]	*/* hdl value is NULL */*
[SCI_ERR_BAD_MODE]	*/* Mode specified not currently supported */*
[SCI_ERR_INSUFFICIENT_SPACE]	*/* Insufficient space in queue to load all data (Asynchronous) */*
[SCI_ERR_XCVR_BUSY]	*/* Channel currently busy (SSPI/Synchronous/Asynchronous) */*

Properties

Prototyped in file “r_sci_rx_if.h”

Description

In asynchronous mode, this function places data into a transmit queue if the transmitter for the SCI channel referenced by the handle is not in use. When circular buffer (SCI_CFG_USE_CIRCULAR_BUFFER (1)) is used, the function allows data to be put on a transmit queue during transmission.

In SSPI and Synchronous modes, no data is queued and transmission begins immediately if the transceiver is not already in use.

Note that the toggling of Slave Select lines when in SSPI mode is not handled by this driver. The Slave Select line for the target device must be enabled prior to calling this function.

Also, toggling of the CTS/RTS pin in Synchronous/Asynchronous mode is not handled by this driver.

Example: Asynchronous Mode

```
    #define STR_CMD_PROMPT "Enter Command: "
    sci_hdl_t Console;
    sci_err_t err;

    err = R_SCI_Send(Console, STR_CMD_PROMPT, sizeof(STR_CMD_PROMPT));

    // Cannot block for this transfer to complete. However, can use TEI
interrupt
    // to determine when there is no more data in queue left to transmit.
```

Example: SSPI Mode

```
    sci_hdl_t  sspiHandle;
    sci_err_t  err;
    uint8_t    flash_cmd,sspi_buf[10];

    // SEND COMMAND TO FLASH DEVICE TO PROVIDE ID */
    FLASH_SS = SS_ON;               // enable gpio flash slave select
    flash_cmd = SF_CMD_READ_ID;

    R_SCI_Send(sspiHandle, &flash_cmd, 1);
    while (SCI_SUCCESS != R_SCI_Control(sspiHandle, SCI_CMD_CHECK_XFER_DONE,
NULL))
    {
    }

    /* READ ID FROM FLASH DEVICE */
    R_SCI_Receive(sspiHandle, sspi_buf, 5);
    while (SCI_SUCCESS != R_SCI_Control(sspiHandle, SCI_CMD_CHECK_XFER_DONE,
NULL))
    {
    }

    FLASH_SS = SS_OFF;              // disable gpio flash slave select
```

Example: Synchronous Mode

```
    #define STRING1 "Test String"
    sci_hdl_t  lcdHandle;
    sci_err_t  err;


    // SEND STRING TO LCD DISPLAY AND WAIT TO COMPLETE */
    R_SCI_Send(lcdHandle, STRING1, sizeof(STRING1));

    while (SCI_SUCCESS != R_SCI_Control(lcdHandle, SCI_CMD_CHECK_XFER_DONE,
NULL))
    {
    }
```

Example: Infrared Data Communication Mode

```
    #define ONETIME_SEND_SIZE 16

    sci_hdl_t Console;
    uint8_t data_send_buf[ONETIME_SEND_SIZE] =
{80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95};

    void main(void)
    {
      sci_err_t err;
      sci_cfg_t config;
      uint16_t cnt;

      config.irda.baud_rate = 115200;
      config.irda.clk_out_width = SCI_IRDA_OUT_WIDTH_3_16;
      config.irda.int_priority = 2; /* 1=lowest, 15=highest */
      err = R_SCI_Open(SCI_CH5, SCI_MODE_IRDA, &config, irdaCallback, &Console);
      if (SCI_SUCCESS != err)
      {
            while(1) { };
      }
```

```
      /* Get the size of the send buffer, if there is free space, passing the
transmitted data. */
      R_SCI_Control(Console, SCI_CMD_TX_Q_BYTES_FREE, (void *)&cnt);
      if (cnt - ONETIME_SEND_SIZE > 0)
      {
     /* Pass the transmitted data. If transmission idle and starts transmission.
*/
            err = R_SCI_Send(Console, &data_send_buf[0], ONETIME_SEND_SIZE);
            if (SCI_SUCCESS != err)
            {
                  while(1) { };
            }
      }
    }
```

Special Notes:
None.

R_SCI_Receive()

In Asynchronous mode, fetches data from a queue which is filled by RXI interrupts. In other modes, initiates reception if transceiver is not in use.

Format

```
sci_err_t       R_SCI_Receive (
                sci_hdl_t const   hdl,
                uint8_t           *p_dst,
                uint16_t const    length
)
```

Parameters

sci_hdl_t const hdl
Handle for channel
Set *hdl* when R_SCI_Open() is successfully processed.

uint8_t p_dst*
Pointer to buffer to load data into

uint16_t const length
Number of bytes to read

Return Values

```
[SCI_SUCCESS]                /* Requested number of bytes were loaded into p_dst
                                (Asynchronous) Clocking in of data initiated (SSPI/Synchronous) */
[SCI_ERR_NULL_PTR]           /* hdl value is NULL */
[SCI_ERR_BAD_MODE]           /* Mode specified not currently supported */
[SCI_ERR_INSUFFICIENT_DATA]  /* Insufficient data in receive queue to fetch all data (Asynchronous) */
[SCI_ERR_XCVR_BUSY]          /* Channel currently busy (SSPI/Synchronous) */
```

Properties

Prototyped in file “r_sci_rx_if.h”

Description

In Asynchronous mode, this function gets data received on an SCI channel referenced by the handle from its receive queue. This function will not block if the requested number of bytes is not available. In SSPI/Synchronous modes, the clocking in of data begins immediately if the transceiver is not already in use. The value assigned to SCI_CFG_DUMMY_TX_BYTE in r_sci_config.h is clocked out while the receive data is being clocked in.

If any errors occurred during reception, the callback function specified in R_SCI_Open() is executed. Check an event passed with the argument of the callback function to see if the reception has been successfully completed. Refer to 2.12, Callback Function for details.

Note that the toggling of Slave Select lines when in SSPI mode is not handled by this driver. The Slave Select line for the target device must be enabled prior to calling this function.

Example: Asynchronous Mode

```
sci_hdl_t Console;
sci_err_t err;
uint8_t   byte;
```

```
    /* echo characters */
    while (1)
    {
        while (SCI_SUCCESS != R_SCI_Receive(Console, &byte, 1))
        {
        }
        R_SCI_Send(Console, &byte, 1);
    }
```

Example: SSPI Mode

```
    sci_hdl_t  sspiHandle;
    sci_err_t  err;
    uint8_t    flash_cmd,sspi_buf[10];


    // SEND COMMAND TO FLASH DEVICE TO PROVIDE ID */

    FLASH_SS = SS_ON;               // enable gpio flash slave select
    flash_cmd = SF_CMD_READ_ID;

    R_SCI_Send(sspiHandle, &flash_cmd, 1);
    while (SCI_SUCCESS != R_SCI_Control(sspiHandle, SCI_CMD_CHECK_XFER_DONE,
NULL))
    {
    }

    /* READ ID FROM FLASH DEVICE */
    R_SCI_Receive(sspiHandle, sspi_buf, 5);
    while (SCI_SUCCESS != R_SCI_Control(sspiHandle, SCI_CMD_CHECK_XFER_DONE,
NULL))
    {
    }

    FLASH_SS = SS_OFF;              // disable gpio flash slave select
```

Example: Synchronous Mode

```
    sci_hdl_t  sensorHandle;
    sci_err_t  err;
    uint8_t    sensor_cmd,sync_buf[10];


    // SEND COMMAND TO SENSOR TO PROVIDE CURRENT READING */

    sensor_cmd = SNS_CMD_READ_LEVEL;

    R_SCI_Send(sensorHandle, &sensor_cmd, 1);
    while (SCI_SUCCESS != R_SCI_Control(sensorHandle, SCI_CMD_CHECK_XFER_DONE,
NULL))
    {
    }

    /* READ LEVEL FROM SENSOR */
    R_SCI_Receive(sensorHandle, sync_buf, 4);
    while (SCI_SUCCESS != R_SCI_Control(sensorHandle, SCI_CMD_CHECK_XFER_DONE,
NULL))
    {
    }
```

Example: Infrared Data Communication Mode

```
    sci_hdl_t Console;
    uint8_t data_recv_buf[80];
```

```
void main(void)
{
  sci_err_t err;
  sci_cfg_t config;
  uint16_t cnt;

  config.irda.baud_rate = 115200;
  config.irda.clk_out_width = SCI_IRDA_OUT_WIDTH_3_16;
  config.irda.int_priority = 2; /* 1=lowest, 15=highest */
  err = R_SCI_Open(SCI_CH5, SCI_MODE_IRDA, &config, irdaCallback, &Console);

  if (SCI_SUCCESS != err)
  {
        while(1) { };
  }
  /* Whether the buffer is receiving data, I want to check. */
  R_SCI_Control(Console, SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ, (void  *)&cnt);
  if (0 != cnt)
  {
        /* Retrieve the data of the size stored. */
        err = R_SCI_Receive(Console,&data_recv_buf[cnt_data],cnt);
        if (SCI_SUCCESS != err)
        {
              while(1) { };
        }
  }
}
```

Special Notes:
See section 2.12 Callback Function for values passed to arguments of the callback function.
In Asynchronous mode, when data match detected, received data stored in a queue and notify to user by callback function with event SCI_EVT_RX_CHAR_MATCH.

R_SCI_SendReceive()

For Synchronous and SSPI modes only. Transmits and receives data simultaneously if the transceiver is not in use.

Format

```
sci_err_t       R_SCI_SendReceive (
                sci_hdl_t const   hdl,
                uint8_t           *p_src,
                uint8_t           *p_dst,
                uint16_t const    length
)
```

Parameters

sci_hdl_t const hdl
Handle for channel
Set *hdl* when R_SCI_Open() is successfully processed.

uint8_t p_src*
Pointer to data to transmit

uint8_t p_dst*
Pointer to buffer to load data into

uint16_t const length
Number of bytes to send

Return Values

[SCI_SUCCESS]	*/* Data transfer initiated */*
[SCI_ERR_NULL_PTR]	*/* hdl value is NULL */*
[SCI_ERR_BAD_MODE]	*/* Channel mode not SSPI or Synchronous */*
[SCI_ERR_XCVR_BUSY]	*/* Channel currently busy */*

Properties

Prototyped in file “r_sci_rx_if.h”

Description

If the transceiver is not in use, this function clocks out data from the p_src buffer while simultaneously clocking in data and placing it in the p_dst buffer.

Note that the toggling of Slave Select lines for SSPI is not handled by this driver. The Slave Select line for the target device must be enabled prior to calling this function.

Also, toggling of the CTS/RTS pin in Synchronous/Asynchronous mode is not handled by this driver.

Example: SSPI Mode

```
    sci_hdl_t  sspiHandle;
    sci_err_t  err;
    uint8_t in_buf[2] = {0x55, 0x55};     // init to illegal values


    /* READ FLASH STATUS USING SINGLE API CALL */

    // load array with command to send plus one dummy byte for clocking in
status reply
```

```
    uint8_t out_buf[2] = {SF_CMD_READ_STATUS_REG, SCI_CFG_DUMMY_TX_BYTE };

    FLASH_SS = SS_ON;

    err = R_SCI_SendReceive(sspiHandle, out_buf, in_buf, 2);
    while (SCI_SUCCESS != R_SCI_Control(sspiHandle, SCI_CMD_CHECK_XFER_DONE,
NULL))
    {
    }

    FLASH_SS = SS_OFF;

    // in_buf[1] contains status
```

Special Notes:
See section 2.12 Callback Function for values passed to arguments of the callback function.

R_SCI_Control()

This function configures and controls the operating mode for the SCI channel.

Format

```
sci_err_t       R_SCI_Control (
                sci_hdl_t const     hdl,
                sci_cmd_t const     cmd,
                void                *p_args
)
```

Parameters

sci_hdl_t const hdl
Handle for channel
Set *hdl* when R_SCI_Open() is successfully processed.

sci_cmd_t const cmd
Command to run (see enumeration below)

*void *p_args*
Pointer to arguments (see below) specific to command, casted to void *

The valid *cmd* values are as follows:

```
typedef enum e_sci_cmd          // SCI Control() commands
{
      /* All modes */
          SCI_CMD_CHANGE_BAUD,              /* change baud/bit rate */
      #if ((SCI_CFG_CH7_FIFO_INCLUDED) || (SCI_CFG_CH8_FIFO_INCLUDED) ||
      (SCI_CFG_CH9_FIFO_INCLUDED)  || (SCI_CFG_CH10_FIFO_INCLUDED) ||
      (SCI_CFG_CH11_FIFO_INCLUDED))
          SCI_CMD_CHANGE_TX_FIFO_THRESH,    /* change TX FIFO threshold */
          SCI_CMD_CHANGE_RX_FIFO_THRESH,    /* change RX FIFO threshold */
      #endif
      #if defined(BSP_MCU_RX64M) || defined(BSP_MCU_RX71M) ||
      defined(BSP_MCU_RX65N) || defined(BSP_MCU_RX66T) || defined(BSP_MCU_RX72T)
      || defined(BSP_MCU_RX72M) || defined(BSP_MCU_RX72N) ||
      defined(BSP_MCU_RX66N)|| defined(BSP_MCU_RX671)
          SCI_CMD_SET_RXI_PRIORITY,         /* change RXI priority level */
          SCI_CMD_SET_TXI_PRIORITY,         /* change TXI priority level */
      #endif

          /* Async commands */
          SCI_CMD_EN_NOISE_CANCEL,          /* enable noise cancellation */
          SCI_CMD_EN_TEI,                   /* SCI_CMD_EN_TEI is obsolete
      command,
                                               but it exists only for
      compatibility with older version. */
          SCI_CMD_OUTPUT_BAUD_CLK,          /* output baud clock on the SCK pin
      */
          SCI_CMD_START_BIT_EDGE,           /* detect start bit as falling edge
      of RXDn pin
                                               (default detect as low level on
      RXDn pin) */
          SCI_CMD_GENERATE_BREAK,           /* generate break condition */
          SCI_CMD_COMPARE_RECEIVED_DATA,    /* Compare received data with
      comparison data */
```

```
        /* Async/IrDA commands */
        SCI_CMD_TX_Q_FLUSH,               /* flush transmit queue */
        SCI_CMD_RX_Q_FLUSH,               /* flush receive queue */
        SCI_CMD_TX_Q_BYTES_FREE,          /* get count of unused transmit
    queue bytes */
        SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ, /* get num bytes ready for reading
    */

        /* Async/Sync commands */
        SCI_CMD_EN_CTS_IN,                /* enable CTS input (default RTS
    output) */

        /* SSPI/Sync commands */
        SCI_CMD_CHECK_XFER_DONE,          /* see if send, rcv, or both are
    done; SCI_SUCCESS if yes */
        SCI_CMD_ABORT_XFER,
        SCI_CMD_XFER_LSB_FIRST,           /* start from LSB bit when sending
    */
        SCI_CMD_XFER_MSB_FIRST,           /* start from MSB bit when sending
    */
        SCI_CMD_INVERT_DATA,              /* logic level of send/receive data
    is invert */

        /* SSPI commands */
        SCI_CMD_CHANGE_SPI_MODE,           /* change clock polarity and phase
    in SSPI mode */
        SCI_CMD_CHECK_TX_DONE,           /* see if tx requests complete;
    SCI_SUCCESS if yes */
        SCI_CMD_CHECK_RX_DONE,            /* see if rx request complete in
    sync mode; SCI_SUCCESS if yes */
        SCI_CMD_CHECK_RX_SYNC_DONE,

    /*Sampling/transition timing adjust commands*/
    SCI_CMD_RX_SAMPLING_ENABLE,
    SCI_CMD_RX_SAMPLING_DISABLE,
    SCI_CMD_TX_TRANSITION_TIMING_ENABLE,
    SCI_CMD_TX_TRANSITION_TIMING_DISABLE,
    SCI_CMD_SAMPLING_TIMING_ADJUST,
    SCI_CMD_TRANSITION_TIMING_ADJUST
} sci_cmd_t;
```

Commands other than the following command do not require arguments and take FIT_NO_PTR for p_args.

The argument for SCI_CMD_CHANGE_BAUD is a pointer to the sci_baud_t variable containing the new bit rate desired. The sci_baud_t structure is shown below.

```
typedef struct st_sci_baud
{
    uint32_t    pclk;       // peripheral clock speed; e.g. 24000000 is 24 MHz
    uint32_t    rate;       // e.g. 9600, 19200, 115200
} sci_baud_t;
```

The argument for SCI_CMD_TX_Q_BYTES_FREE and SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ is a pointer to a uint16_t variable to hold a count value.

The argument for SCI_CMD_CHANGE_SPI_MODE is a pointer to the enumeration (sci_sync_sspi_t) variable containing the new mode desired.

The argument for SCI_CMD_SET_TXI_PRIORITY and SCI_CMD_SET_RXI_PRIORITY (for MCU which can specify different priority levels for TXI and RXI) is a pointer to a uint8_t variable to hold the priority level.

Return Values

[SCI_SUCCESS] */* Successful; channel initialized */*
[SCI_ERR_NULL_PTR] */* hdl or p_args pointer is NULL (when required) */*
[SCI_ERR_BAD_MODE] */* Mode specified not currently supported */*
[SCI_ERR_INVALID_ARG] */* The cmd value or an element of p_args contains an invalid value. */*

Properties

Prototyped in file “r_sci_rx_if.h”

Description

This function is used for configuring special hardware features such as changing driver configuration and obtaining driver status.

The CTS/ RTS pin functions as RTS by default hardware control. By issuing an SCI_CMD_EN_CTS_IN, the pin functions as CTS.

Example: Asynchronous Mode

```
    sci_hdl_t   Console;
    sci_cfg_t   config;
    sci_baud_t  baud;
    sci_err_t   err;
    uint16_t    cnt;

    R_SCI_Open(SCI_CH1, SCI_MODE_ASYNC, &config, MyCallback, &Console);
    R_SCI_Control(Console, SCI_CMD_EN_NOISE_CANCEL, NULL);
    R_SCI_Control(Console, SCI_CMD_EN_TEI, NULL);
      ...
    /* reset baud rate due to low power mode clock switching */
    baud.pclk = 8000000;      // 8 MHz
    baud.rate = 19200;
    R_SCI_Control(Console, SCI_CMD_CHANGE_BAUD, (void *)&baud);
      ...
    /* after sending several messages, determine how much space is left in tx
queue */
    R_SCI_Control(Console, SCI_CMD_TX_Q_BYTES_FREE, (void *)&cnt);
      ...
    /* check to see if there is data sitting in the receive queue */
    R_SCI_Control(Console, SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ, (void *)&cnt);
```

Example: SSPI Mode

```
    sci_cfg_t   config;
    sci_spi_mode_t   mode;
    sci_hdl_t   sspiHandle;
    sci_err_t   err;

    config.sspi.spi_mode     = SCI_SPI_MODE_0;
    config.sspi.bit_rate     = 1000000;           // 1 Mbps
    config.sspi.msb_first    = true;
    config.sspi.invert_data  = false;
    config.sspi.int_priority = 4;
    err = R_SCI_Open(SCI_CH12, SCI_MODE_SSPI, &config, sspiCallback,
&sspiHandle);
      ...
      ...
    // for changing to slave device which operates in a different mode
    mode = SCI_SPI_MODE_3;
    R_SCI_Control(sspiHandle, SCI_CMD_CHANGE_SPI_MODE, (void *)&mode);
```

Special Notes:
When SCI_CMD_CHANGE_BAUD is used, the optimum values for BRR, SEMR.ABCS, and SMR.CKS is calculated based on the bit rate specified. This however does not guarantee a low bit error rate for all peripheral clock/baud rate combinations.

If the command SCI_CMD_EN_CTS_IN is to be used, the pin direction must be selected before calling the R_SCI_Open() function, and the pin function and mode must be selected after calling the R_SCI_Open() function. The following is an example initialization for RX111 channel 1:

```
Before the R_SCI_Open() function call

PORT1.PDR.BIT.B4 = 0;       // set CTS/RTS pin direction to input (dflt)

After the R_SCI_Open() function call

MPC.P14PFS.BYTE = 0x0B;     // Pin Func Select P14 CTS
PORT1.PMR.BIT.B4 = 1;       // set CTS/RTS pin mode to peripheral
```

If the command SCI_CMD_OUTPUT_BAUD_CLK is to be used, the pin direction must be selected before calling the R_SCI_Open() function, and the pin function and mode must be selected after calling the R_SCI_Open() function.
The following is an example initialization for RX111 channel 1:

```
Before the R_SCI_Open() function call

    PORT1.PDR.BIT.B7 = 1;       // set SCK pin direction to output

After the R_SCI_Open() function call

    MPC.P17PFS.BYTE = 0x0A;     // Pin Func Select P17 SCK1
    PORT1.PMR.BIT.B7 = 1;       // set SCK pin mode to peripheral
```

The commands listed below can be executed during transmission. Do not execute the other commands during transmission.

- SCI_CMD_TX_Q_BYTES_FREE
- SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ
- SCI_CMD_CHECK_XFER_DONE
- SCI_CMD_ABORT_XFER

When this function is executed, the TXD pin temporarily becomes Hi-Z. Use any of the following methods to prevent the TXD pin from becoming Hi-Z.

When the SCI_CMD_GENERATE_BREAK command is used:

- Connect the TXD pin to Vcc via a resistor (pull-up).

When a command other than above is used:

Perform one of the following methods:

- Connect the TXD pin to Vcc via a resistor (pull-up).
- Switch the pin function of the TXD pin to general I/O port before the SCI_Control function is executed. Then switch it back to peripheral function after the SCI_Control function has been executed.

R_SCI_GetVersion()

This function returns the driver version number at runtime.

Format
uint32_t R_SCI_GetVersion (void)

Parameters
None

Return Values
Version number.

Properties
Prototyped in file “r_sci_rx_if.h”

Description
Returns the version of this module. The version number is encoded such that the top 2 bytes are the major version number and the bottom 2 bytes are the minor version number.

Example
```
uint32_t   version;
  ...
version = R_SCI_GetVersion();
```

Special Notes:
None.

4. Pin Setting

To use the SCI FIT module, assign input/output signals of the peripheral function to pins with the multi-function pin controller (MPC). The pin assignment is referred to as the “Pin Setting” in this document.

Please perform the pin setting after calling the R_SCI_Open function.

When performing the pin setting in the e^2 studio, the Pin Setting feature of the FIT Configurator or the Smart Configurator can be used. When using the Pin Setting feature, a source file is generated according to the option selected in the Pin Setting window in the FIT Configurator or the Smart Configurator. Then pins are configured by calling the function defined in the source file. Refer to Table 4.1 Function Output by the FIT Configurator for details.

Table 4.1 Function Output by the FIT Configurator

MCU Used	Function to be Output	Remarks
All MCUs	R_SCI_PinSet_SCIx	x: Channel number

5. Demo Projects

Demo projects include function main() that utilizes the FIT module and its dependent modules (e.g. r_bsp). This FIT module includes the following demo projects.

5.1 sci_demo_rskrx113, sci_demo_rskrx113_gcc

This is a simple demo of the RX113 Serial Communications Interface (SCI) for the RSKRX113 starter kit (FIT module "r_sci_rx"). In the demo project, the MCU communicates with the terminal through the SCI channel configured as the UART. The RS232 interface is not on the RSKRX113 in the demo, thus the USB virtual COM interface is used as serial interface for RSKRX113. A PC running the terminal emulation application is required for communicating with the user.

Setup and Execution

1. Prepare jumpers for the RSKRX113 board. Mount J15 jumper between 1 and 2, and J16 jumper between 2 and 3.
2. Build this sample application, download it to the RSK board, and execute the application using a debugger.
3. Connect the serial port on the RSK board to the serial port on the PC.

   This demo project uses the USB virtual COM interface. In this case, connect the serial port to the USB port on the PC where the Renesas USB serial device driver is installed.
4. Open the terminal emulation program on the PC and select the serial COM port allocated to the USB serial virtual COM interface on the RSK.
5. Configure the terminal serial settings so that they correspond to the settings in this sample application listed below:
   115200 bps, 8-bit data, no parity, 1-stop bit, no flow control
6. The software waits for receiving characters from the terminal.
   When the terminal program on the PC is ready, press a key on the keyboard in the PC's terminal window and check the version number of the FIT module output on the terminal.
7. This application is in echo mode. A given key input to the terminal is received by the SCI driver and then the application returns the characters to the terminal.

Boards Supported

RSKRX113

5.2 sci_demo_rskrx231, sci_demo_rskrx231_gcc

This is a simple demo of the RX231 Serial Communications Interface (SCI) for the RSKRX231 starter kit (FIT module "r_sci_rx"). In the demo project, the MCU communicates with the terminal through the SCI channel configured as the UART. The RS232 interface is not on the RSKRX231 in the demo, thus the USB virtual COM interface is used as serial interface for RSKRX231. A PC running the terminal emulation application is required for communicating with the user.

Setup and Execution

1. Build this sample application, download it to the RSK board, and execute the application using a debugger.
2. Connect the serial port on the RSK board to the serial port on the PC.

   This demo project uses the USB virtual COM interface. In this case, connect the serial port to the USB port on the PC where the Renesas USB serial device driver is installed.
3. Open the terminal emulation program on the PC and select the serial COM port allocated to the USB serial virtual COM interface on the RSK.
4. Configure the terminal serial settings so that they correspond to the settings in this sample application listed below:
   115200 bps, 8-bit data, no parity, 1-stop bit, no flow control
5. The software waits for receiving characters from the terminal.
   When the terminal program on the PC is ready, press a key on the keyboard in the PC's terminal window and check the version number of the FIT module output on the terminal.
6. This application is in echo mode. A given key input to the terminal is received by the SCI driver and then the application returns the characters to the terminal.

Boards Supported

RSKRX231

5.3 sci_demo_rskrx64m, sci_demo_rskrx64m_gcc

This is a simple demo of the RX64M Serial Communications Interface (SCI) for the RSKRX64M starter kit (FIT module "r_sci_rx"). In the demo project, the MCU communicates with the terminal through the SCI channel configured as the UART. The RS232 interface is not on the RSKRX64M in the demo, thus the USB virtual COM interface is used as serial interface for RSKRX64M. A PC running the terminal emulation application is required for communicating with the user.

Setup and Execution

1. Prepare jumpers for RSKRX64M board. Mount J16 and J18 jumpers between 2 and 3.
2. Build this sample application, download it to the RSK board, and execute the application using a debugger.
3. Connect the serial port on the RSK board to the serial port on the PC.

   This demo project uses the USB virtual COM interface. In this case, connect the serial port to the USB port on the PC where the Renesas USB serial device driver is installed.
4. Open the terminal emulation program on the PC and select the serial COM port allocated to the USB serial virtual COM interface on the RSK.
5. Configure the terminal serial settings so that they correspond to the settings in this sample application listed below:
   115200 bps, 8-bit data, no parity, 1-stop bit, no flow control
6. The software waits for receiving characters from the terminal.
   When the terminal program on the PC is ready, press a key on the keyboard in the PC's terminal window and check the version number of the FIT module output on the terminal.

7. This application is in echo mode. A given key input to the terminal is received by the SCI driver and then the application returns the characters to the terminal.

Boards Supported

RSKRX64M

5.4 sci_demo_rskrx71m, sci_demo_rskrx71m_gcc

This is a simple demo of the RX71M Serial Communications Interface (SCI) for the RSKRX71M starter kit (FIT module "r_sci_rx"). In the demo project, the MCU communicates with the terminal through the SCI channel configured as the UART. The RS232 interface is not on the RSKRX71M in the demo, thus the USB virtual COM interface is used as serial interface for RSKRX71M. A PC running the terminal emulation application is required for communicating with the user.

Setup and Execution

1. Prepare jumpers for RSKRX71M board. Mount J16 and J18 jumpers between 2 and 3.
2. Build this sample application, download it to the RSK board, and execute the application using a debugger.
3. Connect the serial port on the RSK board to the serial port on the PC.

   This demo program uses the USB virtual COM interface. In this case, connect the serial port to the USB port on the PC where the Renesas USB serial device driver is installed.
4. Open the terminal emulation program on the PC and select the serial COM port allocated to the USB serial virtual COM interface on the RSK.
5. Configure the terminal serial settings so that they correspond to the settings in this sample application listed below:
   115200 bps, 8-bit data, no parity, 1 stop bit, no flow control
6. The software waits for receiving characters from the terminal.
   When the terminal program on the PC is ready, press a key on the keyboard in the PC's terminal window and check the version number of the FIT module output on the terminal.
7. This application is in echo mode. A given key input to the terminal is received by the SCI driver and then the application returns the characters to the terminal.

Boards Supported

RSKRX71M

5.5 sci_demo_rskrx65n, sci_demo_rskrx65n_gcc

This is a simple demo of the RX65N Serial Communications Interface (SCI) for the RSKRX65N starter kit (FIT module "r_sci_rx"). In the demo project, the MCU communicates with the terminal through the SCI channel configured as the UART. The RS232 interface is not on the RSKRX65N in the demo, thus the USB virtual COM interface is used as serial interface for RSKRX65N. A PC running the terminal emulation application is required for communicating with the user.

Setup and Execution

1. Build this sample application, download it to the RSK board, and execute the application using a debugger.
2. Connect the serial port on the RSK board to the serial port on the PC.

   This demo program uses the USB virtual COM interface. In this case, connect the serial port to the USB port on the PC where the Renesas USB serial device driver is installed.
3. Open the terminal emulation program on the PC and select the serial COM port allocated to the USB serial virtual COM interface on the RSK.

4. Configure the terminal serial settings so that they correspond to the settings in this sample application listed below:
   115200 bps, 8-bit data, no parity, 1 stop bit, no flow control
5. The software waits for receiving characters from the terminal.
   When the terminal program on the PC is ready, press a key on the keyboard in the PC's terminal window and check the version number of the FIT module output on the terminal.
6. This application is in echo mode. A given key input to the terminal is received by the SCI driver and then the application returns the characters to the terminal.

Boards Supported

RSKRX65N

5.6 sci_demo_rskrx65n_2m, sci_demo_rskrx65n_2m_gcc

This is a simple demo of the RX65N-2MB Serial Communications Interface (SCI) for the RSKRX65N-2MB starter kit (FIT module "r_sci_rx"). In the demo project, the MCU communicates with the terminal through the SCI channel configured as the UART. The RS232 interface is not on the RSKRX65N-2MB in the demo, thus the USB virtual COM interface is used as serial interface for RSKRX65N-2MB. A PC running the terminal emulation application is required for communicating with the user.

Setup and Execution

1. Build this sample application, download it to the RSK board, and execute the application using a debugger.
2. Connect the serial port on the RSK board to the serial port on the PC.

   This demo program uses the USB virtual COM interface. In this case, connect the serial port to the USB port on the PC where the Renesas USB serial device driver is installed.
3. Open the terminal emulation program on the PC and select the serial COM port allocated to the USB serial virtual COM interface on the RSK.
4. Configure the terminal serial settings so that they correspond to the settings in this sample application listed below:
   115200 bps, 8-bit data, no parity, 1 stop bit, no flow control
5. The software waits for receiving characters from the terminal.
   When the terminal program on the PC is ready, press a key on the keyboard in the PC's terminal window and check the version number of the FIT module output on the terminal.
6. This application is in echo mode. A given key input to the terminal is received by the SCI driver and then the application returns the characters to the terminal.

Boards Supported

RSKRX65N-2MB

5.7 sci_demo_rskrx72m, sci_demo_rskrx72m_gcc

This is a simple demo of the RX72M Serial Communications Interface (SCI) for the RSKRX72M starter kit (FIT module "r_sci_rx"). In the demo project, the MCU communicates with the terminal through the SCI channel configured as the UART. The RS232 interface is not on the RSKRX72M in the demo, thus the USB virtual COM interface is used as serial interface for RSKRX72M. A PC running the terminal emulation application is required for communicating with the user.

Setup and Execution

1. Build this sample application, download it to the RSK board, and execute the application using a debugger.
2. Connect the serial port on the RSK board to the serial port on the PC.

This demo program uses the USB virtual COM interface. In this case, connect the serial port to the USB port on the PC where the Renesas USB serial device driver is installed.

3. Open the terminal emulation program on the PC and select the serial COM port allocated to the USB serial virtual COM interface on the RSK.
4. Configure the terminal serial settings so that they correspond to the settings in this sample application listed below:
   115200 bps, 8-bit data, no parity, 1 stop bit, no flow control
5. The software waits for receiving characters from the terminal.
   When the terminal program on the PC is ready, press a key on the keyboard in the PC's terminal window and check the version number of the FIT module output on the terminal.
6. This application is in echo mode. A given key input to the terminal is received by the SCI driver and then the application returns the characters to the terminal.

5.8 Adding a Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the distribution file for this application note. To add a demo project to a workspace, select *File* >> *Import* >> *General* >> *Existing Projects into Workspace*, then click "Next". From the Import Projects dialog, choose the "Select archive file" radio button. "Browse" to the FITDemos subdirectory, select the desired demo zip file, then click "Finish".

5.9 Downloading Demo Projects

Demo projects are not included in the RX Driver Package. When using the demo project, the FIT module needs to be downloaded. To download the FIT module, right click on this application note and select "Sample Code (download)" from the context menu in the *Smart Browser* >> *Application Notes* tab.

6. Appendices

6.1 Confirmed Operation Environment

This section describes confirmed operation environment for the SCI FIT module.

Table 6.1 Confirmed Operation Environment (Rev.3.90)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2021-07 IAR C/C++ Compiler for Renesas RX version 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202004 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -WI,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.20.3 Compiler option: The default settings of the integrated development environment.
Endian	Little endian/Big endian
Revision of the module	Rev.3.90
Board used	Renesas Starter Kit+ for RX140 (product No.: RTK5RX140xxxxxxxxx)

Table 6.2 Confirmed Operation Environment (Rev.3.80)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2021-07 IAR C/C++ Compiler for Renesas RX version 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202004 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -WI,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.20.3 Compiler option: The default settings of the integrated development environment.
Endian	Little endian/Big endian
Revision of the module	Rev.3.80
Board used	Renesas Starter Kit+ for RX671 (product No.: RTK55671xxxxxxxxxx)

Table 6.3 Confirmed Operation Environment (Rev.3.70)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.8.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.201904 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -WI,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Little endian
Revision of the module	Rev.3.70
Board used	Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx)

Table 6.4 Confirmed Operation Environment (Rev.3.60)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.8.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.201904 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -WI,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Little endian
Revision of the module	Rev.3.60
Board used	Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx) Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2CxxxxxBE) Renesas Solution Starter Kit for RX23W (product No.: RTK5523Wxxxxxxxxxx) Renesas Starter Kit+ for RX113 (product No.: R0K505113SxxxBE) Renesas Starter Kit+ for RX231 (product No.: R0K505231SxxxBE)

Table 6.5 Confirmed Operation Environment (Rev.3.50)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.7.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.201904 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -WI,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.3.50
Board used	Renesas Solution Starter Kit+ for RX23E-A (product No.: RTK0ESXBxxxxxxxxxx)

Table 6.6 Confirmed Operation Environment (Rev.3.40)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.7.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -WI,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.3.40
Board used	Renesas Starter Kit+ for RX72N (product No.: RTK5572Nxxxxxxxxxx)

Table 6.7 Confirmed Operation Environment (Rev.3.30)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.7.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -WI,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.3.30
Board used	RX13T CPU Card (product No.: RTK0EMXA10C00000BJ)

Table 6.8 Confirmed Operation Environment (Rev.3.20)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.5.0 IAR Embedded Workbench for Renesas RX 4.12.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -WI,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.3.20
Board used	Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx)

Table 6.9 Confirmed Operation Environment (Rev.3.10)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.5.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.3.10
Board used	Renesas Solution Starter Kit for RX23W (product No.: RTK5523Wxxxxxxxxxx)

Table 6.10 Confirmed Operation Environment (Rev.3.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.4.0 IAR Embedded Workbench for Renesas RX 4.10.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201803 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.10.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.3.00
Board used	Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565Nxxxxxxxxx)

Table 6.11 Confirmed Operation Environment (Rev.2.20)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.20
Board used	Renesas Starter Kit for RX72T (product No.: RTK5572Txxxxxxxxxx)

Table 6.12 Confirmed Operation Environment (Rev.2.11)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.11
Board used	Renesas Starter Kit for RX66T (product No.: RTK50566T0SxxxxxBE)

Table 6.13 Confirmed Operation Environment (Rev.2.10)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.10
Board used	Renesas Starter Kit for RX66T (product No.: RTK50566T0SxxxxxBE) Renesas Starter Kit+ for RX 65N-2MB (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308SxxxxxBE)

Table 6.14 Confirmed Operation Environment (Rev.2.01)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 6.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.01
Board used	Renesas Starter Kit+ for RX 65N-2MB (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308SxxxxxBE)

Table 6.15 Confirmed Operation Environment (Rev.2.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 5.4.0 (WS Patch)
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.00
Board used	Renesas Starter Kit+ for RX 65N-2MB (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308SxxxxxBE)

Table 6.16 Confirmed Operation Environment (Rev.1.90)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 5.3.0.023
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.06.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.1.90
Board used	Renesas Starter Kit+ for RX24U (product No.: RTK500524USxxxxxBE) Renesas Starter Kit+ for RX24T (product No.: RTK500524TSxxxBE) Renesas Starter Kit+ for RX113 (product No.: R0K505113SxxxBE) Renesas Starter Kit+ for RX65N (product No.: RTK500565NSxxxxxBE)

Table 6.17 Confirmed Operation Environment (Rev.1.80)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 5.0.1.005 Renesas Electronics e^2 studio Version 5.0.0.043 Renesas Electronics e^2 studio Version 4.3.0.007 Renesas Electronics e^2 studio Version 4.2.0.012
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.05.00 Renesas Electronics C/C++ Compiler Package for RX Family V2.04.01 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.1.80
Board used	Renesas Starter Kit+ for RX65N (product No.: RTK500565NSxxxxxBE) (1) Renesas Starter Kit+ for RX64M (product No.: R0K50564MSxxxBE) (2) Renesas Starter Kit+ for RX71M (product No.: R0K50571MSxxxBE) (3) Renesas Starter Kit+ for RX231 (product No.: R0K505231SxxxBE) (4) Renesas Starter Kit+ for RX130 (product No.: RTK5005130SxxxBE) (4) Renesas Starter Kit+ for RX111 (product No.: R0K505111SxxxBE) (4) Renesas Starter Kit+ for RX23T (product No.: RTK500523TSxxxBE) (4) Renesas Starter Kit+ for RX24T (product No.: RTK500524TSxxxBE) (4) Renesas Starter Kit+ for RX113 (product No.: R0K505113SxxxBE) (4) Renesas Starter Kit+ for RX210 (product No.: R0K505210SxxxBE) (4) Renesas Starter Kit+ for RX63N (product No.: R0K50563NSxxxBE) (4)

Note 1. Operation confirmed in e^2 studio Version 5.0.1.005 with C compiler V2.05.00.
Note 2. Operation confirmed in e^2 studio Version 4.3.0.007 with C compiler V2.04.01.
Note 3. Operation confirmed in e^2 studio Version 4.2.0.012 with C compiler V2.04.01.
Note 4. Operation confirmed in e^2 studio Version 5.0.0.043 with C compiler V2.04.01.

6.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file "platform.h".

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- Using CS+:

  Application note "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)"

- Using e^2 studio:

  Application note "Adding Firmware Integration Technology Modules to Projects (R01AN1723)"

When using this FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note "Board Support Package Module Using Firmware Integration Technology (R01AN1685)".

(2) Q: I have added the FIT module to the project and built it. Then I got the error: This MCU is not supported by the current r_sci_rx module.

A: The FIT module you added may not support the target device chosen in your project. Check the supported devices of added FIT modules.

(3) Q: I have added the FIT module to the project and built it. Then I got an error: ERROR - Unsupported channel chosen in r_sci_config.h.

A: The setting in the file "r_sci_rx_config.h" may be wrong. Check the file "r_sci_rx_config.h". If there is a wrong setting, set the correct value for that. Refer to 2.7, Configuration Overview for details.

(4) Q: Transmit data is not output from the TXD pin.

A: The pin setting may not be performed correctly. When using this FIT module, the pin setting must be performed. Refer to 4. "Pin Setting" for details.

7. Reference Documents

User's Manual: Hardware
The latest versions can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

User's Manual: Development Tools
RX Family C/C++ Compiler CC-RX User's Manual (R20UT3248)

The latest version can be downloaded from the Renesas Electronics website.

Related Technical Updates

This module reflects the content of the following technical updates.

TN-RX*-A151A/E

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Nov-15-2013	—	Initial Multi-Mode Release.
1.20	Apr-17-2014	1,3	Added mention of RX110 support.
1.30	Jul-02-2014	-	Fixed RX63N bug that prevented receive errors (Group12) from interrupting except on channel 2.
1.40	Dec-16-2014	1, 7	Added RX113 to list of supported devices. Added section 2.11 Code Size and RAM usage.
1.50	Mar-18-2015	1,3,5	Added RX64M, RX71M to list of supported devices.
1.60	Jun-30-2015	1,3,5	Added RX231 to list of supported devices.
1.70	Sep-30-2015	— 7 11 13 22 22	Added support for the RX23T Group. Updated information of 2.11 Code Size and RAM usage including code sizes in Table 2. Modified the setting procedure in the following sections: • Special Notes in R_SCI_Open(): - When an external clock is used in Asynchronous mode - For settings of the pins used for communications • Special Notes in R_SCI_Control(): - When the command SCI_CMD_EN_CTS_IN is to be used - When the command SCI_CMD_OUTPUT_BAUD_CLK is to be used
1.80	Oct-1-2016	— 3, 4 5 7 8, 9 10 11 11-13 14 18, 19, 21 25 29 30-33 34	Added support for the RX65N Group. Revised the contents in 1. Overview including new sections 1.1 and 1.2. Added the limitation in 2.4 Limitations. Updated the information in 2.5 Supported Toolchains. Added the definitions regarding FIFO as the configuration option in Table 2.1. Updated the table for ROM and RAM minimum sizes and the formula in 2.9 Code Size. Added section 2.10 Parameters. Moved section Return Values from 3.2 to 2.11. Added section 2.12 Callback Function. 3.1 R_SCI_Open() - Added some descriptions in the introduction of the function, and parameters p_callback and p_hdl. - Moved descriptions regarding the callback function to section 2.12. 3.2 R_SCI_Close(),3.3 R_SCI_Send(),3.4 R_SCI_Receive(), 3.5 R_SCI_SendReceive() - Added a description in the parameter hdl. 3.6 R_SCI_Control() - Added a description in the parameter hdl. - Added definitions regarding FIFO in the valid cmd values. Added section 4. Pin Setting. Added section 5. Demo Projects. Added the section for technical update information.

Rev.	Date	Description	
		Page	Summary
1.90	Feb-28-2017	—	Added support for the RX24U Group.
		3	Added RX24U to Table 1.1 SCI Peripheral Functions. Supported by MCU Groups.
		3,7,16	Deleted descriptions regarding usage of the SCI_CMD_EN_TEI command.
		4	Modified the description regarding the FIFO function in the Error Detection.
		4	Modified the description for the R_SCI_Send and R_SCI_Receive functions in Table 1.2 API Function List.
		5	Added RXC v2.06.00 to 2.5 Supported Toolchains. Updated memory sizes in 2.9 Code Size.
		8,9 11,12	2.12 Callback Function: - Modified the existing description and added the description regarding the FIFO function in the overview part. - Added the description of events such that a receive data is not stored in an argument of a callback function when these events occur.
		17, 18	Added the description for handling communication error when the FIFO function is enabled in the Special Notes in 3.1 R_SCI_Open() function. Modified the description in 3.3 R_SCI_Send().
		20 22	Modified the description regarding the callback function for when a receive error occurs in Description in 3.4 R_SCI_Receive().
		26	3.6 R_SCI_Control(): - Modified the description in the overview part. - Parameters: - Added SCI_CMD_SET_RXI_PRIORITY and SCI_CMD_SET_TXI_PRIORITY to commands. - Modified the comment for the SCI_CMD_EN_TEI command. - Added comments for commands that did not have comments in the previous version.
		28, 29	- Special Notes: - Added descriptions for executable commands during transmission. - Added the description regarding the TXD pin when using commands.
		Program	Corrected typo. Modified the SCI_CMD_EN_TEI command to perform no processing. (This command is not necessary anymore, however, kept for compatibility with old versions.) Modified the code to check arguments for both NULL and FIT_NO_PTR. Modified the R_SCI_Control function to return SCI_ERR_INVALID_ARG when the SCI_CMD_EN_CTS_IN command is specified in simple SPI mode. (CTS input is invalid in simple SPI mode.)

	Deleted an unnecessary logical operation before processing to clear an error flag in the sci_error function.

Rev.	Date	Description	
		Page	Summary
1.90	Feb-28-2017	Program	The following issue has been fixed. Target Device: RX110/RX111/RX113/RX130/RX210/RX230/RX231/RX23T/ RX24T/ RX63N/RX631/RX64M/RX651/RX65N/RX71M Description: In reception in clock synchronous mode, the number of data greater than the number of data specified may be received. Condition: In clock synchronous mode, when receiving 2-byte or longer data, time after first dummy data write before the counter is decremented for second dummy data write is 1 frame or longer. Measure: The sci_receive_sync_data function now performs dummy data write only once. (same as the specification in Rev. 1.70) Use Rev. 1.90 or later version of the SCI FIT module. The following issue has been fixed. Target Device: RX110/RX111/RX113/RX130/RX210/RX230/RX231/RX23T/ RX24T/RX63N/RX631/RX64M/RX651/RX65N/RX71M Description: When an error occurs in asynchronous mode, the error interrupt may repeatedly occur and the main processing may not operate correctly. Condition: Parity error, overrun error or framing error occurs when the callback function is not used in asynchronous mode. Measure: In the sci_error function, the error flag was only cleared when the callback function was used. Now the error flag is cleared in any case (same as the specification in Rev. 1.70). Use Rev. 1.90 or later version of the SCI FIT module.
2.00	Jul-24-2017	—	Added support for RX130-512KB and RX65N-2MB.
		1	Related Documents: Added the following document: “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)”
		6 to 10	2.6 Interrupt Vector: Added.
		16	2.13 Callback Function: Modified some description regarding FIFO. With FIFO enabled, the callback function is now called only once.
		18	2.14 Adding the FIT Module to Your Project: Revised.
		19	3.1 R_SCI_Open(): In Special Notes, added description regarding byte queue in Asynchronous mode.
		30	3.6 R_SCI_Control(): SCI_CMD_SET_RXI_PRIORITY and SCI_CMD_SET_TXI_PRIORITY commands can now be used in all modes.
		35	4. Pin Setting: Added the description of “Smart Configurator”.
		39	5.6 Downloading Demo Projects: Added.
		40 to 42	6. Appendices: Added.

Rev.	Date	Description	
		Page	Summary
2.00	Jul-24-2017	Program	The following issue has been fixed. Target Device: RX65N Description: The error flag is not cleared. Thus, an error interrupt occurs all the time. Condition: When opened with FIFO enabled and the callback function not provided, a receive error occurs. Measure: There was no processing to clear the receive error condition when FIFO is enabled. The processing has been added so that the receive error condition is always cleared before exiting an error interrupt handler whether the callback function is provided or not. Use Rev. 2.00 or later version of the SCI FIT module. The following issue has been fixed. Target Device: RX65N Description: When changing the threshold value for transmit/receive FIFO, if the argument is not specified, an unknown value is set to the threshold value. Condition: SCI_CMD_CHANGE_TX_FIFO_THRESH / SCI_CMD_CHANGE_RX_FIFO_THRESH is set as the command in the R_SCI_Control function, and NULL is set to the argument for these commands. Measure: Added NULL check processing for arguments to the R_SCI_Control function. Use Rev. 2.00 or later version of the SCI FIT module. The following issue has been fixed. Target Device: RX65N Description: If a transmission is restarted during transmission, the current transmission is canceled. Then, new transmission does not start. Condition: When FIFO is enabled, a transmission is started during transmission with the channel set as Synchronous mode. Measure: Modified processing. If a transmission is started during transmission, SCI_ERR_XCVR_BUSY is now returned, so the current transmission is not canceled. Use Rev. 2.00 or later version of the SCI FIT module.

Rev.	Date	Description Page	Summary
2.00	Jul-24-2017	Program	The following issue has been fixed. Target Device: RX65N Description: Even if the receive FIFO threshold value is changed, the threshold value becomes “8” after a reception is complete. Condition: When FIFO is enabled in Synchronous mode, a reception is performed with the receive FIFO threshold value set to a value other than the initial value (8). Measure: Modified the code to hold the changed threshold value for the transmit/receive FIFO in the handler. The threshold value is now restored with the value held in the handler instead of the initial value. Use Rev. 2.00 or later version of the SCI FIT module. The following issue has been fixed. Target Device: RX65N Description: Even if the number of bytes received exceeds the receive FIFO threshold value, the receive interrupt does not occur. Condition: With FIFO enabled in Synchronous mode, when the receive FIFO threshold value is changed to a value less than the initial value (8), the number of the received data is less than 8 bytes. Measure: Modified the code to hold the changed threshold value for the transmit/receive FIFO in the handler. The threshold value is now restored with the value held in the handler instead of the initial value. Use Rev. 2.00 or later version of the SCI FIT module. The following issue has been fixed. Target Device: RX65N Description: If the receive FIFO threshold value is 8, the callback function is executed eight times continuously after 8 bytes data are received. Condition: When opened with callback function and FIFO enabled, multiple bytes are received (this occurs even if the number of received bytes is less than 8 bytes). Measure: Modified the code to execute the callback function once per receive interrupt when FIFO is enabled. Also the member “num” which stores the number of bytes to be received has been added to the argument for the callback function. If the number of bytes to be received is greater than the receive buffer size, the data for the buffer size are stored and the rest of the data are discarded (the callback function event is “SCI_EVT_RXBUF_OVFL” for this case). Use Rev. 2.00 or later version of the SCI FIT module.

Rev.	Date	Description	
		Page	Summary
2.00	Jul-24-2017	Program	The following issue has been fixed. Target Device: RX64M/RX71M/RX65N Description: When the priority level for transmission/reception is changed, the priority level set becomes unknown. Condition: SCI_CMD_SET_TXI_PRIORITY / SCI_CMD_SET_RXI_ PRIORITY is set as the command in the R_SCI_Control function and NULL is set to the argument for these commands. Measure: Added NULL check processing for arguments and range check processing for the interrupt priority level in the R_SCI_Control function. Use Rev. 2.00 or later version of the SCI FIT module. The following issue has been fixed. Target Device: RX64M/RX71M/RX65N Description: The interrupt priority level can be changed only in Asynchronous mode. Condition: With Synchronous mode, SCI_CMD_SET_TXI_PRIORITY / SCI_CMD_SET_RXI_PRIORITY is set as the command in the R_SCI_Control function. Measure: Modified the code, so the interrupt priority level can be changed in both Synchronous and Asynchronous modes. Use Rev. 2.00 or later version of the SCI FIT module.
2.01	Oct 31, 2017	39 40 40 41	5.5 sci_demo_rskrx65n: Added 5.6 sci_demo_rskrx65n_2m: Added 5.8 Downloading Demo Projects: Added 6.1 Operation Confirmation Environment: Added Table for Rev.2.01
2.10	Sep 28, 2018	1,3 13 16 46	Added support for RX66T. Added configuration setting for RX66T Added code size corresponding to RX66T 6.1 Confirmed Operation Environment: Added Table for Rev 2.10
2.11	Nov 16, 2018	– 1, 3 46	Added document number in XML Added support for RX651. Changed Renesas Starter Kit Product No for RX66T. Added Table for Rev 2.11
2.20	Feb 01, 2019	Program 1,3,12,14 17,18 25-42 48	Added support for RX72T. Added support for RX72T. Added code size corresponding to RX72T Removed ‘Reentrant’ description in each API function. 6.1 Confirmed Operation Environment: Added Table for Rev 2.20.

3.00	May.20.19	—	Supported the following compilers: - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		1	Deleted the RX210, RX631, and RX63N in Target Devices for end of update these devices. Added the section of Target compilers. Deleted related documents.
		4	1.2 Deleted RX210, RX63N, RX631
		5	1.4 Deleted RX63N, RX631
		6	2.2 Software Requirements Requires r_bsp v5.20 or higher
		7	2.4 Deleted RX210, RX63N, RX631
		14-23	Updated the section of 2.8 Code Size
		53	Table 6.1 Confirmed Operation Environment: Added table for Rev.3.00
		58	Deleted the section of Website and Support.
		Program	Changed below for support GCC and IAR compiler: 1. Deleted the inline expansion of the R_SCI_GetVersion function. 2. Replaced evenaccess with the macro definition of BSP. 3. Replaced nop with the intrinsic functions of BSP. 4. Replaced the declaration of interrupt functions with the macro definition of BSP.
3.10	Jun.28.19	1, 4, 7, 8	Added support for RX23W
		15	Added code size corresponding to RX23W
		54	6.1 Confirmed Operation Environment: Added Table for Rev.3.10
		Program	Added support for RX23W.
3.20	Aug.15.19	1, 4, 12-16	Added support for RX72M
		20, 24, 28	Added code size corresponding to RX72M
		59	6.1 Confirmed Operation Environment: Added Table for Rev.3.20 Table 6.2: Corrected board name for RX23W
		Program	Added support for RX72M.
3.21	Sep.16.19	Program	Fixed issue in RX631/RX63N sci_initialize_ints().
3.30	Nov.25.19	1, 4, 7, 8	Added support for RX13T.
		6	Add new section 2.3 RAM Location Limitations.
		16, 21, 25	Added code size corresponding to RX13T.
		59	6.1 Confirmed Operation Environment: Added Table for Rev.3.30.
		Program	Added support for RX13T. Changed the comment of API functions to the doxygen style. Fixed the “R_SCI_Send” and “R_SCI_SendReceive” issues as mentioned in R20TS0494EJ0100.
3.40	Dec.30.19	1, 4, 12-16	Added support for RX72N, RX66N.
		21-22, 26-27, 31-32	Added code size corresponding to RX72N, RX66N.
		63	6.1 Confirmed Operation Environment: Added Table for Rev.3.40.
		Program	Added support for RX72N, RX66N.

Rev.	Date	Description	
		Page	Summary
3.50	Mar.31.20	1, 4, 8-9	Added support for RX23E-A.
		18, 23-24, 29-30	Added code size corresponding to RX23E-A.
		65	6.1 Confirmed Operation Environment: Added Table for Rev.3.50.
		Program	Added support for RX23E-A.
3.60	Aug.25.20	1,4	Added information that SCI supported by DTC/DMAC Added information Merged IrDA functionality to SCI.
		5,6	Added notes to use SCI with DTC/DMAC support. Added notes to use IrDA functionality.
		15-18	Added configuration setting to use DTC/DMAC. Added configuration setting for IrDA functionality.
		20-39	Added code size corresponding to SCI with DTC/DMAC support Added code size corresponding to Merged IrDA functionality to SCI.
		43-63	Added information Merged IrDA functionality to SCI. Updated and added new demo project
		72-73	Added RSKRX72M to “5. Demo Projects”.
		74	6.1 Confirmed Operation Environment: Added Table for Rev.3.60.
		Program	Added support for SCI with DTC/DMAC support Added support for Merged IrDA functionality to SCI.
3.70	Sep.30.20	74	6.1 Confirmed Operation Environment: Added Table for Rev.3.70.
		Program	Fixed issue of duplicate device group for SCI11 in MDF file. Fixed issue of missing SSCL, SSDA in MDF file.
3.80	Mar.31.21	1,4,11	Added support for RX671.
		6	Updated note when using DTC/DMAC, SCI FIT does not use BYTEQ.
		7	Added 1.5 Using the FIT SCI module. Added 1.5.1 Using FIT SCI module in C++ project.
		15	Added new macro definition SCI_CFG_USE_CIRCULAR_BUFFER to support circular buffer in Asynchronous mode.
		16-19	Added configurations for RX671.
		27,34,41	Added code size corresponding to RX671. Deleted FIFO SPI
		59	Updated description for transfer operation when using circular buffer.
		67	Added configuration setting to use Sampling/Transition timing.
		79	6.1 Confirmed Operation Environment: Added Table for Rev.3.80.
		Program	Added support for RX671. Added support circular buffer in mode asynchronous. Updated macro definitions to enable and disable TXI, RXI, ERI, TEI. Removed using BYTEQ when using DTC/DMAC in Asynchronous mode.

3.90	Apr.15.21	1,4,9 29,37,44 80 Program	Added support for RX140. Added code size corresponding to RX140. 6.1 Confirmed Operation Environment: Added Table for Rev.3.90. Added support for RX140. Added CS+ support for demo project.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.

3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).

7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.