RX Family

SRC Module Using Firmware Integration Technology

Introduction

This application note describes the SRC (Sampling Rate Converter) module using firmware integration technology (FIT) for sampling rate conversion of PCM audio data.

Target Device

- RX64M Group
- RX71M Group

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

Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

For details of the confirmed operation contents of each compiler, refer to “4.1 Confirmed Operation Environment".

Related Documents

- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)

Contents

1. Overview

1.1 SRC FIT Module

The SRC FIT module can be used by being implemented in a project as an API. See section 2.11 Adding the FIT Module to Your Project for details on methods to implement this FIT module into a project.

1.2 Overview of the SRC FIT Module

The SRC module using firmware integration technology (SRC FIT module (*1)) provides ways of sampling rate conversion using the Sampling Rate Converter (SRC) peripheral.

It consists of several functions. Using them in appropriate procedure, the sampling rate conversion can be performed.

Note:

(*1) When the description says “module” in this document, it indicates the SRC FIT module.

1.3 API Overview

The primary use of the module is sampling rate conversion for PCM data produced by various sources.

Generally, SRC is placed between PCM data sources (e.g. MP3 decoder) and SSI (Serial Sound Interface). A sampling rate varies depending on the PCM data source. But it is not preferable for SSI when the sampling rate varies. So using SRC, the sampling rate for the PCM data fed into SSI can be kept constantly.

The following functions are included in this module.

Function	Description
R_SRC_Open ()	Locks to keep SRC peripheral and initializes it according to r_src_api_rx_config.h. This is the function must be called certainly once before starting sampling rate conversion.
R_SRC_Close ()	Unlocks to releases SRC peripheral.
R_SRC_Start ()	Starts sampling rate conversion after configuration corresponding to four arguments. The arguments are Input, Output Sampling Rate, Input and Output Data Endian.
R_SRC_Stop ()	Triggers flush processing to finish sampling rate conversion. Note that after R_SRC_Stop() call, R_SRC_Read() should be continuously called until all data to be read from SRC peripheral to complete flush processing.
R_SRC_Write ()	Writes the PCM data before sampling rate conversion to SRC peripheral's Input Data Register. The number of PCM data samples to write and the input memory address are specified by the arguments. Note that R_SRC_Write() and R_SRC_Read() should be called repeatedly during the sampling rate conversion. And the repeat times to call the functions are different between them.
R_SRC_Read ()	Reads the PCM data after sampling rate conversion from SRC peripheral's Output Data Register. The number of PCM data samples to read and the output data memory address are specified by two arguments. Note that R_SRC_Read() and R_SRC_Write() should be called repeatedly during the sampling rate conversion. And the repeat times to call the functions are different between them.
R_SRC_CheckFlush ()	Checks if flush processing is completed or not.
R_SRC_GetVersion ()	Returns the module version.

1.4 Processing Example

Using the six functions in the table above, the sampling rate conversion can be performed in the following order.

Basic procedure for the sampling rate conversion;

1) SRC peripheral lock and initialization using R_SRC_Open().
2) Sampling rate conversion start using R_SRC_Start().
3) PCM data read after sampling rate conversion using R_SRC_Read() repeatedly.
4) PCM data write before sampling rate conversion using R_SRC_write() repeatedly.
5) Flush processing to finish sampling rate conversion using R_SRC_Stop().
6) SRC peripheral unlock using R_SRC_Close().

2. API Information

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

2.1 Hardware Requirements

The MCU used must support the following functions:

- SRC

2.2 Software Requirements

This driver is dependent upon the following FIT module:

- Renesas Board Support Package (r_bsp) v5.00 or higher

2.3 Supported Toolchain

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

2.4 Interrupt Vector

The input data FIFO empty interrupt is enabled by executing the R_SRC_Start function (while the macro definition SRC_IEN is 1).

The output data FIFO full interrupt is enabled by executing the R_SRC_Start function (while the macro definition SRC_OEN is 1).

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

Table 2.1 Interrupt Vector Used in the SRC FIT Module

Device	Interrupt Vector
RX64M	Input data FIFO empty interrupt (vector no.: 50)
RX71M	Output data FIFO full interrupt (vector no.: 51)

2.5 Header Files

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

2.6 Integer Types

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

2.7 Configuration Overview

Some features or behavior of the software are determined at build-time by configuration options that the user must select. The following table shows the behavior when one of the numbers with “( )” is set to the parameter.

Configuration options in r_src_rx_config.h	
SRC_IEN - **Default value (0)**	Configures IEN bit of SRCIDCTRL register. (0) Input Data FIFO Empty interrupt is disabled. (1) Input Data FIFO Empty interrupt is enabled.
SRC_OEN - **Default value (0)**	Configures OEN bit of SRCODCTRL register. (0) Output Data FIFO Empty interrupt is disabled (1) Output Data FIFO Empty interrupt is enabled.
SRC_IFTG - **Default value (3)**	Configures IFTG bit of SRCIDCTRL register IINT bit of SRCSTAT register is set when the number of data in the input FIFO according to the following condition; (0) the number is 0. (1) the number is 2 or less. (2) the number is 4 or less. (3) the number is 6 or less.
SRC_OFTG - **Default value (3)**	Configures OFTG bit of SRCODCTRL register OINT bit of SRCSTAT register is set when the number of data in the output FIFO according to the following condition; (0) the number is 1 or greater. (1) the number is 4 or greater. (2) the number is 8 or greater. (3) the number is 12 or greater.
SRC_OCH - **Default value (0)**	Configures OCH bit of SRCODCTRL register. (0) Does not exchange the channels. (1) Exchange the channels.

2.8 Code Size

The sizes of ROM, RAM, and maximum stack usage associated with this module are listed below.

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

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

Module Revision: r_src_api_rx rev1.14

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00

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

GCC for Renesas RX 8.3.0.201904

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

IAR C/C++ Compiler for Renesas RX version 4.13.1

(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
RX64M	ROM	**32687 bytes**	**40604 bytes**	**33772 bytes**
	RAM	**7712 bytes**	**7528 bytes**	**2396 bytes**
	STACK	**128 bytes**	**-**	**124 bytes**
RX71M	ROM	**32735 bytes**	**40676 bytes**	**33844 bytes**
	RAM	**7724 bytes**	**7540 bytes**	**2408 bytes**
	STACK	**128 bytes**	**-**	**124 bytes**

Note 1. The size includes BSP.

2.9 Parameters

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

2.10 Return Values

All functions of SRC FIT module are defined as one type of following three.

R_SRC_Write() is “int8_t”, R_SRC_Read() is “int32_t” and the others are all “src_ret_t” type. Note that the R_SRC_Write() returns the number of PCM data word written on Input Data Register when it shows positive value. In addition, it also returns negative value to show some error and progress of flush processing. And the definition of the negative return value is the same meaning as “src_ret_t”. In the same way, R_SRC_Read() returns negative value.

- int8_t
- int32_t
- src_ret_t
- "src_ret_t" defined as an enumerator typedef in file r_src_api_rx_config.h.

```
typedef enum {
    SRC_SUCCESS    =  0,  /* Function is finished successfully. */
    SRC_ERR_PARAM  = -1,  /* Function is finished unsuccessfully because of
                             incorrect argument. */
    SRC_ERR_UNLOCK = -2,  /* Function is finished unsuccessfully because SRC
                             peripheral is unlocked. */
    SRC_ERR_LOCKED = -3,  /* Function is finished unsuccessfully because SRC
                             peripheral is locked. */
    SRC_NOT_END    = -4,  /* Flush process is not completed. */
    SRC_END        = -5,  /* Flush process is completed. */
} src_ret_t;
```

2.11 Adding the FIT Module to Your Project

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

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: e^2 studio (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 “RX Family 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 “RX Smart Configurator User’s Guide: CS+ (R20AN0470)” 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 “RX Family Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

(5) Adding the FIT module to your project using the Smart Configurator in IAREW
By using the Smart Configurator Standalone version, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: IAREW (R20AN0535)” for details.

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

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

The following shows example of description.

while statement example :

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

for statement example :

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

do while statement example :

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

3. API Functions

R_SRC_Open

Locks to keep SRC peripheral, initializes and configures it according to r_src_api_rx_config.h .

Format
src_ret_t R_SRC_Open (void);

Parameters
None

Return Values
SRC_SUCCESS: *Successful, SRC peripheral is configured.*
SRC_ERR_LOCKED: *Not successful, because SRC peripheral is already locked.*

Properties
Prototyped in file r_src_api_rx_if.h

Description
Certainly call this function once before starting to use SRC peripheral.

It does following items for SRC peripheral to use.

- Locks to keep SRC peripheral.
- Cancels SRC peripheral's module stop state.
- Initializes SRC peripheral registers.
- Configures SRC peripheral registers according to file r_src_api_rx_config.h.
- Downloads filter coefficients. (contained in file r_src_api_rx_coef.h)

R_SRC_Close

Unlocks to release SRC peripheral.

Format
src_ret_t R_SRC_Close (void);

Parameters
None

Return Values
SRC_SUCCESS: *Successful, unlocked to release SRC peripheral.*
SRC_ERR_UNLOCK: *Not successful, because the SRC peripheral is not locked yet.*

Properties
Prototyped in file r_src_api_rx_if.h

Description
Call this function when finish to use SRC peripheral.

It does following items for SRC peripheral to use.

- Checks if SRC peripheral is locked and if not, it returns SRC_ERR_UNLOCK.
- Unlocks to release SRC peripheral.
- Sets SRC peripheral to module stop state.

R_SRC_Start

Starts sampling rate conversion corresponding to the arguments.

Format

src_ret_t R_SRC_Start (src_ifs_t fsi, src_ofs_t fso, src_ied_t ied, src_oed_t oed);

Parameters

fsi

It is the sampling frequency of the PCM data before sampling rate conversion. Choose one enumerator member from the enumerator typedef src_ifs_t shown as follows. It is described in file r_src_api_rx_if.h.

```
typedef enum
{
    SRC_IFS_8  = 0,        /* 8kHz     */
    SRC_IFS_11 = 1,        /* 11.02kHz */
    SRC_IFS_12 = 2,        /* 12kHz    */
    SRC_IFS_16 = 4,        /* 16kHz    */
    SRC_IFS_22 = 5,        /* 22.05kHz */
    SRC_IFS_24 = 6,        /* 24.0kHz  */
    SRC_IFS_32 = 8,        /* 32kHz    */
    SRC_IFS_44 = 9,        /* 44.1kHz  */
    SRC_IFS_48 = 10,       /* 48kHz    */
} src_ifs_t;
```

fso

It is the sampling frequency of the PCM data after sampling rate conversion. Choose one enumerator member from the enumerator typedef src_ofs_t shown as follows. It is described in file r_src_api_rx_if.h.

```
typedef enum
{
    SRC_OFS_44 = 0,        /* 44.1kHz  */
    SRC_OFS_48 = 1,        /* 48kHz    */
    SRC_OFS_32 = 2,        /* 32kHz    */
    SRC_OFS_8  = 4,        /* 8kHz     */
    SRC_OFS_16 = 5,        /* 16kHz    */
} src_ofs_t;
```

ied

It specifies the endian format of input PCM data. Choose one enumerator from the enumerator typedef src_ied_t shown as follows. It is described in file r_src_api_rx_if.h.

```
typedef enum {
    SRC_IED_OFF = 0, /* Endian of input data is the same as endian of chip
                        configuration. */
    SRC_IED_ON  = 1, /* Endian of input data is different from endian chip
                        configuration. */
} src_ied_t;
```

oed

It specifies the endian format of output PCM data. Choose one enumerator from the enumerator typedef src_oed_t shown as follows. It is described in file r_src_api_rx_if.h.

```
typedef enum {
    SRC_OED_OFF = 0, /* Endian of output data is the same as endian of chip
                        configuration. */
    SRC_OED_ON  = 1, /* Endian of output data is different from endian chip
                        configuration. */
} src_oed_t;
```

Return Values

SRC_SUCCESS:	*Successful, SRC started sampling rate conversion.*
SRC_PARAM:	*Parameter is Illegal.*
SRC_ERR_UNLOCK:	*SRC peripheral is not locked yet.*
SRC_NOT_END:	*Flush processing is not completed.*

Properties

Prototyped in file r_src_api_rx_if.h

Description

Call this function when starting sampling rate conversion.

It does following processes to start sampling rate conversion.

- Checks if SRC peripheral is locked and if not, it returns SRC_ERR_UNLOCK.
- Checks the legality of parameters, and if they are illegal it returns SRC_ERR_PARAM.
- Checks if no flush process on-going. If the process is on-going, it returns SRC_NOT_END.
- Clears SRC peripheral's internal data.
- Sets the sampling frequency of the PCM data before sampling rate conversion corresponding to *ifs*.
- Sets the sampling frequency of the PCM data after sampling rate conversion corresponding to *ofs*.
- Sets the endian format of input PCM data corresponding to *ied*.
- Sets the endian format of output PCM data corresponding to *oed*.
- Enables or disables input Data FIFO Empty interrupt and output Data FIFO Full interrupt according to r_src_api_rx_config.h.
- Start sampling rate conversion.

R_SRC_Stop

Triggers flush processing to finish sampling rate conversion.

Format
src_ret_t R_SRC_Stop (void);

Parameters
None

Return Values
SRC_SUCCESS:	*Successful, request to stop is accepted.*
SRC_ERR_UNLOCK:	*SRC peripheral is not locked yet.*
SRC_NOT_END:	*Flush processing is not completed.*
SRC_END:	*Flush processing is completed.*

Properties
Prototyped in file r_src_api_rx_if.h

Description
Call this function when trigger flush processing to finish sampling rate conversion.

It does following processes.

- Checks if SRC peripheral is locked. If not, it returns SRC_ERR_UNLOCK.
- Checks if no flush process on-going. If the process is on-going, it returns SRC_NOT_END. Or already completed, it returns SRC_END.
- Disables input Data FIFO Empty interrupt.
- Triggers flush processing.

Note that after R_SRC_Stop() call, R_SRC_Read() should be continuously called until all data to be read from SRC peripheral to complete flush processing.

R_SRC_Write

Write the PCM data before sampling rate conversion to SRC peripheral.

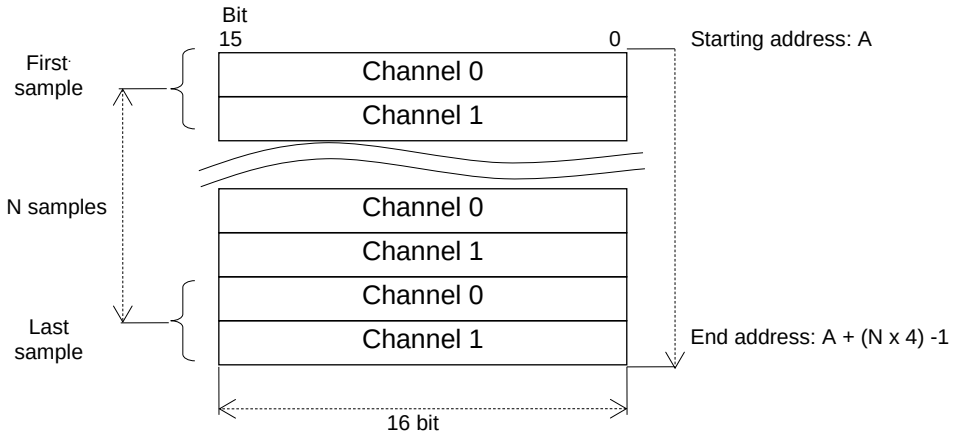
Format

int8_t R_SRC_Write (uint16_t * buf, uint32_t samples);

Parameters

buf

It specifies the starting address of PCM data buffer to write to Input Data Register of SRC peripheral. Note that the PCM data in the buffer must be aligned as following figure. And the one sample is defined as a pair of 16 bit PCM data.



samples

It specifies the number of PCM data samples to write to Input Data Register of SRC peripheral.

Return Values

The number of written samples: Shows the number of PCM data samples written to Input Data Register.
SRC_PARAM: Parameter is Illegal.
SRC_ERR_UNLOCK: SRC peripheral is not locked yet.
SRC_NOT_END: Flush processing is not completed.
SRC_END: Flush processing is completed.

Properties

Prototyped in file r_src_api_rx_if.h

Description

Call this function to write the PCM data before sampling rate conversion to SRC peripheral's Input Data Register. The number of PCM data samples to write and the buffer memory address are specified by the arguments.

It does following processes.

- Checks if SRC peripheral is locked. If not, it returns SRC_ERR_UNLOCK.
- Checks if no flush process on-going. If the process is on-going, it returns SRC_NOT_END. Or already completed, it returns SRC_END.
- Checks the legality of parameters, and if they are illegal it returns SRC_ERR_PARAM.
- Writes the PCM data on Input Data Register corresponding to parameters *buf* and *samples*. Note that there is a limit to writing, so when the input data FIFO is full, it stops writing and returns the number of data written successfully.

Note that R_SRC_Write() and R_SRC_Read() should be called repeatedly during the sampling rate conversion. And the repeat times to call the functions are different between them.

R_SRC_Read

Read the PCM data after sampling rate conversion from SRC peripheral

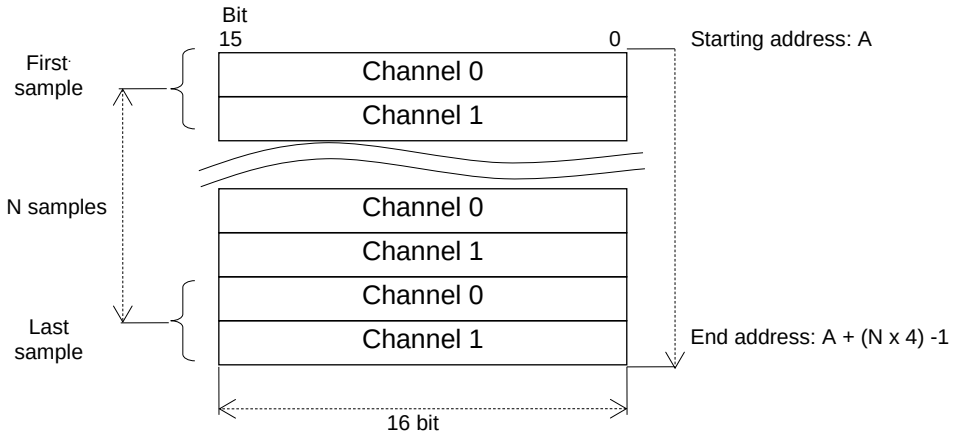
Format

int32_t R_SRC_Read (uint16_t * buf, uint32_t samples);

Parameters

buf

It specifies the starting address of PCM data buffer to store PCM data read from Output Data Register of SRC peripheral. The PCM data is aligned in the buffer as following figure. And the one sample is defined as a pair of 16 bit PCM data.



samples

It specifies the number of the PCM data samples to read from Output Data Register of SRC peripheral.

Return Values

The number of read samples: Shows the number of PCM data samples read from Output Data Register.
SRC_PARAM: Parameter is Illegal.
SRC_ERR_UNLOCK: SRC peripheral is not locked yet.
SRC_END: Flush processing is completed.

Properties

Prototyped in file r_src_api_rx_if.h

Description

Call this function to read the PCM data after sampling rate conversion from SRC peripheral's Output Data Register. The number of PCM data samples to read and the buffer memory address are specified by the arguments.

It does following processes.

- Checks if SRC peripheral is locked. If not, it returns SRC_ERR_UNLOCK.
- Checks the legality of parameters, and if they are illegal it returns SRC_ERR_PARAM.
- Checks if flush process on-going. If the process is already completed, it returns SRC_END.
- Reads the PCM data from Output Data Register corresponding to parameters *buf* and *samples*. Note that there is a limit to reading, so when the output data FIFO is empty, it stops reading and returns the number of data successfully read.

Note that R_SRC_Read() and R_SRC_Write() should be called repeatedly during the sampling rate conversion. And the repeat times to call the functions are different between them.

R_SRC_CheckFlush

Checks if flush processing is completed or not

Format
src_ret_t R_SRC_CheckFlush (void);

Parameters
None

Return Values
SRC_ERR_UNLOCK:	*SRC peripheral is not locked yet.*
SRC_NOT_END:	*Flush processing is not completed.*
SRC_END:	*Flush processing is completed.*

Properties
Prototyped in file r_src_api_rx_if.h

Description
Call this function to check if flush processing is completed. This function is used in case of that DMACs are used instead of R_SRC_Write() and R_SRC_Read() for PCM data transfer to/from SRC peripheral.

It does following processes.

- Checks if SRC peripheral is locked. If not, it returns SRC_ERR_UNLOCK.
- Checks if flush process on-going. If the process is on-going, it returns SRC_NOT_END. Or already completed, it returns SRC_END.

R_SRC_GetVersion

Returns the module version.

Format
uint32_t R_SRC_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_src_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.

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

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

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

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

4. Appendices

4.1 Confirmed Operation Environment

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

Table 4.1 Confirmed Operation Environment (Rev 1.13)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0 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 4.8.4.2018.01 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. 1.13
Board used	Renesas Starter Kit+ for RX64M (product No.: R0K50564MS800BE)

Table 4.2 Confirmed Operation Environment (Rev 1.14)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.7.0 IAR Embedded Workbench for Renesas RX 4.13.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.201904 Compiler option: The following option is added to the default settings of the integrated development environment. -std = gnu99
	IAR C/C++ Compiler for Renesas RX version 4.13.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev. 1.14
Board used	Renesas Starter Kit+ for RX64M (product No.: R0K50564Mxxxxxx)

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 the error: This MCU is not supported by the current r_src_api_rx module.

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

(3) Q: I have added the FIT module to the project and built it. Then I got the error: the value for macro (macro name) must…

A: The setting in the file "r_src_api_rx_config.h" may be wrong. Check the file "r_src_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.

None

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul. 11, 2014	—	First edition issued
1.10	Sep. 9, 2014	4	“2.2.2 memory consumption” is changed corresponding to software version 1.10.
		5	Correction for clerical error in description for SRC_OFTG in Table 1.
		5..7	Renumbering part numbers from 2.7 to 2.10.
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.
1.12	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.
1.13	May 20, 2019	—	Supported the following compilers: - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		1	Added the section of Target Compilers.
		3	Updated the section of 1.3 Outline of the API.
		5	Added the section of 2.4 Interrupt Vector.
		7	Added the section of 2.8 Code Size.
		10	Added the section of 2.12 “for”, “while” and “do while” statements.
		22	Added the section of 4.1 Confirmed Operation Environment.
1.14	Jun 10, 2020	—	Modified comment of API function to Doxygen style.
		Program	Fixed the following. [Target device] All devices. [Description] Changed processing so that there is a register that may be accessed from multiple peripheral functions at the same time, and the atomicity of writing to that register can be ensured.
		1	Deleted R01AN1833 from Related Documents.
		7	2.8 Code Size, amended.
		9	Changed the section of 2.11 Adding the FIT Module to Your Project.
		11..21	Deleted the Reentrant for each API in 3 API Functions.
		22	Added Table 4.2 Confirmed Operation Environment (Rev 1.14)

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/.