RX Family

BYTEQ Module Using Firmware Integration Technology

R01AN1683EU0140
Rev. 1.40
June 30, 2015

Introduction

This module provides functions for creating and maintaining byte-based 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.

Related Documents

- Firmware Integration Technology User’s Manual (R01AN1833EU)
- Board Support Package Firmware Integration Technology Module (R01AN1685EU)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)
- Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826EJ)

Contents

1. Overview

The Byte Queue (BYTEQ) 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 to be supported.

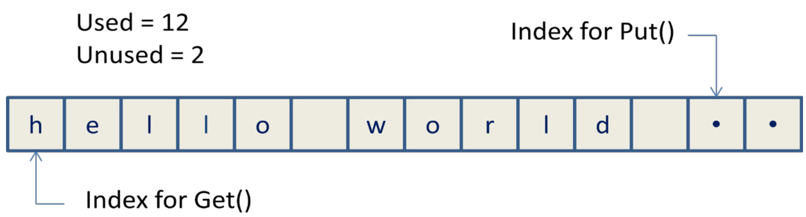
There is one control block per buffer. When an R_BYTEQ_Open() is performed, a pointer to the application's buffer and its length are passed in, and a pointer to a QCB is provided. This pointer, which is called a Handle, is then passed to all of the other API functions. The functions then operate on the 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 BYTEQ Module

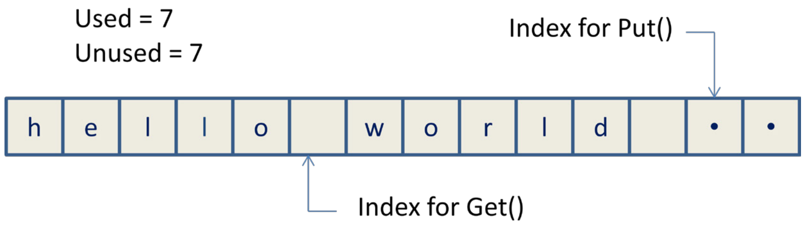
The following illustrates a queue's behavior with API calls:

```
#define BUFSIZE 14

uint8_t       my_buf[BUFSIZE];
byteq_hdl_t   my_que;
byteq_err_t   err;
uint8_t       byte;

err = R_BYTEQ_Open(my_buf, BUFSIZE, &my_que);

// add 12 bytes to queue
R_BYTEQ_Put(my_que, 'h');
R_BYTEQ_Put(my_que, 'e');
R_BYTEQ_Put(my_que, 'l');
R_BYTEQ_Put(my_que, 'l');
R_BYTEQ_Put(my_que, 'o');
R_BYTEQ_Put(my_que, ' ');
R_BYTEQ_Put(my_que, 'w');
R_BYTEQ_Put(my_que, 'o');
R_BYTEQ_Put(my_que, 'r');
R_BYTEQ_Put(my_que, 'l');
R_BYTEQ_Put(my_que, 'd');
R_BYTEQ_Put(my_que, ' ');
```

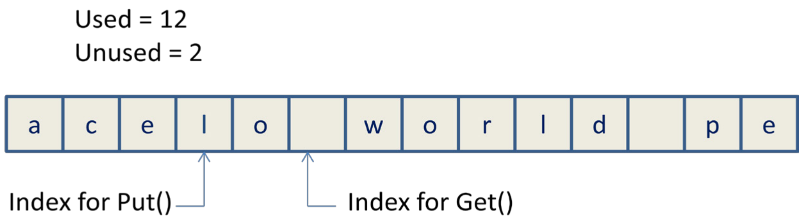



```
// remove 5 bytes from queue
R_BYTEQ_Get(my_que, &byte);     // byte = 'h'
R_BYTEQ_Get(my_que, &byte);     // byte = 'e'
```

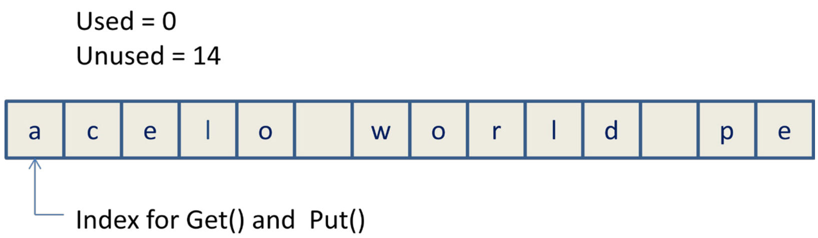
```
R_BYTEQ_Get(my_que, &byte);      // byte = 'l'
R_BYTEQ_Get(my_que, &byte);      // byte = 'l'
R_BYTEQ_Get(my_que, &byte);      // byte = 'o'
```




```
// add 5 bytes to queue
R_BYTEQ_Put(my_que, 'p');
R_BYTEQ_Put(my_que, 'e');
R_BYTEQ_Put(my_que, 'a');
R_BYTEQ_Put(my_que, 'c');
R_BYTEQ_Put(my_que, 'e');
```




```
// flush queue
R_BYTEQ_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

No package dependencies.

2.3 Limitations

No software limitations.

2.4 Supported Toolchains

This driver is tested and working with the following toolchains:

1. Renesas RXC Toolchain v2.02.

2.5 Header Files

Compile time configurable options are located in r_byteq\ref\r_byteq_config_reference.h. This file should be copied into the r_config subdirectory of the project and renamed to r_byteq_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_byteq\r_byteq_if.h. Both this file and r_byteq_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_byteq_config.h". A summary of these settings are provided in the following table:

Configuration options in *r_byteq_config.h*	
`#define BYTEQ_CFG_PARAM_CHECKING_ENABLE 1`	If this equate is set to 1, parameter checking is included in the build. If the equate is set to 0, the parameter checking is omitted from the build. Setting this equate to BSP_CFG_PARAM_CHECKING_ENABLE utilizes the system default setting (must include platform.h).
`#define BYTEQ_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 BYTEQ_CFG_MAX_CTRL_BLKS 4`	Specifies how many control blocks to allocate at compile time. This constant is ignored if BYTEQ_CFG_USE_HEAP_FOR_CTRL_BLKS is 1.

2.8 Code Size

The code size is based on optimization level 2 and optimization type set for size for the supported toolchain. The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options set in the module configuration header file.

ROM and RAM code sizes		
	With Parameter Checking	Without Parameter Checking
Using Heap for Control Blocks	ROM: 248 + 222 for malloc() = 470 bytes	ROM: 164 + 222 for malloc() = 386 bytes
	RAM: 0 + 8 for malloc() = 8 bytes	RAM: 0 + 8 for malloc() = 8 bytes
Using Allocated Control Blocks	ROM: 284 bytes	ROM: 198 bytes
	RAM: 12 x **BYTEQ_CFG_MAX_CTRL_BLKS**	RAM: 12 x **BYTEQ_CFG_MAX_CTRL_BLKS**

2.9 Adding Driver to Your Project

Follow the steps below to add the driver's code to your project:

1. Add the r_byteq and r_config folders to your project.
2. Add a project include path for the "r_byteq" directory.
3. Add a project include path for the "r_byteq\src" directory.
4. Add a project include path for the "r_config" directory.
5. Open "r_config\r_byteq_config.h" file and configure the driver for your project.
6. Add a #include for r_byteq_if.h to any source files that need to use the API functions.

3. API Functions

3.1 Summary

The following functions are included in this design:

Function	Description
R_BYTEQ_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_BYTEQ_Close()	Releases the queue control block associated with the handle.
R_BYTEQ_Put()	Adds a byte of data to the queue.
R_BYTEQ_Get()	Removes the oldest byte of data from the queue.
R_BYTEQ_Flush()	Resets the queue to an empty state.
R_BYTEQ_Used()	Provides the number of bytes used in the queue.
R_BYTEQ_Unused()	Provides the number of bytes unused in the queue.
R_BYTEQ_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_byteq_if.h along with the API function declarations.

```
typedef enum _byteq_err          // BYTEQ API error codes
{
    BYTEQ_SUCCESS = 0,
    BYTEQ_ERR_NULL_PTR,          // received null ptr; missing required arg
    BYTEQ_ERR_INVALID_ARG,       // argument is not valid for parameter
    BYTEQ_ERR_MALLOC_FAIL,       // cannot allocate mem for ctrl block;
                                 // increase heap
    BYTEQ_ERR_NO_MORE_CTRL_BLKS, // no more ctrl blocks;
                                 // increase BYTEQ_MAX_CTRL_BLKS
    BYTEQ_ERR_QUEUE_FULL,        // queue full; cannot add another byte
    BYTEQ_ERR_QUEUE_EMPTY        // queue empty; no byte to fetch
} byteq_err_t;
```

3.3 R_BYTEQ_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

```
byteq_err_t R_BYTEQ_Open(uint8_t * const        p_buf,
                         uint16_t const         size,
                         byteq_hdl_t * const    p_hdl)
```

Parameters

p_buf
Pointer to byte buffer.

size
Buffer size in bytes.

p_hdl
Pointer to a handle for queue (value set here)

Return Values

BYTEQ_SUCCESS:	*Successful; queue initialized*
BYTEQ_ERR_NULL_PTR:	*p_buf is NULL*
BYTEQ_ERR_INVALID_ARG:	*Size is less than or equal to 1.*
BYTEQ_ERR_MALLOC_FAIL:	*Cannot allocate control block. Increase heap size.*
BYTEQ_ERR_NO_MORE_CTRL_BLKS:	*Cannot assign control block. Increase BYTEQ_MAX_CTRL_BLKS in config.h.*

Properties

Prototyped in file "r_byteq_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.

Reentrant

Function is re-entrant for different buffers.

Example

```
#define BUFSIZE 80

uint8_t       tx_buf[BUFSIZE];
byteq_hdl_t   tx_que;
byteq_err_t   byteq_err;

byteq_err = R_BYTEQ_Open(tx_buf, BUFSIZE, &tx_que);
```

Special Notes:

None.

3.4 R_BYTEQ_Close()

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

Format

byteq_err_t R_BYTEQ_Close(byteq_hdl_t const hdl)

Parameters

hdl
Handle for queue.

Return Values

BYTEQ_SUCCESS:	*Successful; control block released.*
BYTEQ_ERR_NULL_PTR:	*hdl is NULL.*

Properties

Prototyped in file "r_byteq_if.h"

Description

If the control block associated with this Handle was allocated dynamically at run time (BYTEQ_USE_HEAP_FOR_CTRL_BLKS set to 1 in config.h), then that memory is free()d by this function. If the control block was statically allocated at compile time (BYTEQ_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.

Reentrant

Function is re-entrant for different queues.

Example

```
byteq_hdl_t   tx_que;
byteq_err_t   byteq_err;

byteq_err = R_BYTEQ_Open(tx_buf, BUFSIZE, &tx_que);

byteq_err = R_BYTEQ_Close(tx_que);
```

Special Notes:

None.

3.5 R_BYTEQ_Put()

This function adds a byte of data to the queue.

Format

```
byteq_err_t R_BYTEQ_Put(byteq_hdl_t const   hdl,
                        uint8_t const       byte)
```

Parameters

hdl
Handle for queue.
byte
Byte to add to queue.

Return Values

BYTEQ_SUCCESS:	*Successful; byte added to queue*
BYTEQ_ERR_NULL_PTR:	*hdl is NULL.*
BYTEQ_ERR_QUEUE_FULL	*Queue full; cannot add byte to queue.*

Properties

Prototyped in file "r_byteq_if.h"

Description

This function adds the contents of *byte* to the queue associated with *hdl*.

Reentrant

Function is re-entrant for different queues.

Example

```
byteq_hdl_t tx_que;
byteq_err_t byteq_err;
uint8_t     byte = 'A';

byteq_err = R_BYTEQ_Open(tx_buf, BUFSIZE, &tx_que);
byteq_err = R_BYTEQ_Put(tx_que, byte);
```

Special Notes:

If the queue is accessed at both the interrupt and application level, it is up to the user to disable and enable the associated interrupt before and after calling this function from the application level. 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.

3.6 R_BYTEQ_Get()

This function removes a byte of data from the queue.

Format

```
byteq_err_t R_BYTEQ_Get(byteq_hdl_t const   hdl,
                        uint8_t * const     p_byte)
```

Parameters

hdl
Handle for queue.

p_byte
Pointer to load byte to.

Return Values

BYTEQ_SUCCESS: *Successful; byte removed from queue*
BYTEQ_ERR_NULL_PTR: *hdl is NULL.*
BYTEQ_ERR_QUEUE_EMPTY: *Queue empty; no data available to fetch*

Properties

Prototyped in file "r_byteq_if.h"

Description

This function removes the oldest byte of data in the queue associated with *hdl* and loads it into the location pointed to by *p_byte*.

Reentrant

Function is re-entrant for different queues.

Example

```
byteq_hdl_t rx_que;
byteq_err_t byteq_err;
uint8_t     byte;

byteq_err = R_BYTEQ_Open(rx_buf, BUFSIZE, &rx_que);

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

byteq_err = R_BYTEQ_Get(rx_que, &byte);
```

Special Notes:

If the queue is accessed at both the interrupt and application level, it is up to the user to disable and enable the associated interrupt before and after calling this function from the application level. 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.

3.7 R_BYTEQ_Flush()

This function resets a queue to an empty state.

Format

byteq_err_t R_BYTEQ_Flush(byteq_hdl_t const hdl)

Parameters

hdl

Handle for queue.

Return Values

BYTEQ_SUCCESS:	*Successful; queue reset*
BYTEQ_ERR_NULL_PTR:	*hdl is NULL.*

Properties

Prototyped in file “r_byteq_if.h”

Description

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

Reentrant

Function is re-entrant for different queues.

Example

```
byteq_hdl_t rx_que;
byteq_err_t byteq_err;


byteq_err = R_BYTEQ_Open(rx_buf, BUFSIZE, &rx_que);

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

byteq_err = R_BYTEQ_Flush(rx_que);
```

Special Notes:

If the queue is accessed at both the interrupt and application level, it is up to the user to disable and enable the associated interrupt before and after calling this function from the application level. 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.

3.8 R_BYTEQ_Used()

This function provides the number of data bytes in the queue.

Format

byteq_err_t R_BYTEQ_Used(byteq_hdl_t const hdl,
uint16_t * const p_cnt)

Parameter

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

Return Values

BYTEQ_SUCCESS: *Successful; *p_cnt loaded with the number of bytes in the queue*
BYTEQ_ERR_NULL_PTR: *hdl is NULL.*

Properties

Prototyped in file "r_byteq_if.h"

Description

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

Reentrant

Yes.

Example

```
byteq_hdl_t rx_que;
byteq_err_t byteq_err;
uint16_t    count;

byteq_err = R_BYTEQ_Open(rx_buf, BUFSIZE, &rx_que);

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

byteq_err = R_BYTEQ_Used(rx_que, &count);
```

Special Notes:

If the queue is accessed at both the interrupt and application level, it is up to the user to disable and enable the associated interrupt before and after calling this function from the application level. 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.

3.9 R_BYTEQ_Unused()

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

Format

byteq_err_t R_BYTEQ_Unused(byteq_hdl_t const hdl,
uint16_t * const p_cnt)

Parameters

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

Return Values

BYTEQ_SUCCESS: *Successful; *p_cnt loaded with the number of bytes not used in the queue*
BYTEQ_ERR_NULL_PTR: *hdl is NULL.*

Properties

Prototyped in file “r_byteq_if.h”

Description

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

Reentrant

Yes.

Example

```
byteq_hdl_t tx_que;
byteq_err_t byteq_err;
uint16_t    count;

byteq_err = R_BYTEQ_Open(tx_buf, BUFSIZE, &tx_que);

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

byteq_err = R_BYTEQ_Unused(tx_que, &count);
```

Special Notes:

If the queue is accessed at both the interrupt and application level, it is up to the user to disable and enable the associated interrupt before and after calling this function from the application level. 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.

3.10 R_BYTEQ_GetVersion()

This function returns the driver version number at runtime.

Format

uint32_t R_BYTEQ_GetVersion(void)

Parameters

None

Return Values

Version number.

Properties

Prototyped in file "r_byteq_if.h"

Description

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

Reentrant

Yes

Example

```
uint32_t   version;

version = R_BYTEQ_GetVersion();
```

Special Notes:

This function is inlined using the "#pragma inline" directive.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jul 24, 2013	—	First edition issued
1.10	Jul 21, 2014	—	Updated XML file for new supported MCUs.
1.20	Nov 21, 2014	—	Removed dependency to BSP. Updated XML file for new supported MCUs.
1.30	Jan 22, 2015	—	Updated XML file for new supported MCUs.
1.40	Jun 30, 2015	—	Added support for the RX231 Group.

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU 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 an MPU or MCU 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.

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