RX Family

IRQ Module Using Firmware Integration Technology

R01AN1668EJ0200
Rev. 2.00
Oct 1, 2016

Introduction

The RX Family MCUs supported by this module have interrupt vectors that can be triggered by external pins. These interrupts are named IRQ0 through IRQ(n). An interrupt request may optionally be generated by either a change of state on the pin from high to low (falling edge), from low to high (rising edge), or from a fixed low level. A digital filter function can also be assigned. These optional settings can be individually assigned to each of the available IRQ interrupts.

This document describes the IRQ Support Module API for the supported RX class MCUs. Software architecture, system interfaces and usage details are provide here to facilitate integration of the IRQ module into a User's application.

Target Device

The following is a list of devices that are currently supported by this API:

- **RX110, RX111, RX113 Groups**
- **RX130 Group**
- **RX210 Group**
- **RX230 Group**
- **RX231 Group**
- **RX23T Group**
- **RX24T Group**
- **RX63N Group**
- **RX64M Group**
- **RX65N 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.

Related Documents

- Firmware Integration Technology User’s Manual (R01AN1833)
- Board Support Package Firmware Integration Technology Module (R01AN1685)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723)
- Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)

Contents

1. Overview

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.

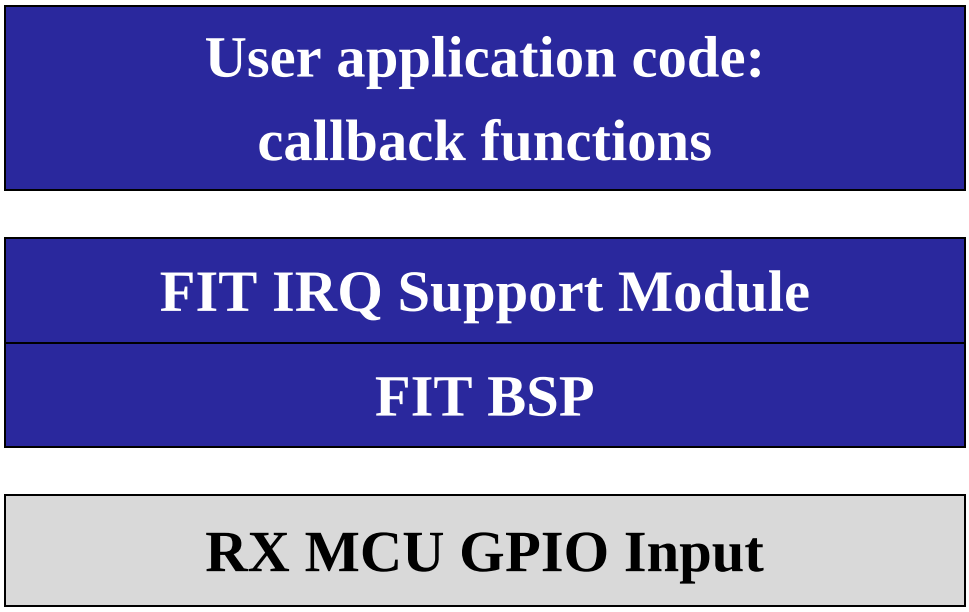

Figure 1 : Example Figure Showing Project Layers

1.1 Using the Firmware Integration Technology (FIT) IRQ support 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. See Section 2.7 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.

1.2 Interrupts

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.

1.3 Callback Functions

The definition of callbacks follows the FIT 1.0 specification rules:

a. Callback functions take one argument. This argument is ‘void * pdata’.

b. Before calling a callback function the function pointer is be checked to be valid. At a minimum the pointer is be checked to be:

i. Non-null

ii. Not equal to the FIT_NO_FUNC macro.

1.3.1 Example callback function prototype declaration

```
void callback(void * pdata)
```

1.3.2 Dereferencing of pdata argument

When the ISR calls your callback function it will pass the pointer to a value containing the IRQ number that triggered the interrupt. This value is of type: **irq_number_t** so the argument passed to the callback will be of type (**irq_number_t *).** Since FIT callbacks take a void pointer, you will need to type-cast the pointer so that it can be dereferenced.

Example:

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

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

2. API Information

The sample code in this application note has been run and confirmed under the following conditions.

2.1 Hardware Requirements

This driver requires that your MCU supports the following peripheral(s):

- 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 packages:

- Renesas Board Support Package (r_bsp). 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 Limitations

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

2.4 Supported Toolchains

This driver is tested and working with the following toolchains:

- Renesas RX Toolchain v.2.02.00 (RX110, RX111, RX113, RX210, RX231, RX63N, RX64M, RX71M)
- Renesas RX Toolchain v.2.03.00 (RX130, RX230, RX23T, RX24T)
- Renesas RX Toolchain v.2.05.00 (RX65N)

2.5 Header Files

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

Build-time configuration options are selected or defined in the file "r_irq_rx_config.h”.

Both of these files should be included by the user’s application.

2.6 Integer Types

This project uses ANSI C99 “Exact width integer types” in order to make the code clearer and more portable. 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.

All configurable options that can be set at build time are located in the file "r_irq_rx_config.h". A summary of these settings are provided in **Table 1**.

Configuration options in r_irq_rx_config.h		
Equate	**Default Value**	**Description**
IRQ_CFG_USE_IRQn	0	Where n is the number for each IRQ. If defined as 1, IRQn is enabled. Disabled = 0. Code for unused IRQs will be excluded from the build.
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
IRQ_PORT_BIT_IRQn
	—	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*  (PORTm)`
		(where m is the port number and the IRQ number replaces *)
		`#define IRQ_PORT_BIT_IRQ* (IRQ_BITn)`
		(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

Table 1: Info about the configuration

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 2.4, 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

Device	Category		Memory Used		Remarks
			With Parameter Checking	Without Parameter Checking	
RX110, RX111	ROM	1 IRQ	701 bytes	448 bytes	
		8 IRQs	1107 bytes	854 bytes	
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	
	Maximum stack usage		48 bytes	40 bytes	
RX130, RX230, RX24T	ROM	1 IRQ	701 bytes	450 bytes	
		8 IRQs	1107 bytes	856 bytes	
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	
	Maximum stack usage		48 bytes	40 bytes	
RX113, RX231	ROM	1 IRQ	701 bytes	448 bytes	
		8 IRQs	1107 bytes	854 bytes	
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	
	Maximum stack usage		48 bytes	40 bytes	

ROM, RAM and Stack Code Sizes					
Device	**Category**		**Memory Used**		**Remarks**
			With Parameter Checking	**Without Parameter Checking**	
RX210	ROM	1 IRQ	701 bytes	448 bytes	
		8 IRQs	1107 bytes	854 bytes	
	RAM	1 IRQ	12 bytes	4 bytes	
		8 IRQs	40 bytes	32 bytes	
	Maximum stack usage		48 bytes	40 bytes	
RX23T	ROM	1 IRQ	961 bytes	444 bytes	
		6 IRQs	981 bytes	734 bytes	
	RAM	1 IRQ	10 bytes	4 bytes	
		6 IRQs	30 bytes	24 bytes	
	Maximum stack usage		48 bytes	40 bytes	
RX63N, RX64M, RX65N, RX71M	ROM	1 IRQ	815 bytes	563 bytes	
		16 IRQs	1685 bytes	1433 bytes	
	RAM	1 IRQ	20 bytes	4 bytes	
		16 IRQs	80 bytes	64 bytes	
	Maximum stack usage		48 bytes	40 bytes	

Table 2: ROM, RAM and Stack Code Sizes

2.9 API Data Types

This section details the data structures that are used with the driver's 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 shows the different values API functions can return. This return type is defined in "r_irq_rx_if.h".

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 Adding a FIT Module to Your Project

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

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

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

To add the FIT module using the plug-in, refer to chapter 2. "Adding FIT Modules to e^2 studio Projects Using FIT Plug-In" in the "Adding Firmware Integration Technology Modules to Projects" application note (R01AN1723).

To add the FIT module manually, refer to chapter 3. "Adding FIT Modules to e^2 studio Projects Manually" in the "Adding Firmware Integration Technology Modules to Projects (R01AN1723)"

When using the FIT module, the BSP FIT module also needs to be added. For details on the BSP FIT module, refer to the "Board Support Package Module Using Firmware Integration Technology" application note (R01AN1685).

3. API Functions

3.1 Summary

The following functions are included in this design:

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.

3.2 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
Number of the IRQ to be initialized
trigger
Enumerated type for trigger type: low level, rising edge, falling edge, both edges.
priority
Enumerated priority level setting for the IRQ.
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().

Reentrant

This function is not designed for reentrant operation on the same IRQ, however it can be reentered for a different IRQ. It protects vulnerable portions of its code from reentrant operation by use of FIT BSP hardware lock functions.

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

3.3 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

handle

Handle for the IRQ

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" .

Reentrant

Yes if locking is enabled. Reentrancy to operate on the same IRQ is rejected by BSP lock. If locking is disabled then reentrancy is only safe for operating on a different IRQ. If locking is enabled special care should be taken to prevent deadlock. Caller should check return value and should handle a locked return status by permitting the process owning the lock to complete.

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

3.4 R_IRQ_Close()

Fully disables the IRQ designated by the handle.

Format

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

Parameters

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.

Reentrant

Yes, however unintentional reentrancy for same IRQ number may result in an unexpected IRQ_ERR_NOT_OPENED return code.

Example

```
/* */
irq_err_t result;

result = R_IRQ_Close(my_handle);
```

Special Notes:

None

3.5 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

handle

Handle for the IRQ

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 real-time 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.

Reentrant

Yes

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

3.6 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

handle
Handle for the IRQ
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.

Reentrant

Yes, however unintentional reentrancy for the same IRQ number is likely to have unintended consequence to application operation.

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

3.7 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.

Reentrant

Yes

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

Regarding the pin setting used for this module, it is strictly recommended to perform the pin setting after calling the R_IRQ_Open () function.

5. Demo Projects

Demo projects are complete stand-alone programs. They include function main() that utilizes the module and its dependent modules (e.g. r_bsp). This FIT module has the following two demo projects:

5.1 irq_demo_rskrx113, irq_demo_rskrx231, irq_demo_rskrx64m, irq_demo_rskrx71m

These are the demo programs for the IRQ FIT module, designed for the Renesas RSKRX113, RSKRX231, RSKRX64M and RSKRX71M 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) 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) 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”.

Related Technical Updates

This module reflects the content of the following technical updates.

None

Website and Support

Renesas Electronics Website

http://www.renesas.com/

Inquiries

http://www.renesas.com/contact

Revision Record

Rev.	Date	Description	
		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”.

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. 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended 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; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor 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 damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate 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. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may 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 should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be 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, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 5.0