RX Family

IrDA Module
Using Firmware Integration Technology

R01AN2175EJ0102
Rev. 1.02
Nov. 15, 2024

Abstract

This document describes the IrDA module using firmware integration technology (FIT).

The module generates IrDA communication waveforms, and transmits and receives data via infrared light using the infrared data association (IrDA) interface and serial communications interface (SCI). This module is hereinafter referred to as the IrDA FIT module.

Products

The module currently supports the following product.

- RX113 Group

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

Related Documents

For additional information associated with this document, refer to the following application notes.

- Firmware Integration Technology User's Manual (R01AN1833EU)
- Board Support Package Module Using Firmware Integration Technology (R01AN1685EU)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)
- Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826EJ)

Contents

1. Overview

This module supports the RX Series peripheral function, IrDA, and transmits and receives IrDA data communication waveforms based on the IrDA (Infrared Data Association) standard 1.0.

The module supports the TXI, TEI, RXI, and ERI interrupts.

The module cannot be used in cooperation with the DMAC, DTC, or ELC, and does not support multiple interrupts. The I flag must be set to 1 for using interrupts in the module.

1.1 IrDA FIT Module

This module is implemented in a project and used as the APIs. Refer to 2.10 Adding the FIT Module to Your Project for details on implementing the module to the project.

1.2 Outline of the APIs

Table 1.1 lists the API functions included in the module and Table 1.2 lists the Required Memory Sizes.

Table 1.1 API Functions

Function	Description
R_IRDA_SCI_Open	Initializes the IrDA for using this module.
R_IRDA_SCI_Close	Releases the IrDA module.
R_IRDA_SCI_Send	Starts IrDA data transmission.
R_IRDA_SCI_Receive	Starts IrDA data reception.
R_IRDA_SCI_Control	Performs internal processing such as releasing the transmit or receive buffer.
R_IRDA_SCI_GetVersion	Returns the module version.

Table 1.2 Required Memory Sizes

Memory Used	Size	Remarks
ROM	2086 bytes	
RAM	184 bytes	
Maximum user stack usage	80 bytes	
Maximum interrupt stack usage	76 bytes	

* The configuration options when measuring each memory size are set to default values listed in 2.6 Configuration Overview.
* The table lists values when the default values are set to the compile options.
  The required memory size varies depending on the C compiler version and compile options.

1.3 Overview of the IrDA FIT Module

In this module, sizes of the transmit and receive buffers, and communication pins can be selected by specifying constants defined in config.h.

The transmit and receive buffer sizes are specified with IRDA_SCI_CFG_CHi_TX_BUFSIZ and IRDA_SCI_CFG_CHi_RX_BUFSIZ, and they can be changed depending on the usage amount.

The communication pins are specified with IRDA_SCI_CFG_CHi_IRTXD_SEL and IRDA_SCI_CFG_CHi_IRRXD_SEL.

The results selected with the constants can be checked in r_irda_sci_rxXXX.h (rxXXX indicates the product used).

The settings required for IrDA communication is configured by calling the R_IRDA_SCI_Open function in the module. The bit rate, high pulse width, and interrupt priority level can be specified by settings in the user program. With these settings, the IrDA is released from the module-stop state, and registers required for using the IrDA are specified.

The R_IRDA_SCI_Open function returns the address for the handle information as the response. Then the handle works as an internal structure that maintains pointers to the I/O registers for the channel, buffers, or other important information until the R_IRDA_SCI_Close function is executed.

To change communication settings such as the bit rate after the communication is complete, execute the R_IRDA_SCI_Close function, specify arguments to be changed, and then execute the R_IRDA_SCI_Open function.

When transmitting data, specify the address for the transmit data and the transmit size, and then call the R_IRDA_SCI_Send function. If no transmission is being performed, write the first byte of data to the TDR register and store the rest of data in the transmit buffer. If the transmission is being performed, all transmit data is stored in the transmit buffer and the data is written to the TDR register in order within the TXI interrupt handler. In the TXI interrupt handler, the size of data stored in the transmit buffer is checked and the data in the transmit buffer is written to the TDR register. The TEI interrupt is enabled with the TXI interrupt while no data is stored in the transmit buffer, and the callback function is called in the TEI interrupt handler.

The RXI interrupt is used for data reception. Data is stored in the receive buffer and then the callback function is called in the RXI interrupt handler. Data in the receive buffer is read with the R_IRDA_SCI_Receive function by specifying the area address for data storage and the size for read operation. If a communication error occurs during communication, the ERI interrupt occurs. In the ERI interrupt handler, the type of communication error and received data are stored, and then the callback function is called.

1.4 State Transitions

The module has the following three states: uninitialized state, idle state, and communicating state.

Figure 1.1 shows the State Transitions in the IrDA FIT Module.

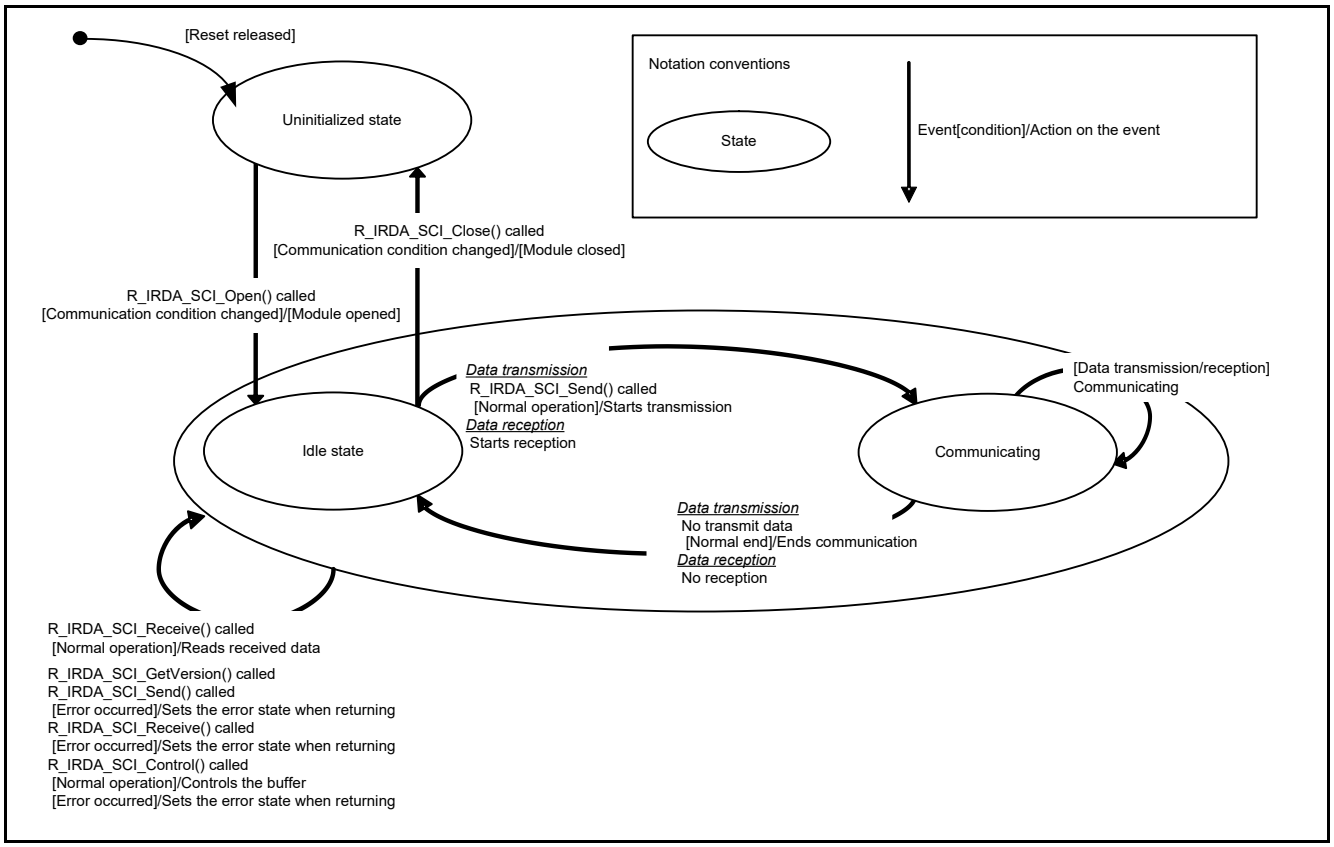

Figure 1.1 State Transitions in the IrDA FIT Module

2. API Information

The sample code accompanying this application note has been run and confirmed under the conditions below.

2.1 Hardware Requirements

This driver requires your MCU support the following feature:

- The IrDA interface

2.2 Software Requirements

This driver is dependent upon the following packages:

- r_bsp
- r_byteq

2.3 Supported Toolchains

This driver is tested and works with the following toolchain:

- Renesas RX Toolchain v.3.06

2.4 Header Files

All API calls and their supporting interface definitions are located in r_irda_sci_rx_if.h. Available setting options when compiling are located in r_irda_sci_rx_config.h. Files r_irda_sci_rx_if.h and r_irda_sci_rx_config.h must be included in the user application.

2.5 Integer Types

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

2.6 Configuration Overview

The configuration options in this module are specified in r_irda_sci_rx_config.h. The option names and setting values are listed in the table below.

Configuration options in *r_irda_sci_rx_config.h*	
#define IRDA_SCI_CFG_PARAM_CHECKING (1)	Selects whether to include parameter checking in the code. The parameter checking is processing to check parameters and is located in the beginning of each function. - When this is set to 0, code for parameter checking is not generated. - When this is set to 1, code for parameter checking is generated and executed.
#define IRDA_SCI_CFG_CHi_INCLUDED **i = 0 to 12** **- When i = 5, the default value = 1** **- When i = other than 5, the default value = 0**	Selects whether to use available channels. - When this is set to 0, code for relevant processes for the channel is not generated. - When this is set to 1, code for relevant processes for the channel is generated.
IRDA_SCI_CFG_CHi_IRTXD_SEL **i = 0 to 12** **- When i = 5, the default value = 1**	Selects the IRTXD pin for each channel. - For channel 5, 1(PC2), 2(PA3), or 3(PA2) can be selected.
IRDA_SCI_CFG_CHi_IRRXD_SEL **i = 0 to 12** **- When i = 5, the default value = 1**	Selects the IRRXD pin for each channel. - For channel 5, 1(PC3) or 2(PA4) can be selected.
#define IRDA_SCI_CFG_CHi_IRTXD_INACTIVE_LEVEL **i = 0 to 12** **- When i = 5, the default value = 1**	Indicates the level of the selected IRTXD pin when the R_IRDA_SCI_Close function is executed. - When this is set to 0, the selected IRTXD pin outputs low. - When this is set to 1, the selected IRTXD pin outputs high.
#define IRDA_SCI_CFG_CHi_IRRXD_INACTIVE_LEVEL **i = 0 to 12** **- When i = 5, the default value = 1**	Indicates the level of the selected IRRXD pin when the R_IRDA_SCI_Close function is executed. - When this is set to 0, the selected IRRXD pin outputs low. - When this is set to 1, the selected IRRXD pin outputs high.
#define IRDA_SCI_CFG_CHi_DATA_POLARITY **i = 0 to 12** **- When i = 5, the default value = 1**	Selects the high pulse width that the IRTXD pin outputs during communication. The value to be selected is the same value of the IrDA Clock Select bit in the IrDA Control Register (IRCR.IRCKS bit). Read the explanation for the bit and specify the value for this option.
#define IRDA_SCI_CFG_CHi_TX_BUFSIZ (80) **i = 0 to 12** **- Default value = 80**	Specifies the size of the buffer used as the transmit queue for each channel. If IRDA_SCI_CFG_CHi_INCLUDED for the corresponding channel is set to 0, the buffer is not maintained.
#define IRDA_SCI_CFG_CH0_RX_BUFSIZ (80) **i = 0 to 12** **- Default value = 80**	Specifies the size of the buffer used as the receive queue for each channel. If IRDA_SCI_CFG_CHi_INCLUDED for the corresponding channel is set to 0, the buffer is not maintained.

2.7 Parameters

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

```
typedef struct st_irda_sci
{
       uint32_t baud_rate; /* Communication bit rate */
       uint8_t clk_out_width; /* Setting value for the high pulse output width of the
                                 IrDA IRTXD pin */
       uint8_t int_priority; /* Interrupt priority level for TXI, TEI, RXI, and ERI;
                                 1 = low, 15 = high */
} irda_sci_t;

typedef struct st_irda_sci_ch_ctrl * irda_sci_hdl_t; /* Handle for the IrDA channel */
```

The structure ‘irda_sci_hdl_t’ has to be declared with the automatic variable or the global variable for each channel. This structure operates using the structure variable specified when calling the R_IRDA_SCI_Open function. After the R_IRDA_SCI_Open function is executed, the area for the structure has to be maintained until the R_IRDA_SCI_Close function is executed.

2.8 Return Values

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

```
typedef enum /* Status code of the IrDA APIs */
{
       IRDA_SCI_SUCCESS,    /* Processing completed successfully. */
       IRDA_SCI_ERR_LOCK_FUNC, /* Selected channel has been hardware-locked. */
       IRDA_SCI_ERR_BAD_CHAN, /* Invalid channel number */
       IRDA_SCI_ERR_OMITTED_CHAN, /* IRDA_SCI_CFG_CHi_INCLUDED in r_irda_sci_rx_config.h
                                     is 0. */
       IRDA_SCI_ERR_CH_NOT_CLOSED, /* Channel currently in operation.
                                      The specified channel has already been used */
       IRDA_SCI_ERR_INVALID_ARG, /* Invalid member included in the structure. */
       IRDA_SCI_ERR_NULL_PTR, /* Specified structure member is NULL. */
       IRDA_SCI_ERR_QUEUE_UNAVAILABLE, /* Transmit queue, receive queue, or both
                                          cannot be opened. */
       IRDA_SCI_ERR_INSUFFICIENT_SPACE, /* Not enough space in the queue to store
                                           all data. */
       IRDA_SCI_ERR_INSUFFICIENT_DATA, /* Receive queue does not have data for
                                          the specified size. */
} irda_sci_err_t;
```

2.9 Callback Function

In this module, the callback function is called when the TEI, RXI, or ERI interrupt occurs.

To specify the callback function, set the address of the function to be registered as the callback function to the argument of the R_IRDA_SCI_Open function.

The callback function has one parameter. The parameter is the pointer to the structure and defined as void type to be consistent with the callback functions in other FIT modules.

```
typedef struct st_irda_sci_cb_args
{
	irda_sci_hdl_t hdl;
	irda_sci_cb_event_t event;
	uint8_t byte; /* Receive data */
} irda_sci_cb_args_t;
```

The ‘hdl’ parameter is the handle for the channel, the ‘event’ parameter is defined in the following enum, and the ‘byte’ parameter stores the receive data.

```
typedef enum e_irda_sci_cb_event
{
	IRDA_SCI_EVT_TEI, /* TEI interrupt occurred; transmitter is in idle state. */
	IRDA_SCI_EVT_RX_CHAR, /* A character received and already placed in the queue. */
	IRDA_SCI_EVT_RXBUF_OVFL, /* Receive queue is full and cannot store data
	                            anymore. */
	IRDA_SCI_EVT_FRAMING_ERR, /* Hardware framing error in the receiver. */
	IRDA_SCI_EVT_OVFL_ERR /* Hardware overrun error in the receiver. */
} irda_sci_cb_event_t;
```

The following shows a sample of the callback function.

```
void MyCallback(void *p_args)
{
	irda_sci_cb_args_t *args;
	args = (irda_sci_cb_args_t *)p_args;
	if (IRDA_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 (IRDA_SCI_EVT_TEI == args->event)
	{
		// from TEI interrupt; transmitter is idle
		// possibly disable external transceiver here
		nop();
	}
#endif
	else if (IRDA_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 (IRDA_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 (IRDA_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();
        }
}
```

2.10 Adding the FIT Module to Your Project

The FIT module must be added to each project in the e^2 studio.

You can use the FIT plug-in to add the FIT module to your project, or the module can be added manually.

It is recommended to use the FIT plug-in as you can add the module to your project easily and also it will automatically update the include file paths for you.

To add the FIT module using the plug-in, refer to chapter 2. “Adding FIT Modules to e2 studio Projects Using FIT Plug-In” in the “Adding Firmware Integration Technology Modules to Projects” application note (R01AN1723EU).

To add the FIT module manually, refer to chapter 3. “Adding FIT Modules to e2 studio Projects Manually” in the “Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)”

When using the FIT module, the BSP FIT module also needs to be added. For details on adding the BSP FIT module, refer to the “Board Support Package Module Using Firmware Integration Technology” application note (R01AN1685EU).

2.11 “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

3.1 R_IRDA_SCI_Open ()

This function configures the SCI to operate the IrDA, enables interrupts, configures port settings for pins IRTXD and IRRXD, and returns the channel handle for use with other API functions.

Format

```
irda_sci_err_t R_IRDA_SCI_Open(uint8_t const          chan,
                               irda_sci_t * const     p_cfg,
                               void                   (* const p_callback)(void *p_args),
                               irda_sci_hdl_t * const p_hdl);
```

Parameters

chan	SCI channel used in the IrDA FIT module (available channel is channel 5)
p_cfg	Pointer to the structure (refer to 2.7 Parameters for details)
p_callback	Pointer to the function called from the interrupt which occurs at completion of a reception/transmission (TEI) or detection of a receive error.
p_hdl	Pointer to the handle for the channel

Return Values

```
IRDA_SCI_SUCCESS:          /* Processing completed successfully. */
IRDA_SCI_ERR_LOCK_FUNC: /* Selected channel has been hardware-locked. */
IRDA_SCI_ERR_BAD_CHAN:   /* Invalid channel number */
IRDA_SCI_ERR_OMITTED_CHAN: /* IRDA_SCI_CFG_CHi_INCLUDED in r_irda_sci_rx_config.h
                              is 0. */
IRDA_SCI_ERR_CH_NOT_CLOSED: /* Channel currently in operation.
                               The specified channel has already been used. */
IRDA_SCI_ERR_INVALID_ARG: /* Invalid member included in the structure. */
IRDA_SCI_ERR_QUEUE_UNAVAILABLE: Transmit queue, receive queue, or both cannot be opened. */
```

Properties

Prototyped in r_irda_sci_rx_if.h.

Description

The initialization is performed to start IrDA communication and returns the handle for use with other API functions to '*p_hdl'. Operations included in the initialization are as follows:

- Enables the IrDA functions and specifies the polarity of the communication pins (set the IRCR.IRE bit to 1 and specify bits IRCR.IRRXINV and IRCR.IRTXINV).
- Initializes registers associated with SCI channel 5 (asynchronous mode, ABCS bit is 0, 2 stop bits).
- Enables SCI5 transmission (set the SCR.TE bit to 1).
- Specifies the Pmn pin function control register (PmnPFS (m = 4, C; n = 2, 3, 4)) for the IRRXD5 pin, and then specifies the port mode register (PMR) to the peripheral function (set the corresponding bit in the PMR register to 1).
- Waits for 18 / (bit rate of 16 × SCI5).
- Specifies the high pulse width of the IRTXD output (specify the IRCR.IRCKS bit).

- Enables the RXI and ERI interrupts (set the corresponding bit in the IER register to 1).
- Enables serial reception (set the SCR.RE bit to 1).
- Specifies the Pmn pin function control register (PmnPFS) for the IRTXD5 pin, and then specifies the port mode register (PMR) to the peripheral function (set the corresponding bit in the PMR register to 1).
- Enables the TXI interrupt (set the corresponding bit in the IER register to 1).

Reentrant
No

Example
```
irda_sci_hdl_t Console;
void main(void)
{
       irda_sci_err_t err;
       irda_control config;

       config.baud_rate = 115200;
       config.clk_out_width = IRDA_SCI_OUT_WIDTH_3_16;
       config.int_priority = 2; /* 1=lowest, 15=highest */
       err = R_IRDA_SCI_Open(IRDA_SCI_CH5, &config, MyCallback, &Console);
       if (IRDA_SCI_SUCCESS != err)
       {
       while(1) { };
       }
              …
}
```

* Refer to 2.9 Callback Function for the sample of the callback function.

Special Notes
This API function implements the algorithm that calculates the optimum values of bits SMR.CKS, SEMR.ABCS, and the BRR register according to BSP_PCLKB_HZ defined in mcu_info.h of the BSP module (1).

This API function also verifies the high pulse width of the IRTXD output is within the specification range (2).

Notes:
1. This does not guarantee a low bit error rate for all peripheral clock/bit rate combinations.
2. The specification range is as follows:
   Minimum: 1.41 µs
   Maximum: (3/16 + 2.5%) × bit rate or (3/16 × bit rate) + 1.08 µs

3.2 R_IRDA_SCI_Close ()

To stop the IrDA, this function performs the initialization for the SCI channel specified with the handle, disables interrupts, and configures port settings for pins IRTXD and IRRXD.

Format

```
irda_sci_err_t R_IRDA_SCI_Close(
        irda_sci_hdl_t const    hdl     /* Structure data */
)
```

Parameters

p_hdl *Handle for the channel*

Return Values

IRDA_SCI_SUCCESS: */* Processing completed successfully. */*
IRDA_SCI_ERR_NULL_PTR: */* Specified structure member is NULL. */*

Properties

Prototyped in r_irda_sci_rx_if.h.

Description

The initialization is performed to stop IrDA communication. Operations included in the initialization are as follows:

- Specifies the Pmn pin function control register (PmnPFS) for the IRTXD5 pin, and then specifies the port mode register (PMR) to a general port (set the corresponding bit in the PMR register to 0).
- Disables the TXI and TEI interrupts (set the corresponding bit in the IER register to 0).
- Disables serial reception (set the SCR.RE bit to 0).
- Disables the RXI and ERI interrupts (set the corresponding bit in the IER register to 0).
- Initializes the high pulse width of the IRTXD output (specify the IRCR.IRCKS bit).
- Specifies the Pmn pin function control register (PmnPFS) for the IRRXD5 pin, and then specifies the port mode register (PMR) to a general port (set the corresponding bit in the PMR register to 0).
- Disables SCI5 transmission (set the TE bit to 0).
- Initializes registers associated with SCI channel 5.
- Clears the IR flags for the TXI, TEI, RXI, and ERI interrupts.
- Disables the IrDA functions and sets the initial setting value to the polarity of the communication pins (set bits IRCR.IRE, IRCR.IRRXINV, and IRCR.IRTXINV to 0).

Reentrant

No

Example

```
irda_sci_hdl_t Console;

void main(void)
{
       irda_sci_err_t err;
       irda_control config;

       config.baud_rate = 115200;
       config.clk_out_width = IRDA_SCI_OUT_WIDTH_3_16;
       config.int_priority = 2; /* 1=lowest, 15=highest */
       err = R_IRDA_SCI_Open(IRDA_SCI_CH5, &config, MyCallback, &Console);
       if (IRDA_SCI_SUCCESS != err)
       {
              while(1) { };
       }
              …
       err = R_IRDA_SCI_Close(Console);
       if (IRDA_SCI_SUCCESS != err)
       {
              while(1) { };
       }
              ……
}
```

* Refer to 2.9 Callback Function for the sample of the callback function.

Special Notes

If this API function is called during transmission or reception, the ongoing transmission or reception is canceled. Execute this function after the TEI interrupt is executed and also when not performing transmission and reception.

3.3 R_IRDA_SCI_Send ()

When no transmission is being performed (IDLE state), this function sets the data to the transmit buffer register and starts transmission. If transmission is being performed, the function stores the data in the transmit data queue, and then performs transmission with an interrupt handler.

Format

```
irda_sci_err_t R_IRDA_SCI_Send(
        irda_sci_hdl_t const    hdl     /* Structure data */
)
```

Parameters

p_hdl *Handle for the channel*

Return Values

IRDA_SCI_SUCCESS: / Processing completed successfully. */*
IRDA_SCI_ERR_NULL_PTR: / Specified structure member is NULL. */*
IRDA_SCI_ERR_INSUFFICIENT_SPACE: / Not enough space in the queue to store all data. */*

Properties

Prototyped in r_irda_sci_rx_if.h.

Description

This API function stores transmit data in the transmit queue to transfer the data from the SCI channel specified by the handle. When the API function is called, if no transmission is being performed, the function executes processing to start transmission.

When the transmission is completed, the callback function specified with R_SCI_Open() is processed.

Reentrant

No

Example

```
#define ONETIME_SEND_SIZE 16

irda_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)
{
       irda_sci_err_t err;
       irda_control config;
       uint16_t cnt;

       config.baud_rate = 115200;
       config.clk_out_width = IRDA_SCI_OUT_WIDTH_3_16;
       config.int_priority = 2; /* 1=lowest, 15=highest */
       err = R_IRDA_SCI_Open(IRDA_SCI_CH5, &config, MyCallback, &Console);
       if (IRDA_SCI_SUCCESS != err)
       {
              while(1) { };
       }
       /* Get the size of the send buffer, if there is free space, passing the
       transmitted data. */
       R_IRDA_SCI_Control(Console, IRDA_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_IRDA_SCI_Send(Console,&data_send_buf[0], ONETIME_SEND_SIZE);
              if (IRDA_SCI_SUCCESS != err)
              {
                     while(1) { };
              }
       }
}
```

* Refer to 2.9 Callback Function for the sample of the callback function.

Special Notes

For the callback function executed at completion of transmission, refer to 2.9 Callback Function.

3.4 R_IRDA_SCI_Receive ()

This API function reads the data stored in the receive queue with the RXI interrupt.

Format

```
irda_sci_err_t R_IRDA_SCI_Receive(
        irda_sci_hdl_t const    hdl     /* Structure data */
)
```

Parameters

p_hdl *Handle for the channel*

Return Values

IRDA_SCI_SUCCESS: */* Processing completed successfully. */*
IRDA_SCI_ERR_NULL_PTR: */* Specified structure member is NULL. */*
IRDA_SCI_ERR_INSUFFICIENT_ DATA: / Receive queue does not have data for the specified size. */*

Properties

Prototyped in r_irda_sci_rx_if.h.

Description

This API function reads the received data from the receive queue corresponding to the SCI channel specified by the handle. If the receive queue does not have data for the specified size, an error is returned and wait processing for reception is not performed.

If a communication error occurs during reception, the callback function specified in R_SCI_Open() is processed. Thus the received data is not read when an error occurs.

Reentrant

No

Example

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

void main(void)
{
       irda_sci_err_t err;
       irda_control config;
       uint16_t cnt;

       config.baud_rate = 115200;
       config.clk_out_width = IRDA_SCI_OUT_WIDTH_3_16;
       config.int_priority = 2; /* 1=lowest, 15=highest */
       err = R_IRDA_SCI_Open(IRDA_SCI_CH5, &config, MyCallback, &Console);
       if (IRDA_SCI_SUCCESS != err)
       {
              while(1) { };
       }
       /* Whether the buffer is receiving data, I want to check. */
       R_IRDA_SCI_Control(Console, IRDA_SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ, (void *)&cnt);
       if (0 != cnt)
       {
              /* Retrieve the data of the size stored. */
              err = R_IRDA_SCI_Receive(Console,&data_recv_buf[cnt_data],cnt);
              if (IRDA_SCI_SUCCESS != err)
              {
                     while(1) { };
              }
       }
}
```

* Refer to 2.9 Callback Function for the sample of the callback function.

Special Notes

For the callback function executed when a communication error occurs during reception, refer to 2.9 Callback Function.

3.5 R_IRDA_SCI_Control ()

This API function verifies the status of the transmit and receive buffers. The usage size of the buffers can be checked or the buffers can be cleared with appropriate commands.

Format

```
irda_sci_err_t R_IRDA_SCI_Control(
        irda_sci_hdl_t const    hdl,    /* Structure data */
        irda_sci_cmd_t const    cmd,    /* Command to be executed */
        void    *p_args         /* Pointer to arguments (content varies depending on the command) */
)
```

Parameters

p_hdl *Handle for the channel*

cmd *Command to be executed (see below)*

p_args *Pointer to arguments and cast into the void* type. The content varies depending on the command.*

Commands which can be specified with ‘cmd’ are as follows:

```
typedef enum e_irda_sci_cmd
{
        IRDA_SCI_CMD_TX_Q_FLUSH,        /* flush transmit queue */
        IRDA_SCI_CMD_RX_Q_FLUSH,        /* flush receive queue */
        IRDA_SCI_CMD_TX_Q_BYTES_FREE, /* get count of unused transmit queue bytes */
        IRDA_SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ, /* get num bytes ready for reading */
} irda_sci_cmd_t;
```

When the IRDA_SCI_CMD_TX_Q_BYTES_FREE or IRDA_SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ is set as the command, ‘p_args’ becomes the pointer to the variable which stores the count value in uint16_t type.Return Values

IRDA_SCI_SUCCESS: / Processing completed successfully. */*
IRDA_SCI_ERR_NULL_PTR: / Specified structure member is NULL. */*
IRDA_SCI_ERR_INVALID_ARG: / Invalid ‘cmd’ or ‘p_args’ value */*

Properties

Prototyped in r_irda_sci_rx_if.h.

Description

This API function is used to read the driver status.

Reentrant

No

Example

```
void MyCallback(void *p_args)
{
	irda_sci_cb_args_t *args;
	args = (irda_sci_cb_args_t *)p_args;
	if (args->event == IRDA_SCI_EVT_RX_CHAR)
	{
		// from RXI interrupt; character placed in queue is in args->byte
		nop();
	}
	else if (args->event == IRDA_SCI_EVT_TEI)
	{
		/* The received data, is an invalid value. Delete the received data. */
		R_IRDA_SCI_Control(Console, IRDA_SCI_CMD_TX_Q_FLUSH,(void *)NULL);
	}
	else if (args->event == IRDA_SCI_EVT_RXBUF_OVFL)
	{
		/* The received data, is an invalid value. Delete the received data. */
		R_IRDA_SCI_Control(Console, IRDA_SCI_CMD_RX_Q_FLUSH,(void *)NULL);
	}
	else if (args->event == IRDA_SCI_EVT_OVFL_ERR)
	{
		/* The received data, is an invalid value. Delete the received data. */
		R_IRDA_SCI_Control(Console, IRDA_SCI_CMD_RX_Q_FLUSH,(void *)NULL);
	}
	else if (args->event == IRDA_SCI_EVT_FRAMING_ERR)
	{
		/* The received data, is an invalid value. Delete the received data. */
		R_IRDA_SCI_Control(Console, IRDA_SCI_CMD_RX_Q_FLUSH,(void *)NULL);
	}
}
```

* For methods to use other commands, refer to the Examples in 3.3 R_IRDA_SCI_Send () and 3.4 R_IRDA_SCI_Receive ().

Special Notes

None

3.6 R_IRDA_SCI_GetVersion ()

This function returns the module version.

Format

uint32_t R_IRDA_SCI_GetVersion(void)

Parameters

None

Return Values

Version number

Properties

Prototyped in r_irda_sci_rx_if.h.

Description

Returns the module version number. The version number is encoded where the top 2 bytes are the major version number and the bottom 2 bytes are the minor version number.

Reentrant

No

Example

```
void main(void)
{
       uint32_t version;
              …
       version = R_IRDA_GetVersion();
       while(1) { };
}
```

Special Notes

This function is inlined using ‘#pragma inline’.

4. Provided Modules

The modules provided can be downloaded from the Renesas Electronics website.

5. Reference Documents

User's Manual: Hardware
The latest version 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
[e^2 studio] RX Family Compiler CC-RX V2.01.00 User's Manual: RX Coding (R20UT2748EJ)
The latest version can be downloaded from the Renesas Electronics website.

Website and Support

Renesas Electronics website
http://www.renesas.com

Inquiries
http://www.renesas.com/contact/

REVISION HISTORY	RX Family Application Note IrDA Module Using Firmware Integration Technology

Rev.	Date	Description	
		Page	Summary
1.00	Dec. 1, 2014	—	First edition issued
1.01	Jan. 27, 2015	14,16,18	Modified the following in the Example of the R_IRDA_SCI_Close function. - Deleted extra "}" in the second if statement. Modified the following in the Example of the R_IRDA_SCI_Send function. - Added "}" to the second if statement. Modified the following in the Example of the R_IRDA_SCI_Receive function. - Added "}" to the second if statement.
1.02	Nov.15,2024	6 11 Program	Updated the toolchains for operation in "2.3 Supported Toolchains" Added the chapter "2.11 About for statements, while statements, and do while statements." Added WAIT_LOOP comments.

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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.

5. 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.

6. 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.

7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.

8. 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.

9. 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.

10. 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.

11. 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.

12. 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.

13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

14. 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.

Corporate Headquarters

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

Trademarks

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

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/.