RX Family

Complex Number Operation Programming with DSP Function Instructions

R01AN0229EJ0100
Rev.1.00
Mar 14, 2011

Introduction

This document explains how to carry out complex number operations with the RX family's DSP function instructions.

Target Device

RX Family

Contents

1. General

The RX family CPU core (hereafter referred to as RX) incorporates a 16 x 16-bit multiply-accumulator. The result of executing a typical 32 x 32-bit integer multiplication instruction (MUL instruction) that is used for multiplicative expressions or address calculations is given by the lower 32 bits of the 64-bit result of multiplying two 32-bit numbers. Accordingly, it is assumed that the result of using an MUL instruction does not exceed 32 bits. However, when a numerical value is expressed as a fixed-point number (For example, refer to [1].), it is common that the valid data of the result of a multiplication or a multiply-accumulation is assigned to the upper bits. Therefore, if a multiplication or a multiply-accumulation of fixed-point numbers is carried out using a MUL instruction, the result must be within 32 bits and only a very limited range of numerical values can be dealt with. To solve this problem, the RX supports the instructions to perform the following: multiply-accumulation (or multiplication) by a 48-bit accumulator, rounding operation of the value stored in an accumulator, and data transfer between an accumulator and a general-purpose register. The combination of these multiply-accumulation and rounding operation instructions allows several high-speed operations on fixed-point numbers and data processing performance equal to DSPs. For details on the RX's multiply-accumulation instruction, refer to "RX Family User's Manual; Software" (REJ09B0435). The application note "How to Use Multiply-Accumulation Instruction" (R01AN0254EJ) explains how to use these multiply-accumulation and rounding operation instructions. In addition, the application note "How to Use Intrinsic Functions for Multiply-Accumulation" (R01AN0255EJ) explains how to use these multiply-accumulation and rounding operation instructions through intrinsic functions that are extended functions of the RX Family C/C++ compiler (hereafter referred to as compiler). The following sections describe complex number operation programming with the RX's multiply-accumulation instructions. The sample program employs a method of using the multiply-accumulation instructions through compiler intrinsic functions (intrinsic functions supporting the multiply-accumulation instruction are available at compiler version 1.01 or later).

Note: [1] Mori, Natori and Torii; "Iwanami Koza Computer Science-18 Numerical Calculation," pp.1-27, Iwanami Shoten, (1982)

2. Complex Number Data Representation

A complex number is a number consisting of a real part and an imaginary part. Since the RX's multiply-accumulation instructions are intended for 16-bit signed integers, in this application note a complex number is represented by a structure that has two 16-bit signed integers as its members. The real part and the imaginary part can be regarded as 16-bit signed integer or fixed-point data. It is expected that since the size of the structure is within 32 bits, calling a function transfers the structure through a register.

The complex number data representation mentioned above and the operation program interface described later are defined collectively in a single header file. Shown below are the contents of the header file (RXComplex.h).

```
#ifndef  _RXCOMPLEX_H
#define  _RXCOMPLEX_H

#include <stdint.h>

/* complex number */
typedef struct {
    int16_t re;   /* real part */
    int16_t im;   /* imaginary part */
} RXComplex;

/* function(s) */
RXComplex complex_add(RXComplex a, RXComplex b);
RXComplex complex_sub(RXComplex a, RXComplex b);
RXComplex complex_mul(RXComplex a, RXComplex b);
RXComplex complex_div(RXComplex a, RXComplex b);

#endif /* ! _RXCOMPLEX_H */
```

3. Four Arithmetic Operations on Complex Numbers

This section describes the four arithmetic operations on complex numbers.

3.1 Addition and Subtraction

This section describes addition and subtraction of complex numbers. First, the definition of addition of complex numbers is given below. In an addition operation on complex numbers, a real part and an imaginary part are added to the other respective parts.

$$(a+bi)+(c+di)=(a+c)+(b+d)i$$

Shown below is a program of complex_add, a function for addition of complex numbers. This function returns the result of adding a complex number a to a complex number b. The decimal point of the result is not shifted (when input is regarded as fixed-point data).

```
/*
   Addition of complex numbers.
   Return the result of adding complex number a to complex number b.
 */
RXComplex complex_add(RXComplex a, RXComplex b)
{
    RXComplex res;

    res.re = (int16_t)(a.re + b.re);
    res.im = (int16_t)(a.im + b.im);
    return res;
}
```

Next, subtraction is discussed. The definition of subtraction of complex numbers is as follows. In a subtraction operation on complex numbers, the real part and the imaginary part are subtracted from the other respective parts.

$$(a+bi)-(c+di)=(a-c)+(b-d)i$$

Shown below is a program of complex_sub, a function for subtraction of complex numbers. This function returns the difference between complex numbers a and b. The decimal point of the result of subtraction is not shifted (when input is regarded as fixed-point data).

```
/*
   Subtraction of complex numbers.
   Return the difference between complex numbers a and b.
 */
RXComplex complex_sub(RXComplex a, RXComplex b)
{
    RXComplex res;

    res.re = (int16_t)(a.re - b.re);
    res.im = (int16_t)(a.im - b.im);
    return res;
}
```

3.2 Multiplication

This section describes multiplication. The definition of multiplication of complex numbers is given below:

$$(a+bi)\times(c+di)=(ac-bd)+(bc+ad)i$$

Shown below is a program of the multiplication function complex_mul that uses the intrinsic multiply-accumulation function macl. complex_mul returns the result of multiplying a complex number a by a complex number b.

```
#include <machine.h>

/*
Multiplication of complex numbers.
Return the result of multiplying complex number a by complex number b.
   Note: The decimal point of the result is shifted by twice the number of
bits (both a and b) to the right.
 */
RXComplex complex_mul(RXComplex a, RXComplex b)
{
    RXComplex res;
    int16_t t;

    t = b.im;
    b.im = (int16_t)(-t);
    /* res.re <-- (a.re * b.re) - (a.im * b.im) */
    res.re = (int16_t)macl(&a.re, &b.re, 2);
    b.im = b.re;
    b.re = t;
    /* res.im <-- (a.re * b.im) + (a.im * b.re) */
    res.im = (int16_t)macl(&a.re, &b.re, 2);
    return res;
}
```

In the program above, in order to perform multiply-accumulation, the sign of b.im (imaginary part of b) is inverted before calculation of the real part, and values are exchanged between b.re (real part of b) and b.im before calculation of the imaginary part.

Note that when the input is fixed-point data, the decimal point of the result of multiplication is shifted to the right. For example, if the input is 16-bit signed fixed-point data with a decimal point between bits 15 and 14, the decimal point of the result of multiplication is shifted to between bits 14 and 13. In the program above, if necessary, replace the intrinsic multiply-accumulation function macl with macw1 or macw2 for fixed-point data.

3.3 Division

This section describes division. The definition of division of complex numbers is given below:

$$(a+bi)\div(c+di)=\left(\frac{ac+bd}{c^2+d^2}\right)+\left(\frac{bc-ad}{c^2+d^2}\right)i$$

Shown below is a program of the division function complex_div that uses the intrinsic multiply-accumulation function macl. complex_div returns the result of dividing a complex number a by a complex number b. Note, however, that if input data causes the dividend to be smaller than the divisor, the result will be incorrect. Also note that the decimal point of the result of division is not shifted (when input is regarded as fixed-point data).

```
#include <machine.h>

/*
   Division of complex numbers.
   Return the result of dividing complex number a by complex number b.
   Note: If the dividend is smaller than the divisor, the result will be
incorrect.
 */
RXComplex complex_div(RXComplex a, RXComplex b)
{
    RXComplex res;
    int16_t d, t;

    /* d <-- (b.re * b.re + b.im * b.im) */
    d = (int16_t)macl(&b.re, &b.re, 2);
    /* res.re <-- (a.re * b.re + a.im * b.im) / d */
    res.re = (int16_t)(macl(&a.re, &b.re, 2) / d);
    /* res.im <-- (a.im * b.re - a.re * b.im) / d */
    t = a.re;
    a.re = a.im;
    a.im = (int16_t)(-t);
    res.im = (int16_t)(macl(&a.re, &b.re, 2) / d);
    return res;
}
```

In the program above, in order to perform multiply-accumulation, values are exchanged between a.re (real part of a) and a.im (imaginary part of a) and at the same time the sign of a.re is inverted, before calculation of the imaginary part.

4. Sample Program: Conformal Map

This section gives a sample program of conformal map as an example of complex number operations. A conformal map is a conversion where angles between local lines are preserved. In most cases, a conformal map is represented by the following complex function as a map in a complex plane (z plane).

$$w = f(z)$$

The above function takes z as the original coordinate system, and maps it to w as the resulting coordinate system (assigns the real part and the imaginary part to the horizontal axis and the vertical axis, respectively). This section describes a program that plots the following z-squared conformal map.

$$w = z^2$$

4.1 Bitmap and Color Palette

This section describes a bitmap used to draw a graph.

The bitmap is a quadrilateral area with a width and a height, and is represented by an array of pixels. The depth of a pixel (data size) is 8 bits, and each pixel stores an index to a 256-color palette. Pixels on the bitmap are specified in a coordinate system where the x and y axes lead leftward and downward, respectively, from the origin placed at the top-right corner of the quadrangle. The horizontal pixel alignment is referred to as scan line, and the pixels on a scan line are aligned in ascending order from left to right. Also, the scan lines are aligned in ascending order from top to bottom in the bitmap.

The color palette to be referred to from the bitmap is represented by a 256-size 32-bit unsigned integer array. A pixel color is specified by the color palette in RGB888 format. In the sample program, the three colors white, red, and green are defined and used.

A program for what is described above is as follows:

```
/* constant(s) */
#define  WIDTH   202
#define  HEIGHT  202
#define  WHITE   0
#define  RED     1
#define  GREEN   2

/* bitmap and palette */
uint8_t bitmap[WIDTH * HEIGHT];
uint32_t palette[256] = {
    0xffffff,  /* [0] --> white */
    0xff0000,  /* [1] --> red */
    0x00ff00,  /* [2] --> green */
};
```

4.2 Drawing Subroutine

This section describes a subroutine used to draw a graph. The function "clear" below initializes the bitmap by painting it white.

```
/* Initialize bitmap (paint bitmap white) */
void clear(void)
{
    memset(bitmap, WHITE, sizeof bitmap);
}
```

The function set_pixel below draws a dot of a color c at the coordinates (x, y) on the bitmap. set_pixel is called from the function draw_line, which is used to draw a straight line.

```
/* Set the value of the pixel at (x, y) on bitmap to c. */
void set_pixel(int x, int y, int c)
{
    if (x >= 0 && x < WIDTH && y >= 0 && y < HEIGHT) {
        bitmap[x + y * WIDTH] = c;
     }
}
```

The function draw_line below draws a straight line of a color c between points (x1, y1) and (x2, y2) on the bitmap.

```
/* Draw straight line of pixel value c between (x1, y1) and (x2, y2) on bitmap.
*/
void draw_line(int x1, int y1, int x2, int y2, int c)
{
    int inc = 1;
    float d;
    float x = x1;
    float y = y1;

    if ((x1 < 0 && x2 < 0) || (y1 < 0 && y2 < 0) ||
        (x1 > WIDTH && x1 > WIDTH) ||
        (y1 > HEIGHT && x2 > HEIGHT)) {
        return; /* Outside the area: nothing is done */
    }
    if (x1 == x2 && y1 == y2) {
        /* dot */
        set_pixel(x1, y1, c);
        return;
    }
    if (abs(x1 - x2) > abs(y1 - y2)) {
        /* Slope is smaller than 45 degrees. (= Horizontally long) */
        d = (float)(y2 - y1) / (float)(x2 - x1);
        if (x1 > x2) {
            inc = -1;
            d = -d;
        }
        while (x1 != x2) {
            set_pixel(x1, (int)y, c);
            x1 += inc;
            y += d;
        }
    }
```

```
    else {
        /* Slope is larger than 45 degrees (= Vertically long) */
        d = (float)(x2 - x1) / (float)(y2 - y1);
        if (y1 > y2) {
            inc = -1;
            d = -d;
        }
        while (y1 != y2) {
            set_pixel((int)x, y1, c);
            y1 += inc;
            x += d;
        }
    }
}
```

4.3 Z-Plane Graph

The function plot_plain below plots as a grid a complex plane (z plane), which is the origin of images under a conformal map.

```
/* Plot a plane grid before mapping */
void plot_plain(void)
{
    int i, j, x, y, x0, y0;

    clear();
    for (j = -100; j <= 100; j += 10) {
        for (i = -100; i <= 100; i += 10) {
            x = i + WIDTH / 2;
            y = j + HEIGHT / 2;
            if (i != -100) {
                draw_line(x0, y0, x, y, RED);
            }
            x0 = x;
            y0 = y;
        }
    }
    for (i = -100; i <= 100; i += 10) {
        for (j = -100; j <= 100; j += 10) {
            x = i + WIDTH / 2;
            y = j + HEIGHT / 2;
            if (j != -100) {
                draw_line(x0, y0, x, y, GREEN);
            }
            x0 = x;
            y0 = y;
        }
    }
}
```

Calling this function draws the graph in figure 1 on the bitmap. In the graph, the horizontal axes (real part) and vertical axes (imaginary part) on the z plane are drawn in red and green, respectively.

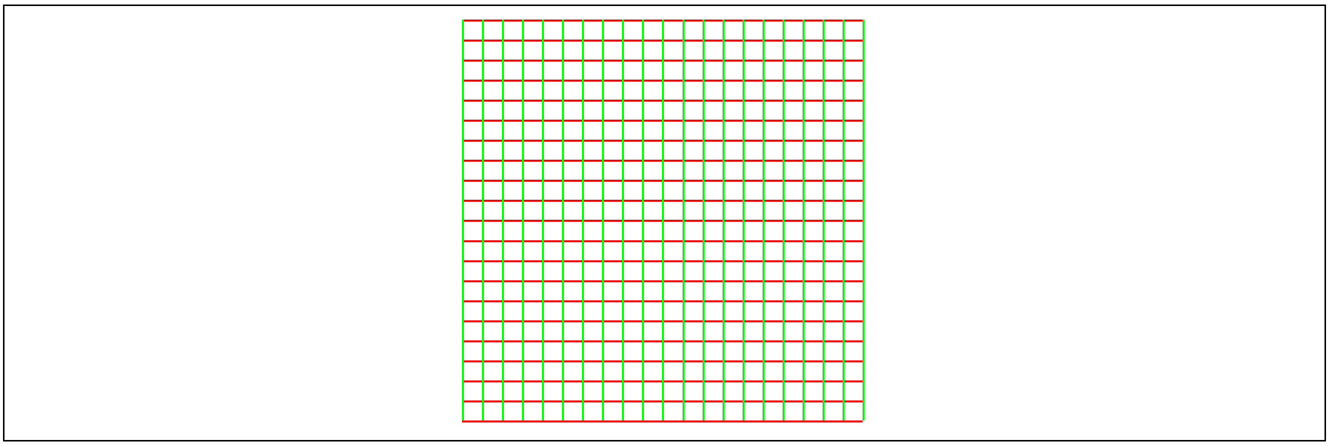

Figure 1 Z-Plane Graph (Red is Horizontal Axes, Green is Vertical Axes)

4.4 Z-Squared Conformal Map Graph

The function plot_square below draws a graph as a grid to show how a complex plane (z plane) is mapped through the following z-squared conformal map.

$$w = z^2$$

```
/* Plot a plane grid mapped by the mapping expression (z^2). */
void plot_square(void)
{
    int i, j, x, y, x0, y0;
    RXComplex z;
    const float scale_x = 0.005;
    const float scale_y = 0.005;
    const int translate_x = 100;
    const int translate_y = 100;

    clear();
    for (j = -100; j <= 100; j += 10) {
        for (i = -100; i <= 100; i++) {
            z.re = i;
            z.im = j;
            z = complex_mul(z, z);
            x = translate_x + z.re * scale_x;
            y = translate_y + z.im * scale_y;
            if (i != -100) {
                draw_line(x0, y0, x, y, RED);
            }
            x0 = x;
            y0 = y;
        }
    }
    for (i = -100; i <= 100; i += 10) {
        for (j = -100; j <= 100; j += 1) {
            z.re = i;
            z.im = j;
            z = complex_mul(z, z);
            x = translate_x + z.re * scale_x;
            y = translate_y + z.im * scale_y;
            if (j != -100) {
                draw_line(x0, y0, x, y, GREEN);
            }
            x0 = x;
            y0 = y;
        }
    }
}
```

Calling this function draws the graph in figure 2 on the bitmap. In the graph, the mapped horizontal axes (real part) and the mapped vertical axes (imaginary part) on the z plane are drawn in red and green, respectively.

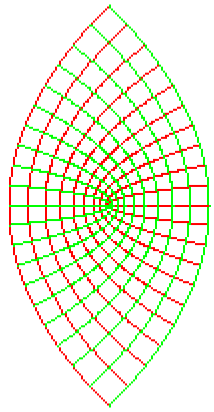

Figure 2 Z-Squared Conformal Map Graph (Red is Horizontal Axes, Green is Vertical Axes)

4.5 Main Program

The main program (main function) is shown below. The main program merely sequentially calls functions that are used to display a graph. When the sample program is executed in the RX's integrated development environment (High-performance Embedded Workshop), setting a breakpoint at the function pause allows the program to be paused immediately after the graph is displayed.

```
/* Set a breakpoint at this function and check the displayed graph */
void pause(void)
{
}

void main(void)
{
    plot_plain();
    pause();
    plot_square();
    pause();
}
```

4.6 How to Display Bitmap Image

The procedure below allows the bitmap image to be displayed on the window of the RX's integrated development environment (High-performance Embedded Workshop).

1. Select "Screen" -> "Graphic" -> "Image" from the High-performance Embedded Workshop's menu bar.
2. The "Image Property" dialog is displayed (See figure 3).
3. Select "RGB" from "Mode" in "Color Info" on the dialog.
4. Select "8 bits (Index Color)" from "Bits/Pixels" on the dialog.
5. Set the symbol "_bitmap" for "Data Address" on the dialog.
6. Set the symbol "_palette" for "Pallet Address" on the dialog.
7. Enter 202 (width of bitmap) for "Width" on the dialog.
8. Enter 202 (height of bitmap) for "Height" on the dialog.
9. Click "OK" to close the dialog.
10. The High-performance Embedded Workshop's window to display the image is opened.

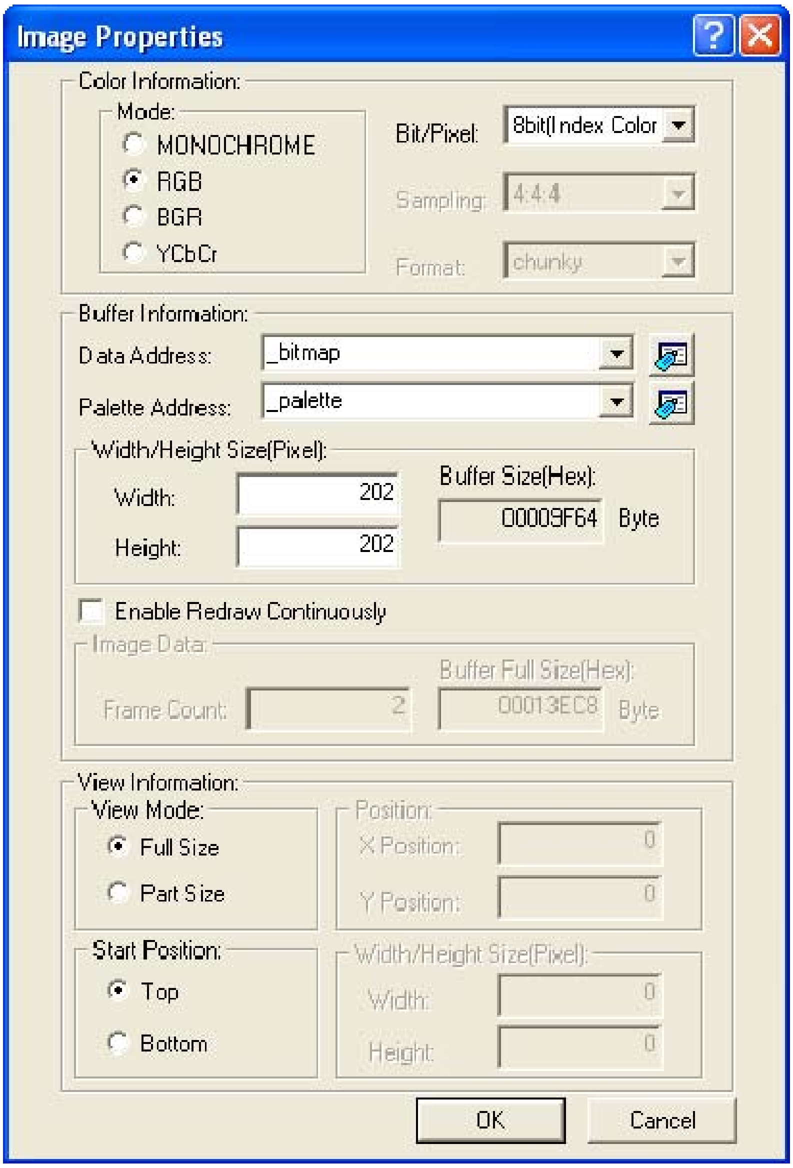

Figure 3 "Image Property" Dialog

Website and Support

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

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

Revision Record

		Description	
Rev.	Date	Page	Summary
1.00	Mar 14, 2011	—	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 accord 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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, 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: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

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: +852-2886-9318, Fax: +852 2886-9022/9044

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

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

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.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141