APPLICATION NOTE

RX Family

Application Example of Multiple precision Arithmetic: Computing Mersenne Primes

R01AN0230EJ0100
Rev.1.00
Mar 14, 2011

Introduction

This document explains how to compute Mersenne primes as an example of a multiple precision arithmetic program using the DSP instructions for the RX family microcomputers.

Target Device

RX Family

Contents

1. General 2

2. Data Representation Multiple precision Numbers 2

3. Mersenne Primes 3

4. Multiple precision Arithmetic Program 4

5. Computing Mersenne Primes 11

1. General

Multiple precision arithmetic refers to numerical computations on numbers of a precision higher than that which can be handled directly with the hardware instructions of a computer. The range of numbers that can be handled directly by a 32-bit microcontroller such as the RX family is limited to from 0 to 2^{32} – 4294967295 (assuming that the numbers are unsigned integers). A program that carries out multiple precision arithmetic is required to make calculations beyond such hardware limitations. Generally, multiple precision arithmetic is used in applications in which the precision provided by hardware-implemented fixed-precision arithmetic would be inadequate or any overflows of computation results would bring about some problems. Some four multiple precision arithmetic operation programs using the RX's DSP instructions are described in the application note entitled Multiple precision Multiplication Program Using the DSP instructions (R01AN0226EJ). This document describes a program for finding Mersenne primes as an application example of a multiple precision arithmetic program.

For details on the multiple precision arithmetic algorithm, refer to the following document.

Note: [1] D. E. Knuth, Seminumerical Algorithms, The Art of Computer Programming, Vol. 2, pp.265-284, 3rd edition, Addison Wesley, 1997.

2. Data Representation Multiple precision Numbers

Numbers of which precision exceeds the limits of values that a computer hardware can handle directly are called multiple precision numbers. This chapter describes the data representation of multiple precision numbers that are subject to multiple precision arithmetic.

The multiple precision numbers are assumed to be unsigned integers. The RX has a 16 bits × 16 bits multiply-accumulate instruction. To take advantage of this multiply-accumulate instruction to implement a multiplication program for multiple precision numbers, a multiple precision number is assumed to be represented by an array of 16-bit unsigned integers in this application note.

A 16-bit unsigned integer representation can represent 2^{16} numbers. By regarding each element of the array as a digit in a notation system of base 2^{16}, an array of 16-bit unsigned integers with a length of N can be used to represent an integer of N digits in a notation system of base 2^{16}. In addition, it is predefined that the elements of the array for a number of N digits are arranged in the ascending order toward the uppermost digit, with the element with a subscript of 0 being designated as the lowermost digit.

This is illustrated in figure 1.

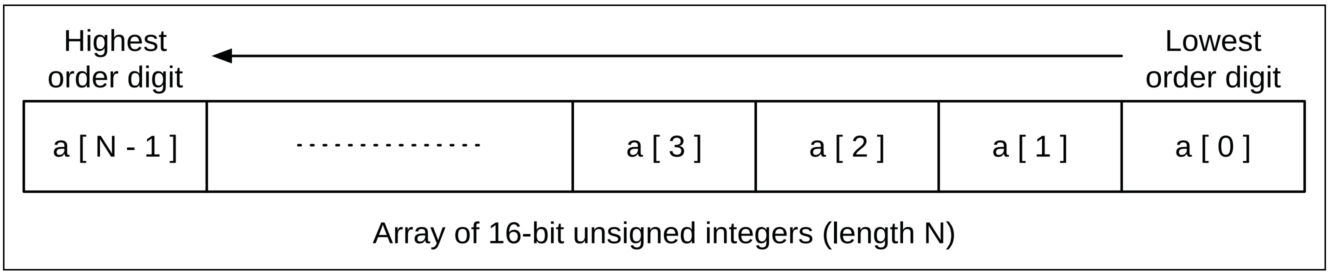

Figure 1 Data Representation of a Multiple precision Number Using an Array of 16-bit Unsigned Integers

The numerical value stored in the multiple precision number a[N] can be represented by the following expression:

$$a[N-1]\times 2^{16\times(N-1)} + \ldots + a[2]\times 2^{32} + a[1]\times 2^{16} + a[0]$$

An example of a C program fragment for a multiple precision number that corresponds to figure 1 is given below. In this program, the number of digits N is set to 500 and unsigned integers with a maximum of $2^{8000} - 1$ of magnitude are assumed to be handled.

```
#include <stdint.h>
#define N       500      /* Length of multiple precision number (for an array
of 16-bit unsigned integers) */
uint16_t a[N];
```

3. Mersenne Primes

A Mersenne number is a number that is equal to one less than a power of 2, or that is represented in the following form:

$$M_p = 2^p - 1$$

A Mersenne number which is a prime number is especially called a Mersenne prime. Mersenne primes that satisfy $p <$ 3000 are known to be the following:

p = 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281

As conjectured from the above numerical sequence, it is known that p must be a prime number for a Mersenne number to be a prime number. The rest of this document focuses only on Mersenne numbers in which p is a prime number.

For reference's sake, on a digital computer which is based on the binary number system, the internal representation of a Mersenne number is a sequence of consecutive 1 bits. For example, the Mersenne number in which p = 13 is represented by 1111111111111 in the binary system. This is equal to the value that is obtained simply by subtracting 1 from the results of shifting a 1 13 bits to the left. In this way, a digital computer can easily make Mersenne numbers using left shift and subtraction. It is, however, not so easy to determine whether a Mersenne number is a prime number.

3.1 Lucas-Lehmer Primality Test

Since Mersenne numbers generally turn to be very huge integers, it is difficult to determine if they are primary numbers with any ordinary methods. Determining if a Mersenne number is a primary is accomplished by a method that is known as the Lucas-Lehmer Primality Test. A description of the Lucas-Lehmer Primality Test follows.

First, define a numerical sequence S_i (i = 0, 1, 2, …) for the odd primary p as follows:

$$S_0 = 4$$
$$S_{i+1} = S_i^2 - 2$$

In this case, the necessary and sufficient condition for the Mersenne number M_p to be a primary number is given as follows:

$$S_{p-2} \equiv 0 \bmod M_p$$

That is, M_p is a primary number if S_p–2 is divisible by M_p (the remainder is 0).

In an actual program, however, the Lucas-Lehmer Primality Test is used in the way as shown below. First, define a numerical sequence S_i (i = 0, 1, 2, …) for the odd primary p as follows:

$$S_0 = 4$$
S_{i+1} = Remainder obtained by dividing $(S_i^2 - 2)$ by M_p

In this case, the necessary and sufficient condition for the Mersenne number M_p to be a prime number is as follows:

$$S_{p-2} = 0$$

Since a remainder is taken for each term with this method, S_i can never be too large to be computable.

4. Multiple precision Arithmetic Program

This chapter introduces multiple precision arithmetic programs that are used for computing Mersenne primes. This is a worked out and improved version of the arithmetic program that is explained in the application note entitled Multiple precision Multiplication Program Making Use of the DSP Instructions (R01AN0226EJ).

4.1 Zero-clearing and Copying of Values

To improve program readability, define the function long_clr that zero-clears multiple precision numbers and the function long_cpy that copies values.

```
#include <string.h>

/*
  Clears multiple precision number a to 0.
 */
void long_clr(uint16_t *a)
{
    memset(a, 0, sizeof(uint16_t) * N);
}

/*
  Copies multiple precision number b to a.
 */
void long_cpy(uint16_t *a, uint16_t *b)
{
    memcpy(a, b, sizeof(uint16_t) * N);
}
```

4.2 Addition, Subtraction, and Comparison

The program for the multiple precision addition function long_add is given below. This function adds the value of multiple precision number b to multiple precision number a and places the results in a. This function is coded in assembly language. Accordingly, the #pragma inline_asm declaration is used.

```
/*
  Adds together multiple precision numbers
  Results are placed in a.
 */
#pragma inline_asm long_add
void long_add(uint16_t *a, uint16_t *b)
{
        mov.l   #0,r4
        mov.l   #N,r5
  ?:
        movu.w  [r1],r3
        add     r3,r4
        movu.w  [r2+],r3
        add     r3,r4
        mov.w   r4,[r1+]
        shlr    #16,r4
        sub     #1,r5
        bnz     ?-
}
```

Next, the program for multiple precision subtraction function long_sub is shown below. This function subtracts the value of multiple precision number b from multiple precision number a and places the results in a. However, the inequality a ≥ b must be observed. This function is coded in assembly language. Accordingly, the #pragma inline_asm declaration is used.

```
/*
  Subtraction on multiple precision numbers (a >= b must be observed)
  Results are placed in a.
 */
#pragma inline_asm long_sub
void long_sub(uint16_t *a, uint16_t *b)
{
        mov.l   #0,r4
        mov.l   #N,r5
  ?:
        movu.w  [r1],r3
        add     r3,r4
        movu.w  [r2+],r3
        sub     r3,r4
        mov.w   r4,[r1+]
        shar    #16,r4
        sub     #1,r5
        bnz     ?-
}
```

Finally, the program for the multiple precision number comparison function long_cmp is given below. This function compares two multiple precision number a and b and returns 0 if a = b, –1 if a < b, and 1 if a > b.

```
/*
  Compares between multiple precision numbers
  Returns 1 if a > b, 0 if a == b, and -1 if a < b.
 */
int long_cmp(uint16_t *a, uint16_t *b)
{
    int i;
    int32_t c;

    for (i = N - 1; i >= 0; i--) {
        c = (int32_t)a[i] - (int32_t)b[i];
        if (c < 0) {
            return -1;
        }
        if (c > 0) {
            return 1;
        }
    }
    return 0;
}
```

4.3 Multiplication Program

This section gives a program for the function long_mul which performs multiplications on multiple precision numbers. This function performs a multiplication on multiple precision numbers a and b and places the results in a. This function references mul16 and add16 as auxiliary function. The function mul16 is coded in assembly language. Accordingly, the #pragma inline_asm declaration is used.

```
/*
  Multiplies multiple precision numbers
  Results are placed in a.
 */
void long_mul(uint16_t *a, uint16_t *b)
{
    int i, j;
    uint32_t x;
    static uint16_t res[N];

    memset(res, 0, sizeof res);
    for (i = 0; i < N; i++) {
        if (a[i] != 0) {
            for (j = 0; j < N; j++) {
                if (b[j] != 0 && i + j < N) {
                    x = mul16(a[i], b[j]);
                    add16(res, i + j, (x & 0xffff));
                    add16(res, i + j + 1, (x >> 16));
                }
            }
        }
    }
    memcpy(a, res, sizeof res);
}

/*
  Multiplies 16-bit unsigned integers.
  Returns a 32-bit unsigned integer as results.
 */
#pragma inline_asm mul16
static uint32_t mul16(uint16_t a, uint16_t b)
{
        push.l  r6
        mov.l   r1,r3
        and     #7fffh,r3
        mov.l   r2,r4
        and     #7fffh,r4
        mov.l   #0,r5
        tst     #8000h,r1
        bz      ?+
        mov.l   r4,r6
        shll    #15,r6
        add     r6,r5
  ?:
        tst     #8000h,r2
        bz      ?+
        mov.l   r3,r6
        shll    #15,r6
        add     r6,r5
        tst     #8000h,r1
        bz      ?+
```

```
        add     #40000000h,r5
  ?:
        mullo   r3,r4
        mvfacmi r1
        add     r5,r1
        pop     r6
}

/*
  Adds 16-bit unsigned integer b to ith digit of multiple precision number a.
 */
static void add16(uint16_t *a, int i, uint16_t b)
{
    uint32_t c;

    for (c = b ; c > 0 && i < N; i++) {
        c += a[i];
        a[i] = c;  // Only lower 16 bits of c are transferred.
        c >>= 16;  // Upper 16 bits of c hold the value of the carry.
    }
}
```

4.4 Division

The program for the multiple precision number division function long_div is shown below. This function divides the value of multiple precision number a by the value of multiple precision number b and places the quotient in q and the remainder in r. However, the inequality b > 0 must be observed. This function references guess, lshl, lshr, and llen as auxiliary function. The function guess is coded in assembly language. Accordingly, the #pragma inline_asm declaration is used.

```
/*
  Performs division on multiple precision numbers (inequality b > 0 must be
observed).
  Places the quotient in q and the remainder in r.
 */
void long_div(uint16_t *a, uint16_t *b, uint16_t *q, uint16_t *r)
{
    int i, m, n, shift;
    uint32_t u, quot;
    static uint16_t c[N], d[N], e[N];

#define ZERO(x)        memset(x, 0, sizeof(uint16_t) * N)
#define COPY(x, y)      memcpy(x, y, sizeof(uint16_t) * N)

    /* initialize */
    ZERO(e);
    ZERO(q);
    COPY(r, a);
    n = llen(b) - 1;
    if (long_cmp(a, b) < 0 || n < 0) {
        return;
    }
    /* normalize */
    for (shift = 0, u = b[n]; (u & 0x8000) == 0; u <<= 1) {
        shift++;
    }
    lshl(r, shift);
```

```
    lshl(b, shift);
    /* loop */
    while (long_cmp(r, b) >= 0) {
        m = llen(r) - 1;
        if (r[m] >= b[n]) {
            ZERO(c);
            for (i = 0; i <= n; i++) { c[m - n + i] = b[i]; }
            if (long_cmp(r, c) >= 0) {
                q[m - n] = 1;
                long_sub(r, c);
                continue;
            }
        }
        quot = guess(r, b, m, n);
        ZERO(c);
        for (i = 0; i <= n; i++) { c[m - n - 1 + i] = b[i]; }
        COPY(d, c);
        e[0] = quot;
        long_mul(c, e);
        while (long_cmp(r, c) < 0) {
            long_sub(c, d);
            quot--;
        }
        q[m - n - 1] = quot;
        long_sub(r, c);
    }
    /* unnormalize */
    lshr(r, shift);
    lshr(b, shift);

#undef  ZERO
#undef  COPY
}

/* Auxiliary functions for long_div */
#pragma inline_asm guess
static uint32_t guess(uint16_t *a, uint16_t *b, int c, int d)
{
        shll    #01h,r3,r5
        add     r1,r5
        movu.w  [r5],r1
        sub     #02h,r5
        shll    #10h,r1
        add     [r5].uw,r1
        movu.w  [r4,r2],r5
        divu    r5,r1
        cmp     #0ffffh,r1
        bleu    ?+
        mov.l   #0ffffh,r1
  ?:
}

/*
  Shifts multiple precision number a n bits to the left.
  0 <= n <= 15 must be observed.
 */
static void lshl(uint16_t *a, int n)
{
```

```
    int i;
    uint32_t c = 0;
    uint32_t t;

    if (n == 0) {
        return;
    }
    for (i = 0; i < N; i++) {
        t = (uint32_t)a[i];
        t <<= n;
        t |= c;
        a[i] = t;
        c = (t >> 16);
    }
}

/*
  Shifts multiple precision number a n bits to the right.
  0 <= n <= 15 must be observed.
 */
static void lshr(uint16_t *a, int n)
{
    int i;
    uint16_t c = 0;
    uint16_t t;

    if (n == 0) {
        return;
    }
    for (i = N - 1; i >= 0; i--) {
        t = a[i];
        a[i] = (c | (t >> n));
        c = (t << (16 - n));
    }
}

/*
  Returns number of digits of a multiple precision number
  Returns 0 if a == 0
 */
static int llen(uint16_t *a)
{
    int i;

    for (i = N - 1; i >= 0; i--) {
        if (a[i] != 0) {
            return i + 1;
        }
    }
    return 0;
}
```

4.5 Bit Shifting

Given below are functions long_shl and long_shr which shift a multiple precision number left or right on a bit basis. These functions shift multiple precision number a n bits to the left or right on a bit basis. The number of bits n to shift may be 15 or greater. These functions reference lshl, lshr, and llen which are the auxiliary functions for the division program introduced in the preceding section.

```
/*
  Shifts multiple precision number a n bits to the left
 */
void long_shl(uint16_t *a, int n)
{
    int i, j, k;
    static uint16_t t[N];

    if (n <= 15) {
        lshl(a, n);
    }
    else {
        j = n / 16;     /* Number of bits to shift left */
        n -= j * 16;    /* Number of remaining bits to shift */
        k = llen(a);    /* Number of digits of a */
        memset(t, 0, sizeof t);
        for (i = 0; i < k && i + j < N; i++) {
            t[i + j] = a[i];
        }
        lshl(t, n);
        memcpy(a, t, sizeof t);
    }
}

/*
  Shifts multiple precision number a n bits to the right
 */
void long_shr(uint16_t *a, int n)
{
    int i, j, k;
    static uint16_t t[N];

    if (n <= 15) {
        lshr(a, n);
    }
    else {
        j = n / 16;     /* Number of bits to shift right */
        n -= j * 16;    /* Number of remaining bits to shift */
        k = llen(a);    /* Number of digits of a */
        memset(t, 0, sizeof t);
        for (i = k - 1; i - j >= 0; i--) {
            t[i - j] = a[i];
        }
        lshr(t, n);
        memcpy(a, t, sizeof t);
    }
}
```

5. Computing Mersenne Primes

This chapter presents a program that finds Mersenne primes. The main program (main function) tests to determine whether Mersenne numbers ($2^p - 1$) corresponding to the odd primes *p* that start at 5 and smaller than 3000 sequentially using the Lucas-Lehmer Primality Test.

The main program references the three auxiliary functions, i.e., divisor, next_prime, and print_mersenne_prime.

divisor is a function that returns the smallest divisor of integer n. divisor tries to divide n by odd numbers with magnitudes from 3 up to sqrt(n) sequentially and finds the smallest divisor (other than 1) of n.

The next_prime function returns the smallest prime number that is greater than the given integer n. next_prime calls divisor and verifies that the smallest divisor other than 1 equals that number, whereby identifying the prime number.

print_mersenne_prime is a function that is called by the main program when a Mersenne prime is found. When the program is to be run in the integrated development environment HEW, the user can verify the Mersenne prime that has been found by setting a breakpoint in this function.

```
#include <stdint.h>
#include <math.h>
#include "RXlong.h"

/* Returns smallest divisor of n */
static int divisor(n)
{
    int i, root;

    root = (int)sqrt(n);
    for (i = 3; i <= root; i += 2) {
        if (n % i == 0) {
            return i;
        }
    }
    return n;
}

/* Returns smallest prime that is greater than or equal to n */
static int next_prime(int n)
{
    if (n <= 3) {
        return 3;
    }
    if (n % 2 == 0) {
        n += 1;/* Odd prime */
    }
    while (divisor(n) != n) {
        n += 2;
    }
    return n;
}

/* Displays Mersenne prime p */
static void print_mersenne_prime(int p)
{
    /* printf("mersenne prime: p = %d\n", p); */
}

void main(void)
{
    int p, i;
```

```
    static uint16_t M[N], S[N], a[N], q[N];

    /*
Uses the Lucas-Lehmer Primality Test to determine whether (2 ^ p - 1) is a
Mersenne prime for odd primes p that starts at 5 and grows in ascending order
and not exceeds 3000.
    */
    for (p = 5; p < 3000; p = next_prime(p + 2)) {
        /* M <-- 2 ^ p - 1 */
        long_clr(M);
        M[0] = 1;
        long_shl(M, p);
        long_clr(a);
        a[0] = 1;
        long_sub(M, a);

        /* S <-- 4 */
        long_clr(S);
        S[0] = 4;
        for (i = 1; i < n - 1; i++) {
            /* S <-- S ^ 2 - 2 */
            long_cpy(a, S);
            long_mul(S, a);
            long_clr(a);
            a[0] = 2;
            long_sub(S, a);
            /* S <-- S mod M */
            long_div(S, M, q, a);
            long_cpy(S, a);
        }
        if (long_nil(S)) {
            /* A Mersenne prime if S == 0*/
            print_mersenne_prime(p);
        }
    }
}
```

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	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.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system 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. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

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

Renesas Electronics America Inc.
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