

# RX Family

R21AN0003EJ0100

Rev.1.00

## RX Fixed-point Library

---

Mar 03, 2011

### Introduction

This document describes the usage of RX Fixed-point Library

### Target Device

RX Family

**Content**

1. Fixed-point Library..... 4

1.1 Overview..... 4

1.1 Format of Fixed-point Data..... 4

1.3 Library Files ..... 4

1.4 Example of Usage ..... 5

1.5 Notes on Library Usage..... 6

2. Specification of Fixed-point Library ..... 7

2.1 "fixmath.h" ..... 7

2.2 Description of Functions ..... 12

2.2.1 Multiplication (macro) ..... 12

2.2.2 Division (macro)..... 12

2.2.3 Conversion (macro)..... 13

2.2.4 Multiplication..... 15

2.2.5 Sine Function..... 15

2.2.6 Cosine Function..... 16

2.2.7 Arctangent Function ..... 16

2.2.8 Square Root Function ..... 17

2.2.9 Multiplication (fraction part) (macro)..... 17

2.2.10 Multiplication (saturated) (macro)..... 18

2.2.11 Multiplication (FIX-type) (macro) ..... 18

2.2.12 Multiplication (FIX-type)..... 19

2.2.13 Multiplication (fraction part, FIX-type) (macro) ..... 19

2.2.14 Multiplication (fraction part, FIX-type) (macro) ..... 20

2.2.15 Multiplication (saturated, FIX-type) (macro) ..... 20

2.2.12 Multiplication (saturated, FIX-type)..... 21

3. Performance and Precision ..... 22

3.1 Evaluation Condition ..... 22

3.2 Execution Cycles ..... 22

3.3 Precision ..... 22

## 1. Fixed-point Library

### 1.1 Overview

This library provides real-number operations using fixed-point format<sup>1</sup> for RX Family.

The fixed-point library enables fast real-number operations, especially on CPU's without FPU.

This library supports the following functions for fixed-point type with 16, 24, or 29 fraction bits.

1. Multiplication and division
2. Mathematical functions (sin, cos, atan, and sqrt)
3. Conversion between floating point data.

Use 16-bit or 24-bit depending on the required precision of your application. 29-bit precision is supported as the most precise type which can represent the input range of trigonometric functions ( $-\pi \sim +\pi$ ).

In fixed-point arithmetic, the range of values is restricted compared with floating point. So appropriate precision should be selected according to the input/output values of each operations. For this reason, this library supports multiplication, division, conversion for all the precision (from 1 to 31) of fixed-point type.

### 1.1 Format of Fixed-point Data

Following is the format of fixed-point data supported in this library.

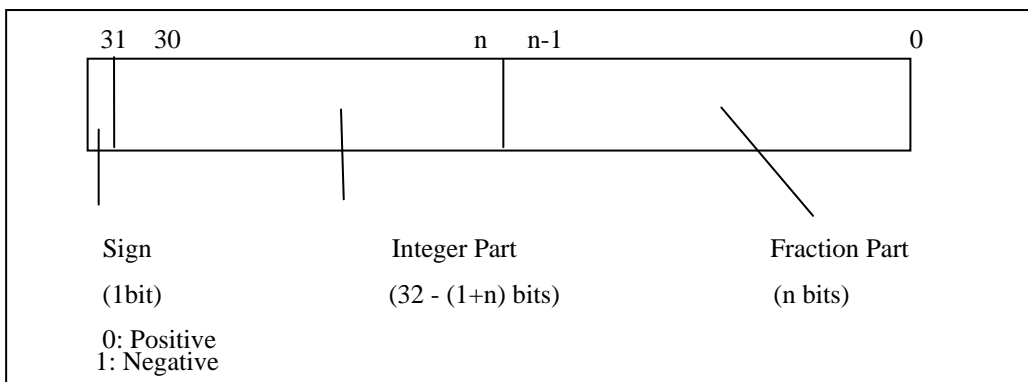


Figure 1. Fixed-point Data Format

According to the number of bits in fraction part, types from FIX1 to FIX31 are supported. The number indicates the number of bits in the fraction type.

Generic fixed-point type FIX is also supported, and generic fixed-point operations are supported for this type.

### 1.3 Library Files

The following include file and library files are provided.

When using this library, include the file indicated in table 1, and link the library file (corresponding to the compiler option) indicated in table 2.

<sup>1</sup> Fixed-point format represents a real number by assuming a decimal point at some fixed bit position.

Table 1. Include File for Fixed-point Library

Library	Function	
Fixed-point library	Implements fixed-point operations	"fixmath.h"

Table 2. Fixed-point Libraries

Library name	Compiler Option	
	cpu	endian
RXfixmath_big.lib	RX	big
RXfixmath_little.lib	RX	little

Before using, copy these files into your local include or library directories.

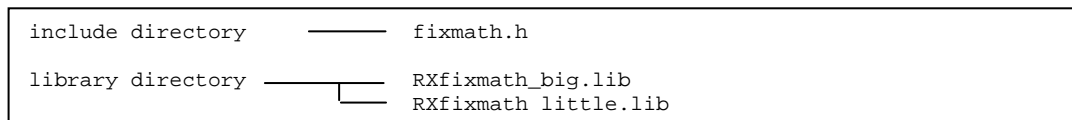


Figure 2. Sample Configuration

### 1.4 Example of Usage

The following example shows a program using FIX16 operation and how to specify the library undef High-performance Embedded-Workshop.

[Source Program]

```

#include <stdio.h>
#include "fixmath.h" // Necessary when using
                    // fixed-point library

void main()
{
    float r_flt;
    FIX16 d_fix16, r_fix16;

    d_fix16 = FIX16_fromfloat(3.14f); // convert float type constant
                                    // to FIX16
    r_fix16 = FIX16_sin(d_fix16);    // computes sin
    r_flt = FIX16_tofloat(r_fix16);  // Convert back for printing
    printf("%f\n", r_flt);
}
    
```

[How to specify the library under High-Performance Embedded-Workshop]

Select [RX Standard Toolchain ...] in [build] menu. In the dialog box [RX Standard Toolchain ...], select tab [optimizing linker], and specify "input" for [category] and specify "library file" for [option item], and click [add] button. In the dialog box [Add library file], select the library to be linked.

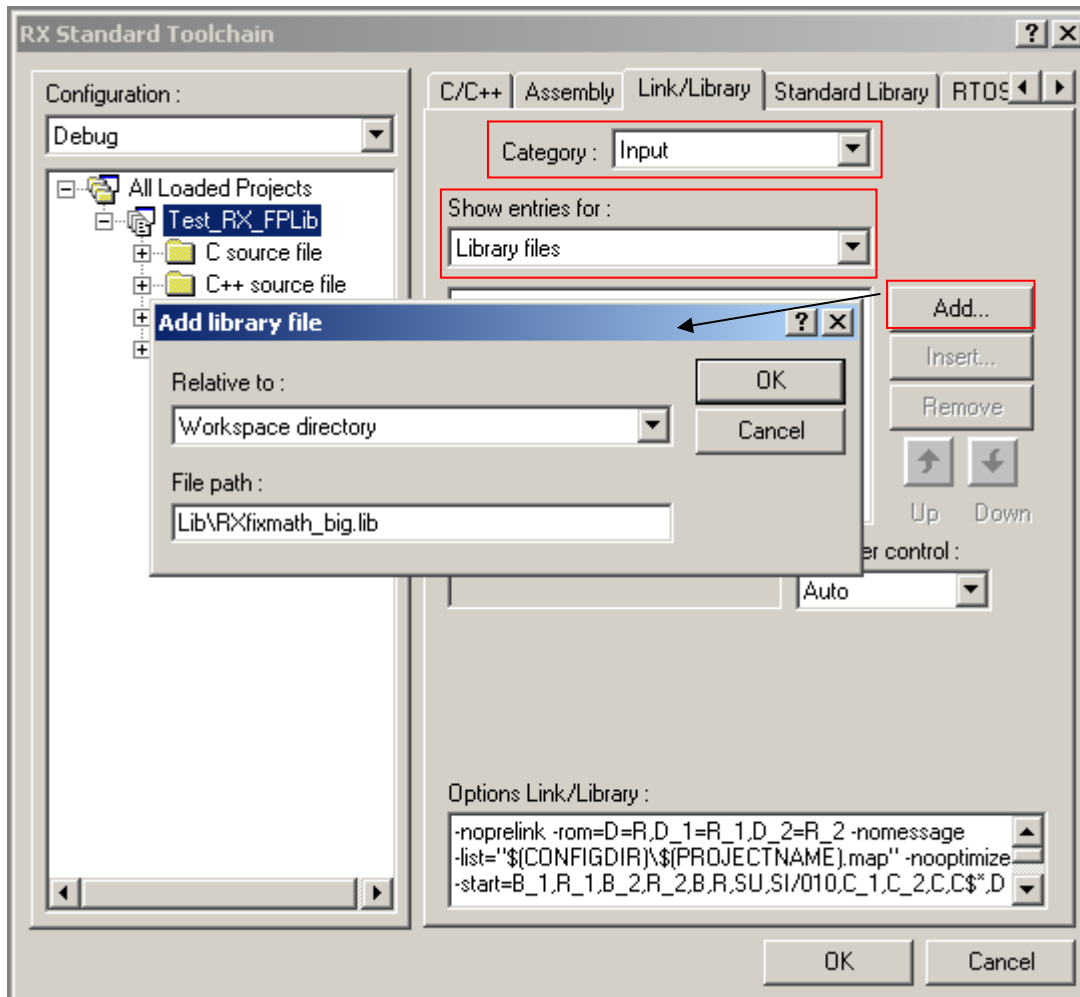


Figure 3. Specifying library

[To be replaced with English version window]

### 1.5 Notes on Library Usage

If the result of operation or conversion exceeds the range of fixed-point type, the result is not guaranteed.

## 2. Specification of Fixed-point Library

### 2.1 "fixmath.h"

This header file defines types and functions for fixed-point operations.

Table 3 shows the types defined in the file and supported functions (macros).

NOTATION: The notation <n> in type, function, or macro names represents a number from 1 to 31. The number in the function or macro name corresponds to the number in the type name.

Table 3. Types and Supported Functions

Type	Supported functions and macros
FIX1-FIX31 (except FIX16, FIX24 or FIX29)	FIX<n>_mul_short, FIX<n>_mul, FIX<n>_div, FIX<n>_tfloat, FIX<n>_fromfloat, FIX<n>_todouble, FIX<n>_fromdouble, FIX<n>_mul_frac, FIX<n>_mul_sat
FIX16, FIX24, FIX29	FIX<n>_mul_short, FIX<n>_mul, FIX<n>_div, FIX<n>_tfloat, FIX<n>_fromfloat, FIX<n>_todouble, FIX<n>_fromdouble, FIX<n>_mul_frac, FIX<n>_mul_sat, FIX<n>_sin, FIX<n>_cos, FIX<n>_atan, FIX<n>_sqrt
FIX	FIX_mul_scale<n>, FIX_mul_frac_scale<n>, FIX_mul_sat_scale<n>, FIX_mul_scale, FIX_mul_frac_scale, FIX_mul_sat_scale

These types are defined as long type.

When the operands and the result of an operation are the same type (FIX<n>), use the function corresponding to that type. Otherwise, use a function corresponding to the generic fixed-point type FIX.

#### [Hints on Fixed-point Library Usage]

- (1) Select one of the standard fixed-point type (FIX16 or FIX24) according to the requirement of your application.
- (2) Compared with floating-point types, fixed-point types have limited range of values. It is recommended to select appropriate fixed-point types according to the range of input or intermediate result, or required precision of arithmetic.
- (3) When converting data between different fixed-point types, use shift operator of C language.

Example: Conversion from FIX16 to FIX24

```
FIX16 x, FIX24 y;  
x=y>>8;
```

- (4) When adding or subtracting between data of the same fixed-point type, use integer addition or subtraction of the C language.

Example: Addition of FIX16.

```
FIX16 x, y, z;  
z=x+y;
```

- (5) Conversion between floating-point types and fixed-point types should be done only when required. Unnecessary conversions reduces the efficiency. But the conversion function applied to a constant generates a constant expression by expanding a macro, and fixed-point constant can be specified without any overhead.

Example: Fixed-point constant.

```
FIX16 x;  
x=FIX16_fromfloat(3.14f);
```

Table 4 shows the representations and ranges of fixed types.

Table 4. Representation and Ranges of Fixed Types

Type	Size (byte)	Alignment (byte)	Sign	Range	
				Minimum Value	Maximum Value
FIX1	4	4	signed	$-2^{30}(-1073741824.0)$	$2^{30}-2^{-1}(1073741823.5)$
FIX2	4	4	signed	$-2^{29}(-536870912.0)$	$2^{29}-2^{-2}(536870911.75)$
FIX3	4	4	signed	$-2^{28}(-268435456.0)$	$2^{28}-2^{-3}(268435455.875)$
FIX4	4	4	signed	$-2^{27}(-134217728.0)$	$2^{27}-2^{-4}(134217727.9375)$
FIX5	4	4	signed	$-2^{26}(-67108864.0)$	$2^{26}-2^{-5}(67108863.96875)$
FIX6	4	4	signed	$-2^{25}(-33554432.0)$	$2^{25}-2^{-6}(33554431.984375)$
FIX7	4	4	signed	$-2^{24}(-16777216.0)$	$2^{24}-2^{-7}(16777215.9921875)$
FIX8	4	4	signed	$-2^{23}(-8388608.0)$	$2^{23}-2^{-8}(8388607.99609375)$
FIX9	4	4	signed	$-2^{22}(-4194304.0)$	$2^{22}-2^{-9}(4194303.998046875)$
FIX10	4	4	signed	$-2^{21}(-2097152.0)$	$2^{21}-2^{-10}(2097151.9990234375)$
FIX11	4	4	signed	$-2^{20}(-1048576.0)$	$2^{20}-2^{-11}(1048575.99951171875)$
FIX12	4	4	signed	$-2^{19}(-524288.0)$	$2^{19}-2^{-12}(524287.999755859375)$
FIX13	4	4	signed	$-2^{18}(-262144.0)$	$2^{18}-2^{-13}(262143.9998779296875)$
FIX14	4	4	signed	$-2^{17}(-131072.0)$	$2^{17}-2^{-14}(131071.99993896484375)$
FIX15	4	4	signed	$-2^{16}(-65536.0)$	$2^{16}-2^{-15}(65535.999969482421875)$
FIX16	4	4	signed	$-2^{15}(-32768.0)$	$2^{15}-2^{-16}(32767.9999847412109375)$
FIX17	4	4	signed	$-2^{14}(-16384.0)$	$2^{14}-2^{-17}(16383.99999237060546875)$
FIX18	4	4	signed	$-2^{13}(-8192.0)$	$2^{13}-2^{-18}(8191.999996185302734375)$
FIX19	4	4	signed	$-2^{12}(-4096.0)$	$2^{12}-2^{-19}(4095.9999980926513671875)$
FIX20	4	4	signed	$-2^{11}(-2048.0)$	$2^{11}-2^{-20}(2047.99999904632568359375)$
FIX21	4	4	signed	$-2^{10}(-1024.0)$	$2^{10}-2^{-21}(1023.999999523162841796875)$
FIX22	4	4	signed	$-2^9(-512.0)$	$2^9-2^{-22}(511.9999997615814208984375)$
FIX23	4	4	signed	$-2^8(-256.0)$	$2^8-2^{-23}(255.99999988079071044921875)$
FIX24	4	4	signed	$-2^7(-128.0)$	$2^7-2^{-24}(127.999999940395355224609375)$
FIX25	4	4	signed	$-2^6(-64.0)$	$2^6-2^{-25}(63.9999999701976776123046875)$
FIX26	4	4	signed	$-2^5(-32.0)$	$2^5-2^{-26}(31.99999998509883880615234375)$
FIX27	4	4	signed	$-2^4(-16.0)$	$2^4-2^{-27}(15.999999992549419403076171875)$
FIX28	4	4	signed	$-2^3(-8.0)$	$2^3-2^{-28}(7.9999999962747097015380859375)$
FIX29	4	4	signed	$-2^2(-4.0)$	$2^2-2^{-29}(3.99999999813735485076904296875)$
FIX30	4	4	signed	$-2^1(-2.0)$	$2^1-2^{-30}(1.999999999068677425384521484375)$
FIX31	4	4	signed	$-2^0(-1.0)$	$2^0-2^{-31}(0.9999999995343387126922607421875)$
FIX	4	4	signed	Represents one of above ranges, depending on the number of fraction bits assumed.	

The macros defined are listed in table 5.

Table 5. List of Macros

Category	Name	Parameter Type	Return Type	Description
Multiplication	FIX<n>_mul_short	FIX<n> n=1~31	FIX<n> n=1~31	Computes multiplication of fixed-point data (If the multiplication result exceeds 32-bits, the result is not guaranteed).
	FIX<n>_mul_frac	FIX<n> n=1~31	FIX<n> n=1~31	Computes fractional part f of the fixed-point multiplication ( $0 \leq f < 1.0$ ).
Division	FIX<n>_div	FIX<n> n=1~31	FIX<n> n=1~31	Computes division of fixed-point data.
Conversion	FIX<n>_tfloat	FIX<n> n=1~31	float	Converts FIX<n> to float.
	FIX<n>_fromfloat	float	FIX<n> n=1~31	Converts float to FIX<n>.
	FIX<n>_todouble	FIX<n> n=1~31	double	Converts FIX<n> to double.
	FIX<n>_fromdouble	double	FIX<n> n=1~31	Converts double to FIX<n>.
Multiplication of generic fixed-point	FIX_mul_scale<n>	FIX	FIX	Computes multiplication of generic fixed-point data.
	FIX_mul_frac_scale<n>	FIX	FIX	Computes fractional part f of the generic fixed-point multiplication ( $0 \leq f < 1.0$ ).
	FIX_mul_sat_scale<n>	FIX	FIX	Computes multiplication of generic fixed-point data. When overflow occurs, the result is maximum or minimum value of the range.
	FIX_mul_frac_scale	FIX	FIX	Computes fractional part f of the generic fixed-point multiplication ( $0 \leq f < 1.0$ ).

If the result of operation is outside the range of the data type, its value is not guaranteed.

The functions declared are listed in table 6.

Table 6. List of Functions

Category	Name	Parameter Type	Return Type	Description
Multiplication	FIX_mul	FIX	FIX	Computes multiplication of fixed-point data.
	FIX<n>_mul_sat	FIX<n> n=1~31	FIX<n> n=1~31	Computes multiplication of fixed-point data. When overflow occurs, the result is maximum or minimum value of the range.
Sine	FIX<n>_sin	FIX<n> n=16, 24, 29	FIX<n> n=16, 24, 29	Computes sine of fixed-point data (radian)
Cosine	FIX<n>_cos	FIX<n> n=16, 24, 29	FIX<n> n=16, 24, 29	Computes cosine of fixed-point data (radian).
Arctangent	FIX<n>_atan	FIX<n> n=16, 24, 29	FIX<n> n=16, 24, 29	Computes radian value of arctangent of fixed-point data.
Square Root	FIX<n>_sqrt	FIX<n> n=16, 24, 29	FIX<n> n=16, 24, 29	Computes square root of fixed-point data
Multiplication of generic fixed-point	FIX_mul_scale	FIX	FIX	Computes multiplication of generic fixed-point data.
	FIX_mul_sat_scale	FIX	FIX	Computes multiplication of generic fixed-point data. When overflow occurs, the result is maximum or minimum value of the range.

If the result of operation is outside the range of the data type, its value is not guaranteed.

## 2.2 Description of Functions

### 2.2.1 Multiplication (macro)

[Interface]    `FIX<n> FIX<n>_mul_short(FIX<n> x, FIX<n> y)`  
                  n: 1~31

[Description]    Two 32-bit data are multiplied and shifted right by n bits. This computes the multiplication of two fixed-point data of `FIX<n>` type.

[Header]        `"fixmath.h"`

[Return Value]    Result of multiplication

[Parameters]    x:     Fixed-point data.  
                  y:     Fixed-point data

[Example]        

```
#include "fixmath.h"
fix16 x, y, ret;

ret=FIX16_mul_short(x, y);
```

[Note]         Short multiplication uses 32-bit integer arithmetic. The result is not guaranteed if the intermediate result exceeds 32 bits.

### 2.2.2 Division (macro)

[Interface]    `FIX<n> FIX<n>_div(FIX<n> x, FIX<n> y)`  
                  n: 1~31

[Description]    Computes the quotient of two fixed-point data.

[Header]        `"fixmath.h"`

[Return Value]    Result of division

[Parameters]    x:     Dividend.  
                  y:     divisor

[Example]        

```
#include "fixmath.h"
fix16 x, y, ret;

ret=FIX16_div(x, y);
```

### 2.2.3 Conversion (macro)

#### (1) Conversion from float type to fixed-point

[Interface]    `FIX<n> FIX<n>_fromfloat(float x)`  
                  `n: 1~31`

[Description]    Converts float type data to fixed-point type.

[Header]         `"fixmath.h"`

[Return Value]    Result of conversion

[Parameters]    `x:`        Source of conversion

[Example]        `#include "fixmath.h"`  
                  `float x;`  
                  `FIX16 ret;`  
  
                  `ret=FIX16_fromfloat(x);`

#### (2) Conversion from double type to fixed-point

[Interface]    `FIX<n> FIX<n>_fromdouble(double x)`  
                  `n: 1~31`

[Description]    Converts double type data to fixed-point type.

[Header]         `"fixmath.h"`

[Return Value]    Result of conversion

[Parameters]    `x:`        Source of conversion

[Example]        `#include "fixmath.h"`  
                  `double x;`  
                  `FIX16 ret;`  
  
                  `ret=FIX16_fromdouble(x);`

## (3) Conversion from fixed-point type to float

[Interface] float FIX<n>\_tofloat(FIX<n> x)  
n: 1~31

[Description] Converts fixed-point data to float.

[Header] "fixmath.h"

[Return Value] Result of conversion

[Parameters] x: Source of conversion

```
[Example] #include "fixmath.h"
          FIX16 x;
          float ret;

          ret=FIX16_tofloat(x);
```

## (3) Conversion from fixed-point type to double

[Interface] double FIX<n>\_todouble(FIX<n> x)  
n: 1~31

[Description] Converts fixed-point data to double.

[Header] "fixmath.h"

[Return Value] Result of conversion

[Parameters] x: Source of conversion

```
[Example] #include "fixmath.h"
          FIX16 x;
          double ret;

          ret=FIX16_todouble(x);
```

## 2.2.4 Multiplication

[Interface] `FIX FIX_mul(FIX x, FIX y, int n)`  
n: 1~31

[Description] Computes the multiplication of two fixed-point data of FIX type. 64-bit intermediate result is used. Supposing fraction part both of x and y is n-bit, computes the product of two fixed-point data and the values of x and y are multiplied as long data, and shifted n bits to the right.

[Header] "fixmath.h"

[Return Value] Result of multiplication

[Parameters] x: Fixed-point data.  
y: Fixed-point data  
n: bit size of Fraction Part

[Example] 

```
#include "fixmath.h"
FIX16 x, y, ret;

ret=(FIX16)FIX_mul((FIX)x, (FIX)y, 16);
```

## 2.2.5 Sine Function

[Interface] `FIX<n> FIX<n>_sin(FIX<n> x)`  
n: 16, 24, 29

[Description] Computes the sine function of FIX<n> fixed-point data (radian value).

[Header] "fixmath.h"

[Return Value] Result of sine.

[Parameters] x: Fixed-point data (radian)

[Example] 

```
#include "fixmath.h"
FIX16 x, ret;

ret=FIX16_sin(x);
```

## 2.2.6 Cosine Function

[Interface] `FIX<n> FIX<n>_cos(FIX<n> x)`  
n: 16, 24, 29

[Description] Computes the cosine function of `FIX<n>` fixed-point data (radian value).

[Header] "fixmath.h"

[Return Value] Result of cosine.

[Parameters] x: Fixed-point data (radian)

[Example] 

```
#include "fixmath.h"
FIX16 x, ret;

ret=FIX16_cos(x);
```

## 2.2.7 Arctangent Function

[Interface] `FIX<n> FIX<n>_atan(FIX<n> x)`  
n: 16, 24, 29

[Description] Computes the arctangent function of `FIX<n>` fixed-point data. The result is radian value.

[Header] "fixmath.h"

[Return Value] Result of arctangent (in radian).

[Parameters] x: Fixed-point data.

[Example] 

```
#include "fixmath.h"
FIX16 x, ret;

ret=FIX16_atan(x);
```

## 2.2.8 Square Root Function

[Interface] `FIX<n> FIX<n>_sqrt(FIX<n> x)`  
n: 16, 24, 29

[Description] Computes the square root of FIX<n> fixed-point data.

[Header] "fixmath.h"

[Return Value] Result of square root.

[Parameters] x: Fixed-point data.

```
[Example] #include "fixmath.h"
          FIX16 x, ret;

          ret=FIX16_sqrt(x);
```

## 2.2.9 Multiplication (fraction part) (macro)

[Interface] `FIX<n> FIX<n>_mul_frac(FIX<n> x, FIX<n> y)`  
n: 1~31

[Description] Computes the fraction part of the product. The result will always be positive ( $0 \leq \text{result} < 1.0$ ).

[Header] "fixmath.h"

[Return Value] Fractional part of the product.

[Parameters] x: Fixed-point data.  
y: Fixed-point data.

```
[Example] #include "fixmath.h"
          FIX16 x, y, ret;

          ret=FIX16_mul_frac(x, y);
```

### 2.2.10 Multiplication (saturated) (macro)

[Interface] `FIX<n> FIX<n>_mul_sat(FIX<n> x, FIX<n> y)`  
 n: 1~31

[Description] Computes the product of two fixed-point data. When the result overflows, the return value will be the maximum or minimum value, according to the sign of the result.

[Header] "fixmath.h"

[Return Value] Product of fixed point data.

[Parameters] x: Fixed-point data.  
 y: Fixed-point data.

[Example] 

```
#include "fixmath.h"
FIX16 x, y, ret;

ret=FIX16_mul_sat(x, y);
```

### 2.2.11 Multiplication (FIX-type) (macro)

[Interface] `FIX FIX_mul_scale<n>(FIX x, FIX y)`  
 n: 1~31

[Description] Computes the product of two generic fixed-point data. The values of x and y are multiplied as long data, and shifted n bits to the right.

[Header] "fixmath.h"

[Return Value] Product of fixed point data.

[Parameters] x: Fixed-point data.  
 y: Fixed-point data.

[Example] 

```
#include "fixmath.h"
FIX x, y, ret;

ret=FIX_mul_scale16(x, y);
```

[Note] If the result cannot be represented in 32-bit, its value is not guaranteed. When multiplying `FIX<n1>` and `FIX<n2>` to get `FIX<n3>` type, shift count is  $n_1+n_2-n_3$ , and you can specify this value (1~31) as `<n>`.

## 2.2.12 Multiplication (FIX-type)

[Interface] `FIX FIX_mul_scale(FIX x, FIX y, int n)`  
n: 1~31

[Description] Computes the product of two generic fixed-point data. The values of x and y are multiplied as long data, and shifted n bits to the right.

[Header] "fixmath.h"

[Return Value] Product of fixed point data.

[Parameters] x: Fixed-point data.  
y: Fixed-point data.

[Example] 

```
#include "fixmath.h"
FIX x, y, ret;
int n=16;

ret=FIX_mul_scale(x, y, n);
```

[Note] If the result cannot be represented in 32-bit, its value is not guaranteed. When multiplying `FIX<n1>` and `FIX<n2>` to get `FIX<n3>` type, shift count is  $n_1+n_2-n_3$ , and you can specify this value (1~31) as `<n>`.

## 2.2.13 Multiplication (fraction part, FIX-type) (macro)

[Interface] `FIX FIX_mul_frac_scale<n>(FIX x, FIX y)`  
n: 1~31

[Description] Computes the fraction part of the product of two generic fixed-point data. The values of x and y are multiplied as long data, shifted n bits to the right, and lower n-bits are returned.

[Header] "fixmath.h"

[Return Value] Product of fixed point data.

[Parameters] x: Fixed-point data.  
y: Fixed-point data.

[Example] 

```
#include "fixmath.h"
FIX x, y, ret;

ret=FIX_mul_frac_scale16(x, y);
```

[Note] This function can be used to compute the fractional part of `FIX<n>` type by multiplying `FIX<n+d>` type and `FIX<n-d>` type.

### 2.2.14 Multiplication (fraction part, FIX-type) (macro)

[Interface]    `FIX FIX_mul_frac_scale(FIX x, FIX y, int n)`  
                   n: 1~31

[Description]    Computes the fraction part of the product of two generic fixed-point data. The values of x and y are multiplied as long data, shifted n bits to the right, and lower n-bits are returned.

[Header]        "fixmath.h"

[Return Value]    Product of fixed point data.

[Parameters]    x:        Fixed-point data.  
                   y:        Fixed-point data.

[Example]        `#include "fixmath.h"`  
                   `FIX x, y, ret;`  
                   `int n=16;`  
  
                   `ret=FIX_mul_frac_scale(x, y, n);`

[Note]         This function can be used to compute the fractional part of FIX<n> type by multiplying FIX<n+d> type and FIX<n-d> type.

### 2.2.15 Multiplication (saturated, FIX-type) (macro)

[Interface]    `FIX FIX_mul_sat_scale<n>(FIX x, FIX y)`  
                   n: 1~31

[Description]    Computes the product of two generic fixed-point data. The values of x and y are multiplied as long data, and shifted n bits to the right. When the result overflows, the return value will be the maximum or minimum value, according to the sign of the result.

[Header]        "fixmath.h"

[Return Value]    Product of fixed point data.

[Parameters]    x:        Fixed-point data.  
                   y:        Fixed-point data.

[Example]        `#include "fixmath.h"`  
                   `FIX x, y, ret;`  
  
                   `ret=FIX_mul_sat_scale16(x, y);`

[Note]         If the result cannot be represented in 32-bit, its value is not guaranteed. When multiplying FIX<n<sub>1</sub>> and FIX<n<sub>2</sub>> to get FIX<n<sub>3</sub>> type, shift count is n<sub>1</sub>+n<sub>2</sub>-n<sub>3</sub>, and you can specify this value (1~31) as <n>.

## 2.2.12 Multiplication (saturated, FIX-type)

[Interface] `FIX FIX_mul_sat_scale(FIX x, FIX y, int n)`  
n: 1~31

[Description] Computes the product of two generic fixed-point data. The values of x and y are multiplied as long data, and shifted n bits to the right. When the result overflows, the return value will be the maximum or minimum value, according to the sign of the result.

[Header] "fixmath.h"

[Return Value] Product of fixed point data.

[Parameters] x: Fixed-point data.  
y: Fixed-point data.

[Example] 

```
#include "fixmath.h"
FIX x, y, ret;
int n=16;

ret=FIX_mul_sat_scale(x, y, n);
```

[Note] If the result cannot be represented in 32-bit, its value is not guaranteed. When multiplying FIX<n<sub>1</sub>> and FIX<n<sub>2</sub>> to get FIX<n<sub>3</sub>> type, shift count is n<sub>1</sub>+n<sub>2</sub>-n<sub>3</sub>, and you can specify this value (1~31) as <n>.

### 3. Performance and Precision

#### 3.1 Evaluation Condition

Compiler: Renesas RX Standard Compiler V.1.0.0.0

Configuration: See table 7 for library configuration.

Table 7. Library Configuration

Condition	Compilation Options	
	cpu	endian
1	RX600	big
2	RX600	little

#### 3.2 Execution Cycles

The execution cycles of fixed-point mathematical functions are shown in table 8.

Table 8. Execution Cycles of Fixed-point Mathematical Functions

CPU Configuration (see table 7)	RX600		
	1	2	
Sine	FIX16_sin	56	56
	FIX24_sin	56	56
	FIX29_sin	55	55
Cosine	FIX16_cos	52	52
	FIX24_cos	52	52
	FIX29_cos	53	53
Arctangent	FIX16_atan	118	118
	FIX24_atan	118	118
	FIX29_atan	112	112
Square	FIX16_sqrt	82	82
Root	FIX24_sqrt	82	82
	FIX29_sqrt	81	81

[Note] The numbers are in cycles. Measurement may include some error.

#### 3.3 Precision

The maximum error of these mathematics functions is  $\pm 2$  in the last place except FIX29\_sqrt. The precision of FIX29\_sqrt is  $\pm 3$  in the last place.

## Website and Support

Renesas Technology Website

<http://www.renesas.com/>

Inquiries

<http://www.renesas.com/inquiry>

[csc@renesas.com](mailto:csc@renesas.com)

All trademarks and registered trademarks are the property of their respective owners.

## Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Mar.03.11	—	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 manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

### 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 one with a different type number, confirm that the change will not lead to problems.

— The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

## Notice

- 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.
- 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.
- You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- 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.
- This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
- 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.



### 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: +49-211-6503-0, 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.

7F, No. 363 Fu Shing North Road Taipei, Taiwan, R.O.C.  
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