RX Family

SSI Module using Firmware Integration Technology

Introduction

This application note describes the SSI (Serial Sound Interface) module using firmware integration technology (FIT).

This software module provides ways of PCM data transmit and receive operations using SSI.

Target Device

- RX64M Group
- RX71M Group
- RX113 Group
- RX231 Group
- RX230 Group
- RX23W Group

The SSI module supports two channels of SSI. The number of available SSI channel varies depending on the target device.

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 “4.1 Confirmed Operation Environment".

Related Documents

- Firmware Integration Technology User’s Manual (R01AN1833)
- Board Support Package Module Using Firmware Integration Technology (R01AN1685)

Contents

1. Overview

The SSI (Serial Sound Interface) module using firmware integration technology provides ways of PCM data transmit and receive operation using the SSI. It consists of several functions. Using them in appropriate procedure, PCM data transmit and/or receive operations are performed. When the description says “module” in this document, it indicates the SSI module.

1.1 Using the SSI module

The primary use of the module is PCM data transmit and/or receive operation with external DAC and/or ADC device(s) in the specified data transfer format and sampling frequency. When the description says “Fs” in this document, it indicates sampling frequency. Transfer format could be different depending on the external device(s), so several transfer formats (e.g. I^2S) and sampling frequencies are available to connect with them.

And when using this module, firstly add it to your target project in the following steps.

1) add the module to your target project,

2) modify the file r_ssi_api_rx_config.h to suit it for your application,

1.2 Outline of the API

The following functions are included in this module.

Function	Description
R_SSI_Open ()	Locks to keep a specified SSI channel and initializes it corresponding to r_ssi_api_rx_config.h This function must be called certainly once before starting to use the specified SSI channel individually.
R_SSI_Close ()	Unlocks to releases a specified SSI channel.
R_SSI_Start ()	Enables PCM data transmit and/or receive operations for a specified SSI channel.
R_SSI_Stop ()	Disables PCM data transmit and/or receive operations for a specified SSI channel.
R_SSI_Write ()	Writes PCM data to a specified SSI channel for transmit operation. This function should be called repeatedly during transmit operation.
R_SSI_Read ()	Reads PCM data from a specified SSI channel for receive operation. This function should be called repeatedly during receive operation.
R_SSI_Mute ()	Sets or releases mute on a specified SSI channel during transmit operation. During mute, the transmitting PCM data of the specified SSI channel is turned 0.
R_SSI_GetVersion ()	Returns the module version.
R_SSI_R_SSI_GetFlagTxUnderFlow ()	Returns a state of whether transmit underflow occurs or not. The state is the value corresponding to a specified SSI channel's TUIRQ flag of SSISR.
R_SSI_R_SSI_GetFlagTxOverFlow ()	Returns a state of whether transmit overflow occurs or not. The state is the value corresponding to a specified SSI channel's TOIRQ flag of SSISR.
R_SSI_R_SSI_GetFlagRxUnderFlow ()	Returns a state of whether receive underflow occurs or not. The state is the value corresponding to a specified SSI channel's RUIRQ flag of SSISR.
R_SSI_R_SSI_GetFlagRxOverFlow ()	Returns a state of whether receive overflow occurs or not. The state is the value corresponding to a specified SSI channel's ROIRQ flag of SSISR.

Function	Description
R_SSI_R_SSI_ClearFlagTxUnderFlow ()	Clears a specified SSI channel's TUIRQ flag of SSISR to 0.
R_SSI_R_SSI_ClearFlagTxOverFlow ()	Clears a specified SSI channel's TOIRQ flag of SSISR to 0.
R_SSI_R_SSI_ClearFlagRxUnderFlow ()	Clears a specified SSI channel's RUIRQ flag of SSISR to 0.
R_SSI_R_SSI_ClearFlagRxOverFlow ()	Clears a specified SSI channel's ROIRQ flag of SSISR to 0.

Using functions in the above table with a specified SSI channel, the PCM data transmit and/or receive operation can be performed in the following order.

Basic procedure to transmit PCM data;

1) Call R_SSI_Open() to lock to keep and initialize a specified SSI channel.
2) Call R_SSI_Start() to start transmit operation with a specified SSI channel.
3) Call R_SSI_Write() repeatedly every after Transmit Data Empty Flag being set to maintain transmit operation with a specified SSI channel.
4) Call R_SSI_Stop() to stop transmit operation with a specified SSI channel.
5) Call R_SSI_Close() to unlock to release a specified SSI channel.

Basic procedure to receive PCM data;

1) Call R_SSI_Open() to lock to keep and initialize a specified SSI channel.
2) Call R_SSI_Start() to start receive operation with a specified SSI channel.
3) Call R_SSI_Read() repeatedly every after Receive Data Full Flag being set to maintain receive operation with a specified SSI channel.
4) Call R_SSI_Stop() to stop receive operation with a specified SSI channel.
5) Call R_SSI_Close() to unlock to release a specified SSI channel.

2. API Information

2.1 Hardware Requirements

SSI module requires RX MCU with the SSI.

2.2 Hardware Resource Requirements

2.2.1 MCU Peripheral

The module does not require any MCU peripherals other than SSI.

2.2.2 Memory Consumption

The sizes of ROM, RAM and maximum stack usage associated with this module are listed below. Information is listed for a single representative device of the one channel and two channels, respectively.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7, Configuration Overview.

The values in the table below are confirmed under the following conditions.

Module Revision: r_ssi_api_rx rev1.23

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00

(The option of "-lang = c99" is added to the default settings of the integrated development environment.)

GCC for Renesas RX 4.8.4.201801

(The option of "-std = gnu99" is added to the default settings of the integrated development environment.)

IAR C/C++ Compiler for Renesas RX version 4.10.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings

ROM, RAM and Stack Code Sizes

Device	Category	Memory Used		
		Renesas Compiler	GCC	IAR Compiler
RX231	ROM	2196bytes	3990bytes	2683bytes
	RAM	0bytes	0bytes	0bytes
	STACK	36bytes	-	44bytes
RX64M	ROM	2434bytes	4917bytes	3261bytes
	RAM	0bytes	0bytes	0bytes
	STACK	40bytes	-	44bytes

2.3 Software Requirements

The module is dependent upon the following packages:

- Renesas Board Support Package (r_bsp) v5.10 or higher.

2.4 Supported Toolchains

The module has been confirmed to work with the toolchain listed in 4.1, Confirmed Operation Environment.

2.5 Header Files

All API functions and their supporting interface definitions are located in file r_ssi_api_rx_if.h.

2.6 Integer Types

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

2.7 Configuration Overview

Behavior of SSI module is determined at build-time by configuration options that the user must select. This module supports two types of configuration as Table 1 shows. Therefore, firstly user must choose one configuration from the two. And next, in order to determine the behavior, user must select one of the numbers with "()" for each option shown in Table 2 or Table 3.

- Standard configuration
  This is an easier way of configuration which user chooses values among the alternatives for the options generally known for PCM data transfer. The values are converted suitable value for SSI's I/O registers when building user's target project.
- User unique configuration
  Use this way of configuration in case of that user would like to set values directly to SSI's I/O registers.

The configuration type is chosen by putting macro definition on r_ssi_rx_config.h as shown in Table 1.

Table 1 : Configuration type

Configuration type	Macro definition must be written on r_ssi_rx_config.h
Standard configuration - default	#define SSI_STANDARD_CONFIG
User unique configuration	#define SSI_USER_UNIQUE_CONFIG

Note, carefully refer to the using target device's "User's Manual: Hardware" for setting appropriate value to I/O registers before configuration in any ways.

2.7.1 Standard configuration

Table 2 shows options which user certainly set when Standard configuration is chosen. Chose preferable value among alternatives shown in the right column.

Note, put macro definition "SSI_STANDARD_CONFIG" certainly before use this configuration.

Table 2 : Configuration options for Standard configuration

Configuration options in r_ssi_rx_config.h	
SSI_CH0_IO_MODE - **Default value (2)**	Configures SSI channel 0/1 as transmitter and/or receiver. (0) Not used. (1) Receiver. (2) Transmitter. (3) Transmitter and Receiver (inhibited for channel 1)
SSI_CH1_IO_MODE - **Default value (0)**	
SSI_CH0_SERIAL_IF_FMT - **Default value (0)**	Configures SSI channel 0/1's serial audio interface format. (0) I^2S (1) Left-justified. (2) Right-justified.
SSI_CH1_SERIAL_IF_FMT - **Default value (0)**	
SSI_CH0_DATA_WIDTH - **Default value (16)**	Configures SSI channel 0/1's PCM data width. (8) 8bit

SSI_CH1_DATA_WIDTH - **Default value (16)**	(16) 16bit (18) 18bit (20) 20bit (22) 22bit (24) 24bit
SSI_CH0_BCLK - **Default value (64)**	Configures SSI channel 0/1's bit clock rate. (16) 16Fs (32) 32Fs (48) 48Fs (64) 64Fs
SSI_CH1_BCLK - **Default value (64)**	
SSI_MCLK - **Default value (256)**	Specifies the master clock frequency. The setting value must be integral multiple of SSI_CHn_BCLK. This configuration is common for both SSI channels. (16) 16Fs (32) 32Fs : : (256) 256Fs : : (8192) 8192Fs
SSI_CH0_CLK_MODE - **Default value (0)**	Configures SSI channel 0/1's BCLK and LRCK clock direction. Choose Master mode when SSI feeds those clocks to external DAC and/or ADC device(s). Choose Slave mode, when those clocks are fed SSI from the external device(s). (0) Master mode (1) Slave mode.
SSI_CH1_CLK_MODE - **Default value (0)**	
SSI_CH0_TTRG_NUMBER - **Default value (4)**	Configures condition when Transmit Data Empty flag (TDE) to be set. The setting value is different depending on PCM data width as follows; PCM data width: 8bit TDE is set when the number of PCM data in transmit FIFO is; (12) 12 or less (8) 8 or less (4) 4 or less PCM data width: 16bit TDE is set when the number of PCM data in transmit FIFO is; (6) 6 or less (4) 4 or less (2) 2 or less PCM data width: 18, 20, 22, 24bit
SSI_CH1_TTRG_NUMBER - **Default value (4)**	

	TDE is set when the number of PCM data in transmit FIFO is; (3) 3 or less (2) 2 or less (1) 1 or lesser
SSI_CH0_RTRG_NUMBER - **Default value (4)**	Configures condition when receive Data Full flag (RDF) to be set. The setting value is different depending on PCM data width as follows; PCM data width: 8bit RDF is set when the number of PCM data in receive FIFO is; (4) 4 or greater (8) 8 or greater (12) 12 or greater PCM data width: 16bit RDF is set when the number of PCM data in receive FIFO is; (2) 2 or greater (4) 4 or greater (6) 6 or greater PCM data width: 18, 20, 22, 24bit RDF is set when the number of PCM data in receive FIFO is; (1) 1 or greater (2) 2 or greater (3) 3 or greater
SSI_CH1_RTRG_NUMBER - **Default value (4)**	

2.7.2 User unique configuration

Table 3 shows options which user certainly set when User unique configuration is chosen. Set preferable value according to the using target device's " User's Manual: Hardware".

Note, put macro definition "SSI_USER_UNIQUE_CONFIG" certainly before use this configuration.

Table 3 : Configuration options for User unique configuration

Configuration options in r_ssi_rx_config.h	
SSI_CH0_IO_MODE - **Default value (3)**	Configures SSI channel 0/1 as transmitter and/or receiver. (0) Not used. (1) Receiver. (2) Transmitter. (3) Transmitter and Receiver (inhibited for channel 1)
SSI_CH1_IO_MODE - **Default value (3)**	
SSI_CH0_TTRG - **Default value (3)**	Sets value to SSI channel 0/1's SSIFCR.TTRG register.

SSI_CH1_TTRG - **Default value (3)**	Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH0_RTRG - **Default value (3)**	Sets value to SSI channel 0/1's SSIFCR.RTRG register. Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH1_RTRG - **Default value (3)**	
SSI_CH0_DEL - **Default value (0)**	Sets value to SSI channel 0/1's SSICR.DEL register. Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH1_DEL - **Default value (0)**	
SSI_CH0_PDTA - **Default value (0)**	Sets value to SSI channel 0/1's SSICR.PDTA register. Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH1_PDTA - **Default value (0)**	
SSI_CH0_SDTA - **Default value (0)**	Sets value to SSI channel 0/1's SSICR.SDTA register. Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH1_SDTA - **Default value (0)**	
SSI_CH0_SPDP - **Default value (0)**	Sets value to SSI channel 0/1's SSICR.SPDP register. Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH1_SPDP - **Default value (0)**	
SSI_CH0_SWSP - **Default value (0)**	Sets value to SSI channel 0/1's SSICR.SWSP register. Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH1_SWSP - **Default value (0)**	
SSI_CH0_SCKP - **Default value (0)**	Sets value to SSI channel 0/1's SSICR.SCKP register. Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH1_SCKP - **Default value (0)**	
SSI_CH0_SWL - **Default value (0)**	Sets value to SSI channel 0/1's SSICR.SWL register.

SSI_CH1_SWL - **Default value (0)**	Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH0_DWL - **Default value (0)**	Sets value to SSI channel 0/1's SSICR.DWL register. Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH1_DWL - **Default value (0)**	
SSI_CH0_DWSD - **Default value (0)**	Sets value to SSI channel 0/1's SSICR.DWSD register. Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH1_DWSD - **Default value (0)**	
SSI_CH0_SCKD - **Default value (0)**	Sets value to SSI channel 0/1's SSICR.SCKD register. Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH1_SCKD - **Default value (0)**	
SSI_CH0_AUCKE - **Default value (0)**	Sets value to SSI channel 0/1's SSICR.AUCKE register. Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH1_AUCKE - **Default value (0)**	
SSI_CH0_CKDV - **Default value (0)**	Sets value to SSI channel 0/1's SSICR.CKDV register. Refer to the using target device's "User's Manual: Hardware" for the value to set.
SSI_CH1_CKDV - **Default value (0)**	

2.8 API Data Structures

This section shows the data structures that are used with the SSI module API functions.

2.8.1 Data Types

Some parameters used in API functions are defined in r_ssi_api_rx_if.h. It is the public interface file and allowable values are defined in it.

2.9 Return Values

All API functions of this module are defined as one type of following three.

- int8_t
- int32_t
- ssi_ret_t

  "ssi_ret_t" defined as an enumerator typedef in file r_ssi_api_rx_if.h.

R_SSI_Write() and R_SSI_Read() are “int8_t”, R_SSI_GetVersion() is “int32_t” and the others are all “ssi_ret_t”type. Note that the R_SSI_Write() returns the number of PCM data samples written on transmit FIFO when it shows positive value. In addition, it also returns negative value to show some errors. And the definition of the negative return value is the same meaning as “ssi_ret_t”. In the same way, R_SSI_Read() returns the number of PCM data samples read from receive FIFO or errors.

```
typedef enum {
    SSI_SET         =  1,  /* A specified error flag of SSISR is 1. */
    SSI_CLR         =  0,  /* A specified error flag of SSISR is 0. */
    SSI_SUCCESS     =  0,  /* Function is finished successfully. */
    SSI_ERR_PARAM   = -1,  /* Function is finished unsuccessfully because of
                              incorrect argument. */
    SSI_ERR_CHANNEL = -2, /* Function is finished unsuccessfully because the
                              specified SSI channel is already occupied. */
    SSI_ERR_EXEPT   = -3,  /* Function is finished unsuccessfully because of
                              unwanted hardware condition. */
} ssi_ret_t;
```

2.10 Adding SSI 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.

2.11 “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_SSI_Open

Locks to keep a specified SSI channel and initializes it corresponding to r_ssi_api_rx_config.h.

Format

```
ssi_ret_t R_SSI_Open ( const ssi_ch_t Channel );
```

Parameters

Channel

It specifies an SSI channel to lock. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESS:	*Successful, the specified SSI channel is configured.*
SSI_ERR_PARAM:	*Not successful, the parameter is illegal.*
SSI_ERR_CHANNEL:	*Not successful, the specified SSI channel can't be locked.*
SSI_ERR_EXCEPT:	*Not successful, the specified SSI channel is in unwanted hardware condition.*

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function certainly once before starting to use a specified SSI channel individually.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Locks to keep the specified SSI channel.
- Release the specified SSI channel from the module stop state.
- Initializes I/O registers of the specified SSI channel.
- Sets values to I/O registers of the specified SSI channel corresponding to the file r_ssi_api_rx_config.h.

Reentrant

This function is reentrant.

3.2 R_SSI_Close

Unlocks to release a specified SSI channel.

Format

```
ssi_ret_t R_SSI_Close ( const ssi_ch_t Channel );
```

Parameters

Channel

It specifies an SSI channel to release. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESS: Successful, the specified SSI channel is released.
SSI_ERR_PARAM: Not successful, the parameter is illegal.
SSI_ERR_CHANNEL: Not successful, the specified SSI channel can't be unlocked or has not been locked.

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function when finish to use a specified SSI channel.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked.
- Unlocks to release the specified SSI channel.
- Sets the specified SSI to module stop state.

Reentrant

This function is reentrant.

3.3 R_SSI_Start

Enables PCM data transmit and/or receive operations for a specified SSI channel.

Format

```
ssi_ret_t R_SSI_Start ( const ssi_ch_t Channel );
```

Parameters

Channel

It specifies an SSI channel to enable PCM data transmit and/or receive operations. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESS: *Successful, PCM data transmit and/or receive operations are enabled for the specified SSI channel.*
SSI_ERR_PARAM: *Not successful, the parameter is illegal.*
SSI_ERR_CHANNEL: *Not successful, the specified SSI channel has not been locked.*

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to enable PCM data transmit and/or receive operation with a specified SSI channel.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked.
- Clears the specified SSI channel's transmit FIFO when the specified SSI channel is used as a transmitter.
- Clears the specified SSI channel's receive FIFO when the specified SSI channel is used as a receiver.
- Sets interrupt enable bits TIE, TOIRQ and TUIRQ of SSIFCR and SSICR registers when the specified SSI channel is used as a transmitter.
- Sets interrupt enable bits RIE, ROIRQ and RUIRQ of SSIFCR and SSICR registers when the specified SSI channel is used as a receiver.
- Enables PCM data transmit operation by setting TEN bit of SSICR register of the specified SSI channel when the specified SSI channel is used as a transmitter.
- Enables PCM data receive operation by setting REN bit of SSICR register of the specified SSI channel when the specified SSI channel is used as a receiver.

Use ICU to unmask those interrupts as above when designing SSI module application software which works interrupt mechanism.

Reentrant

This function is not reentrant.

3.4 R_SSI_Stop

Disables PCM data transmit and/or receive operations for a specified SSI channel.

Format

```
ssi_ret_t R_SSI_Stop ( const ssi_ch_t Channel );
```

Parameters

Channel

It specifies an SSI channel to disable PCM data transmit and/or receive operations. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESS: *Successful, PCM data transmit and/or receive operations are disabled for the specified SSI channel.*
SSI_ERR_PARAM: *Not successful, the parameter is illegal.*
SSI_ERR_CHANNEL: *Not successful, the specified SSI channel has not been locked.*
SSI_ERR_EXCEPT: *Not successful, the specified SSI channel is in unwanted hardware condition.*

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to disable PCM data transmit and/or receive operation with a specified SSI channel.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked.
- Clears interrupt enable bits TIE, TOIRQ and TUIRQ of SSIFCR and SSICR registers when the specified SSI channel is used as a transmitter.
- Clears interrupt enable bits RIE, ROIRQ and RUIRQ of SSIFCR and SSICR registers when the specified SSI channel is used as a receiver.
- Disables PCM data transmit operation by clearing TEN bit of SSICR register of the specified SSI channel when the specified SSI channel is used as a transmitter.
- Disables PCM data receive operation by clearing REN bit of SSICR register of the specified SSI channel when the specified SSI channel is used as a receiver.

Use ICU to mask those interrupts as above when designing SSI module application software which works interrupt mechanism.

Reentrant

This function is not reentrant.

3.5 R_SSI_Write

Writes PCM data to a specified SSI channel for transmit operation. For the operation, adequate amount of buffer memory must be declared in user application software.

Format

```
int8_t R_SSI_Write ( const ssi_ch_t Channel, const void * pBuf, const uint8_t Samples );
```

Parameters

Channel

It specifies an SSI channel to write PCM data. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

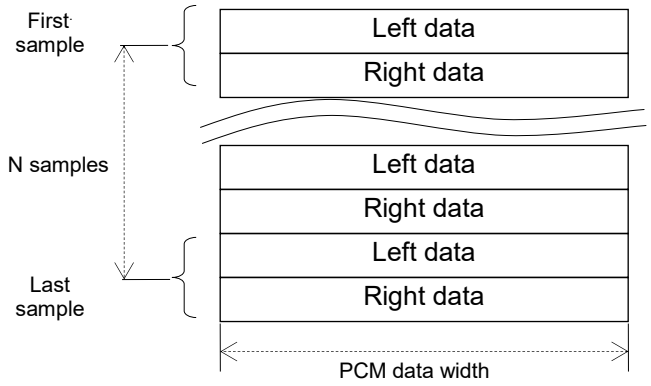
```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

pBuf

It specifies the beginning address of PCM buffer memory storing PCM data to writes to Transmit FIFO Data Register.

PCM buffer memory

- The storing order of PCM data is shown as following figure.
- In case 8 or 16 bit width PCM data, width of buffer memory is the same as PCM data width.
- In case 18, 20, 22 or 24 bit width PCM data, width of buffer memory must be 32bit (LSB justified).
- Total bytes of the buffer memory must be multiple of 8.



Samples

It specifies the request number of PCM data samples to write to Transmit FIFO Data Register.

Return Values

The number of written samples: Shows the number of PCM data samples written to Transmit FIFO Data Register of a specified SSI channel.
SSI_ERR_PARAM: Not successful, the parameter is illegal.
SSI_ERR_CHANNEL: Not successful, the specified SSI channel has not been locked. Or the SSI channel is configured for receive operation.
SSI_ERR_EXCEPT: Not successful, the specified SSI channel is in unwanted hardware condition.

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to write PCM data to a specified SSI channel's Transmit FIFO Data Register.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.

- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked. Or returns SSI_ERR_CHANNEL when the SSI channel is configured for receive operation.
- Writes the PCM data to Transmit FIFO Data Register corresponding to two parameters *pBuf* and *Samples*. However, when the Transmit Data FIFO is become full, this function stops writing before requested number of data writing completion and returns the number of samples successfully written.

Note that R_SSI_Write() should be called repeatedly to write PCM data to the specified SSI channel during transmit operation.

Reentrant

This function is not reentrant.

3.6 R_SSI_Read

Reads PCM data from a specified SSI channel for receive operation. For the operation, adequate amount of buffer memory must be declared in user application software.

Format

```
int8_t R_SSI_Read ( const ssi_ch_t Channel, const void * pBuf, const uint8_t Samples );
```

Parameters

Channel

It specifies an SSI channel to read PCM data. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

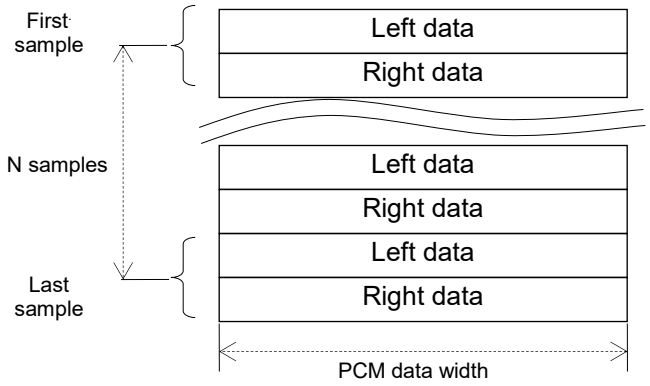
```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

pBuf

It specifies the beginning address of buffer memory storing PCM data read from Receive FIFO Data Register.

PCM data buffer memory

- The storing order of PCM data is shown as following figure.
- In case 8 or 16 bit width PCM data, width of buffer memory is the same as PCM data width.
- In case 18, 20, 22 or 24 bit width PCM data, width of buffer memory must be 32bit (LSB justified).
- Total bytes of the buffer memory must be multiple of 8.



Samples

It specifies the request number of PCM data samples to read from Receive FIFO Data Register.

Return Values

The number of read samples:	*Shows the number of PCM data samples read from Receive FIFO Data Register of a specified SSI channel.*
SSI_ERR_PARAM:	*Not successful, the parameter is illegal.*
SSI_ERR_CHANNEL:	*Not successful, the specified SSI channel has not been locked. Or returns SSI_ERR_CHANNEL when the SSI channel is configured for transmit operation.*
SSI_ERR_EXCEPT:	*Not successful, the specified SSI channel is in unwanted hardware condition.*

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to read PCM data from a specified SSI channel's Receive FIFO Data Register.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.

- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked. Or returns SSI_ERR_CHANNEL when the SSI channel is configured for transmit operation.
- Reads the PCM data from Receive FIFO Data Register corresponding to two parameters *pBuf* and *Samples*. However, when the Receive Data FIFO is become empty, this function stops reading and returns the number of samples successfully read.
  So when the Transmit Data FIFO is full, this function stops reading before requested number of data reading completion and returns the number of samples successfully read.

Note that R_SSI_Read() should be called repeatedly to read PCM data from the specified SSI channel during receive operation.

Reentrant
This function is not reentrant.

3.7 R_SSI_Mute

Sets or releases mute on a specified SSI channel during transmit operation.

During mute, the transmitting PCM data of the specified SSI channel is turned 0.

Format

```
ssi_ret_t R_SSI_Mute ( const ssi_ch_t Channel, const ssi_mute_t OnOff );
```

Parameters

Channel

It specifies an SSI channel to set or release mute. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

OnOff

It specifies to set or release mute. Choose one enumerator member from the enumerator typedef ssi_mute_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_MUTE_ON  = 0,
    SSI_MUTE_OFF = 1,
} ssi_mute_t;
```

Return Values

SSI_SUCCESS:	*Successful, for the specified SSI channel.*
SSI_ERR_PARAM:	*Not successful, the parameter is illegal.*
SSI_ERR_CHANNEL:	*Not successful, the specified SSI channel has not been locked. Or the SSI channel is configured for operations other than transmit.*
SSI_ERR_EXCEPT:	*Not successful, the specified SSI channel is in unwanted hardware condition.*

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to set or release mute for an SSI channel during transmit operation.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked or when it is configured for operations other than transmit.
- Setting SSI_MUTE_ON, disables interrupts related to transmission and configures the specified SSI channel to transmit PCM data 0 as WS continue mode.
  Resister settings are shown as follows.
  Clears TUIEN of SSICR and TIE of SSIFCR, writes 0 on SSIFTDR, sets CONT of SSITDMR, clears TEN of SSICR and TUIRQ bit of SSISR.
- Setting SSI_MUTE_OFF, enables interrupts related to transmission and configures the specified SSI channel to transmit PCM data written on transmit FIFO data register.
  Resister settings are shown as follows.
  Writes 0 on SSIFTDR, sets TEN of SSICR, clears CONT of SSITDMR, clears TDE of SSIFSR and TUIRQ of SSISR, sets TIE of SSIFCR and TUIEN of SSICR.

Reentrant

This function is not reentrant.

3.8 R_SSI_GetVersion

Returns the module version.

Format

```
uint32_t R_SSI_GetVersion ( void );
```

Parameters

None

Return Values

Version number with major and minor version digits packed into a single 32-bit value.

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

The function returns the version of this module. The version number is encoded such that the top two bytes are the major version number and the bottom two bytes are the minor version number.

Reentrant

This function is reentrant.

Example

```
/* Retrieve the version number and convert it to a string. */

uint32_t   version, version_high, version_low;
char       version_str[9];

version = R_SSI_GetVersion();
version_high = (version >> 16)&0xf;
version_low  =  version & 0xff;

sprintf(version_str, "SSI v%1.1hu.%2.2hu", version_high, version_low);
```

3.9 R_SSI_GetFlagTxUnderFlow

Returns a state of whether transmit underflow or not. The state is the value corresponding to a specified SSI channel's TUIRQ flag of SSISR.

Format

```
ssi_ret_t R_SSI_GetFlagTxUnderFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

It specifies an SSI channel to get a state of the TUIRQ flag. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_CLR:	*The specified SSI channel's TUIRQ flag is 0.*
SSI_SET:	*The specified SSI channel's TUIRQ flag is 1.*
SSI_ERR_CHANNEL:	*Not successful, the specified SSI channel has not been locked.*
SSI_ERR_PARAM:	*Not successful, the parameter is illegal.*

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to read a state of the TUIRQ flag of the SSISR register of a specified SSI channel.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked.
- Reads the state of the TUIRQ flag and returns SSI_SET or SSI_CLR depending on the state.

Reentrant

This function is reentrant.

3.10 R_SSI_GetFlagTxOverFlow

Returns a state of whether transmit overflow occurs or not. The state is the value corresponding to a specified SSI channel's TOIRQ flag of SSISR.

Format

```
ssi_ret_t R_SSI_GetFlagTxOverFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

It specifies an SSI channel to get a state of the TOIRQ flag. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_CLR:	*The specified SSI channel's TOIRQ flag is 0.*
SSI_SET:	*The specified SSI channel's TOIRQ flag is 1.*
SSI_ERR_CHANNEL:	*Not successful, the specified SSI channel has not been locked.*
SSI_ERR_PARAM:	*Not successful, the parameter is illegal.*

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to read a state of the TOIRQ flag of the SSISR register of a specified SSI channel.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked.
- Reads the state of the TOIRQ flag and returns SSI_SET or SSI_CLR depending on the state.

Reentrant

This function is reentrant.

3.11 R_SSI_GetFlagRxUnderFlow

Returns a state of whether receive underflow occurs or not. The state is the value corresponding to a specified SSI channel's RUIRQ flag of SSISR.

Format

```
ssi_ret_t R_SSI_GetFlagRxUnderFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

It specifies an SSI channel to get a state of the RUIRQ flag. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_CLR:	*The specified SSI channel's RUIRQ flag is 0.*
SSI_SET:	*The specified SSI channel's RUIRQ flag is 1.*
SSI_ERR_CHANNEL:	*Not successful, the specified SSI channel has not been locked.*
SSI_ERR_PARAM:	*Not successful, the parameter is illegal.*

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to read a state of the RUIRQ flag of the SSISR register of a specified SSI channel.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked.
- Reads the state of the RUIRQ flag and returns SSI_SET or SSI_CLR depending on the state.

Reentrant

This function is reentrant.

3.12 R_SSI_GetFlagRxOverFlow

Returns a state of whether receive overflow occurs or not. The state is the value corresponding to a specified SSI channel's ROIRQ flag of SSISR.

Format

```
ssi_ret_t R_SSI_GetFlagRxOverFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

It specifies an SSI channel to get a state of the ROIRQ flag. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_CLR:	*The specified SSI channel's ROIRQ flag is 0.*
SSI_SET:	*The specified SSI channel's ROIRQ flag is 1.*
SSI_ERR_CHANNEL:	*Not successful, the specified SSI channel has not been locked.*
SSI_ERR_PARAM:	*Not successful, the parameter is illegal.*

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to read a state of the ROIRQ flag of the SSISR register of a specified SSI channel.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked.
- Reads the state of the ROIRQ flag and returns SSI_SET or SSI_CLR depending on the state.

Reentrant

This function is reentrant.

3.13 R_SSI_ClearFlagTxUnderFlow

Clears a specified SSI channel's TUIRQ flag of SSISR to 0

Format

```
ssi_ret_t  R_SSI_ClearFlagTxUnderFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

It specifies an SSI channel to clear the TUIRQ flag. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESSFUL: *Successful, this function cleared the TUIRQ flag of the specified SSI channel.*
SSI_ERR_CHANNEL: *Not successful, the specified SSI channel has not been locked.*
SSI_ERR_PARAM: *Not successful, the parameter is illegal.*

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to clear the TUIRQ flag when the flag is 1.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked.
- Clears the TUIRQ flag to 0 after reading the specified SSISR register and returns SSI_SUCCESS.

Reentrant

This function is reentrant.

3.14 R_SSI_ClearFlagTxOverFlow

Clears transmitter's overflow status by setting the TOIRQ flag to 0.

Format

```
ssi_ret_t R_SSI_ClearFlagTxOverFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

It specifies an SSI channel to clear the TOIRQ flag. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESSFUL: Successful, this function cleared the TOIRQ flag of the specified SSI channel.
SSI_ERR_CHANNEL: Not successful, the specified SSI channel has not been locked.
SSI_ERR_PARAM: Not successful, the parameter is illegal.

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to clear the TOIRQ flag when the flag is 1.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked.
- Clears the TOIRQ flag to 0 after reading the specified SSISR register and returns SSI_SUCCESS.

Reentrant

This function is reentrant.

3.15 R_SSI_ClearFlagRxUnderFlow

Clears a specified SSI channel's RUIRQ flag of SSISR to 0.

Format

```
ssi_ret_t R_SSI_ClearFlagRxUnderFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

It specifies an SSI channel to clear the RUIRQ flag. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESSFUL: Successful, this function cleared the RUIRQ flag of the specified SSI channel.
SSI_ERR_CHANNEL: Not successful, the specified SSI channel has not been locked.
SSI_ERR_PARAM: Not successful, the parameter is illegal.

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to clear the RUIRQ flag when the flag is 1.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked.
- Clears the RUIRQ flag to 0 after reading the specified SSISR register and returns SSI_SUCCESS.

Reentrant

This function is reentrant.

3.16 R_SSI_ClearFlagRxOverFlow

Clears a specified SSI channel's ROIRQ flag of SSISR to 0.

Format

```
ssi_ret_t R_SSI_ClearFlagRxOverFlow ( const ssi_ch_t Channel );
```

Parameters

Channel

It specifies an SSI channel to clear ROIRQ flag. Choose one enumerator member from the enumerator typedef ssi_ch_t shown as follows. It is described in file r_ssi_api_rx_if.h.

```
typedef enum
{
    SSI_CH0 = 0,      /* SSI channel 0 */
    SSI_CH1 = 1,      /* SSI channel 1, available depending on the target MCU */
} ssi_ch_t;
```

Return Values

SSI_SUCCESSFUL: *Successful, this function cleared the ROIRQ flag of the specified SSI channel.*
SSI_ERR_CHANNEL: *Not successful, the specified SSI channel has not been locked.*
SSI_ERR_PARAM: *Not successful, the parameter is illegal.*

Properties

Prototyped in file r_ssi_api_rx_if.h

Description

Call this function to clear the ROIRQ flag when the flag is 1.

This function executes following items for the specified SSI channel.

- Checks the legality of parameter, returns SSI_ERR_PARAM when illegal.
- Checks if the specified SSI channel is locked or not, returns SSI_ERR_CHANNEL when it is not locked.
- Clears the ROIRQ flag to 0 after reading the specified SSISR register and returns SSI_SUCCESS.

Reentrant

This function is reentrant.

4. Appendices

4.1 Confirmed Operation Environment

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

Table 4.1 Confirmed Operation Environment (Rev. 1.23)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version V7.3.0 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 4.8.4.201801 Compiler option: The following option is added to the default settings of the integrated development environment. -std = gnu99
	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.1.23
Board used	Renesas Starter Kit for RX231 (product No.: R0K505231SxxxBE)

Table 4.2 Confirmed Operation Environment (Rev. 1.24)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version V7.5.0
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.1.24
Board used	Renesas Solution Starter Kit for RX23W

4.2 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 an error "#error "ERROR !!! The value set to SSI_CH0_IO_MODE is invalid."".

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

5. 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*-A133B/E

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Aug.1.2014	—	First edition issued.
1.10	Dec.5.2014	3, 21..28	• Added following functions; R_SSI_GetFlagTxUnderFlow (); R_SSI_GetFlagTxOverFlow (); R_SSI_GetFlagRxUnderFlow (); R_SSI_GetFlagRxOverFlow (); R_SSI_ClearFlagTxUnderFlow (); R_SSI_ClearFlagTxOverFlow (); R_SSI_ClearFlagRxUnderFlow (); R_SSI_ClearFlagRxOverFlow ();
		10	• Add following members to the enumeration “ssi_ret_t”; SSI_FLAG_SET = 1, SSI_FLAG_CLR = 0,
1.11	Dec.12.2014	—	• Added support for the RX71M Group.
		4	• Changed “2.3 Software Requirement” Required r_bsp version is changed v2.60 to v2.80 or higher.
1.20	Apr.28.2015	—	• Added support for the RX113, RX231 and RX230 Group.
		1	• Added comment regarding available number of available SSI channel.
		4	• Changed memory consumption. • Changed required r_bsp version from v2.80 to v2.90 or higher.
		12..19, 20..28	• Changed comments in areas showing typedef enum ssi_ch_t.
1.21	Apr.7.2017	—	• Corrected wrong descriptions of this whole document.
		29	• Added chapter 4.
1.22	Feb.1.2019	—	Changes associated with functions: Added support setting function of configuration option Using GUI on Smart Configurator. [Description] Added a setting file to support configuration option setting function by GUI.
1.23	May 20. 19	—	• Added support for the GCC and IAR compiler.
		3, 23	• Added descriptions for R_SSI_Mute.
		33, 34	• Added “4. Appendices”.
1.24	Jun. 20. 19	—	• Added support for the RX23W Group.

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

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/.