RX Family

IRQ Module Using Firmware Integration Technology

Introduction

This application note describes the Interrupt Request (IRQ) module which uses Firmware Integration Technology (FIT). This module uses IRQ to provide a unified, abstracted interface for handling events from external pin interrupts. In this document, this module is referred to as the IRQ FIT module.

Target Devices

- RX110, RX111, RX113 Groups
- RX130 Group
- RX230 Group
- RX231 Group
- RX23T Group
- RX23W Group
- RX24T Group
- RX24U Group
- RX64M Group
- RX651, RX65N Group
- RX66T Group
- RX71M Group
- RX72T Group
- RX72M Group

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

Target Compilers

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

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

Contents

1. Overview

1.1 IRQ FIT Module

The IRQ FIT module can be used by being implemented in a project as an API. See section 2.12, Adding the FIT Module to Your Project for details on methods to implement this FIT module into a project.

1.2 Overview of the IRQ FIT Module

This software provides a unified, abstracted interface for handling events from external pin interrupts. These events are mapped to the IRQ vectors. The operations needed to prepare the IRQs for handling interrupts are performed in the R_IRQ_Open() API function. IRQ vectors supported by each MCU can be used, so a means to identify a particular IRQ vector in each API function is provided. The software makes use of a data structure assigned to each IRQ that stores information specific to that vector required to perform the various IRQ API functions. One such data structure is allocated for each IRQ in use. Each of these data structures is referred to as an IRQ handle, and each has a handle pointer.

When an IRQ is initialized through the R_IRQ_Open() function, its handle pointer is returned to the caller. Thereafter, the application must provide the handle pointer for the selected IRQ when calling any of the remaining IRQ API functions. When called, the API functions extract the IRQ number from the handle, as well as other information linked to that IRQ and contained in the handle structure.

When an IRQ event is triggered, an interrupt handler is invoked that passes control to a user defined "Callback" function. Since Callbacks are executed in the interrupt state, further interrupts are disabled until the callback completes and the program returns from the ISR. Interrupt processing may be enabled or disabled by the user application at any time after the IRQ vector has been initialized.

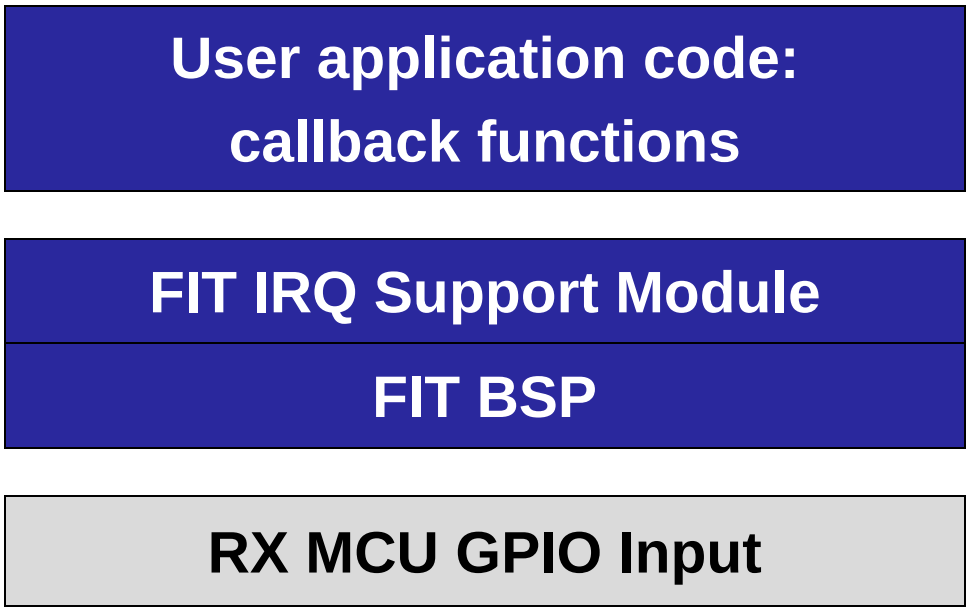

Figure 1 : Example Figure Showing Project Layers

1.3 Using the FIT IRQ module

The primary use of the IRQ module is to make it easy to generate interrupt events that are triggered by a change in state of an MCU GPIO input pin. The user's application can arbitrarily assign a callback function to the event that will execute upon event detection.

After adding the IRQ module to your project you will need to modify the *r_irq_rx_config.h* file to configure the software for your installation.

If Smart Configurator is used, at least one IRQ pin must be chosen, otherwise build error would happen. Make sure that in *r_irq_rx_config.h*, the following are not commented out:

```
#include "r_irq_rx_pinset.h"
#ifndef R_IRQ_RX_H
#error "Please add IRQ Pin setting in Smart Configurator, or if you are using
FIT Configurator remove the comment of following macros"
#endif
```

and the following are commented out:

```
#define IRQ_PORT_IRQ0_PORT      ('m')
#define IRQ_PORT_IRQ0_BIT       ('n')
…
```

If FIT Configurator is used, make sure that in *r_irq_rx_config.h*, the section to comment/uncomment is reverse to the case where Smart Configurator is used

See Section 2.7, Configuration Overview for details on configuration options.

Control registers for input pins to be used for IRQ sources must be correctly set up before the IRQ module can communicate with them. The IRQ module does not provide any means to initialize the pin registers--that needs to be done externally prior to calling IRQ API functions. Typically this would be accomplished at system startup time as part of a general pin initialization routine.

There is no need to set up interrupt vectors for IRQs as this software takes care of that automatically based on the IRQs that have been enabled for use at build time through the configuration options.

The first step in using IRQs at run time is to call the R_IRQ_Open() function, passing the desired IRQ number and other required settings. On completion, the IRQ will be active and ready to respond to the specified transition or state of the input pin. On the occurrence of an IRQ event, the ISR will call the callback function that you provided as an argument in your R_IRQ_Open() call.

Two convenient commands are provided in the R_IRQ_Control() function to allow changing the interrupt trigger mode, and the priority level. This permits the interrupt event to be adjusted to adapt to current conditions as desired. Interrupts for the selected IRQ number are disabled briefly while the R_IRQ_Control() function is making the changes.

Other IRQ settings are only configured at the R_IRQ_Open(); these are the settings that are not expected to frequently change. If they need to be changed later during run-time, the R_IRQ_Close() function must be called so that R_IRQ_Open() can be called again with new settings.

Generally, most of the IRQ API functions will require a 'handle' argument. This is used to identify the IRQ number that is selected for the operation. A handle is obtained by first calling the R_IRQ_Open() function. You must provide the address of a location where you will store the handle to R_IRQ_Open(), and on completion the handle will be available for use. Thereafter, simply pass the provided handle value for that IRQ number to the other IRQ functions when calling them. In your application you will need to keep track of which handle belongs to a given IRQ, as each IRQ will be assigned its own handle.

1.4 API Overview

Table 1.1 lists the API functions included in this module.

Table 1.1 API Functions

Function	Description
R_IRQ_Open()	Initializes the associated registers required to prepare the specified IRQ for use, enables interrupts, and provides the handle for use with other API functions. Takes a callback function pointer for responding to interrupt events. This function must be called before calling any other API functions.
R_ IRQ_Close()	Disables the specified IRQ and its associated interrupt.
R_IRQ_Control()	Handles special hardware or software operations for the IRQ.
R_IRQ_ReadInput()	Reads the current level of the pin assigned to the specified IRQ.
R_IRQ_InterruptEnable()	Enables or disables the ICU interrupt for the specified IRQ.
R_IRQ_GetVersion()	Returns the driver version number.

1.5 Limitations

This driver is applicable only to IRQ type interrupts; i.e.: external input pin interrupts.

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:

- IRQ
- One or more available GPIO input pins that are configurable as interrupt sources

2.2 Software Requirements

This driver is dependent upon the following FIT modules:

- Renesas Board Support Package (r_bsp) v5.20 or higher. It assumes that the related I/O ports have been correctly initialized elsewhere prior to calling this software's API functions.
- Use of the digital filtering features requires that the peripheral clock (PCLK) has been initialized by an external procedure prior to calling the APIs of this module.

2.3 Supported Toolchain

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

2.4 Interrupt Vector

When a change of state on an IRQ input pin occurs that matches the trigger mode setting an interrupt request is generated. If interrupts are enabled, the interrupt ISR will execute which will call your assigned callback function. It is within the callback that you will place the code that you want to occur immediately in response to the ISR. Since callbacks are being processed within the context of the interrupt, and interrupts are disabled at this time, it is strongly recommended that your callback function complete as quickly as possible to avoid missing other interrupts that might occur in the system.

IRQ interrupt is enabled by executing the **R_IRQ_Open()** function.

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

Table 2.1 Interrupt Vector Used in the IRQ FIT Module

Device	Interrupt Vector
RX110 *1	IRQ0 interrupt (vector no.: 64)
RX111 *1	IRQ1 interrupt (vector no.: 65)
RX113 *1	IRQ2 interrupt (vector no.: 66)
RX130 *1	IRQ3 interrupt (vector no.: 67)
RX230 *1	IRQ4 interrupt (vector no.: 68)
RX231 *1	IRQ5 interrupt (vector no.: 69)
RX23T *2	IRQ6 interrupt (vector no.: 70)
RX23W*3	IRQ7 interrupt (vector no.: 71)
RX24T *1	IRQ8 interrupt (vector no.: 72)
RX24U *1	IRQ9 interrupt (vector no.: 73)
RX64M	IRQ10 interrupt (vector no.: 74)
RX651	IRQ11 interrupt (vector no.: 75)
RX65N	IRQ12 interrupt (vector no.: 76)
RX66T	IRQ13 interrupt (vector no.: 77)
RX71M	IRQ14 interrupt (vector no.: 78)
RX72T	IRQ15 interrupt (vector no.: 79)
RX72M	

Note 1. Only have IRQ0 to IRQ7.
Note 2. Only have IRQ0 to IRQ5.
Note 3. Only have IRQ0, IRQ1, IRQ4 to IRQ7.

2.5 Header Files

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

2.6 Integer Types

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

2.7 Configuration Overview

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

Configuration options in r_irq_rx_config.h	
IRQ_CFG_FILT_EN_IRQn 0	If defined as 1 digital filtering is enabled for the IRQ number n. 0 = not enabled.
IRQ_CFG_FILT_PCLK_IRQn	PCLK digital filter clock divisor setting for the IRQ number n. Select from one of the predefined constants IRQ_CFG_PCLK_DIVxx. Example: /* Filter sample clock divisor for IRQ 0 = PCLK/64. */ #define IRQ_CFG_FILT_PLCK_IRQ0 (IRQ_CFG_PCLK_DIV64)
IRQ_CFG_REQUIRE_LOCK 1	If defined as 1 then the R_IRQ_Open() function will attempt to obtain a BSP lock for the duration of the function execution. This is to protect its internal state from reentrant access. On exit the lock will be released. This option may be set to 0 if BSP locking is not required or available in the system.
IRQ_CFG_PARAM_CHECKING * Default = BSP_CFG_PARAM_ CHECKING_ENABLE	Checking of arguments passed to IRQ API functions can be enabled or disabled. Disabling argument checking is provided for systems that absolutely require faster and smaller code. By default the module is configured to use the setting of the system-wide BSP_CFG_PARAM_CHECKING_ENABLE macro. This can be locally overridden for the IRQ module by redefining IRQ_CFG_PARAM_CHECKING. To control parameter checking locally, set IRQ_CFG_PARAM_CHECKING to 1 to enable it, otherwise set to 0 skip checking.
IRQ_PORT_IRQn_PORT IRQ_PORT_IRQn_BIT	The port and port-bit assignments to each IRQ must be defined so that the IRQ module can perform port specific operations, such as R_IRQ_ReadInput(). Set these as required according to the following format: #define IRQ_PORT_IRQ*_PORT (‘m’) (where m is the port number and the IRQ number replaces *) #define IRQ_PORT_IRQ*_BIT (‘n’) (where n is the bit number and the IRQ number replaces *) Note: Port assignments here must match the port configuration settings performed externally for them by the BSP

2.8 Code Size

Typical code sizes associated with this module are listed below.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7, Configuration Overview. The table lists reference values when the C compiler's compile options are set to their default values, as described in 2.3, Supported Toolchains. The compile option default values are optimization level: 2, optimization type: for size, and data endianness: little-endian. The code size varies depending on the C compiler version and compile options.

ROM, RAM and Stack Code Sizes (1/3)

Device	Category		Memory Used: Renesas Compiler: With Parameter Checking	Memory Used: Renesas Compiler: Without Parameter Checking	Remarks
RX110, RX111	ROM	1 IRQ	777 bytes	527 bytes	
		8 IRQs	1377 bytes	1127 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		48 bytes	46 bytes	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.
RX130, RX230, RX24T, RX24U	ROM	1 IRQ	777 bytes	527 bytes	
		8 IRQs	1366 bytes	1116 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		48 bytes	36 bytes	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.

ROM, RAM and Stack Code Sizes (2/3)					
Device	Category		Memory Used		Remarks
			Renesas Compiler		
			With Parameter Checking	Without Parameter Checking	
RX23T	ROM	1 IRQ	767 bytes	521 bytes	
		6 IRQs	1207 bytes	961 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	10 bytes	4 bytes	
		6 IRQs	30 bytes	24 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		48 bytes	36 bytes	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.
RX23W	ROM	1 IRQ	696 bytes	448 bytes	
		6 IRQs	986 bytes	738 bytes	ROM size is the sum of Program, Constant, and Initialized data.
	RAM	1 IRQ	12 bytes	4 bytes	
		6 IRQs	32 bytes	24 bytes	RAM size is the sum of Uninitialized data, Data, Stack and Others.
	Maximum stack usage		84 bytes	76 bytes	Maximum stack usage is the sum of the maximum size of peripheral API and maximum stack size of peripheral interrupt.
RX64M, RX65N, RX66T, RX71M	ROM	1 IRQ	892 bytes	640 bytes	
		16 IRQs	2180 bytes	1928 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	20 bytes	4 bytes	
		16 IRQs	80 bytes	64 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		48 bytes	40 bytes	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.

ROM, RAM and Stack Code Sizes (3/3)					
Device	Category		Memory Used		Remarks
			Renesas Compiler		
			With Parameter Checking	Without Parameter Checking	
RX113, RX231	ROM	1 IRQ	777 bytes	529 bytes	
		8 IRQs	1366 bytes	1118 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		48 bytes	36 bytes	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.
RX72T	ROM	1 IRQ	892 bytes	640 bytes	
		16 IRQs	2181 bytes	1929 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	20 bytes	4 bytes	
		16 IRQs	80 bytes	64 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		48 bytes	40 bytes	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.
RX72M	ROM	1 IRQ	879 bytes	620 bytes	
		16 IRQs	2160 bytes	1901 bytes	ROM size is the sum of Program, Constant, and Initialized data.
	RAM	1 IRQ	20 bytes	4 bytes	
		16 IRQs	80 bytes	64 bytes	RAM size is the sum of Uninitialized data, Data, Stack and Others.
	Maximum stack usage		84 bytes	80 bytes	Maximum stack usage is the sum of the maximum size of peripheral API and maximum stack size of peripheral interrupt.

ROM, RAM and Stack Code Sizes (1/3)					
Device	Category		Memory Used		Remarks
			GCC		
			With Parameter Checking	Without Parameter Checking	
RX110, RX111	ROM	1 IRQ	1532 bytes	1140 bytes	
		8 IRQs	2592 bytes	2200 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		-	-	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.
RX130, RX230, RX24T, RX24U	ROM	1 IRQ	1532 bytes	1140 bytes	
		8 IRQs	2592 bytes	2200 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		-	-	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.

ROM, RAM and Stack Code Sizes (2/3)					
Device	Category		Memory Used		Remarks
			GCC		
			With Parameter Checking	Without Parameter Checking	
RX23T	ROM	1 IRQ	1532 bytes	1140 bytes	
		6 IRQs	4048 bytes	3656 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	20 bytes	4 bytes	
		6 IRQs	80 bytes	64 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		-	-	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.
RX64M, RX65N, RX66T, RX71M	ROM	1 IRQ	1772 bytes	1380 bytes	
		16 IRQs	2288 bytes	1904 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	8 bytes	4 bytes	
		16 IRQs	28 bytes	24 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		-	-	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.

ROM, RAM and Stack Code Sizes (3/3)					
Device	Category		Memory Used		Remarks
			GCC		
			With Parameter Checking	Without Parameter Checking	
RX113, RX231	ROM	1 IRQ	1540 bytes	1148 bytes	
		8 IRQs	2600 bytes	2216 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		-	-	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.
RX72T	ROM	1 IRQ	1772 bytes	1380 bytes	
		16 IRQs	4048 bytes	3656 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	20 bytes	4 bytes	
		16 IRQs	80 bytes	64 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		-	-	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.
RX72M	ROM	1 IRQ	1892 bytes	1484 bytes	
		16 IRQs	4168 bytes	3760 bytes	ROM size is the sum of Program, Constant, and Initialized data.
	RAM	1 IRQ	20 bytes	4 bytes	
		16 IRQs	80 bytes	64 bytes	RAM size is the sum of Uninitialized data, Data, Stack and Others.
	Maximum stack usage		-	-	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.

ROM, RAM and Stack Code Sizes (1/3)					
Device	Category		Memory Used		Remarks
			IAR Compiler		
			With Parameter Checking	Without Parameter Checking	
RX110, RX111	ROM	1 IRQ	1120 bytes	792 bytes	
		8 IRQs	1676 bytes	1348 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		152 bytes	152 bytes	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.
RX130, RX230, RX24T, RX24U	ROM	1 IRQ	1120 bytes	792 bytes	
		8 IRQs	1672 bytes	1344 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		152 bytes	152 bytes	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.

ROM, RAM and Stack Code Sizes (2/3)					
Device	Category		Memory Used		Remarks
			IAR Compiler		
			With Parameter Checking	Without Parameter Checking	
RX23T	ROM	1 IRQ	1112 bytes	784 bytes	
		6 IRQs	1504 bytes	1176 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	8 bytes	4 bytes	
		6 IRQs	28 bytes	24 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		152 bytes	152 bytes	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.
RX64M, RX65N, RX66T, RX71M	ROM	1 IRQ	1338 bytes	1039 bytes	
		16 IRQs	2523 bytes	2289 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	20 bytes	4 bytes	
		16 IRQs	80 bytes	64 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		168 bytes	168 bytes	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.

ROM, RAM and Stack Code Sizes (3/3)					
Device	Category		Memory Used IAR Compiler With Parameter Checking	Memory Used IAR Compiler Without Parameter Checking	Remarks
RX113, RX231	ROM	1 IRQ	1120 bytes	792 bytes	
		8 IRQs	1674 bytes	1346 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		152 bytes	152 bytes	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.
RX72T	ROM	1 IRQ	1319 bytes	984 bytes	
		16 IRQs	2504 bytes	2167 bytes	ROM size can be calculated with the following formula: ROM size for 1 channel + (58 bytes x number of additional channels)
	RAM	1 IRQ	20 bytes	4 bytes	
		16 IRQs	80 bytes	64 bytes	RAM size can be calculated with the following formula: RAM size for 1 channel + (4 bytes x number of additional channels)
	Maximum stack usage		168 bytes	168 bytes	Nested interrupts are prohibited, so the maximum value when one channel is used is listed.
RX72M	ROM	1 IRQ	1320 bytes	992 bytes	
		16 IRQs	2505 bytes	2174 bytes	ROM size is the sum of readonly code memory and readonly data memory.
	RAM	1 IRQ	20 bytes	4 bytes	
		16 IRQs	80 bytes	64 bytes	RAM size is readwrite data memory.
	Maximum stack usage		172 bytes	172 bytes	The stack size is the sum of interrupt, Program entry and Uncalled function.

2.9 Parameters

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

2.9.1 Special Data Types

To provide strong type checking and reduce errors, many parameters used in API functions require arguments to be passed using the provided type definitions. Allowable values are defined in the public interface file *r_irq_rx_if.h*. The following special types have been defined:

Enumeration of IRQ numbers

Type: irq_number_t

Macro: IRQ_NUM_n

Values (n): (all MCUs) 0 through 7 **(and for RX MCUs with 16 IRQs)** 8 through 15

Example: IRQ_NUM_2

IRQ control command codes

Type: irq_cmd_t

Values: IRQ_CMD_SET_PRIO, IRQ_CMD_SET_TRIG

IRQ interrupt priority settings.

Type: irq_prio_t

Macro: IRQ_PRI_n

Value s (n): 0 through 15

Example: IRQ_PRI_3

IRQ trigger mode settings

Type: irq_trigger_t

Values: IRQ_TRIG_LOWLEV, IRQ_TRIG_FALLING, IRQ_TRIG_RISING, IRQ_TRIG_BOTH_EDGE

Handle

Type: irq_handle_t

Values: User provides pointer to storage for this type for a handle. Handle value is automatically assigned by R_IRQ_Open function

2.10 Return Values

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

Return Type: irq_err_t

Values:	Cause
IRQ_SUCCESS	Function completed without errors
IRQ_ERR_BAD_NUM	Invalid IRQ number was passed
IRQ_ERR_NOT_OPENED	IRQ not yet opened. Function cannot be completed.
IRQ_ERR_NOT_CLOSED	IRQ still open from previous open.
IRQ_ERR_UNKNOWN_CMD	Control command is not recognized.
IRQ_ERR_INVALID_ARG	Argument is not valid for parameter.
IRQ_ERR_INVALID_PTR	Received null pointer; missing required argument.
IRQ_ERR_LOCK	A lock procedure failed.

2.11 Callback Function

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

The callback function is specified by storing the address of the user function in the “void (*const pcallback)(void *pargs))” structure member (see 2.9, Parameters). When the callback function is called, the variable which stores the channel number is passed as the argument.

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

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

```
void my_irq_callback(void * pdata)
{
    irq_number_t  my_triggered_irq_number;

    my_triggered_irq_number = *((irq_number_t *)pdata);
    ...
}
```

2.12 Adding the FIT Module to Your Project

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

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

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

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

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

2.13 “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_IRQ_Open()

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

Format

```
irq_err_t       R_IRQ_Open (
        irq_number_t    irq_number,
                irq_trigger_t     trigger,
                irq_prio_t        priority,
        irq_handle_t     *phandle,
        void  (*const  pcallback)(void *pargs)
)
```

Parameters

irq_number_t irq_number
Number of the IRQ to be initialized.

irq_trigger_t trigger
Enumerated type for trigger type: low level, rising edge, falling edge, both edges.

irq_prio_t priority
Enumerated priority level setting for the IRQ.

irq_handle_t phandle
Pointer to a location for handle for IRQ. Handle value will be set by this function.

pcallback
Pointer to function called from interrupt.

Return Values

```
[IRQ_SUCCESS]          /* Successful; IRQ initialized                              */
[IRQ_ERR_BAD_NUM]      /* IRQ number is invalid or unavailable                     */
[IRQ_ERR_NOT_CLOSED]   /* IRQ currently in operation; Perform R_IRQ_Close() first  */
[IRQ_ERR_INVALID_PTR]  /* phandle pointer is NULL                                  */
[IRQ_ERR_INVALID_ARG]  /* An invalid argument value was passed.                    */
[IRQ_ERR_LOCK]         /* The lock could not be acquired.                          */
```

Properties

Prototyped in file “r_irq_rx_if.h”.

Description

The Open function is responsible for preparing an IRQ for operation. After completion of the Open function the IRQ shall be enabled and ready to service interrupts. This function must be called once prior to calling any other IRQ API functions. Once successfully completed, the status of the selected IRQ will be set to "open". After that this function should not be called again for the same IRQ without first performing a "close" by calling R_IRQ_Close().

Example

```
/* Allocate a handle that will be used for access to other IRQ API functions. */
irq_handle_t      my_handle;
irq_err_t         result;

/* Prepare IRQ0 for use. Trigger interrupt on falling edge, priority level 3. */
result = R_IRQ_Open (IRQ_NUM_0,
                     IRQ_TRIG_FALLING,
                     IRQ_PRI_3,
                     &my_handle,
                     &my_callback);

if(IRQ_SUCCESS != result)
{
    // Handle the error.
}
```

Special Notes:

None.

R_IRQ_Control()

The Control function is responsible for handling special hardware or software operations for the IRQ.

Format

```
irq_err_t       R_IRQ_Control (
        irq_handle_t  const     handle,
        irq_cmd_t               const cmd,
        void                    *pcmd_data
)
```

Parameters

irq_handle_t const handle

Handle for the IRQ.

irq_cmd_t cmd

Enumerated command codes:

IRQ_CMD_SET_PRIO - Changes the interrupt priority level.

IRQ_CMD_SET_TRIG - Changes the interrupt triggering mode.

pcmd_data

Pointer to the command-data structure parameter of type void that is used to reference the location of any data specific to the command that is needed for its completion.

Return Values

```
[IRQ_SUCCESS]           /* Command successfully completed.                          */
[IRQ_ERR_NOT_OPENED]    /* The IRQ has not been opened.  Perform R_IRQ_Open() first */
[IRQ_ERR_BAD_NUM]       /* IRQ number is invalid or unavailable                     */
[IRQ_ERR_UNKNOWN_CMD]   /* Control command is not recognized.                       */
[IRQ_ERR_INVALID_PTR]   /* pcmd_data pointer or handle is NULL                      */
[IRQ_ERR_INVALID_ARG]   /* An element of the pcmd_data structure contains an invalid value. */
[IRQ_ERR_LOCK]          /* The lock could not be acquired                           */
```

Properties

Prototyped in file “r_irq_rx_if.h”

Description

This function is responsible for handling special hardware or software operations for the IRQ . It takes an IRQ handle to identify the selected IRQ, an enumerated command value to select the operation to be performed, and a void pointer to a location that contains information or data required to complete the operation. This pointer must point to storage that has been type-cast by the caller for the particular command using the appropriate type provided in "r_irq_rx_if.h".

Example

```
/* Change trigger mode to rising edge. */
irq_trigger_t     my_trig_mode = IRQ_TRIG_RISING;

result = R_IRQ_Control(my_handle, IRQ_CMD_SET_TRIG, &my_trig_mode);
```

```
/* Change the priority. */
irq_prio_t my_priority = IRQ_PRI_10;

result = R_IRQ_Control(my_handle, IRQ_CMD_SET_PRIO, &my_priority);
```

Special Notes:
None.

R_IRQ_Close()

Fully disables the IRQ designated by the handle.

Format

```
irq_err_t       R_IRQ_Close (
        irq_handle_t    handle
)
```

Parameters

irq_handle_t handle
Handle for the IRQ.

Return Values

[IRQ_SUCCESS]	*/* Successful; IRQ closed*	**/*
[IRQ_ERR_NOT_OPENED]	*/* The IRQ has not been opened so closing has no effect.*	**/*
[IRQ_ERR_BAD_NUM]	*/* IRQ number is invalid or unavailable*	**/*
[IRQ_ERR_INVALID_PTR]	*/* A required pointer argument is NULL*	**/*

Properties

Prototyped in file “r_irq_rx_if.h”

Description

This function frees the IRQ by clearing its assignment to a port, and disables the associated interrupts. The IRQ handle is modified to indicate that it is no longer in the 'open' state. The IRQ cannot be used again until it has been reopened with the R_IRQ_Open function. If this function is called for an IRQ that is not in the open state then an error code is returned.

Example

```
/* */
irq_err_t result;

result = R_IRQ_Close(my_handle);
```

Special Notes:

None.

R_IRQ_ReadInput()

This function reads the current level of the pin assigned to the specified IRQ.

Format

```
irq_err_t       R_IRQ_ReadInput (
        irq_handle_t const      handle,
        uint8_t                 *plevel
)
```

Parameters

irq_handle_t const handle
Handle for the IRQ.

uint8_t plevel
Pointer to location where the input pin state can be returned.

Return Values

```
[IRQ_SUCCESS]           /* Operation successfully completed.                    */
[IRQ_ERR_NOT_OPENED]    /* The IRQ has not been opened.  Perform R_IRQ_Open() first */
[IRQ_ERR_BAD_NUM]       /* IRQ number is invalid or unavailable                 */
[IRQ_ERR_INVALID_PTR]   /* plevel data pointer or handle is NULL                */
```

Properties

Prototyped in file “r_irq_rx_if.h”

Description

This function reads the current level of the pin assigned to the specified IRQ. This is a realtime read which may indicate a different value than the level that initially triggered an interrupt. One example use is for cases in which a switch has triggered an interrupt and then needs to be polled for debounce.

Example

```
/* What logic level does the input currently see? */
uint8_t irq_pin_level;

result = R_IRQ_ReadInput(my_handle, (uint8_t*)&irq_pin_level);
```

Special Notes:

None.

R_IRQ_InterruptEnable()

This function enables or disables the ICU interrupt for the specified IRQ.

Format
```
irq_err_t       R_IRQ_InterruptEnable (
        irq_handle_t const      handle,
        bool                    enable
)
```

Parameters
irq_handle_t const handle
Handle for the IRQ.

bool enable
true = enable the interrupt.
false = disable interrupt.

Return Values
[IRQ_SUCCESS]	*/* Operation successfully completed.*	**/*
[IRQ_ERR_NOT_OPENED]	*/* The IRQ has not been opened. Perform R_IRQ_Open() first*	**/*
[IRQ_ERR_BAD_NUM]	*/* IRQ number is invalid or unavailable*	**/*
[IRQ_ERR_INVALID_PTR]	*/* handle is NULL*	**/*

Properties
Prototyped in file “r_irq_rx_if.h”

Description
The function enables or disables the ICU interrupt for the IRQ specified by the handle argument. This function is potentially called frequently and is expected to execute quickly.

Example
```
irq_err_t result;

/* Enable interrupt */
result = R_IRQ_InterruptEnable (my_handle, true);

/* Disable interrupt */
result = R_IRQ_InterruptEnable (my_handle, false);
```

Special Notes:
None.

R_IRQ_GetVersion()

This function returns the driver version number at runtime.

Format
uint32_t R_IRQ_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_irq_rx_if.h”.

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

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

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

version = R_IRQ_GetVersion();

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

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

Special Notes:
None.

4. Pin Setting

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

When performing the Pin Setting in the e2 studio, the Pin Setting feature of the FIT Configurator or the Smart Configurator can be used. When using the Pin Setting feature, a source file is generated according to the option selected in the Pin Setting window in the FIT Configurator or the Smart Configurator. Pins are configured by calling the function defined in the source file.

5. Demo Projects

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

5.1 irq_demo_rskrx113, irq_demo_rskrx231, irq_demo_rskrx64m, irq_demo_rskrx71m, irq_demo_rskrx65n, irq_demo_rskrx65n_2m

These are the demo programs for the IRQ FIT module, designed for the Renesas RSKRX113, RSKRX231, RSKRX64M, RSKRX71M, RSKRX65N and RSKRX65N-2MB demo boards. The programs demonstrate how to use the R_IRQ_Open API call to configure a port bit as an interrupt input and how to set up a callback function to handle the interrupt. They also demonstrate how to use the R_IRQ_Control API call to reconfigure the interrupt trigger conditions, how to use the R_IRQ_ReadInput API call and how to dereference the callback argument to obtain the interrupt number. IRQ2 (IRQ4 on the RX231, IRQ9 on the RX65N, IRQ13 on the RX65N-2MB) is chosen as the interrupt and is used to detect key presses on SW2. All three demo programs operate the same, once the code is compiled and down-loaded to the target board and running, SW2 can be pressed to cause IRQ2 (IRQ4 on the RX231, IRQ9 on the RX65N, IRQ13 on the RX65N-2MB) interrupts to occur. LED3 will turn on in response to a falling edge when SW2 is pressed and, will turn off in response to a rising edge when SW2 is released.

5.2 Adding a Demo to a Workspace

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

5.3 Downloading Demo Projects

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

6. Appendices

6.1 Confirmed Operation Environment

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

Table 6.1 Confirmed Operation Environment (Rev.3.20)

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

Table 6.2 Confirmed Operation Environment (Rev.3.10)

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

Table 6.3 Confirmed Operation Environment (Rev.3.00)

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

Table 6.4 Confirmed Operation Environment (Rev.2.40)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.40
Board used	Renesas Starter Kit for RX72T (product No.: RTK5572Txxxxxxxxx)

Table 6.5 Confirmed Operation Environment (Rev.2.31)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.31
Board used	Renesas Starter Kit for RX66T (product No.: RTK50566T0SxxxxxBE) Renesas Starter Kit+ for RX 65N-2MB (product No.: RTK50565N2CxxxxxBR) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308CxxxxxBR)

Table 6.6 Confirmed Operation Environment (Rev.2.30)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.30
Board used	Renesas Starter Kit for RX66T (product No.: RTK50566T0SxxxxxBE) Renesas Starter Kit+ for RX 65N-2MB (product No.: RTK50565N2CxxxxxBR) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308CxxxxxBR)

Table 6.7 Confirmed Operation Environment (Rev.2.21)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 6.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.21
Board used	Renesas Starter Kit+ for RX 65N-2MB (product No.: RTK50565N2CxxxxxBR) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308CxxxxxBR)

Table 6.8 Confirmed Operation Environment (Rev.2.20)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 6.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.2.20
Board used	Renesas Starter Kit+ for RX 65N-2MB (product No.: RTK50565N2CxxxxxBR) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308CxxxxxBR)

6.2 Troubleshooting

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

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

- Using CS+:

  Application note "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)"
- Using e^2 studio:

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

When using a 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_irq_rx module.

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

(3) Q: I have added the FIT module to the project and built it. Then I got an error for when the configuration setting is wrong.

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

(4) Q: Interrupt does not occur.

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

7. Reference Documents

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

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

User's Manual: Development Tools
RX Family Compiler CC-RX User's Manual (R20UT3248)
The latest versions can be downloaded from the Renesas Electronics website.

Related Technical Updates

This module reflects the content of the following technical updates.

None

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Nov 15, 2013	--	First edition issued
1.20	April 10, 2014	1	Added RX110 to list of tested MCUs
		4	Added sections 1.2, 1.3 callback functions usage
			Updated Colophon to 4.0
1.30	July 23, 2014	Various	Added mention of RX64M in supported MCUs, added code size information, updated format.
1.40	Jan 08,2015	Various	Updated to current template.
		—	Added support for the RX113 Group.
1.50	Mar 24,2015	—	Added support for the RX71M Group.
1.60	June 30,2015	—	Added support for the RX231 Group.
1.70	Sep 30,2015	—	Added support for the RX23T Group.
		—	Fixed typo in the Return value: IRQ_ERR_NOT_OPEN -> IRQ_ERR_NOT_OPENED
		7	Updated code sizes in 2.8 Code Size for the RX23T Group.
1.80	Oct 1, 2015	—	Added support for the RX130 Group.
		7	Updated code sizes in 2.8 Code Size for the RX130 Group.
1.90	Dec 1, 2015	—	Added support for the RX230 and the RX24T Groups.
		1, 9	Changed the document number for the “Board Support Package Firmware Integration Technology Module” application note.
		5	Changed the description in section 2.
		7	Updated 2.8 Code Size for the RX230 and the RX24T Groups.
		17	Added “4. Demo Projects”.
1.91	June 15, 2016	17	Added RSKRX64M to “4. Demo Projects”.
		18	Added “Related Technical Updates”.
2.00	Oct 1, 2016	—	Added support for the RX65N Group.
		7, 8	Changed 2.8 Code Size for the tabular format of Code Size. Updated 2.8 Code Size for the RX65N Group.
		18	Added “4. Pin Setting”.
2.10	Feb 28, 2017	—	Added support for the RX24U Group.
		7	Updated 2.8 Code Size for the RX24U Group.
2.20	July 21, 2017	—	Added support for the RX130-512KB and RX65N-2MB.
		5	Added RXC v2.07.00 to “2.4 Supported Toolchains”.
		6	Added “2.5 Interrupt Vector”
		11	Added “2.12 Adding the FIT Module to Your Project”.
		19	Added “4. Pin Setting”.
2.21	Oct 31, 2017	20	Added RSKRX65N, RSKRX65N-2MB to “5. Demo Projects”
		20	Added 5.3 Downloading Demo Project
		21	Added 6. Appendices

2.30	Sep 28, 2018	—	Changed the structure of macros in r_irq_rx_config.h, r_irq_rx_private.h
		1, 7	Added support for the RX66T.
		4	Added instruction on how to comment/uncomment macros when Smart Configurator or FIT Configurator is used.
		8	In “Configuration options in r_irq_rx_config.h” table, removed IRQ_CFG_USE_IRQn 0, changed IRQ_PORT_IRQn to IRQ_PORT_IRQn_PORT changed IRQ_PORT_BIT_IRQn to IRQ_PORT_IRQn_BIT
		10	Added code size corresponding to RX66T
		26	6.1 Confirmed Operation Environment: Added table for Rev.2.30
2.31	Nov 16, 2018	—	Added document number in XML
		26	Changed Renesas Starter Kit Product No for RX66T. Added table for Rev.2.31
2.40	Feb 01, 2019	Program	Added support for the RX72T
		1, 7, 11	Added support for the RX72T
		16-23	Removed ‘Reentrant’ description in each API function.
		26	6.1 Confirmed Operation Environment: Added table for Rev.2.40
3.00	May.20.19	—	Supported the following compilers: - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		1	Deleted the RX210, and RX63N in Target Devices for end of update these devices. Added the section of Target compilers. Deleted related documents.
		6	2.2 Software Requirements Requires r_bsp v5.20 or higher
		9-14	Updated the section of 2.8 Code Size
		7	2.4 Interrupt Vector: Deleted RX210, RX63N
		32	Table 6.1 Confirmed Operation Environment: Added table for Rev.3.00
		36	Deleted the section of Website and Support.
		Program	Changed bellow for support GCC and IAR compiler: 1. Deleted the inline expansion of the R_IRQ_GetVersion function. 2. Replaced nop with the intrinsic functions of BSP. 3. Replaced the declaration of interrupt functions with the macro definition of BSP.
3.10	Jun.28.19	1, 7	Added support for RX23W
		10	Added code size corresponding to RX23W
		32	6.1 Confirmed Operation Environment: Added Table for Rev.3.10
		Program	Added support for RX23W
3.20	Aug.15.19	1, 7	Added support for RX72M
		11, 14, 17	Added code size corresponding to RX72M
		32	6.1 Confirmed Operation Environment: Added Table for Rev.3.20 Table 6.2: Corrected board name for RX23W
		Program	Added support for RX72M

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

2. Processing at power-on

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

3. Input of signal during power-off state

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

4. Handling of unused pins

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

5. Clock signals

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

6. Voltage application waveform at input pin

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

7. Prohibition of access to reserved addresses

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

8. Differences between products

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

Notice

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

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

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

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

(Rev.4.0-1 November 2017)

Corporate Headquarters

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

Contact information

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

Trademarks

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