RX Family

LONGQ Module Using Firmware Integration Technology

Introduction

This module provides functions for creating and maintaining uint32_t circular buffers.

Target Device

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

- **All RX MCUs**

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 “5.1 Confirmed Operation Environment".

Contents

1. Overview

The Long Queue (LONGQ) module provides basic circular buffer services for buffers provided by the application.

The module allocates a Queue Control Block (QCB) for each buffer passed to the Open() function. A QCB maintains the buffer's "in" and "out" indexes for adding and removing data from the queue. The Queue Control Blocks can be allocated statically at compile time or dynamically at run time using malloc. An equate in config.h determines whether they are statically or dynamically created. If they are statically allocated, an additional equate is utilized which specifies the maximum number of buffers/queues to be supported.

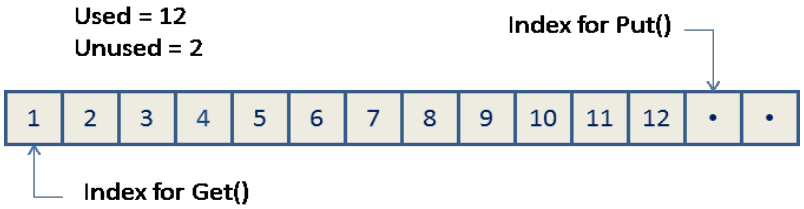
There is one control block per buffer. When an R_LONGQ_Open() is performed, a pointer to the application's buffer and its length are passed in, and a pointer to a QCB is provided. This pointer, which is called a Handle, is then passed to all of the other API functions. The functions then operate on the queue referenced by this Handle. Because there is no global or static data shared between the queues, the API functions are re-entrant for different queues.

This module does not make use of any interrupts. If a queue can be modified at both the interrupt and application level, it is up to the application to ensure that the appropriate related interrupt is disabled whenever the queue is being accessed. Similarly, if the queue is accessed by tasks of different priorities, it is up to the user to prevent task switching or to utilize a mutex or semaphore to reserve the queue.

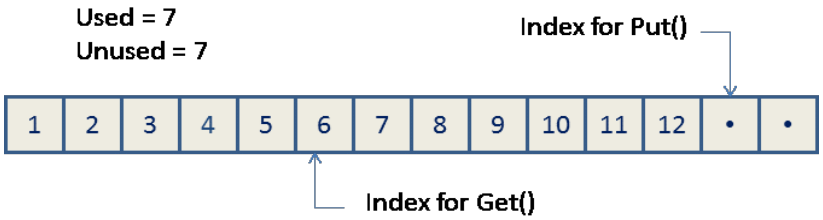
1.1 Using the LONGQ Module

The following illustrates a queue's behavior with API calls:

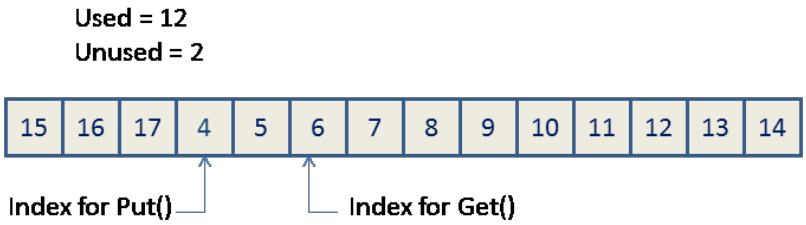
```
#define BUFSIZE 14

uint32_t      my_buf[BUFSIZE];
longq_hdl_t   my_que;
longq_err_t   err;
uint32_t      data,i;

err = R_LONGQ_Open(my_buf, BUFSIZE, false, &my_que);

// add 12 entries to queue
for (i=0; i < 12; i++)
{
    R_LONGQ_Put(my_que, i+1);
}
```



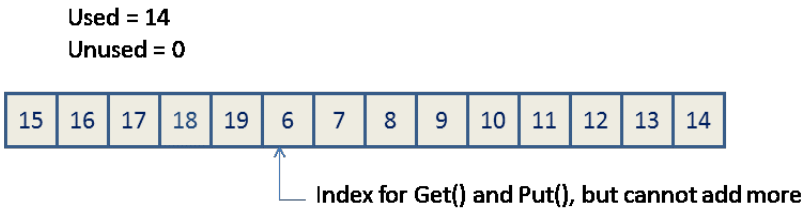

```
/* remove 5 entries from queue */
R_LONGQ_Get(my_que, &data);     // data = 1
R_LONGQ_Get(my_que, &data);     // data = 2
R_LONGQ_Get(my_que, &data);     // data = 3
R_LONGQ_Get(my_que, &data);     // data = 4
R_LONGQ_Get(my_que, &data);     // data = 5
```



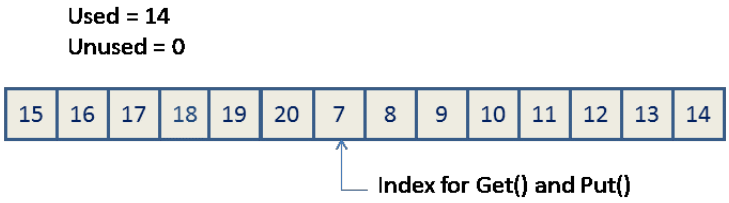

```
/* add 5 entries to queue */
R_LONGQ_Put(my_que, 13);
R_LONGQ_Put(my_que, 14);
R_LONGQ_Put(my_que, 15);
R_LONGQ_Put(my_que, 16);
R_LONGQ_Put(my_que, 17);
```



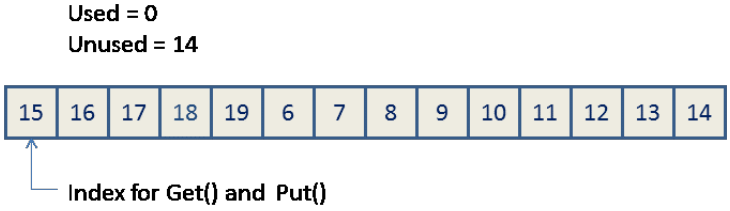

```
/* attempt to add 3 more entries to queue */
err = R_LONGQ_Put(my_que, 18);
err = R_LONGQ_Put(my_que, 19);
err = R_LONGQ_Put(my_que, 20);  // err=LONGQ_ERR_QUEUE_FULL
```




```
/* Note: If Open() was called with ignore_overflow = true, last Put() would
succeed */
```




```
/* flush queue */
R_LONGQ_Flush(my_que);
```



2. API Information

This Driver API follows the Renesas API naming standards.

2.1 Hardware Requirements

No hardware requirements.

2.2 Software Requirements

This driver is dependent upon the following FIT packages:

- Renesas Board Support Package (r_bsp) v3.10.or higher

2.3 Limitations

No software limitations.

2.4 Supported Toolchain

This driver has been confirmed to work with the toolchain listed in 5.1 Confirmed Operation Environment.

2.5 Header Files

Compile time configurable options are located in r_longq\ref\r_longq_config_reference.h. This file should be copied into the r_config subdirectory of the project and renamed to r_longq_config.h. It is this renamed file that should be modified if needed and the original kept as a reference.

All API calls and their supporting interface definitions are located in r_longq\r_longq_if.h. Both this file and r_longq_config.h should be included by the User's application.

2.6 Integer Types

If your toolchain supports C99 then *stdint.h* should be described as shown below. If not, then there should be *typedefs.h* file that is included with your project as defined by the Renesas Coding Standards document.

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

All configurable options that can be set at build time are located in the file “r_longq_config.h”. A summary of these settings are provided in the following table:

Configuration options in *r_longq_config.h*	
#define LONGQ_CFG_PARAM_CHECKING_ENABLE	= 1: Include parameter checking in the build. = 0: Omit parameter checking from the build. = BSP_CFG_PARAM_CHECKING_ENABLE (default): Use the system default setting. Note: Code size can be reduced by excluding parameter checking from the build.
#define LONGQ_CFG_USE_HEAP_FOR_CTRL_BLKS 0	A control block is needed for each queue to maintain in/out indexes. By default, these control blocks are allocated at compile time. To dynamically allocate memory at run time, set this equate to 1.
#define LONGQ_CFG_MAX_CTRL_BLKS 32	Specifies how many control blocks to allocate at compile time. This constant is ignored if LONGQ_CFG_USE_HEAP_FOR_CTRL_BLKS is 1.
#define LONGQ_CFG_PROTECT_QUEUE 0	This #define is used to protect the queue so that it can be accessed from both application and interrupt level without being corrupted. = 1: Use disable interrupt to protect queue. = 0: The queue is not protected. Note 1: Set BSP_CFG_RUN_IN_USER_MODE = 0 when setting LONGQ_CFG_PROTECT_QUEUE = 1. If not, build error would occur.
#define LONGQ_CFG_CRITICAL_SECTION 0	This #define is used to protect critical section of APIs R_LONGQ_Put(), R_LONGQ_Get() so that it can be accessed from both application and interrupt level without being corrupted. = 1: Use disable interrupt to protect critical section. = 0: The critical section is not protected. Note 1: Set BSP_CFG_RUN_IN_USER_MODE = 0 when setting LONGQ_CFG_CRITICAL_SECTION = 1. If not, build error would occur.

2.8 Code Size

The sizes of ROM and RAM of this module are listed below.

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

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

Module Revision: r_longq rev1.90

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00
Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99

GCC for Renesas RX 8.3.0.202102
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 4.20.3
Compiler option: The default settings of the integrated development environment.

Configuration Options: Default settings.

ROM, RAM and Stack Code Sizes

Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
Using Heap for Control Blocks	ROM	277 bytes+ 269 bytes (Note 1)	211 bytes+ 273 bytes (Note 1)	1120 bytes+ 208 bytes (Note 1)	1016 bytes+ 200 bytes (Note 1)	728 bytes+ 290 bytes (Note 1)	656 bytes+ 282 bytes (Note 1)
	RAM	16 bytes for malloc() x Control Blocks		16 bytes for malloc() x Control Blocks		16 bytes for malloc() x Control Blocks	
Using Allocated Control Blocks	ROM	324 bytes+ 269 bytes (Note 1)	257 bytes+ 273 bytes (Note 1)	696 bytes+ 520 bytes (Note 1)	584 bytes+ 520 bytes (Note 1)	424 bytes+ 282 bytes (Note 1)	352 bytes+ 274 bytes (Note 1)
	RAM	1 byte + 16 bytes x LONGQ_CFG_MAX_CTRL_BLKS		16 bytes x LONGQ_CFG_MAX_CTRL_BLKS		16 bytes x LONGQ_CFG_MAX_CTRL_BLKS	

Note 1: Only when LONGQ_CFG_PROTECT_QUEUE and LONGQ_CFG_CRITICAL_SECTION set 1.

2.9 Adding the 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) or (5) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

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

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

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: CS+ (R20AN0470)” for details.

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

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

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

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

The following shows example of description.

while statement example :

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

for statement example :

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

do while statement example :

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

3. API Functions

3.1 Summary

The following functions are included in this design:

Function	Description
R_LONGQ_Open()	Allocates and initializes a queue control block for a buffer provided by the user. Provides a queue handle for use with other API functions.
R_LONGQ_Close()	Releases the queue control block associated with the handle.
R_LONGQ_Put()	Adds an entry to the queue.
R_LONGQ_Get()	Removes the oldest entry from the queue.
R_LONGQ_Flush()	Resets the queue to an empty state.
R_LONGQ_Used()	Provides the number of elements used in the queue.
R_LONGQ_Unused()	Provides the number of elements unused in the queue.
R_LONGQ_GetVersion()	Returns at runtime the module version number.

3.2 Return Values

These are the different error codes API functions can return. The enum is found in r_longq_if.h along with the API function declarations.

```
typedef enum LONGQ_err       // LONGQ API error codes
{
    LONGQ_SUCCESS = 0,
    LONGQ_ERR_NULL_PTR,      // received null ptr; missing required argument
    LONGQ_ERR_INVALID_ARG,   // argument is not valid for parameter
    LONGQ_ERR_MALLOC_FAIL,   // cannot allocate mem for ctrl block;
                             // increase heap
    LONGQ_ERR_NO_MORE_CTRL_BLKS, // no more ctrl blocks, inc LONGQ_MAX_CTRL_BLKS
    LONGQ_ERR_QUEUE_FULL,    // queue full; cannot add another entry
    LONGQ_ERR_QUEUE_EMPTY    // queue empty; no entry to fetch
} longq_err_t;
```

3.3 R_LONGQ_Open()

This function allocates and initializes a queue control block for a buffer provided by the user. A queue handle is provided for use with other API functions.

Format

```
longq_err_t R_LONGQ_Open(uint32_t * const       p_buf,
                         uint16_t const         size,
                         bool const             ignore_overflow,
                         longq_hdl_t * const    p_hdl)
```

Parameters

p_buf
Pointer to buffer.
size
Buffer size in number of elements.
ignore_overflow
true = continue to add entries when queue is full by overwriting oldest entry
false = return error when a Put() call is made and the queue is full
p_hdl
Pointer to a handle for queue (value set here)

Return Values

LONGQ_SUCCESS:	*Successful; queue initialized*
LONGQ_ERR_NULL_PTR:	*Received null ptr; missing required argument*
LONGQ_ERR_INVALID_ARG:	*Size is less than or equal to 1.*
LONGQ_ERR_MALLOC_FAIL:	*Cannot allocate control block. Increase heap size.*
LONGQ_ERR_NO_MORE_CTRL_BLKS:	*Cannot assign control block. Increase LONGQ_MAX_CTRL_BLKS in config.h.*

Properties

Prototyped in file “r_longq_if.h”

Description

This function allocates or assigns a queue control block for the buffer pointed to by *p_buf*. Initializes the queue to an empty state and provides a Handle to its control structure in *p_hdl* which is then used as a queue ID for the other API functions.

Example

```
#define BUFSIZE 80

uint32_t      tx_buf[BUFSIZE];
longq_hdl_t   tx_que;
longq_err_t   longq_err;

longq_err = R_LONGQ_Open(tx_buf, BUFSIZE, false, &tx_que);
```

Special Notes:

None.

3.4 R_LONGQ_Close()

This function releases the queue control block associated with a handle.

Format

longq_err_t R_LONGQ_Close(longq_hdl_t const hdl)

Parameters

hdl
Handle for queue.

Return Values

LONGQ_SUCCESS:	*Successful; control block freed*
LONGQ_ERR_NULL_PTR:	*hdl is NULL*

Properties

Prototyped in file “r_longq_if.h”

Description

If the control block associated with this Handle was allocated dynamically at run time (LONGQ_USE_HEAP_FOR_CTRL_BLKS set to 1 in config.h), then that memory is freed by this function. If the control block was statically allocated at compile time (LONGQ_USE_HEAP_FOR_CTRL_BLKS set to 0 in config.h), then this function marks the control block as available for use by another buffer. Nothing is done to the contents of the buffer referenced by this Handle.

Example

```
longq_hdl_t   tx_que;
longq_err_t   longq_err;

longq_err = R_LONGQ_Open(tx_buf, BUFSIZE, false, &tx_que);

R_LONGQ_Close(tx_que);
```

Special Notes:

None.

3.5 R_LONGQ_Put()

This function adds an entry to the queue.

Format

```
longq_err_t R_LONGQ_Put(longq_hdl_t const   hdl,
                        uint32_t const      datum)
```

Parameters

hdl
Handle for queue.
datum
Entry to add to queue.

Return Values

LONGQ_SUCCESS:	*Successful; entry added to queue*
LONGQ_ERR_NULL_PTR:	*hdl is NULL*
LONGQ_ERR_QUEUE_FULL:	*Queue full; cannot add entry to queue.*

Properties

Prototyped in file “r_longq_if.h”

Description

This function adds the contents of *datum* to the queue associated with *hdl*. If the queue is full and *ignore_overflow* was set to *false* during Open(), then *LONG_ERR_QUEUE_FULL* is returned. If the queue is full and *ignore_overflow* was set to *true* during Open(), then *datum* overwrites the oldest entry in the queue and *LONGQ_SUCCESS* is returned.

Example

```
    longq_hdl_t tx_que;
    longq_err_t longq_err;
    uint32_t    data = 0x0056;

    longq_err = R_LONGQ_Open(tx_buf, BUFSIZE, false, &tx_que);
    longq_err = R_LONGQ_Put(tx_que, data);
```

Special Notes:

None.

3.6 R_LONGQ_Get()

This function removes an entry from the queue.

Format

```
longq_err_t R_LONGQ_Get(longq_hdl_t const   hdl,
                        uint32_t * const    p_datum)
```

Parameters

hdl
Handle for queue.
p_datum
Pointer to load entry to.

Return Values

LONGQ_SUCCESS:	*Successful; entry removed from queue*
LONGQ_ERR_NULL_PTR:	*hdl or p_datum is NULL*
LONGQ_ERR_QUEUE_EMPTY:	*Queue empty; no data available to fetch*

Properties

Prototyped in file “r_longq_if.h”

Description

This function removes the oldest entry (if available) in the queue associated with *hdl* and loads it into the location pointed to by *p_datum*.

Example

```
    longq_hdl_t rx_que;
    longq_err_t longq_err;
    uint32_t    data;

    longq_err = R_LONGQ_Open(rx_buf, BUFSIZE, false, &rx_que);

    /* queue filled with data by R_LONGQ_Put()elsewhere */

    longq_err = R_LONGQ_Get(rx_que, &data);
```

Special Notes:

None.

3.7 R_LONGQ_Flush()

This function resets a queue to an empty state.

Format

longq_err_t R_LONGQ_Flush(longq_hdl_t const hdl)

Parameters

hdl
Handle for queue.

Return Values

LONGQ_SUCCESS:	*Successful; queue reset*
LONGQ_ERR_NULL_PTR:	*hdl is NULL*

Properties

Prototyped in file “r_longq_if.h”

Description

This function resets the queue identified by *hdl* to an empty state.

Example

```
longq_hdl_t rx_que;
longq_err_t longq_err;


long_err = R_LONGQ_Open(rx_buf, BUFSIZE, false, &rx_que);

/* queue filled with data by R_LONGQ_Put()elsewhere */

R_LONGQ_Flush(rx_que);
```

Special Notes:

None.

3.8 R_LONGQ_Used()

This function provides the number of entries in the queue.

Format

```
longq_err_t R_LONGQ_Used(longq_hdl_t const   hdl,
                         uint16_t * const    p_cnt)
```

Parameters

hdl
Handle for queue.
p_cnt
Pointer to load queue data count to.

Return Values

LONGQ_SUCCESS:	*Successful; *p_cnt loaded with number of entries in queue*
LONGQ_ERR_NULL_PTR:	*hdl or p_cnt is NULL.*

Properties

Prototyped in file “r_longq_if.h”

Description

This function loads the number of entries in the queue associated with *hdl* and into the location pointed to by *p_cnt*.

Example

```
longq_hdl_t rx_que;
longq_err_t longq_err;
uint16_t    count;

longq_err = R_LONGQ_Open(rx_buf, BUFSIZE, false, &rx_que);

/* queue filled with data by R_LONGQ_Put()elsewhere */

R_LONGQ_Used(rx_que, &count);
```

Special Notes:

None.

3.9 R_LONGQ_Unused()

This function provides the number of elements available for storage in the queue.

Format

```
longq_err_t R_LONGQ_Unused(longq_hdl_t const   hdl,
                           uint16_t * const    p_cnt)
```

Parameters

hdl
Handle for queue.
p_cnt
Pointer to load queue unused element count to.

Return Values

LONGQ_SUCCESS:	*Successful; *p_cnt loaded with number of elements available in queue.*
LONGQ_ERR_NULL_PTR:	*hdl or p_cnt is NULL.*

Properties

Prototyped in file “r_longq_if.h”

Description

This function loads the number of unused elements in the queue associated with *hdl* and into the location pointed to by *p_cnt*.

Example

```
longq_hdl_t tx_que;
longq_err_t longq_err;
uint16_t    count;

longq_err = R_LONGQ_Open(tx_buf, BUFSIZE, false, &tx_que);

/* queue filled with data by R_LONGQ_Put()elsewhere */

R_LONGQ_Unused(tx_que, &count);
```

Special Notes:

None.

3.10 R_LONGQ_GetVersion()

This function returns the driver version number at runtime.

Format

uint32_t R_LONGQ_GetVersion(void)

Parameters

None

Return Values

Version number.

Properties

Prototyped in file “r_longq_if.h”

Description

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

```
uint32_t   version;

version = R_LONGQ_GetVersion();
```

Special Notes:

None.

4. 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 demo projects.

4.1 longq_demo_rskrx231

The longq_demo_rskrx71m project demonstrates how to use some of the LONGQ API calls. The demo project opens and initializes a queue, puts 32-bit data into the queue, queries the number of elements in the queue, gets the data from the queue and prints them to the virtual console. The demo project also prints out the version number of the LONGQ module. Virtual console can be enabled by selecting Open Console > Renesas Debug Virtual Console.

4.2 longq_demo_rskrx71m

The longq_demo_rskrx71m project demonstrates how to use some of the LONGQ API calls. The demo project opens and initializes a queue, puts 32-bit data into the queue, queries the number of elements in the queue, gets the data from the queue and prints them to the virtual console. The demo project also prints out the version number of the LONGQ module. Virtual console can be enabled by selecting Open Console > Renesas Debug Virtual Console.

4.3 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”.

4.4 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 Brower* >> *Application Notes* tab.

5. Appendices

5.1 Confirmed Operation Environment

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

Table 5.1 Confirmed Operation Environment (Rev. 1.60)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 4.2.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.04.01
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.1.60
Board used	Renesas Starter Kit+ for RX65N (product No.: RTK500565NSxxxxxBE)

Table 5.2 Confirmed Operation Environment (Rev. 1.70)

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 V2.08.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.1.70
Board used	Renesas Starter Kit for RX231 (product No.: R0K505231SxxxBE) Renesas Starter Kit+ for RX71M (product No.: R0K50571MSxxxBE)

Table 5.3 Confirmed Operation Environment (Rev. 1.71)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.1.0
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.1.71

Table 5.4 Confirmed Operation Environment (Rev. 1.80)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.2.0 IAR Embedded Workbench for Renesas RX 4.10.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201801 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	IAR C/C++ Compiler for Renesas RX version 4.10.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.1.80
Board used	Renesas Starter Kit for RX231 (product No.: R0K505231xxxxxx)

Table 5.5 Confirmed Operation Environment (Rev. 1.82)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2020-10 (20.10.0)
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.1.82
Board used	Renesas Starter Kit for RX231 (product No.: R0K505231SxxxBE) Renesas Starter Kit+ for RX71M (product No.: R0K50571MSxxxBE)

Table 5.6 Confirmed Operation Environment (Rev. 1.90)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio Version 2021-07 IAR C/C++ Compiler for Renesas RX version 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202102 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 4.20.3 Compiler option: The default settings of the integrated development environment.
Endian	Little endian/Big endian
Revision of the module	Rev.1.90
Board used	Renesas Starter Kit+ for RX671 (product No.: RTK55671xxxxxxxxxx)

5.2 Troubleshooting

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

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

- Using CS+:
  Application note “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)”
- Using e^2 studio:
  Application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)”

When using this FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note “Board Support Package Module Using Firmware Integration Technology (R01AN1685)”.

6. Reference Documents

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

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

User's Manual: Development Tools
RX Family C/C++ Compiler CC-RX User's Manual (R20UT3248)

The latest version can be downloaded from the Renesas Electronics website.

Related Technical Updates

This module reflects the content of the following technical updates.
None

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Dec.11.13	—	First edition issued
1.10	Jul.21.14	—	Removed dependency to BSP. Updated XML file for new supported MCUs.
1.20	Nov.26.14	—	Updated XML file for new supported MCUs.
1.30	Jan.28.15	—	Updated XML file for new supported MCUs.
1.40	Jun.30.15	—	Added support for the RX231 Group.
1.50	Sep.30.15	—	Added support for the RX23T Group.
		5	Added r_bsp in Section 2.2 Software Requirements.
		6	Updated code sizes in 2.8 Code Size.
		7	Deleted the “LONGQ_ERR_SUCCESS” definition from the 3.2 Return Values.
1.60	Jan.29.16	6	Updated code sizes in 2.8 Code Size.
		16	Added the section of 4. Demo Projects.
		program	Added support for the RX Family. Changed the XML in order not to depend on the series / group / board of the RX Family. Fixed the initial setting procedure in the R_LONGQ_Open function. Fixed a program according to the Renesas coding rules.
1.70	Jun.01.18	—	Added support setting function of configuration option Using GUI on Smart Configurator.
		—	Updated Demo projects.
		5	Changed toolchain in 2.4 Supported Toolchain.
		6	Changed the default value of LONGQ_CFG_MAX_CTRL_BLKS in 2.7 Configuration Overview. Updated code sizes in 2.8 Code Size.
		7	Added the section of 2.9 Adding the Module to Your Project.
		8	Added the section of 2.10 “for”, “while” and “do while” statements.
		18	Added the section of 4.4 Downloading Demo Projects.
		19	Added the section of 5. Appendices. Added the section of 5.1 Confirmed Operation Environment.
		20	Added the section of 5.2 Troubleshooting.
		21	Added the section of 6. Reference Documents.
		program	Changed the default value of the following macro definition: - LONGQ_CFG_MAX_CTRL_BLKS: Value: (2) -> (32)
1.71	Dec.03.18	19	5.1 Operation Confirmation Environment: Added Table 5.3 Confirmed Operation Environment (Rev. 1.71).
		program	Added document number of the application note accompanying the sample program of the FIT module to xml file.
1.80	Feb.07.19	—	Supported the following compilers: - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		1	Added the section of Target compiler. Deleted related documents.

Rev.	Date	Description	
		Page	Summary
1.80	Feb.07.19	7	Updated the section of 2.8 Code Size
		18	Updated the section of 3.10 R_LONGQ_GetVersion().
		21	Updated the section of 5.1 Confirmed Operation Environment.
		23	Deleted the section of Website and Support.
		program	Deleted the inline expansion of the R_LONGQ_GetVersion function.
1.81	Jun.10.20	—	Modified comment of API function to Doxygen style.
		8	Changed Section 2.9 Adding the Module to Your Project.
		11..18	Deleted the Reentrant for each API in 3. API Functions.
1.82	Nov.30.20	—	Updated the sample code project due to the upgrade of the development environment.
1.90	Oct.29.21	6	Updated the section 2.7 Configuration Overview Added new macro definition LONGQ_CFG_PROTECT_QUEUE and LONGQ_CFG_CRITICAL_SECTION.
		7	Updated the section of 2.8 Code Size.
		13..17	3. API Functions Deleted “Special Notes” for 5 APIs: R_LONGQ_Put, R_LONGQ_Get, R_LONGQ_Flush, R_LONG_Used, R_LONG_Unused.
		22	5.1 Confirmed Operation Environment Added Table 5.6 Confirmed Operation Environment (Rev. 1.90).
		program	Updated for queue protection in R_LONGQ_Put, R_LONGQ_Get, R_LONGQ_Flush, R_LONGQ_Used, R_LONGQ_Unused functions.

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

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

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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