RX Family

Coding Example of Wait Processing by Software

R01AN1852EJ0100
Rev. 1.00
Feb. 3, 2014

Abstract

This document describes a coding example of wait processing by software.

Products

- RX600 Series: RX610 Group, RX62N/621 Group, RX62T Group, RX62G Group, RX630 Group, RX63N/631 Group, RX63T Group
- RX200 Series: RX210 Group, RX220 Group, RX21A Group
- RX100 Series: RX110 Group, RX111 Group

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

Contents

1. Wait Processing

When performing wait processing using a loop such as the Do-while statement, the wait time may not match the intended time for the following reasons.

- In C language, instructions to be output or the number of instructions vary depending on the compiler optimization options and version, in the result, the number of execution cycles also varies.
- In assembly language, when an instruction straddles the alignment, the number of instruction fetches increases, then the number of execution cycles will change.

For these reasons, assembly language must be used to have wait processing with a fixed number of execution cycles, so instructions are not affected by the alignment.

The following pages explain coding examples of wait processing with a fixed number of execution cycles.

2. Operation Confirmation Conditions

The sample code accompanying this application note has been run and confirmed under the conditions below.

Table 2.1 Operation Confirmation Conditions (High-performance Embedded Workshop)

Item	Contents
MCU used	R5F563NBDDFC (RX63N Group)
Integrated development environment	Renesas Electronics High-performance Embedded Workshop Version 4.09.01
C compiler	Renesas Electronics RX Standard Toolchain Version 1.2.1.0
	Options [Compiler] -cpu=rx600 -output=obj="$(CONFIGDIR)¥$(FILELEAF).obj" -debug -nologo [Assembler] -cpu=rx600 -output="$(CONFIGDIR)¥$(FILELEAF).obj" –debug -nologo [Linker] -noprelink -rom=D=R,D_1=R_1,D_2=R_2 –nomessage -list="$(CONFIGDIR)¥$(PROJECTNAME).map" -nooptimize -start=B_1,R_1,B_2,R_2,B,R,SU,SI/04,PResetPRG/0FFFF8000,C_1,C_2,C,C$*,D_1,D_2,D,P,PIntPRG,W*,L/0FFFF8100,FIXEDVECT/0FFFFFFD0 -nologo -output="$(CONFIGDIR)¥$(PROJECTNAME).abs" –end -input="$(CONFIGDIR)¥$(PROJECTNAME).abs" -form=stype -output="$(CONFIGDIR)¥$(PROJECTNAME).mot" -exit
Endian	Little endian
Sample code version	Version 1.00

Table 2.2 Operation Confirmation Conditions (e2 studio)

Item	Contents
MCU used	R5F563NBDDFC (RX63N Group)
Integrated development environment	Renesas Electronics e2 studio Version 2.2.0.13
C compiler	Renesas Electronics RX Standard Toolchain Version 2.1.0
	Options [Compiler] -cpu=rx600 -include="${TCINSTALL}¥include" –debug –nologo -change_message=warning -define=__RX [Assembler] -cpu=rx600 -nolistfile -debug -nologo [Linker] -library="${CONFIGDIR}¥${ProjName}.lib" -noprelink -list="${ProjName}.map" -show -nooptimize -nomessage -nologo -output="${CONFIGDIR}¥${ProjName}.abs" -rom=D=R -rom=D_1=R_1 -rom=D_2=R_2
Endian	Little endian
Sample code version	Version 1.00

3. Software

The sample code accompanying this application note has two kinds of wait processing.

- Inline function that specifies the number of loops
- Function that specifies the execution time

3.1 Operation Overview

- Inline function that specifies the number of loops
  Loops for the number of loops specified are performed.
  This processing is the assembly language inline function, and one loop is executed in five cycles.
  The branch instruction placed before entering the loop processing is to clear the CPU instruction queue and used to match the number of execution cycles between the first loop and the second loop. When the branch instruction is executed, the instruction queue is cleared, and the CPU starts an instruction fetch from the branch destination.
  The NOP instruction is placed in the branch destination to fix the number of cycles regardless of the alignment. The number of execution cycles for the loop can always be the same by fetching the subsequent instruction while executing the NOP instruction.

- Function that specifies the execution time
  The execution time and the frequency of the system clock (ICLK) are used as arguments, and a waits for the specified execution time is performed. The number of loops is calculated using the execution time (µs) and the frequency of the ICLK (kHz) specified in the arguments, and the inline function that specifies the number of loops is called. As approximately 20 cycles are needed to for a function call and exit, and calculation for the number of loops, overhead for the cycles is taken into account when calculating the number of loops.

3.2 Coding Examples of Wait Processing

Figure 3.1 shows an example of Coding for the Inline Function That Specifies the Number of Loops.

C source code

```
void main(void)
{
      :
   R_DELAY(LOOP_COUNT);                    ← Set the number of loops (LOOP_COUNT).
      :
}

#pragma inline_asm R_DELAY
static void R_DELAY (unsigned long loop_cnt)
{
   BRA    ?+
   NOP
?:                                         ⎫
   NOP                                     ⎪
   SUB    #01H,R1                          ⎬ The number of cycles becomes fixed regardless of the alignment
   BNE    ?-                               ⎭
}
```

Figure 3.1 Coding for the Inline Function That Specifies the Number of Loops

In the processing for calling the R_DELAY function, the number of loops (LOOP_COUNT) is specified in the argument. In the R_DELAY function, one loop is decremented from the number of loop, the loop is repeated until R1 reaches 0. Assuming the number of loops (LOOP_COUNT) is five loops, the execution cycles for the R_DELAY function is calculated as:
5 loops × 5 cycles = 25 cycles

Figure 3.2 shows Coding for the Function That Specifies the Execution Time.

```
C source code

#include "r_delay.h"                                  ← Include the header file.

void main(void)
{
        :
    R_DELAY_Us(WAIT_TIME_US, BSP_ICLK_HZ);            ← Set the execution time (WAIT_TIME_US) and ICLK frequency (BSP_ICLK_HZ).
        :
}

C source code (r_delay.c)

#pragma inline_asm R_DELAY
static void R_DELAY (unsigned long loop_cnt)
{
    BRA    ?+
    NOP
?:
    NOP
    SUB    #01H,R1
    BNE    ?-

}

void R_DELAY_Us (unsigned long us, unsigned long khz)
{

    signed long loop_cnt;

    loop_cnt = us * khz;                              ┐
    loop_cnt = ( loop_cnt / 5000 );                   ├Calculate the number of loops.
    loop_cnt = loop_cnt ? 4;                          ┘

    if( loop_cnt > 0 )
    {
        R_DELAY((unsigned long)loop_cnt);
    }
}
```

Figure 3.2 Coding for the Function That Specifies the Execution Time

In the processing for calling the R_DELAY_Us function, the execution time (WAIT_TIME_US) and ICLK frequency (BSP_ICLK_HZ) are specified in the argument. In the R_DELAY_Us function, the number of loops is calculated, then the R_DELAY function is executed using the calculation result as the argument. The number of loops is calculated as follows:
Number of loops = Execution time (µs) × ICLK (kHz) ÷ 5,000 [execution cycles for 1 loop × 1,000] – 4 loops [overhead of 20 cycles]
When an execution time of 100 µs and an ICLK frequency of 10,000 kHz (10 MHz)
The number of loops: 100 × 10,000 ÷ 5,000 – 4 = 196 loops
The number of execution cycles: 196 loops × 5 cycles = 980 cycles
Execution time (µs): 10,000 kHz (100 ns) × (980 cycles + 20 cycles [overhead]) = 100 µs

3.3 Notes on Using Functions

This section describes notes on using the functions.

- Inline function that specifies the number of loops
  — Do not set the number of loops to 0.
  — When executing the function in external memory, the number of cycles for a loop does not become 5 cycles.

- Function that specifies the execution time
  — Do not set the execution time and ICLK frequency to 0.
  — Set the execution time and ICLK frequency to a whole number.
  — The 20-cycle overhead may increase depending on the execution time (µs) and ICLK frequency (kHz) values.
  — The calculated result for the number of loops is rounded off to the nearest whole number. Specify the execution time taking the rounding off into consideration.

3.4 File Composition

Table 3.1 lists the Files Used in the Sample Code.

Table 3.1 Files Used in the Sample Code

File Name	Outline	Remarks
r_delay.c	Wait processing by software	
r_delay.h	Header file for r_delay.c	

3.5 Functions

Table 3.2 lists the Functions.

Table 3.2 Functions

Function Name	Outline
R_DELAY	Inline function that specifies the number of loops
R_DELAY_Us	Function that specifies the execution time

3.6 Function Specifications

The following tables list the sample code function specifications.

R_DELAY

Outline	Inline function that specifies the number of loops
Header	None
Declaration	static void R_DELAY (unsigned long loop_cnt)
Description	Wait processing that loops at a fixed five cycles.
Arguments	loop_cnt : Number of loops
Return Value	None
Remarks	This function is the assembly language inline function. When using this function, write the function codes in the appropriate source file. By adding a NOP instruction in the start of the loop processing, the number of cycles for a loop can be adjusted.

R_DELAY_Us

Outline	Function that specifies the execution time
Header	r_delay.h
Declaration	void R_DELAY_Us (unsigned long us, unsigned long khz)
Description	The number of loops are calculated based on the execution time (µs) and ICLK frequency, and the inline function that specifies the number of loops is called.
Arguments	us : Execution time khz : ICLK frequency when calling the function
Return Value	None

3.7 Flowcharts

3.7.1 Function That Specifies the Number of Loops

Figure 3.3 shows the Function That Specifies the Number of Loops.

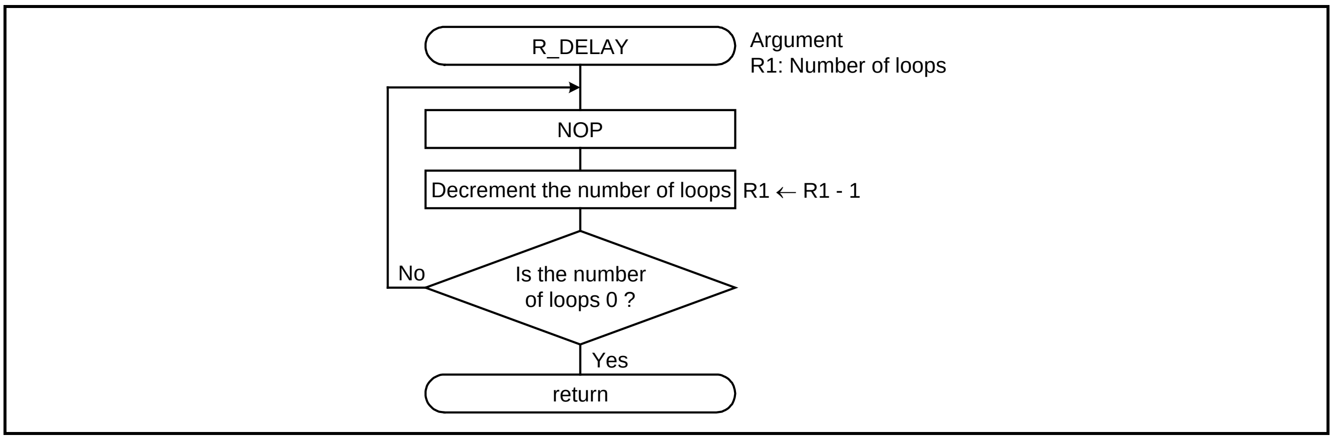

Figure 3.3 Function That Specifies the Number of Loops

3.7.2 Function That Specifies Execution Time

Figure 3.4 shows the Function That Specifies Execution Time.

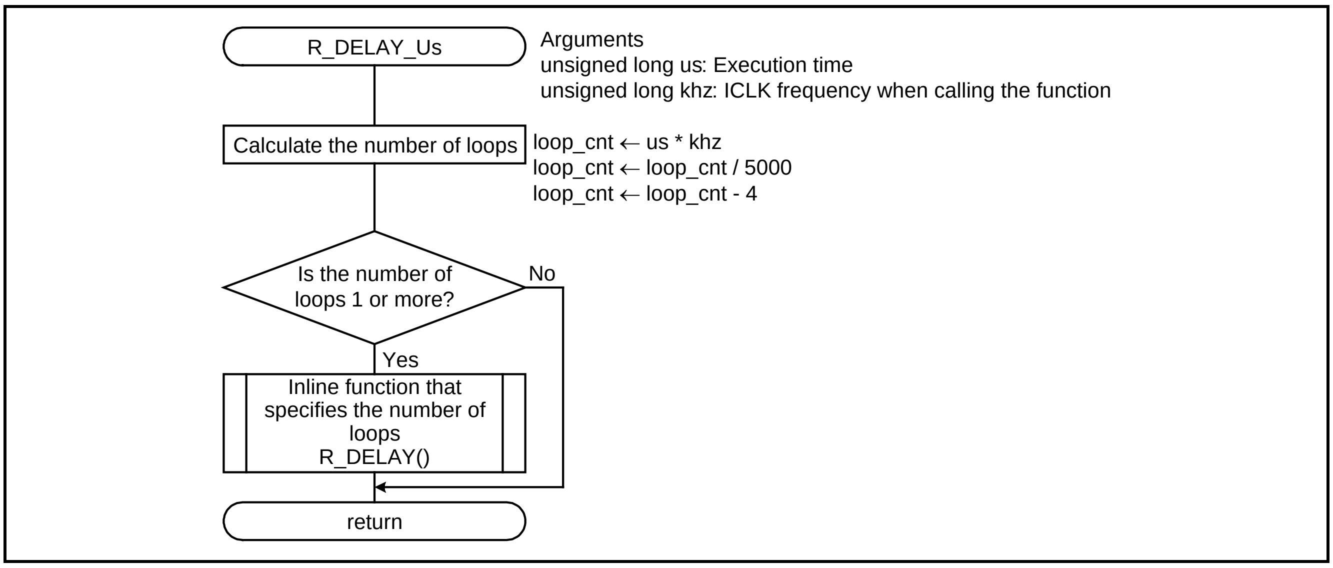
R_DELAY_Us
Arguments
unsigned long us: Execution time
unsigned long khz: ICLK frequency when calling the function
Calculate the number of loops
loop_cnt ← us * khz
loop_cnt ← loop_cnt / 5000
loop_cnt ← loop_cnt - 4
Is the number of loops 1 or more?
No
Yes
Inline function that specifies the number of loops R_DELAY()
return

Figure 3.4 Function That Specifies Execution Time

4. Reference

The following explains how the number of execution cycles varies.

4.1 Influence of Optimization Options on Instruction Codes

When using C language to write wait processing, if the optimization options or compiler version on the compile process differ, the type and number of instructions to be output differ, and then the number of execution cycles in the wait processing changes.

Table 3.1 lists examples of optimization options and instructions that are output.

Table 4.1 Optimization Options and Instructions That are Output

C Language Source		

Wait processing using a Do-while statement	void Software_delay (unsigned long count)
	{
	do
	{
	count--;
	} while (count);
	}	

Example of Compile Results		
Optimization level: 0 1 loop: 6 instructions	Optimization level: 1 Optimization type: Size prioritized 1 loop: 4 instructions	Optimization level: 2 Optimization type: Size prioritized 1 loop: 2 instructions
_Software_delay: .STACK _Software_delay=8 SUB #04H, R0 MOV.L R1, [R0] L1: MOV.L [R0], R1 SUB #01H, R1 MOV.L R1, [R0] L2: MOV.L [R0], R1 CMP #00H, R1 BNE L1 L3: RTSD #04H	_Software_delay: .STACK _Software_delay=4 L1: ADD 0FFFFFFFFH, R1, R14 CMP #01H, R1 MOV.L R14, R1 BNE L1 L2: RTS	_Software_delay: .STACK _Software_delay=4 L1: SUB #01H, R1 BNE L1 L2: RTS

4.2 Influence of the Instruction Allocation Address on the Number of Instruction Execution Cycles

When the code size of the instruction code is 2 bytes or more, and the allocation address of the instruction code straddles the alignment, the fetch for the instruction is performed twice, then the number of execution cycles may increase by one cycle.

The sample code below is an example to show how the number of execution cycles increases in wait processing.

In Figure 4.1, when the sample shown on the left is compiled, instruction codes are output as shown on the right.

When the SUB instruction address is allocated to an address that does not straddle the alignment as shown in “Pattern A” of Figure 4.2, the instruction fetch is only performed once, then the number of execution cycles for the SUB instruction is one cycle.

When the SUB instruction address is allocated to an address that straddles the alignment as shown in “Pattern B” of Figure 4.2, the instruction fetch is performed twice, then the number of execution cycles for the SUB instruction becomes two cycles.

Sample code

```
count = XX;
do
 {
      count--;
 } while (count);

(XX: Number of loops)
```

Compile result (assembler)

```
Address          Instruction code
FFFFXXXX                             Label:
FFFFXXXX         6011                          SUB #01H, R1
FFFFXXXX+2       21rr                          BNE Label
```

Figure 4.1 Compile Result of the Sample Code

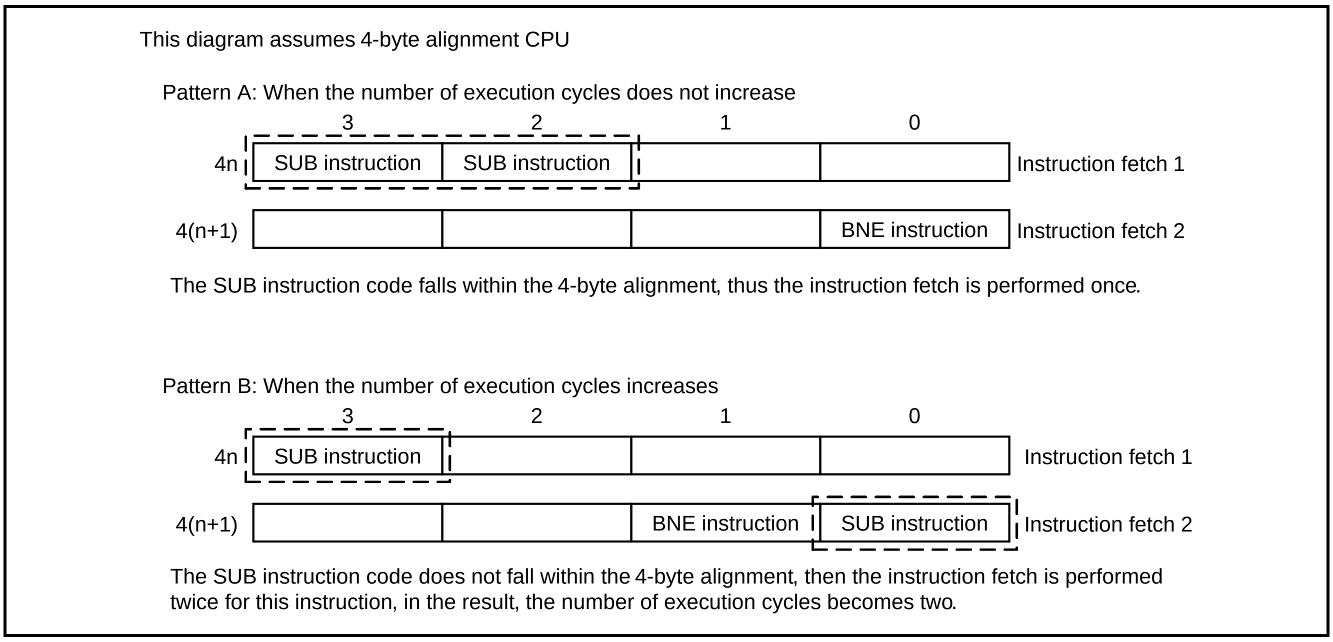
This diagram assumes 4-byte alignment CPU
Pattern A: When the number of execution cycles does not increase
3
2
1
0
4n
SUB instruction
SUB instruction
Instruction fetch 1
4(n+1)
BNE instruction
Instruction fetch 2
The SUB instruction code falls within the 4-byte alignment, thus the instruction fetch is performed once.
Pattern B: When the number of execution cycles increases
3
2
1
0
4n
SUB instruction
Instruction fetch 1
4(n+1)
BNE instruction
SUB instruction
Instruction fetch 2
The SUB instruction code does not fall within the 4-byte alignment, then the instruction fetch is performed twice for this instruction, in the result, the number of execution cycles becomes two.

Figure 4.2 Allocation Address of the Instruction and the Number of Instruction Execution Cycles

5. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

6. Reference Documents

User’s Manual: Software
RX Family User’s Manual: Software Rev.1.20 (R01US0032EJ)
The latest version can be downloaded from the Renesas Electronics website.

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

User’s Manual: Development Tools
RX Family C/C++ Compiler Package V.1.01 User’s Manual Rev.1.00 (R20UT0570EJ)
The latest version can be downloaded from the Renesas Electronics website.

RX Family CC-RX V2.01.00 User’s Manual: RX Coding (R20UT2748EJ)
The latest version can be downloaded from the Renesas Electronics website.

Website and Support

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

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

REVISION HISTORY	RX Family Application Note Coding Example of Wait Processing by Software

Rev.	Date	Description	
		Page	Summary
1.00	Feb. 3, 2014	—	First edition issued

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.

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
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

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 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

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 906, 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 Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141