

## KF32 系列

### Iqmath 定点浮点库使用说明 V1.0

2020 年 7 月

## 目录

IQMATH 定点浮点库使用说明 V1.0 .....	1
目录 .....	2
1. 概述 .....	4
2. 库组成与使用配置 .....	4
2.1 定点浮点格式说明与数据类型 .....	4
2.2 定点浮点格式表达数据范围与精度 .....	5
2.3 C 语言开发调用 IQMATH 库方法 .....	6
2.4 C++语言开发调用 IQMATH 库方法 .....	7
3. 库方法介绍 .....	7
3.1 字符定点转换浮点定点转换实现 .....	9
3.1.1 浮点到定点的转换函数 IQ_IQN .....	9
3.1.2 定点到浮点的转换函数 IQtoF_IQNtoF .....	10
3.1.3 定点字符串解析到浮点 atoIQ_atoIQN .....	11
3.1.4 转换定点到字符串输出 IQtoa_IQNtoa .....	12
3.1.5 获取 IQN 格式值的整数值 IQint_IQNint .....	13
3.1.6 获取 IQN 格式值的 IQ 格式小数值 IQfrac_IQNfrac .....	14
3.1.7 全局格式转换指定格式定点浮点 IQtoIQN .....	14
3.1.8 指定格式转换全局格式定点浮点 IQNtoIQ .....	15
3.1.9 全局格式转 16 位格式定点浮点 IQtoQN .....	15
3.1.10 16 位格式转全局格式定点浮点 QntoIQ .....	16
3.2 算术运算乘除 .....	17
3.2.1 同格式定点浮点乘 IQmpy_IQNmpy .....	17
3.2.2 类四舍五入定点浮点乘 IQrmpy_IQNrmpy .....	17
3.2.3 类四舍五入饱和定点浮点乘 IQrsmpy_IQNrsmpy .....	18
3.2.4 定点浮点乘以整数的乘 IQmpyI32_IQNmpyI32 .....	19
3.2.5 定点浮点乘整数的定点整数 IQmpyI32int_IQNmpyI32int .....	20
3.2.6 定点浮点乘整数的定点小数 IQmpyI32frac_IQNmpyI32frac .....	21
3.2.7 不同格式定点浮点乘 IQNmpyIQX .....	21
3.2.8 同格式定点浮点除法 IQdiv_IQNdiv .....	22
3.2.9 定点浮点乘以 2 的幂优化 IQmpy2, 4, 8..64 .....	26
3.2.10 定点浮点除以 2 的幂优化 IQdiv2, 4, 8..64 .....	26
3.3 三角函数 SIN COS TAN .....	26
3.3.1 定点浮点表达的反向正弦值 IQasin_IQNasin .....	26
3.3.2 定点浮点表达的正弦值 IQsin_IQNsin .....	27

3.3.3	定点浮点表达的正弦值_IQsinPU _IQNsinPU .....	30
3.3.4	定点浮点表达的反向余弦值_IQacos _IQNacos .....	32
3.3.5	定点浮点表达的余弦值_IQcos _IQNcos .....	32
3.3.6	定点浮点表达的余弦值_IQcosPU _IQNcosPU .....	35
3.3.7	定点浮点表达的反切值_IQatan2 _IQNatan2 .....	37
3.3.8	定点浮点表达的反切值_IQatan2PU _IQNatan2PU .....	39
3.3.9	定点浮点表达的反切值_IQatan _IQNatan .....	41
3.4	数学实现 EXP LOG SQRT MAG .....	41
3.4.1	计算 e 的定点浮点指数 IQexp IQNexp .....	41
3.4.2	计算 2 的定点浮点指数 IQexp2 IQNexp2 .....	42
3.4.3	计算底数 e 的定点浮点真数的对数_IQlog _IQNlog .....	43
3.4.4	定点浮点表达的平方根_IQsqrt _IQNsqrt .....	43
3.4.5	定点浮点表达的平方根倒数_IQisqrt _IQNisqrt .....	46
3.4.6	定点浮点平方和的平方根 _IQmag _IQNmag .....	49
3.5	其他实现 .....	50
3.5.1	定点浮点数绝对值_IQabs _IQNabs .....	50
3.5.2	饱和处理定点浮点值到给定的正与负值限定 _IQsat .....	50
4.	历史记录 .....	51

## 1. 概述

在精细数据控制逻辑中往往使用浮点数，在无浮点运算单元的内核处理器上，浮点的运算采用软件算法模拟，该代码量大，运算效率低。定点浮点算法选取有效小数点位数的格式，内部计算往往可做整数处理的使用内核的 MULS、DIVS、ADD、SUB、LSR、LSL、ASR 指令使用处理，使用定点浮点可以加速计算效率。

## 2. 库组成与使用配置

声明头文件：IQmathLib.h (C)、IQmathCPP.h (C++)

运算库文件：libiqmath.a

以上文件已集成在 ChipON IDE KF32 开发工具中，在使用中仅需要包含对应语言的头文件并使用对应的浮点定点库方法即可。

如果代码兼容但使用更高精度的浮点时，仅需要更改 IQmathLib.h 文件中

`#define IQ_MATH 0` 为 `#define IQ_MATH 1`

如果代码使用全局格式的精简写法，默认使用 IQ24，如果需要更改 IQmathLib.h 文件中为对应的格式，如修改为 25：

`#define GLOBAL_Q 24` 为 `#define GLOBAL_Q 25`

### 2.1 定点浮点格式说明与数据类型

通过设定小数点在 32 位数中的不同位置，就可以表示不同大小和不同精度的小数了。数的定标有 Q 表示法和 S 表示法两种这里，如 Q15、S17.15。文件介绍与库名均采用 Q 表示法，Q15/Q30 小数格式示例如下：

Q15 小数格式：

数据位	.....	bit15	Bit14	.....	bit2	bit1	bit0
权值	.....	$2^0$	$2^{-1}$	.....	$2^{-13}$	$2^{-14}$	$2^{-15}$

↑  
小数点

Q30 小数格式：

数据位	bit31	bit30	bit29	.....	bit2	bit1	bit0
权值	$2^0$	$2^{-1}$	$2^{-2}$	.....	$2^{-29}$	$2^{-30}$	$2^{-31}$

↑  
小数点

定义不同小数格式的数据类型名如下，即 4 字节的 32 位数据：

```
typedef long _iq; /* Fixed point data type: GLOBAL_Q format */
typedef long _iq30; /* Fixed point data type: Q30 format */
typedef long _iq29; /* Fixed point data type: Q29 format */
typedef long _iq28; /* Fixed point data type: Q28 format */
typedef long _iq27; /* Fixed point data type: Q27 format */
typedef long _iq26; /* Fixed point data type: Q26 format */
typedef long _iq25; /* Fixed point data type: Q25 format */
typedef long _iq24; /* Fixed point data type: Q24 format */
typedef long _iq23; /* Fixed point data type: Q23 format */
typedef long _iq22; /* Fixed point data type: Q22 format */
typedef long _iq21; /* Fixed point data type: Q21 format */
typedef long _iq20; /* Fixed point data type: Q20 format */
typedef long _iq19; /* Fixed point data type: Q19 format */
typedef long _iq18; /* Fixed point data type: Q18 format */
typedef long _iq17; /* Fixed point data type: Q17 format */
typedef long _iq16; /* Fixed point data type: Q16 format */
typedef long _iq15; /* Fixed point data type: Q15 format */
typedef long _iq14; /* Fixed point data type: Q14 format */
typedef long _iq13; /* Fixed point data type: Q13 format */
typedef long _iq12; /* Fixed point data type: Q12 format */
typedef long _iq11; /* Fixed point data type: Q11 format */
typedef long _iq10; /* Fixed point data type: Q10 format */
typedef long _iq9; /* Fixed point data type: Q9 format */
typedef long _iq8; /* Fixed point data type: Q8 format */
typedef long _iq7; /* Fixed point data type: Q7 format */
typedef long _iq6; /* Fixed point data type: Q6 format */
typedef long _iq5; /* Fixed point data type: Q5 format */
typedef long _iq4; /* Fixed point data type: Q4 format */
typedef long _iq3; /* Fixed point data type: Q3 format */
typedef long _iq2; /* Fixed point data type: Q2 format */
typedef long _iq1; /* Fixed point data type: Q1 format */
```

## 2.2 定点浮点格式表达数据范围与精度

数据类型	范围		精度分辨率
	最小	最大	
_iq30	-2	1.999 999 999	0.000 000 001
_iq29	-4	3.999 999 998	0.000 000 002
_iq28	-8	7.999 999 996	0.000 000 004
_iq27	-16	15.999 999 993	0.000 000 007
_iq26	-32	31.999 999 985	0.000 000 015

_iq25	-64	63.999 999 970	0.000 000 030
_iq24	-128	127.999 999 940	0.000 000 060
_iq23	-256	255.999 999 981	0.000 000 119
_iq22	-512	511.999 999 762	0.000 000 238
_iq21	-1024	1023.999 999 523	0.000 000 477
_iq20	-2048	2047.999 999 046	0.000 000 954
_iq19	-4096	4095.999 998 093	0.000 001 907
_iq18	-8192	8191.999 996 185	0.000 003 815
_iq17	-16384	16383.999 992 371	0.000 007 629
_iq16	-32768	32767.999 984 741	0.000 015 259
_iq15	-65536	65535.999 969 482	0.000 030 518
_iq14	-131072	131071.999 938 965	0.000 061 035
_iq13	-262144	262143.999 877 930	0.000 122 070
_iq12	-524288	524287.999 755 859	0.000 244 141
_iq11	-1048576	1048575.999 511 719	0.000 488 281
_iq10	-2097152	2097151.999 023 437	0.000 976 563
_iq9	-4194304	4194303.998 046 875	0.001 953 125
_iq8	-8388608	8388607.996 093 750	0.003 906 250
_iq7	-16777216	16777215.992 187 500	0.007 812 500
_iq6	-33554432	33554431.984 375 000	0.015 625 000
_iq5	-67108864	67108863.968 750 000	0.031 250 000
_iq4	-134217728	134217727.937 500 000	0.062 500 000
_iq3	-268435456	268435455.875 000 000	0.125 000 000
_iq2	-536870912	536870911.750 000 000	0.250 000 000
_iq1	-1073741824	1 073741823.500 000 000	0.500 000 000

## 2.3 C 语言开发调用 IQMath 库方法

□ 包含 IQmathLib.h 头文件.

□ 使用 \_iq 和 \_iqN 数据类型的函数原型方法.

举例： 调用 *IQ25sin*

```
#include "IQmathLib.h"

#define PI 3.14159

_iq input, sin_out;
void main(void )
{
    /* 0.25 x PI radians represented in Q29 format */

    input=_IQ29(0.25*PI);

    sin_out =_IQ29sin(input);
}
```

## 2.4 C++语言开发调用 IQMath 库方法

在 C++ 项目中 `_iq` 类型变化为 `iq` 类. 它允许操作重载, 如乘法和除法. 使用方法如下:

- 如下格式包含头文件 `IQmathLib.h` 和 `IQmathCPP.h`

```
extern "C" {  
    #include "IQmathLib.h"  
}  
#include "IQmathCPP.h"
```

- 调用 C++ 代码函数的 `iq` 和 `iqN` 类型方法. C++ 函数调用 C 的库函数

C Code	C++ Code	Note
<code>_iq, _iqN</code>	<code>iq, iqN</code>	IQ Data Types
<code>_IQ(A), _IQN(A)</code>	<code>IQ(A), IQN(A)</code>	Convert float to IQ
<code>_IQdiv(A,B)</code>	<code>A/B</code>	Division

举例: 调用 ***IQ25sin***

```
extern "C" {  
    #include "IQmathLib.h"  
}  
  
#include "IQmathCPP.h"  
  
#define PI 3.14159  
  
iq input, sin_out;  
void main(void )  
{  
  
    /* 0.25 x PI radians represented in Q29 format */  
  
    input = IQ29(0.25*PI);  
  
    sin_out = IQ29sin(input);  
}
```

## 3. 库方法介绍

函数命名说明:

IQ 表明为全局 IQ 格式

IQN 表明指定 (1 ≤ N ≤ 30) 小数点位数

**C 语言声明功能函数摘要：**

类型	函数名	功能描述	格式
转换	_iq_IQ(float F)	浮点转定点	Q=GLOBAL_Q
转换	_iqN_IQN(float F)		Q=1:30
转换	float_IQtoF(_iq A)	定点转浮点	Q=GLOBAL_Q
转换	float_IQNtoF(_iqN A)		Q=1:30
转换	double_IQtoD(_iq A)	定点转双精度浮点	Q=GLOBAL_Q
转换	double_IQNtoD(_iqN A)		Q=1:30
转换	_iq_atoIQ(char *S)	浮点字符串转定点	Q=GLOBAL_Q
转换	_iqN_atoIQN(char *S)		Q=1:30
转换	int_IQtoa(char *S, const *format, long x)	定点转字符串	Q=GLOBAL_Q
转换	int_IQNtoa(char *S, const *format, long x)		Q=1:30
转换	long_IQint(_iq A)	定点转整数	Q=GLOBAL_Q
转换	long_IQNint(_iqN A)		Q=1:30
转换	_iq_IQfrac(_iq A)	返回定点的定点小数部分	Q=GLOBAL_Q
转换	_iqN_IQNfrac(_iqN A)		Q=1:30
转换	_iqN_IQtoIQN(_iq A)	全局格式与指定格式的定点浮点转换	Q=GLOBAL_Q
转换	_iq_IQNtoIQ(_iqN A)		Q=GLOBAL_Q
转换	int_IQtoQN(_iq A)	32 位定点浮点与 16 位定点浮点的转换	Q=GLOBAL_Q
转换	_iq_QNtoIQ(int A)		Q=GLOBAL_Q
算术	_iq_IQmpy(_iq A, _iq B)	IQ 定点浮点乘法	Q=GLOBAL_Q
算术	_iqN_IQNmpy(_iqN A, _iqN B)		Q=1:30
算术	_IQXXmpyIQX(A, IQA, B, IQB)	不同格式定点浮点乘法	Q=1:30
算术	_iq_IQrmpy(_iq A, _iq B)	带舍入进位的定点浮点乘法	Q=GLOBAL_Q
算术	_iqN_IQNrmpy(_iqN A, _iqN B)		Q=1:30
算术	_iq_IQrsmpy(_iq A, _iq B)	带舍入进位和饱和处理的定点浮点乘法	Q=GLOBAL_Q
算术	_iqN_IQNrsmpy(_iqN A, _iqN B)		Q=1:30
算术	_iq_IQmpyI32(_iq A, long B)	定点浮点与整数的乘法	Q=GLOBAL_Q
算术	_iqN_IQNmpyI32(_iqN A, long B)		Q=1:30
算术	long_IQmpyI32int(_iq A, long B)	定点浮点与整数的乘法，返回整数值	Q=GLOBAL_Q
算术	long_IQNmpyI32int(_iqN A, long B)		Q=1:30
算术	long_IQmpyI32frac(_iq A, long B)	定点浮点与整数乘法，返回定点格式小数值	Q=GLOBAL_Q
算术	long_IQNmpyI32frac(_iqN A, long B)		Q=1:30
算术	_iq_IQmpyIQX(_iqN1 A, N1, _iqN2 B, N2)	不同格式定点浮点乘法	Q=GLOBAL_Q
算术	_iqN_IQmpyIQX(_iqN1 A, N1, _iqN2 B, N2)		Q=1:30
算术	_iq_IQdiv(_iq A, _iq B)	定点浮点除法	Q=GLOBAL_Q
算术	_iqN_IQNdiv(_iqN A, _iqN B)		Q=1:30
算术	_iq_IQmpy2(_iq A)	乘以 2 通过左移 1 代替	Q=1:30
算术	_iq_IQmpy4(_iq A)	乘以 4 通过左移 2 代替	Q=1:30
算术	_iq_IQmpy8(_iq A)	乘以 8 通过左移 3 代替	Q=1:30
算术	_iq_IQmpy16(_iq A)	乘以 16 通过左移 4 代替	Q=1:30
算术	_iq_Iqmpy32(_iq A)	乘以 32 通过左移 5 代替	Q=1:30
算术	_iq_Iqmpy64(_iq A)	乘以 2 通过左移 6 代替	Q=1:30
算术	_iq_IQdiv2(_iq A)	除以 2 通过右移 1 代替	Q=1:30
算术	_iq_Iqdiv4(_iq A)	除以 4 通过右移 2 代替	Q=1:30
算术	_iq_Iqdiv8(_iq A)	除以 8 通过右移 3 代替	Q=1:30
算术	_iq_Iqdiv16(_iq A)	除以 16 通过右移 4 代替	Q=1:30
算术	_iq_Iqdiv32(_iq A)	除以 32 通过右移 5 代替	Q=1:30
算术	_iq_Iqdiv64(_iq A)	除以 64 通过右移 6 代替	Q=1:30



三角	_iq_IQasin( _iq A)	高精度 ASIN (弧度输出)	Q=GLOBAL_Q
三角	_iqN_IQNasin( _iqN A)		Q=1:30
三角	_iq_IQsin( _iq A)	高精度 SIN (弧度输入)	Q=GLOBAL_Q
三角	_iqN_IQNsin( _iqN A)		Q=1:30
三角	_iq_IQsinPU( _iq A)	高精度 SIN (弧度占比输入)	Q=GLOBAL_Q
三角	_iqN_IQNsinPU( _iqN A)		Q=1:30
三角	_iq_IQacos( _iq A)	高精度 ACOS (弧度输出)	Q=GLOBAL_Q
三角	_iqN_IQNacos( _iqN A)		Q=1:30
三角	_iq_IQcos( _iq A)	高精度 COS (弧度输入)	Q=GLOBAL_Q
三角	_iqN_IQNcos( _iqN A)		Q=1:30
三角	_iq_IQcosPU( _iq A)	高精度 COS (弧度占比输入)	Q=GLOBAL_Q
三角	_iqN_IQNcosPU( _iqN A)		Q=1:30
三角	_iq_IQatan2( _iq A, _iq B)	4 相限反切 (弧度输出)	Q=GLOBAL_Q
三角	_iqN_IQNatan2( _iqN A, _iqN B)		Q=1:30
三角	_iq_IQatan2PU( _iq A, _iq B)	4 相限反切 (弧度占比输出)	Q=GLOBAL_Q
三角	_iqN_IQNatanPU( _iqN A, _iqN B)		Q=1:30
三角	_iq_IQatan( _iq A, _iq B)	反切, $-\pi/2 \sim \pi/2$	Q=GLOBAL_Q
三角	_iqN_IQNatan( _iqN A, _iqN B)		Q=1:30
数学	_iq_IQexp( _iq A)	计算 e 的 IQ 值的指数	Q=GLOBAL_Q
数学	_iqN_IQNexp( _iqN A)		Q=1:30
数学	_iqN_IQexp2( _iqN A)	计算 2 的 IQ 值的指数	Q=GLOBAL_Q
数学	_iqN_IQNexp2( _iqN A)		Q=1:30
数学	_iq_IQsqrt( _iq A)	计算平方根	Q=GLOBAL_Q
数学	_iqN_IQNsqrt( _iqN A)		Q=1:30
数学	_iq_IQisqrt( _iq A)	计算平方根倒数	Q=GLOBAL_Q
数学	_iqN_IQNisqrt( _iqN A)		Q=1:30
数学	_iq_IQmag( _iq A, _iq B)	计算平方和的平方根	Q=GLOBAL_Q
数学	_iqN_IQNmag( _iqN A, _iqN B)		Q=1:30
其他	_iq_IQsat( _iq A, long P, long N)	饱和和定点浮点到到最小和最大值	Q=1:30
其他	_iq_IQabs( _iq A)	返回定点浮点值的绝对定点浮点值	Q=GLOBAL_Q
其他	_iqN_IQNabs( _iqN A)		Q=1:30

### 3.1 字符定点转换浮点定点转换实现

#### 3.1.1 浮点到定点的转换函数\_IQ\_IQN

**描述：**宏定义转换浮点常数或变量到对应的定点浮点数。

**声明全局 IQ 宏 (IQ 格式= GLOBAL\_Q)**

C        \_iq        \_IQ(float F)

C++     iq        IQ(float F)

**Q 格式指定 IQ 宏 (IQ 格式= IQ1 ~IQ30)**

C        \_iqN     \_IQN(float F)

C++     iq        IQN(float F)

**输入：**浮点变量或常量

输出：全局 IQ 宏 (IQ 格式= GLOBAL\_Q)

全局格式的等效定点浮点数值

Q 格式宏 (IQ 格式= IQ1 ~IQ30)

IQN 格式的等效定点浮点数值

使用：转换浮点变量或常量到等效的 IQ 格式数值。

样例 1：使用定点浮点赋值

```
// Floating-point equation
Y = M*1.26 + 2.345

// IQmath equation using the GLOBAL_Q value
Y = _IQmpy(M, _IQ(1.26)) + _IQ(2.345)

// IQmath equation specifying the Q value
Y = _IQ23mpy(M, _IQ23(1.26)) + _IQ23(2.345)
```

样例 2：转换浮点变量到 IQ 数据类型。

```
#include "IQmathLib.h"

float x=3.343;

_iq y1;
_iq23 y2

y1=_IQ(x)    // Uses the GLOBAL_Q value
y2=_IQ23(x)  // Specifies the Q value
```

样例 3：初始化全局变量或表

```
#include "IQmathLib.h"

// IQmath using GLOBAL_Q
_iq Array[4] ={_IQ(1.0), _IQ(2.5) _IQ(-0.2345), _IQ(0.0) }

// IQmath specifying the Q value
_iq23 Array[4] ={_IQ23(1.0), _IQ23(2.5) _IQ23(-0.2345), _IQ23(0.0) }
```

### 3.1.2 定点到浮点的转换函数\_IQtoF \_IQNtoF

描述 转换 IQ 数据到 IEEE754 格式的浮点数。

声明 全局IQ 函数(IQ格式= GLOBAL\_Q)

C float \_IQtoF(\_iq A)

C++ float IQtoF(const iq &A)

Q 格式函数 (IQ 格式= IQ1 ~IQ30)

C float \_IQNtoF(\_iqN A)

C++ float IQNtoF(const iqN &A)

输入 全局IQ 函数(IQ 格式= GLOBAL\_Q)

全局格式的定点浮点值。

Q格式函数 (IQ 格式= IQ1 ~IQ30)

指定格式的定点浮点值。

输出 等效浮点数值。

**使用** 需要将数据转换回浮点进行操作或显示时调用。

**样例:**

转换定点浮点数组值到浮点数值中。

```
_iq DataIQ[N];
float DataF[N];
for(i = 0; i < N; i++)
{
    DataF[i] = _IQtoF(DataIQ[i]);
}
```

### 3.1.3 定点字符串解析到浮点 `_atoIQ` `_atoIQN`

**描述** 转换字符串到定点浮点值。

**声明** 全局IQ 函数(IQ 格式= GLOBAL\_Q)

C float `_atoIQ(char *S)`

C++ float `atoIQ(char *S)`

**Q 格式函数 (IQ 格式= IQ1 ~IQ30)**

C float `_atoIQN(char *S)`

C++ float `atoIQN(char *S)`

**输入** 按顺序识别可选的符号, 包含10进制下任意数字的字符串. 有效输入字符串示例:

“12.23456”, “-12.23456”, “0.2345”, “0.0”, “0”, “127”, “-89”

**输出** 遇到首个异常字符下, 结果返回0. 如果输入的内容超出对应格式的定点浮点的最大或最小范围, 返回值将限定到其格式下返回。

**全局IQ 函数(IQ 格式= GLOBAL\_Q)**

GLOBAL\_Q 格式等效定点浮点值

**Q 格式函数 (IQ 格式= IQ1 ~IQ30)**

IQN 格式等效定点浮点值

**使用** 程序需要的解析外部输入或ASCII 字符串。

**样例:**

下面代码处理用户输入的X的值:

```
char buffer[N];
_iq X;
printf("Enter value X = ");
gets(buffer);
X = _atoIQ(buffer); // IQ value (GLOBAL_Q)
```

### 3.1.4 转换定点到字符串输出 `_IQtoa` `_IQNtoa`

**描述** 转换 IQ 数值到字符串.

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

```
C int _IQtoa(char *string, const char *format, _iq x)
```

```
C++ int IQtoa(char *string, const char *format, const iq &x)
```

**Q 格式函数 (IQ 格式= IQ1 -IQ30)**

```
C int _IQNtoa(char *string, const char *format, _iqN x)
```

```
C++ int IQNtoa(char *string, const char *format, const iqN &x)
```

**参数 string:** 输出字符串

**参数 format:** 转换格式. 必须样例为 "%xx.yyf" 并且 xx 和 yy 最多 2 个数字的长度. 举例: "%10.12f", "%2.4f", "%11.6f". 最大支持 (xx) 是 11 (包含负符号), 即支持全整数范围 I2Q30 到 I31Q1.

**参数 x:** 全局 IQ 函数: GLOBAL\_Q 格式定点浮点数值

Q 格式函数: IQN 格式定点浮点数值

**输出** 输出字符串在指针 "string" 参数中.

如果使用 MATH\_TYPE 设定为 IQ\_MATH, 返回整数值确定错误, 错误码如下:

0 = 正确

1 = 宽度太小不能保存整数长度内容

2 = 错误的指定格式

如果使用 MATH\_TYPE 设定为 FLOAT\_MATH, 函数 `sprintf()` 被调用并返回输出字符的个数.

**使用**

整数部分的前面 0 不打印输出. 因此, 格式限定整数的最大宽度. 区域可能偏小.

输出字符串使用 '\0' 字符标记结束.

"format" 中整数宽度包含负数的符号, 如 -12.3456 是 "%3.5f".

"format" 中小数宽度包含小数点. 如 -12.3456 是 "%3.5f"

"string" 必须足够大来保存输出 (包括负数符号和终止字符). 代码不检测溢出.

如果 "string" 太小, 将发生内存溢出问题.

非 0 的返回意味着输出字符串存在问题 "string"

**样例:**

```
char buffer[30];
_iq x1 = _IQ(1.125);
_iq1 x2 = _IQ1(-6789546.3);
_iq14 x3 = _IQ14(-432.6778);
_iq30 x4 = _IQ30(1.127860L);
int error;
// Global_Q
```

```

error = _IQtoa(buffer, "%10.10f", x1);
// IQ1
error = _IQ1toa(buffer, "%8.2f", x2);
// IQ14
error = _IQ14toa(buffer, "%6.6f", x3);
// IQ30
error = _IQ30toa(buffer, "%11.12f", x4);

```

### 3.1.5 获取 IQN 格式值的整数值\_IQint \_IQNint

**描述** 返回定点浮点数的整数值.

**声明** 全局IQ函数(IQ 格式= GLOBAL\_Q)

```

C      long    _IQint(_iq A)
C++    long    IQint(const iq &A)

```

**Q 格式函数(IQ 格式= IQ1 ~IQ30)**

```

C      long    _IQNint(_iqN A)
C++    long    IQNint(const iqN &A)

```

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式定点浮点数值.

**Q 格式函数 (IQ 格式= IQ1 ~IQ30)**

IQN 格式定点浮点数值.

**输出** IQ 值的整数值

**样例 1:**

获取定点浮点数的整数和定点小数部分

```

_iq Y0 = 2.3456;
_iq Y1 = -2.3456
long Y0int, Y1int;
_iq Y0frac, Y1frac;
Y0int = _IQint(Y0); // Y0int = 2
Y1int = _IQint(Y1); // Y1int = -2
Y0frac = _IQfrac(Y0); // Y0frac = 0.3456
Y1frac = _IQfrac(Y1); // Y1frac = -0.3456

```

**样例 2:**

从整数和定点小数构建定点浮点数

```

_iq Y;
long Yint;
_iq Yfrac;
Y = _IQmpyI32(_IQ(1.0), Yint) + Yfrac;

```

### 3.1.6 获取 IQN 格式值的 IQ 格式小数值 IQfrac IQNfrac

**描述:** 返回IQ数据的小数值结果.

**声明** 全局IQ函数 (IQ 格式= GLOBAL\_Q)

```
C      _iq      _IQfrac(_iq A)
C++    iq      IQfrac(const iq &A)
```

**Q 格式函数 (IQ 格式= N = 1 ~30)**

```
C      _iqN      _IQNfrac(_iqN A)
C++    iqN      IQNfrac(const iqN &A)
```

**输入:** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式定点浮点数.

**QN 格式函数(IQ 格式= IQ1 ~IQ30)**

IQN 格式定点浮点数.

**输出:** 定点浮点数的小数部分值

**使用举例 1:**

解析 IQ 数据的整数部分和小数部分

```
_iq Y0 = _IQ(2.3456);
_iq Y1 = _IQ(-2.3456);
long Y0int, Y1int;
_iq Y0frac, Y1frac;

Y0int = _IQint(Y0); // Y0int = 2
Y1int = _IQint(Y1); // Y1int = -2

Y0frac = _IQfrac(Y0); // Y0frac = 0.3456
Y1frac = _IQfrac(Y1); // Y1frac = -0.3456
```

**使用举例 2:**

从小数与整数构建 IQ 数据.

```
_iq Y;
long Yint;
_iq Yfrac;
Y = _IQmpyI32(_IQ(1.0), Yint) + Yfrac;
```

### 3.1.7 全局格式转换指定格式定点浮点 IQtoIQN

**描述** 定义宏转换IQ 数值 GLOBAL\_Q 格式到 IQN格式.

**声明**

```
C      _iqN      _IQtoIQN(_iq A)
C++    iqN      IQtoIQN(const iq &A)
```

**输入** GLOBAL\_Q 格式定点浮点值

**输出** IQN 格式的等效定点浮点值

**使用** 当计算 IQ 格式值可能溢出他的范围，因此需要不同的 IQ 格式值来中间操作。

**举例：**

下面例子计算复杂数据 (X+jY) 到 Q26 格式量级：

$$Z = \sqrt{X^2 + Y^2}$$

符号 Z, X, Y 使用 GLOBAL\_Q = 26 给定，但等式自身可能产生溢出。

为了预防发生，中间计算使用 Q = 23，并值结果转换回设定全局格式：

```
#include "IQmathLib.h"
_iq Z, Y, X; // GLOBAL_Q = 26
_iq23 temp;
temp = _IQ23sqrt( _IQ23mpy(_IQtoIQ23(X), _IQtoIQ23(X)) +
    _IQ23mpy(_IQtoIQ23(Y), _IQtoIQ23(Y)) );
Y = _IQ23toIQ(temp);
```

### 3.1.8 指定格式转换全局格式定点浮点 \_IQNtoIQ

**描述** 宏转换 IQN 格式定点浮点到 GLOBAL\_Q 格式定点浮点。

**声明**

C	<code>_iq _IQNtoIQ(_iqN A)</code>
C++	<code>iq IQNtoIQ(const iqN &amp;A)</code>

**输入** IQN 格式定点浮点数

**输出** GLOBAL\_Q 格式等效定点浮点数

**使用** 宏使用转换 IQ 精度范围结果到 GLOBAL\_Q 格式定点浮点数。

**样例：**

下面样例使用 Q26 格式计算复杂数据 (X+jY)：

$$Z = \sqrt{X^2 + Y^2}$$

Z, X, Y 值给定定义为 GLOBAL\_Q = 26，b 等式自身可能溢出，因此中间计算使用 Q=23 格式，并最终数值转换回全局格式：

```
#include "IQmathLib.h"
_iq Z, Y, X; // GLOBAL_Q = 26
_iq23 temp;
temp = _IQ23sqrt( _IQ23mpy(_IQtoIQ23(X), _IQtoIQ23(X)) +
    _IQ23mpy(_IQtoIQ23(Y), _IQtoIQ23(Y)) );
Y = _IQ23toIQ(temp);
```

### 3.1.9 全局格式转 16 位格式定点浮点\_IQtoQN

**描述** 宏转换 32-bit GLOBAL\_Q 格式定点浮点到 16-bitQN 格式定点浮点。

**声明**

C	<code>int _IQtoQN(_iq A)</code>
C++	<code>int IQtoQN(const iq &amp;A)</code>

**输入** GLOBAL\_Q 格式定点浮点数

**输出** 输入的等效 QN 格式定点浮点数 (16-bit)

**使用** 使用宏处理输入输出是 16-bits, 但中间操作使用 32-bits IQ 数据类型的转换.

**样例:**

乘积加和使用输入序列不是全局格式 GLOBAL\_Q:

```
Y = X0*C0 + X1*C1 + X2*C2 // X0, X1, X2 in Q15 format
                          //C0, C1, C2 in GLOBAL_Q format
```

转换 Q15 值到 IQ, 使用 IQ 计算结果, 并存贮结果到 Q15 格式:

```
short X0, X1, X2; // Q15 short
iq C0, C1, C2; // GLOBAL_Q
short Y; // Q15
_iq sum // IQ (GLOBAL_Q)

sum = _IQmpy(_Q15toIQ(X0), C0);
sum += _IQmpy(_Q15toIQ(X1), C1);
sum += _IQmpy(_Q15toIQ(X2), C2);
Y = _IQtoQ15(sum);
```

### 3.1.10 16 位格式转全局格式定点浮点 \_QntoIQ

**描述** 宏转换 QN 格式的 16bit 数值到全局格式 GLOBAL\_Q 的 32bit 数值

<b>声明</b>	C	_iq	_QntoIQ(int A)
	C++	iq	QntoIQ(int A)

**输入** 16bit 定点浮点格式 N

**输出** 输入值的 32bit GLOBAL\_Q 格式数据结果

**使用** 主要在输入和输出数据是 16bit, 但中间操作使用 IQ 数据类型操作处理.

**举例:**

一些计算的输入序列不是全局 GLOBAL\_Q 格式:

```
Y = X0*C0 + X1*C1 + X2*C2 // X0, X1, X2 in Q15 format
                          // C0, C1, C2 in GLOBAL_Q format
```

转换 Q15 到 IQ, 执行 IQ 的中间计算, 存贮会 Q15 格式结果

```
short X0, X1, X2; // Q15 short
iq C0, C1, C2; // GLOBAL_Q
short Y; // Q15
_iq sum // IQ (GLOBAL_Q)

sum = _IQmpy(_Q15toIQ(X0), C0);
sum += _IQmpy(_Q15toIQ(X1), C1);
sum += _IQmpy(_Q15toIQ(X2), C2);
Y = _IQtoQ15(sum);
```



## 3.2 算术运算乘除

### 3.2.1 同格式定点浮点乘\_IQmpy\_IQNmpy

**描述** 两个定点浮点的 C 编译模式相乘. 不处理饱和与舍入。更多时候两个定点浮点数相乘结果不超过其格式范围, 该函数消耗更少的运算周期和代码空间, 因此不超出范围时最常使用。

**声明** 全局 IQ 模式(IQ 格式= GLOBAL\_Q)

```
C      _iq      _IQmpy(_iq A, _iq B)
C++    iq      operator * (const iq &A, const iq &B)
        iq      &iq :: operator *= (const iq &A)
```

**Q 格式模式 (IQ 格式= IQ1 to IQ30)**

```
C      _iqN      _IQNmpy(_iqN A, _iqN B)
C++    iqN      operator * (const iqN &A, const iqN &B)
        iqN      &iqN :: operator *= (const iqN &A)
```

**输入** 全局 IQ 模式(IQ 格式= GLOBAL\_Q)

参数 A、B 为全局格式定点浮点数

**Q 格式模式 (IQ 格式= IQ1 ~IQ30)**

参数 A、B 为指定格式定点浮点数

**输出** 全局 IQ 模式(IQ 格式= GLOBAL\_Q)

乘结果的 GLOBAL\_Q 格式的定点浮点数

**Q 格式模式 (IQ 格式= IQ1 ~IQ30)**

乘结果的 IQN 格式的定点浮点数

**举例 1**

计算  $Y = M * X + B$  使用 GLOBAL\_Q 格式, 不使用进位和饱和处理.

```
_iq Y, M, X, B;
Y = _IQmpy(M, X) + B;
```

**举例 2:**

计算  $Y = M * X + B$  使用 IQ10 格式, 不使用进位和饱和处理

M, X, B 定义为 IQ10 格式.

```
_iq10 Y, M, X, B;
Y = _IQ10mpy(M, X) + B;
```

### 3.2.2 类四舍五入定点浮点乘\_IQrmpy\_IQNrmpy

**描述** 两个定点浮点的相乘. 不处理饱和但处理舍入. 需要绝对精度时, 处理 IQ 相乘, 并返回时执行舍入。等于附加 1/2 LSB 精度。

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

C        \_iq        \_IqrmPy(\_iq A, \_iq B)

C++     iq        IqrmPy(const iq &A, const iq &B)

**Q 格式函数(IQ 格式= IQ1 ~IQ30)**

C        \_iqN        \_IQNrmpy(\_iqN A, \_iqN B)

C++     iqN        IQNrmpy(const iqN &A, const iqN &B)

**输入    全局 IQ 函数(IQ 格式= GLOBAL\_Q)**

参数 A、B 为全局格式定点浮点数

**Q 格式函数(IQ 格式= IQ1 ~IQ30)**

参数 A、B 为指定格式定点浮点数

**输出    全局 IQ 函数(IQ 格式= GLOBAL\_Q)**

乘结果的 GLOBAL\_Q 格式的定点浮点数

**Q 格式函数(IQ 格式= IQ1 ~IQ30)**

乘结果的 IQN 格式的定点浮点数

**举例 1:**

计算  $Y = M * X + B$  的 GLOBAL\_Q 格式, 处理舍入但忽略饱和处理.

\_iq Y, M, X, B;

$Y = \_IqrmPy(M, X) + B;$

**举例 2:**

计算  $Y = M * X + B$  的 IQ10 格式, 处理输入但忽略饱和处理.

\_iq10 Y, M, X, B;

$Y = \_IQ10rmpy(M, X) + B;$

### 3.2.3 类四舍五入饱和定点浮点乘\_IQrsmPy \_IQNrsmPy

**描述** 两个定点浮点的相乘. 处理饱和和处理舍入. 即计算时可能操作格式所标定的范围, 该功能执行舍入, 并饱和结果到对应格式可表达的范围。

**声明    全局 IQ 函数(IQ 格式= GLOBAL\_Q)**

C        \_iq        \_IQrsmPy(\_iq A, \_iq B)

C++     iq        IQrsmPy(iq &A, iq &B)

**Q 格式函数(IQ 格式= IQ1 ~IQ30)**

C        \_iqN        \_IQNrsmPy(\_iqN A, \_iqN B)

C++     iqN        IQNrsmPy(const iqN &A, const iqN &B)

**输入    全局 IQ 函数(IQ 格式= GLOBAL\_Q)**

参数 A、B 为全局格式定点浮点数

**Q 格式函数(IQ 格式= IQ1 ~IQ30)**

参数 A、B 为指定格式定点浮点数

**输出    全局 IQ 函数(IQ 格式= GLOBAL\_Q)**

乘结果的 GLOBAL\_Q 格式的定点浮点数

**Q 格式函数(IQ 格式= IQ1 ~IQ30)**

乘结果的 IQN 格式的定点浮点数

**使用** 假定使用 IQ26 作为 GLOBAL\_Q 全局格式. 这意味着数据的大约范围  $[-32.0, 32.0]$ . 如

果两个这样的数相乘, 结果的最大范围是  $[-1024, 1024]$ . 计算确保饱和到格式的范围 $\pm 32$ 。

**举例 1:**

计算  $Y = M \times X$  使用 GLOBAL\_Q 格式, 处理舍入并执行饱和处理。

```
_iq Y, M, X;  
M=_IQ(10.9); // M=10.9  
X=_IQ(4.5); // X=4.5  
Y = _IQrmpy(M, X); // Y= ~32.0, 输出饱和到 MAX
```

**举例 2:**

计算  $Y = M \times X$  使用 IQ26 格式, 处理舍入并执行饱和处理

```
_iq26 Y, M, X;  
M=_IQ26(-10.9); // M=-10.9  
X=_IQ26(4.5); // X=4.5  
Y = _IQ26rmpy(M, X); // Y= -32.0, 饱和到 MIN
```

### 3.2.4 定点浮点乘以整数的乘\_IQmpyI32 \_IQNmpyI32

**描述** 宏方法计算定点浮点与整数的相乘结果。

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

```
C      _iq      _IQmpyI32(_iq A, long B)  
C++    iq      IQmpyI32(const iq &A, long B)
```

**Q 格式函数(IQ 格式= IQ1 ~IQ30)**

```
C      _iqN      _IQNmpyI32(_iqN A, long B)  
C++    iqN      IQNmpyI32(const iqN &A, long B)
```

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

参数 A 为 GLOBAL\_Q 格式定点浮点, 参数 B 为 32 位符号整数

**Q 格式函数(IQ 格式= IQ1 ~IQ30)**

参数 A 为 IQN 格式定点浮点, 参数 B 为 32 位符号整数

**输出** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

乘结果的 GLOBAL\_Q 格式的定点浮点数

**Q 格式函数(IQ 格式= IQ1 ~IQ30)**

乘结果的 IQN 格式的定点浮点数

**举例 1:**

计算  $Y = 5 \times X$  使用 GLOBAL\_Q 个格式. 假定 GLOBAL\_Q = IQ26。

```
_iq Y, X;  
X = _IQ(5.1); // X=5.1 in GLOBAL_Q format  
Y = IQmpyI32(X, 5); // Y= 25.5 in GLOBAL_Q format
```

**举例 2:**

计算  $Y = 5 \times X$  使用 IQ26 格式。

```
_iq26 Y, X;  
long M;
```

```
M=5; // M=5
X = _IQ26(5.1); // X=5.1 in IQ29 format
Y = _IQ26mpyI32(X,M); // Y=25.5 in IQ29 format
```

### 3.2.5 定点浮点乘整数的定点整数\_IQmpyI32int \_IQNmpyI32int

**描述** 定点浮点乘以整数并返回整数部分

**声明** 全局 IQ 模式(IQ 格式= GLOBAL\_Q)

```
C      long      _IQmpyI32int(_iq A, long B)
```

```
C++    long      IQmpyI32int(const iq &A, long B)
```

**Q 格式模式 (IQ 格式= IQ1 ~ IQ30)**

```
C      long      _IQNmpyI32int(_iqN A, long B)
```

```
C++    long      IQNmpyI32int(const iqN &A, long B)
```

**输入** 全局 IQ 模式(IQ 格式= GLOBAL\_Q)

参数 A 为 GLOBAL\_Q 格式定点浮点, 参数 B 为 32 位符号整数.

**Q 格式模式 (IQ 格式= IQ1 ~ IQ30)**

参数 A 为 IQN 格式定点浮点, 参数 B 为 32 位符号整数

**输出** 全局 IQ 模式(IQ 格式= GLOBAL\_Q)

结果的整数部分整数值

**Q 格式模式 (IQ 格式= IQ1 ~ IQ30)**

结果的整数部分整数值

**举例**

转换范围 [- 1.0, +1.0]的定点浮点值 到范围为[ 0, 1023]的 DAC 值:

```
_iq Output;
long temp;
short OutputDAC;
// value converted to +/- 512
temp = _IQmpyI32int(Output, 512);
// value scaled to 0 to 1023
temp += 512;
// saturate within range of DAC
if( temp > 1023 )temp = 1023;
if( temp < 0 ) temp = 0;
// output to DAC value
OutputDAC = (int )temp;
```

**注意:** 整数部分的提取设计先计算的 64 位结果, 再进行整数提取. 因此避免可能溢出的条件.

### 3.2.6 定点浮点乘整数的定点小数\_IQmpyI32frac\_IQNmpyI32frac

**描述** 定点浮点乘以整数并返回小数定点部分

**声明** 全局 IQ 模式(IQ 格式= GLOBAL\_Q)

```
C      _iq      _IQmpyI32frac(_iq A, long B)
```

```
C++    iq      IQmpyI32frac(const iq &y, long x)
```

**Q 格式模式 (IQ 格式= IQ1 ~ IQ30)**

```
C      _iqN     _IQNmpyI32frac(_iqN A, long B)
```

```
C++    iqN     IQNmpyI32frac(const iqN &A, long B)
```

**输入** 全局 IQ 模式(IQ 格式= GLOBAL\_Q)

参数 A 为 GLOBAL\_Q 格式定点浮点, 参数 B 为 32 位符号整数.

**Q 格式模式 (IQ 格式= IQ1 ~ IQ30)**

参数 A 为 IQN 格式定点浮点, 参数 B 为 32 位符号整数.

**输出** 全局 IQ 模式(IQ 格式= GLOBAL\_Q)

结果的小数部分全局定点格式值

**Q 格式模式 (IQ 格式= IQ1 ~ IQ30)**

结果的小数部分 IQN 定点格式值

**举例:**

提取乘计算结果的小数部分。(假定 GLOBAL\_Q=IQ26)

```
_iq X1= _IQ(2.5);
```

```
_iq X2= _IQ26(-1.1);
```

```
_iq Y1frac, Y2frac;
```

```
long M1=5, M2=9;
```

```
// Y1frac = 0.5 in GLOBAL_Q
```

```
Y1frac = IQmpyI32frac(X1, M1);
```

```
// Y2frac = -0.9 in GLOBAL_Q
```

```
Y2frac = IQ26mpyI32frac(X2, M2);
```

### 3.2.7 不同格式定点浮点乘\_IQNmpyIQX

**描述** 不同格式的定点浮点的 C 编译模式相乘方法

**声明** 全局 IQ 模式(IQ 格式= GLOBAL\_Q)

```
C      _iq _IQmpyIQX(_iqN1 A, int N1, _iqN2 B, int N2)
```

```
C++    iq IQmpyIQX(iqN1 A, int N1, iqN2 B, int N2)
```

**Q 格式模式 (IQ 格式= IQ1 ~ IQ30)**

```
C      _iqN _IQNmpyIQX(_iqN1 A, int N1, _iqN2 B, int N2)
```

```
C++    iqN IQNmpyIQX(iqN1 A, int N1, iqN2 B, int N2)
```

**输入** 参数为 IQN1 格式, 参数 B 为 IQN2 格式.

**输出** 全局 IQ 模式(IQ 格式= GLOBAL\_Q)

乘结果的 GLOBAL\_Q 格式定点浮点值

**Q 格式模式 (IQ 格式= IQ1 ~IQ30)**

乘结果的 IQN 格式定点浮点值

**使用** 适合计算不同格式定点浮点的乘运算.

**举例:**

计算等式:  $Y = X0 * C0 + X1 * C1 + X2 * C2$

其中 X0, X1, X2 值为 IQ30 格式(范围-2 to +2)

C0, C1, C2 值为 IQ28 格式(范围 - 8 to +8)

Y 的最大范围是 -48 到+48. 因此存在数据的格式至少需要为 IQ25.

**情形 1: GLOBAL\_Q=IQ25**

```
_iq30 X0, X1, X2; // All values IQ30
_iq28 C0, C1, C2; // All values IQ28
_iq Y; // 结果 GLOBAL_Q = IQ25
Y = _IQmpyIQX(X0, 30, C0, 28);
Y += _IQmpyIQX(X1, 30, C1, 28);
Y += _IQmpyIQX(X2, 30, C2, 28);
```

**情形 2: IQ 格式计算**

```
_iq30 X0, X1, X2; // All values IQ30
_iq28 C0, C1, C2; // All values IQ28
_iq25 Y; // 结果 GLOBAL_Q = IQ25
Y = _IQ25mpyIQX(X0, 30, C0, 28);
Y += _IQ25mpyIQX(X1, 30, C1, 28);
Y += _IQ25mpyIQX(X2, 30, C2, 28);
```

### 3.2.8 同格式定点浮点除法\_IQdiv \_IQNdiv

**描述** 提供 2 个定点浮点相除, 商为 32-bit 对应定点浮点格式

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

```
C      _iq _IQdiv(_iq A, _iq B)
C++    iq operator / (const iq &A, const iq &B)
        iq &iq :: operator /= (const iq &A)
```

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

```
C      _iqN _IQNdiv(_iqN A, iq B)
C++    iqN operator / (const iqN &A, const iqN &B)
        iqN &iqN :: operator /= (const iqN &A)
```

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

参数 A、B 为全局格式定点浮点数.

**Q 格式函数(IQ 格式= IQ1 ~IQ30)**

参数 A、B 为指定格式定点浮点数

**输出** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

除结果全局格式定点浮点值 A/B .

Q 格式函数(IQ 格式= IQ1 ~IQ30)

除结果指定格式定点浮点数 A/B

精度  $= 20\log_2(2^{31}) - 20\log_2(7) = 28 \text{ bits}$

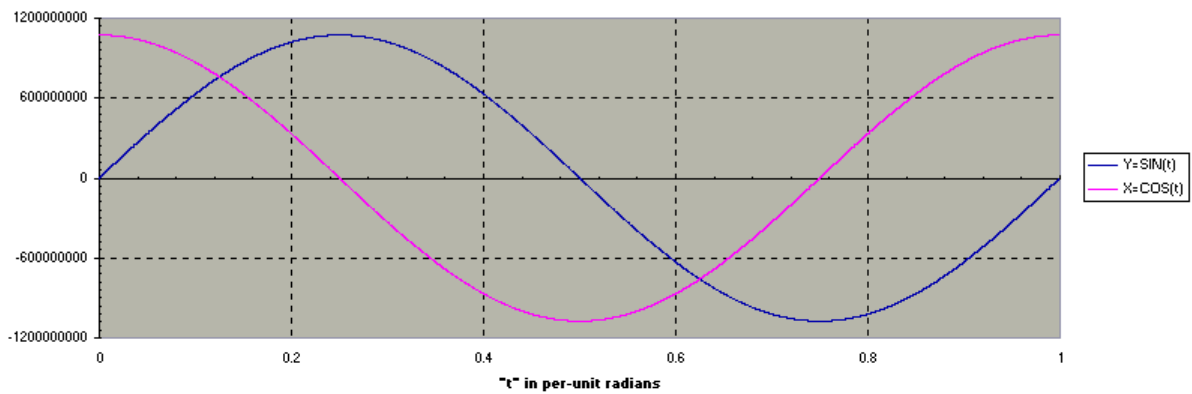
举例：

样例获取  $1/15=0.666$  使用 GLOBAL\_Q 和 Q28 格式.

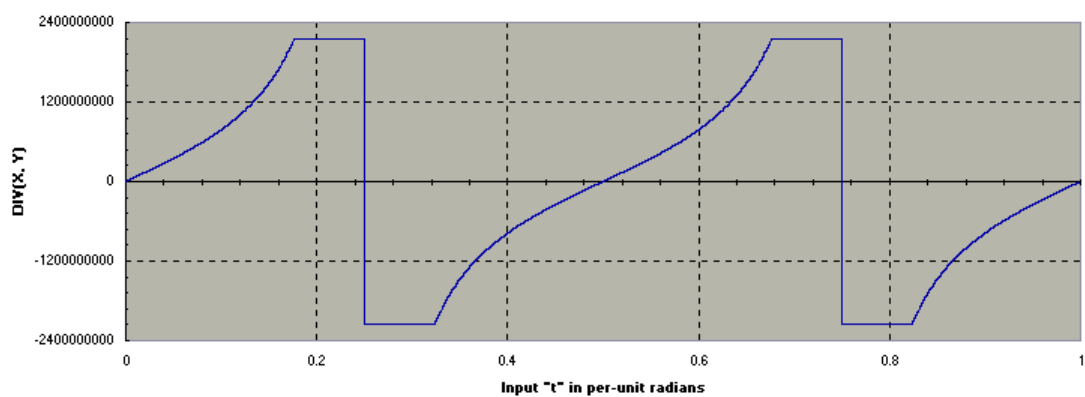
```
#include "IQmathLib.h"
_iq in1 out1;
_iq28 in2 out2;
void main(void )
{
    in1 = _IQ(1.5);
    out1 = _IQdiv(_IQ(1.0), in1);
    in2 = _IQ28(1.5);
    out2 = _IQ28div(_IQ28(1.0), in2);
}
```

定点浮点除法 对比 C 语言浮点除法：

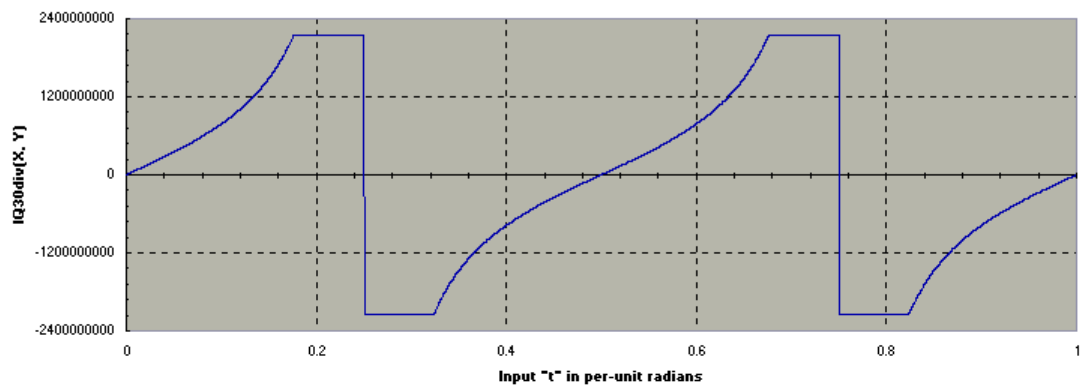
Division function input :  $X=\sin(t)$  &  $Y=\cos(t)$  in Q30 format



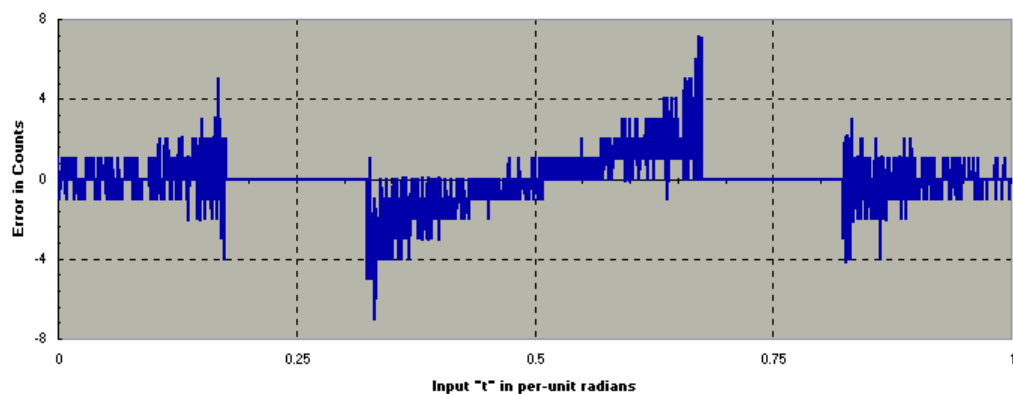
DIV(X, Y)-Floating Point Computation (Q30)



IQ30div(X, Y) - Floating Point Computation(Q30)

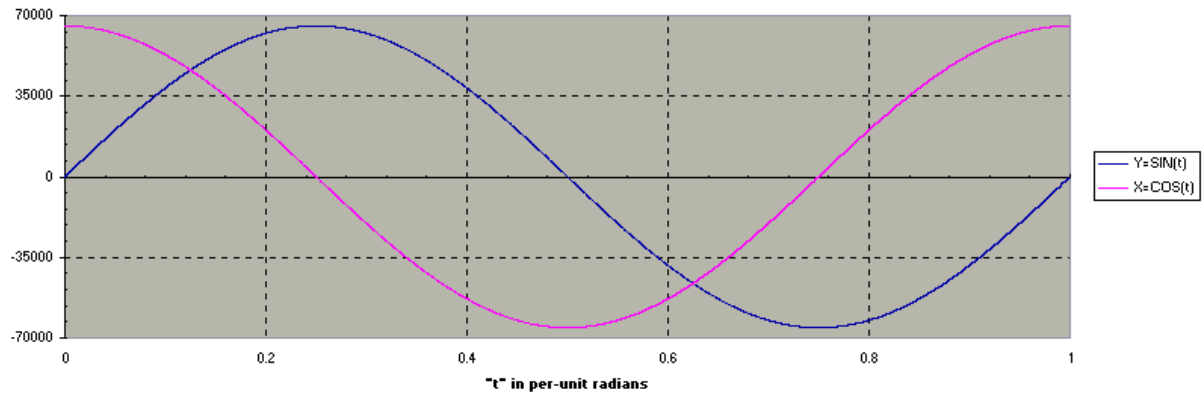


Error=IQ30div(X, Y) - DIV(X, Y)

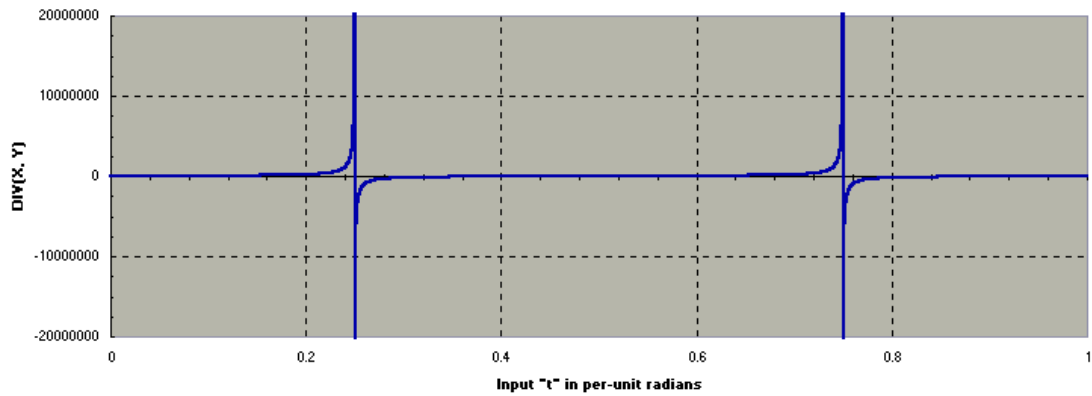




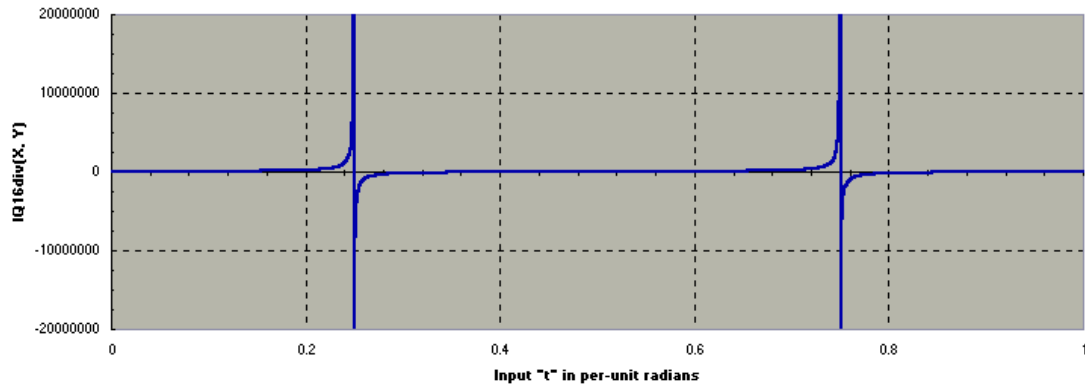
Division function input : X=SIN(t) & Y=COS(t) in Q16 format



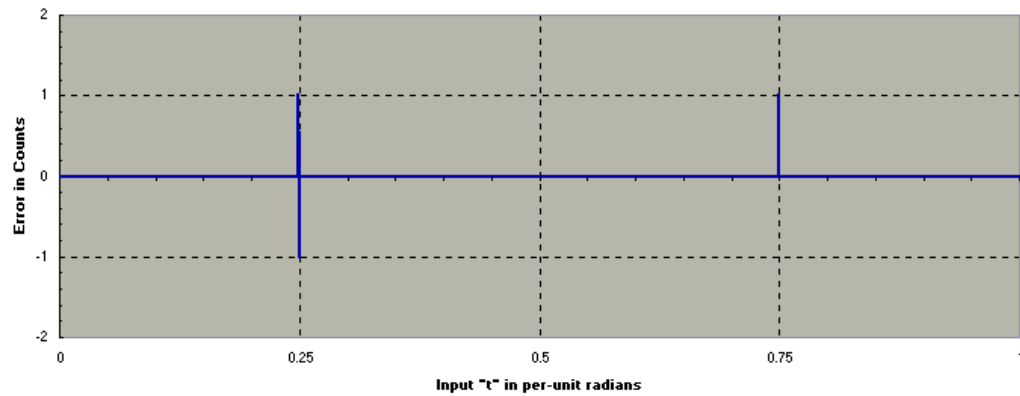
DIV(X, Y)-Floating Point Computation (Q16)



IQ16div(X, Y)-Floating Point Computation(Q16)



Error=IQ16div(X, Y) - DIV(X, Y)



### 3.2.9 定点浮点乘以 2 的幂优化 IQmpy2, 4, 8..64

**描述** 在 IQmathLib.h 中宏定义, 通过左移实现乘以 2 的操作。提供乘 2/4/8/16/32/64 支持。当使用 FLOAT\_MATH 时, 仍为相应的乘法运算。

**输入** 指定格式定点浮点数。

**输出** 指定格式定点浮点数。

**举例**

计算  $Y = 2 * X = X \ll 1$

```
_iq Y;
```

```
Y = _IQmpy2(X);
```

计算  $Y = 16 * X = X \ll 4$

```
_iq Y;
```

```
Y = _IQmpy16(X);
```

### 3.2.10 定点浮点除以 2 的幂优化 IQdiv2, 4, 8..64

**描述** 在 IQmathLib.h 中宏定义, 通过右移 1 位实现乘以 1/2。提供除 2/4/8/16/32/64 支持 4, 8 16, 32 and 64.

当使用 FLOAT\_MATH 时, 仍为相应的乘小数运算, 如  $*(0.5/0.25/0.125/0.0625/0.03125/0.015625)$ .

**输入** 指定格式定点浮点数。

**输出** 指定格式定点浮点数。

**举例**

计算  $Y = 2 * X = X \ll 1$

```
_iq Y;
```

```
Y = _IQdiv2(X);
```

计算  $Y = 16 * X = X \ll 4$

```
_iq Y;
```

```
Y = _IQdiv16(X);
```

## 3.3 三角函数 SIN COS TAN

函数的参数弧度与角度转换关系为: 弧度=角度\* $\pi/180$ 。即  $0^{\circ} \sim 360^{\circ}$  对应  $0^{\circ} \sim 2\pi$ 。 $-180^{\circ} \sim 180^{\circ}$  对应  $-\pi \sim \pi$ 。代码编写使用 PI 替代  $\pi$ 。

### 3.3.1 定点浮点表达的反向正弦值 \_IQasin IQNasin

**描述** 计算输入的反向正弦结果, 单位为弧度。

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

	C	<code>_iq</code>	<code>_IQasin(_iq A)</code>
	C++	<code>iq</code>	<code>IQasin(const iq &amp;A)</code>
	<b>Q 格式函数 (IQ 格式= IQ1 ~ IQ30)</b>		
	C	<code>_iqN</code>	<code>_IQNasin(_iqN A)</code>
	C++	<code>iqN</code>	<code>IQNasin(const iqN &amp;A)</code>
输入	<b>全局 IQ 函数(IQ 格式= GLOBAL_Q)</b>		
	GLOBAL_Q 格式表达的弧度正弦值		
	<b>Q 格式函数 (IQ 格式= IQ1 ~ IQ30)</b>		
	IQN 格式表达的弧度正弦值		
输出	<b>全局 IQ 函数(IQ 格式= GLOBAL_Q)</b>		
	返回 GLOBAL_Q 格式的反向正弦弧度		
	<b>Q 格式函数 (IQ 格式= IQ1 ~ IQ30)</b>		
	返回IQN格式的反向正弦弧度		

**使用** 实际有效IQN格式为1~29，因为IQ30格式数据范围不能表达 $[-\pi:\pi]$ 直接的数，IQ30个数据表达范围为 $[-2:2]$ 。

**举例** 获取等式结果  $\text{asin}(0.70710678) = (0.25 * \pi)$ 。

假定 IQmath 头文件配置 GLOBAL\_Q = Q29 。

```
#include "IQmathLib.h"
_iq in1, out1;
_iq29 in2, out2;
void main(void )
{
    // in1 = in2 = 0.70710678L x 2^29 = 0x16A09E65
    // out1 = out2 = asin(0.70710678) = 0x1921FB53
    in1 = _IQ(0.70710678L);
    out1 = _IQasin(in1);
    in2 = _IQ29(0.70710678L)
    out2 = _IQ29asin(in2);
}
```

### 3.3.2 定点浮点表达的正弦值\_IQsin \_IQNsin

**描述** 查表计算输入值的 sin 值(单位弧度)，表入口之间使用泰勒级数扩展。

**声明** **全局 IQ 函数(IQ 格式= GLOBAL\_Q)**

```
C      _iq      _IQsin(_iq A)
C++    iq      IQsin(const iq &A)
```

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

```
C      _iqN      _IQNsin(_iqN A)
C++    iqN      IQNsin(const iqN &A)
```

**输入** **全局 IQ 函数(IQ 格式= GLOBAL\_Q)**

GLOBAL\_Q 格式定点浮点表达的弧度值

Q 格式函数 (IQ 格式= IQ1 ~ IQ30)

IQN 格式定点浮点表达的弧度值

输出 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式的正弦计算结果

Q 格式函数 (IQ 格式= IQ1 ~ IQ30)

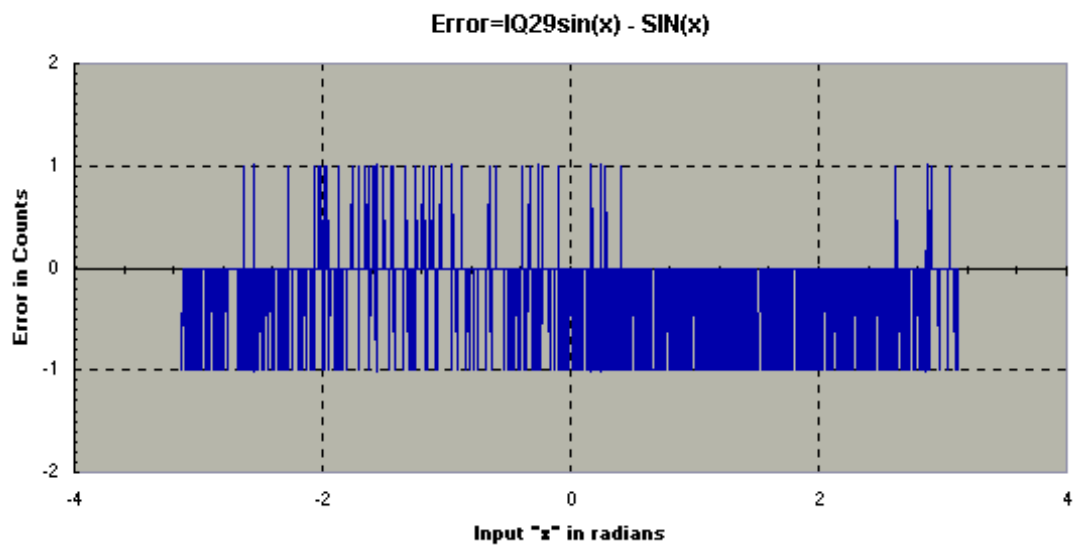
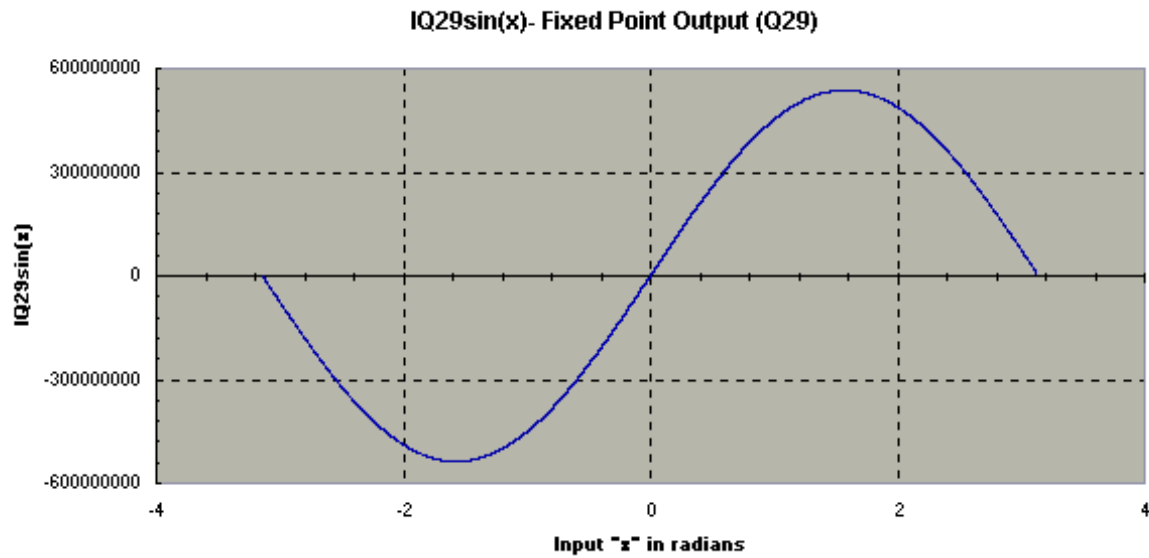
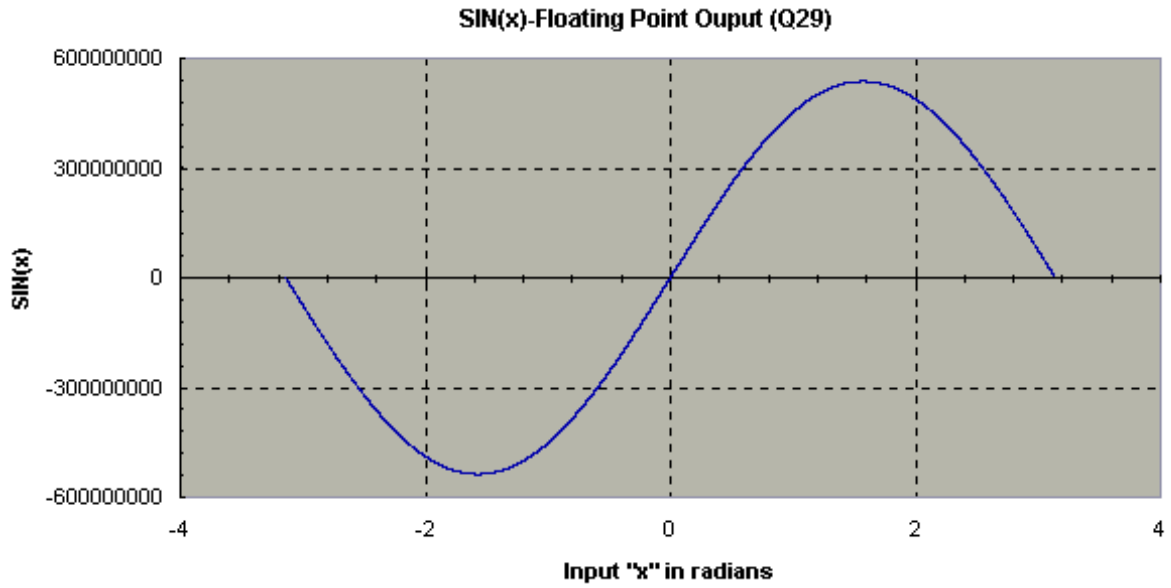
IQN 格式的正弦计算结果

精度  $= 20\log_2(\pi \times 2^{29}) - 20\log_2(1) = 30 \text{ bits}$

举例 获取  $\sin(0.25 * \pi) = 0.707$  。假定 IQmath 头文件配置 GLOBAL\_Q = Q29 。

```
#include "IQmathLib.h"
#define PI 3.14159265161
_iq in1, out1;
_iq28 in2, out2;
void main(void ){
    // in1 = 0.25 x pi x 2^29 = 0x1921FB54
    // out1 = sin(0.25 x pi)x 2^29 = 0x16A09E66
    // in2 = 0x25 x pi x 2^29 = 0x1921FB54
    // out2 = sin(0.25 x pi)x 2^29 = 0x16A09E66
    in1=_IQ(0.25*PI);
    out1=_IQsin(in1);
    in2=_IQ29(0.25*PI);
    out2=_IQ29sin(in2);
}
```

**IQNsin 函数对比 C 语言浮点 SIN: 输入参数【 $-\pi$  to  $\pi$ 】：**



### 3.3.3 定点浮点表达的正弦值\_IQsinPU \_IQNsinPU

**描述** 查表计算输入值的 sin 值(弧度占比)，表入口之间使用泰勒级数扩展.

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

```
C      _iq      _IQsinPU(_iq A)
C++    iq      IQsinPU(const iq &A)
```

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

```
C      _iqN      _IQNsinPU(_iqN A)
C++    iqN      IQNsinPU(const iqN &A)
```

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式单位弧度的定点浮点数

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

IQN 格式单位弧度的定点浮点数

**输出** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式的正弦计算结果.

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

IQN 格式的正弦计算结果

**精度**  $= 20\log_2(1 \times 2^{30}) - 20\log_2(1) = 30 \text{ bits}$

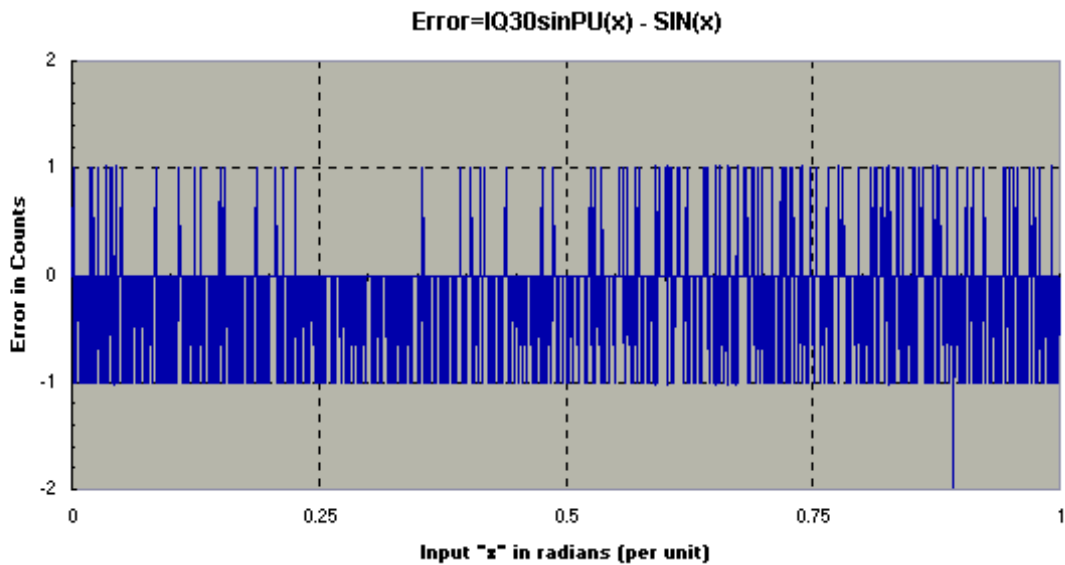
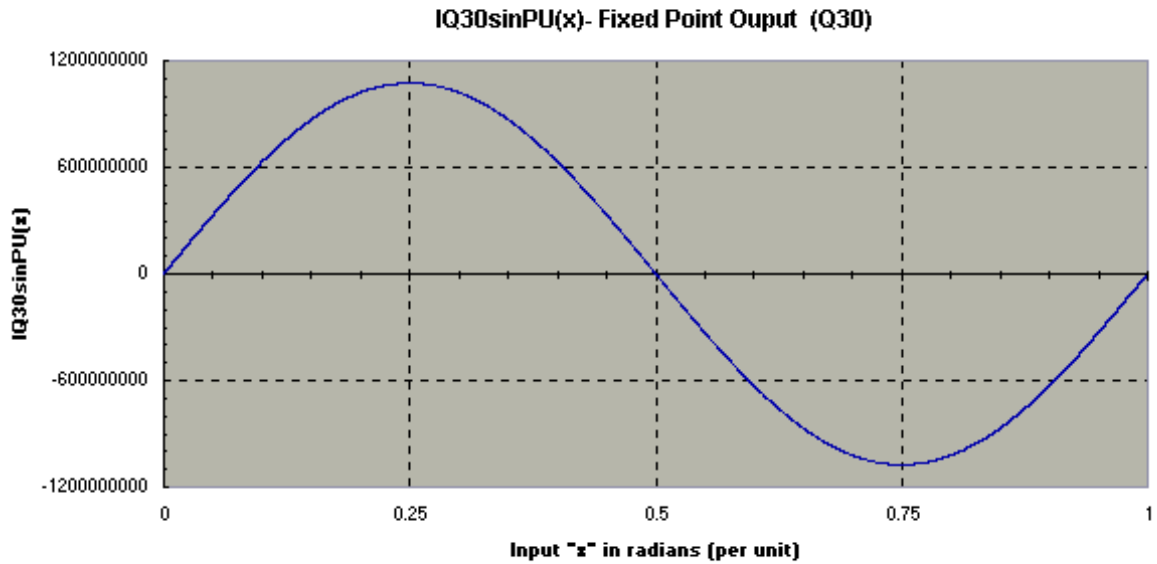
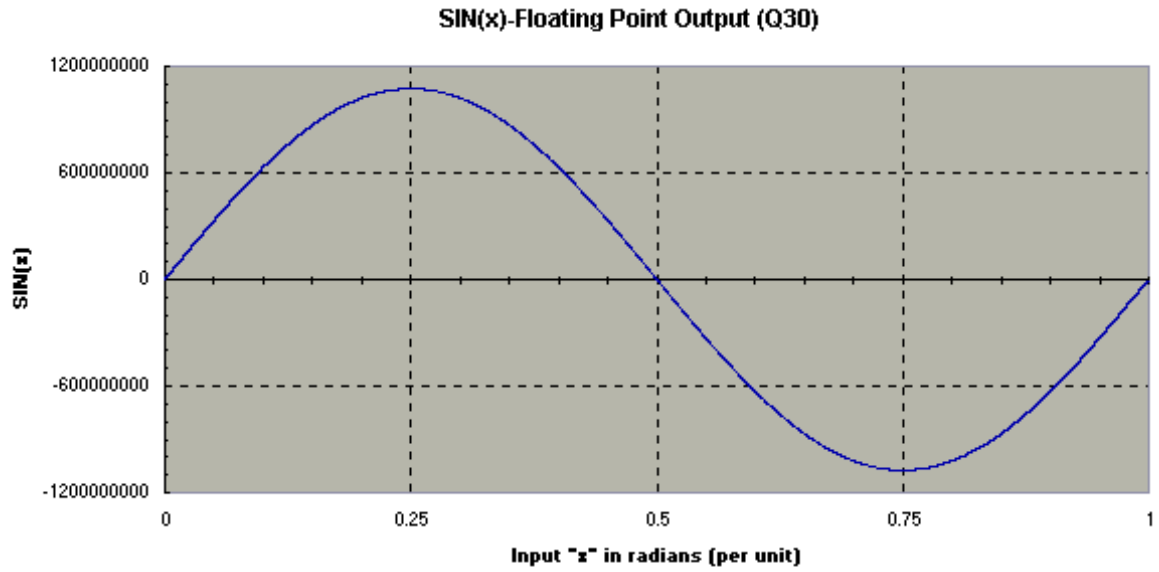
**举例:**

获取  $\sin(0.25 \times \pi) = 0.707$  假定 IQmath 头文件配置 GLOBAL\_Q = Q30。

```
#include "IQmathLib.h"
#define PI 3.14159265161
_iq in1, out1;
_iq30 in2, out2;
void main(void ) {
    // in1 = (0.25 x PI)/(2PI) x 2^30 = 0x08000000
    // out1 = sin(0.25 x PI) x 2^30 = 0x2D413CCD
    // in2 = (0.25 x PI)/(2PI) x 2^30 = 0x08000000
    // out2 = sin(0.25 x PI) x 2^30 = 0x2D413CCD
    in1 = _IQ(0.25*PI/(2*PI));
    out1 = _IQsinPU(in1);
    in2 = _IQ30(0.25*PI/(2*PI));
    out2 = _IQ30sinPU(in2);
}
```

即 sin 与 sinPU 的参数关系为  $A=B \times 2\pi$

**IQNsinPU 函数对比 C 语言浮点 SIN: 输入参数【 $(0 \sim 2\pi)/2\pi$ 】(单位比):**



### 3.3.4 定点浮点表达的反向余弦值\_IQacos\_IQNacos

**描述** 计算输入值的反向余弦值，反向余弦结果值单位为弧度。该实现为宏定义：

```
#define _IQXXacos(A) (_IQXX(1.570796327) - _IQXXasin(A))
```

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

```
C      _iq      _IQacos(_iq A)
```

```
C++    iq      IQacos( const iq & A)
```

Q 格式函数 (IQ 格式= IQ1 ~ IQ30)

```
C      _iqN      _IQNacos(_iqN A)
```

```
C++    iqN      IQNacos(const iqN &A)
```

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式表达的弧度余弦值

Q 格式函数 (IQ 格式= IQ1 ~ IQ30)

IQN 格式表达的弧度余弦值

**输出** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

返回 GLOBAL\_Q 格式的反向余弦弧度

Q 格式函数 (IQ 格式= IQ1 ~ IQ30)

返回 IQN 格式的反向余弦弧度

**举例：**获取  $\text{asin}(0.70710678) = (0.25 \times \pi)$ ，假定 IQmath 头文件配置 GLOBAL\_Q = Q29.

```
#include "IQmathLib.h"
_iq in1, out1;
_iq29 in2, out2;
void main(void )
{
    // in1 = 0x70710678 x 2^29 = 0x16A09E65
    // out1 = acos(0.70710678L) x 2^29 = 0x1921FB55
    // in2 = 0x70710678 x 2^29 = 0x16A09E65
    // out2 = acos(0.70710678L) x 2^29 = 0x1921FB55
    in1 = _IQ(0.70710678L);
    out1 = _IQacos(in1);
    in2 = _IQ29(0.70710678L);
    out2 = _IQ29acos(in2);
}
```

### 3.3.5 定点浮点表达的余弦值\_IQcos\_IQNcos

**描述** 查表计算输入值的 cos 值(弧度输入)，表入口之间使用泰勒级数扩展.

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

```
C      _iq      _IQcos(_iq A)
```

```
C++    iq      IQcos(const iq &A)
```



Q 格式函数 (IQ 格式= IQ1 ~ IQ30)

C        \_iqN        \_IQNcos(\_iqN A)

C++      iqN        IQNcos(const iqN &A)

输入    全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式定点浮点表达的弧度值

Q 格式函数 (IQ 格式= IQ1 ~ IQ30)

IQN 格式定点浮点表达的弧度值

输出    全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式的余弦计算结果

Q 格式函数 (IQ 格式= IQ1 ~ IQ30)

IQN 格式的余弦计算结果

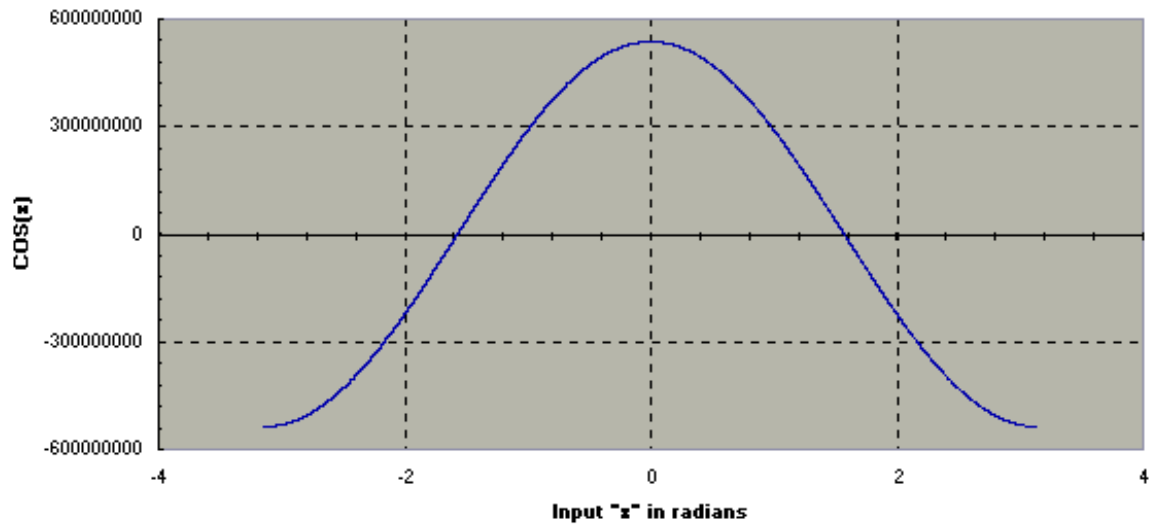
精度     $= 20\log_2(\pi \times 2^{29}) - 20\log_2(2) = 30 \text{ bits}$

举例    获取  $\cos(0.25 \times \pi) = 0.707$  , 假定 IQmath 头文件配置 GLOBAL\_Q = Q29.

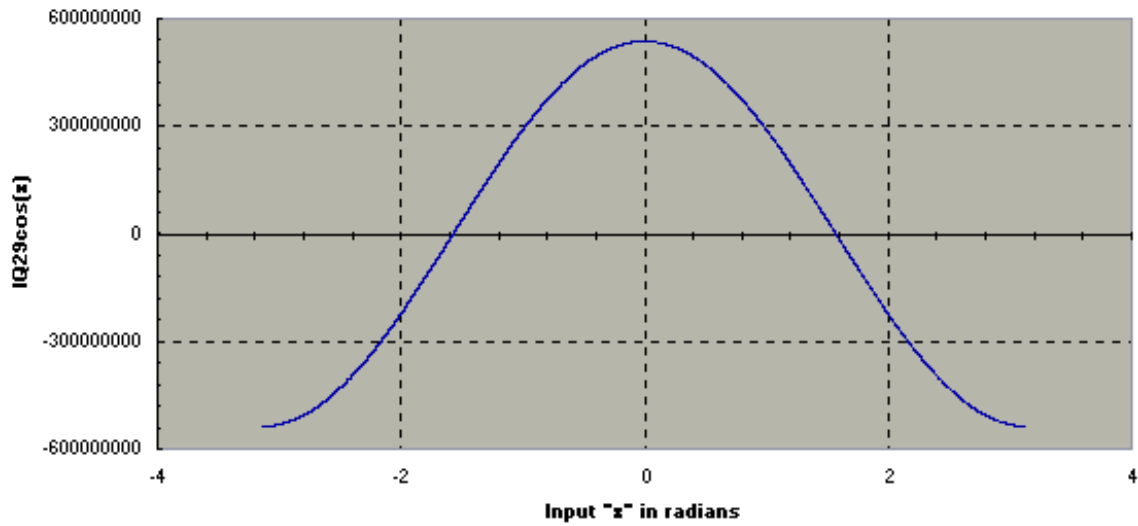
```
#include "IQmathLib.h"
#define PI 3.14159265161
_iq in1, out1;
_iq29 in2 out2;
void main(void ){
    // in = 0.25 x PI x 2^29 = 0x1921FB54
    // out1 = cos(0.25 x PI) x 2^29 = 0x16A09E67
    // in = 0.25 x PI x 2^29 = 0x1921FB54
    // out1 = cos(0.25 x PI) x 2^29 = 0x16A09E67
    in1 = _IQ(0.25*PI);
    out1 = _IQcos(in1);
    in2 = _IQ29(0.25*PI);
    out2 = _IQ29cos(in2);
}
```

定点浮点 COS 函数对比 C 语言浮点 COS: 输入参数 $[-\pi \sim \pi]$  :

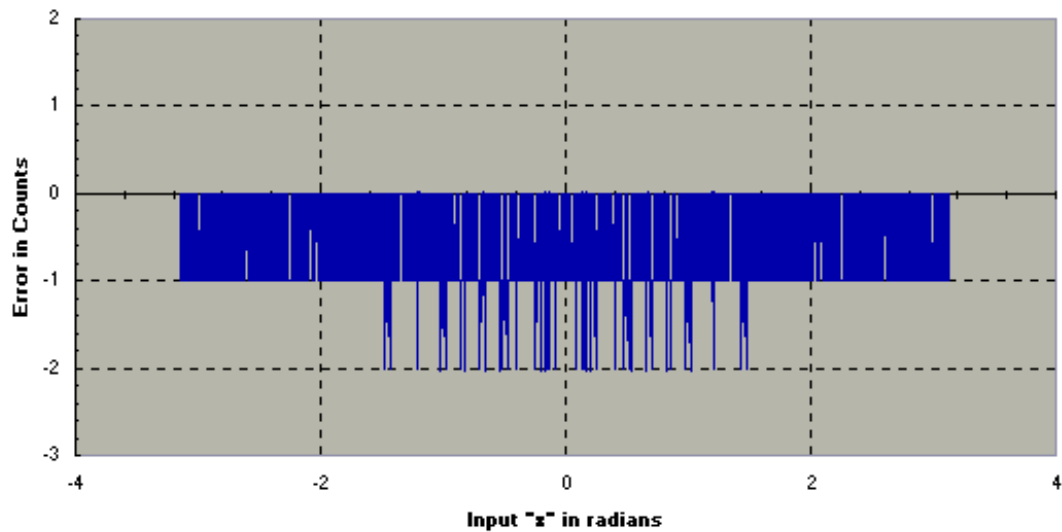
COS(x)-Floating Point Output (Q29)



IQ29cos- Fixed Point Output (Q29)



Error=IQ29cos(x) - COS(x)



### 3.3.6 定点浮点表达的余弦值\_IQcosPU \_IQNcosPU

**描述** 查表计算输入值的 cos 值(弧度占比输入)，表入口之间使用泰勒级数扩展.

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

```
C      _iq      _IQcosPU(_iq A)
C++    iq      IQcosPU(const iq &A)
```

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

```
C      _iqN      _IQNcosPU(_iqN A)
C++    iqN      IQNcosPU(const iqN &A)
```

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式定点浮点表达的弧度值

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

IQN 格式定点浮点表达的弧度值

**输出** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式的余弦计算结果

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

IQN 格式的余弦计算结果

**精度**

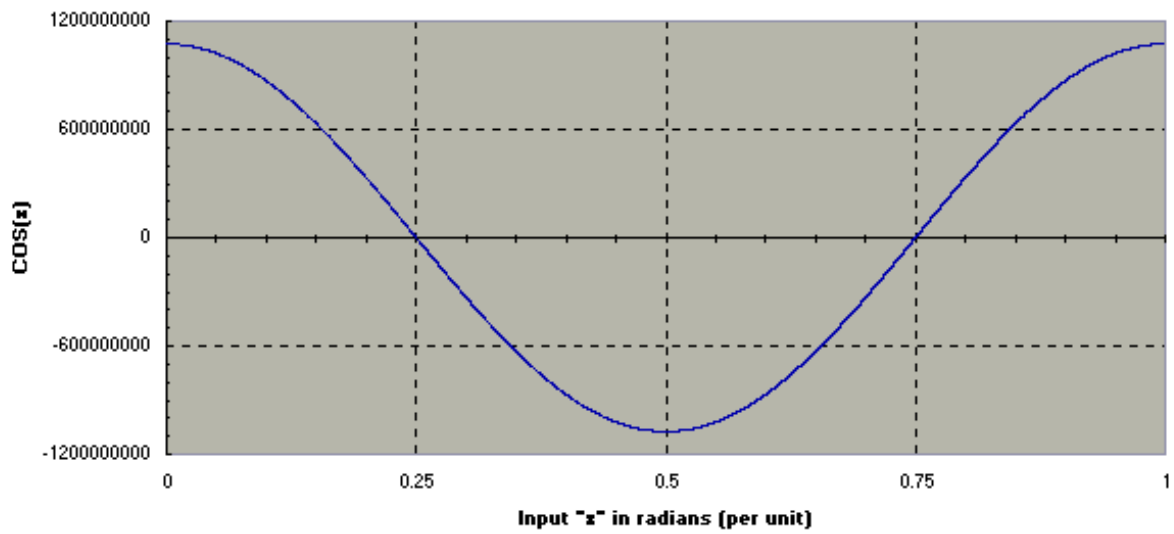
**举例：**获取  $\cos(0.25 \times \pi) = 0.707$ ，假定 IQmath 头文件配置 GLOBAL\_Q = Q30.

```
#include "IQmathLib.h"
#define PI 3.14159265161
_iq in1, out1;
_iq30 in2, out2
void main(void ){
    // in1 = in2 = (0.25 x PI)/(2PI) x 2^30 = 0x08000000
    // out1 = out2 = cos(0.25 x PI) x 2^30 = 0x2D413CCD
    in1 = _IQ(0.25*PI/(2*PI));
    out1 = _IQcosPU(in1);
    in2 = _IQ30(0.25*PI/(2*PI));
    out2 = _IQ30cosPU(in2);
}
```

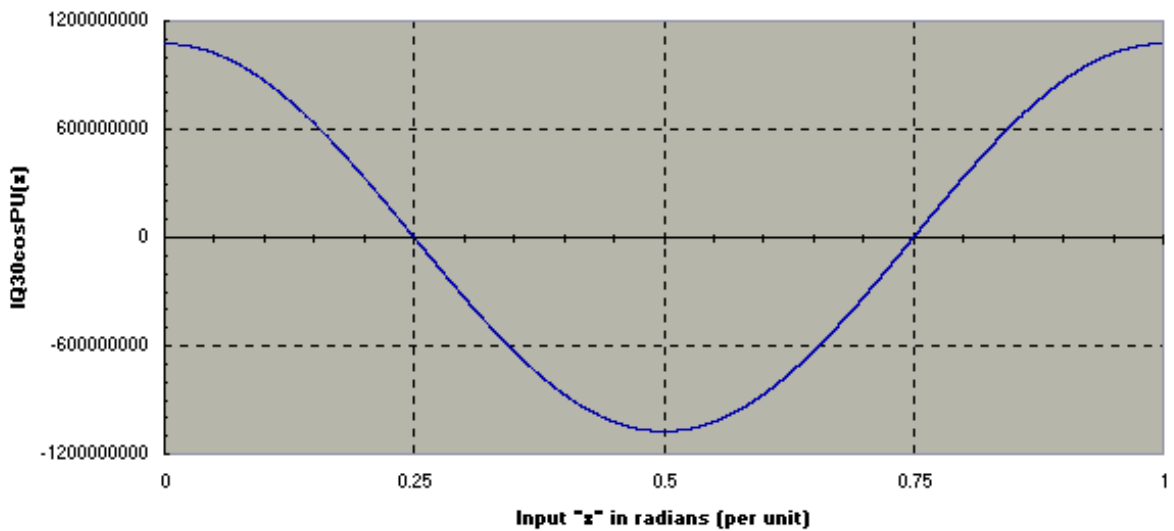
即 cos 与 cosPU 的参数关系为  $A=B \times 2\pi$

**定点浮点 COSPU 函数对比 C 语言浮点 COSPU：**输入参数  $[(0 \sim 2\pi)/2\pi]$  (单位比)：

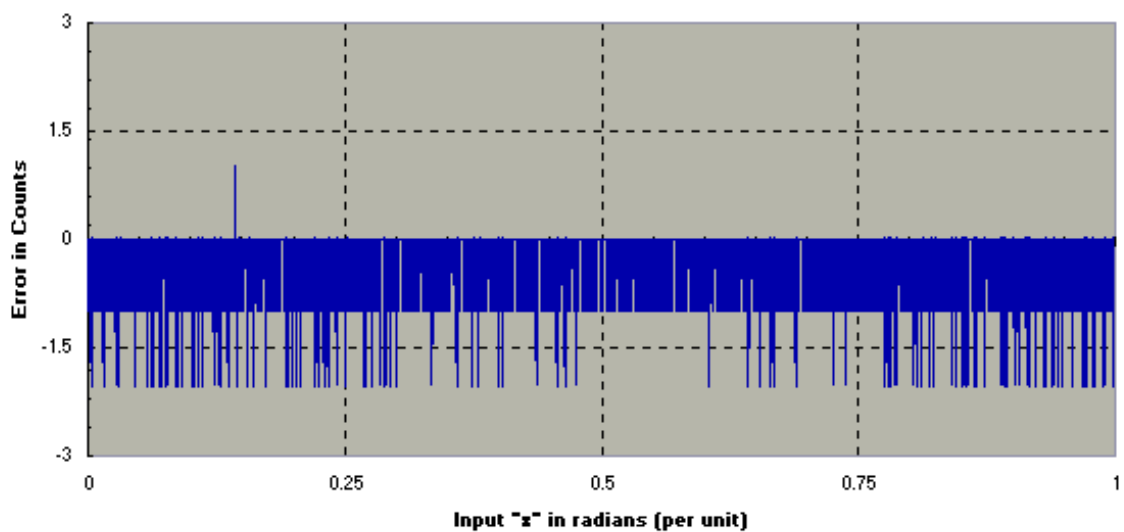
COS(x)-Floating Point Output (Q30)



IQ30cosPU(x)- Fixed Point Output (Q30)



Error=IQ30cosPU(x) - COS(x)



### 3.3.7 定点浮点表达的反切值\_IQatan2\_IQNatan2

**描述** 计算四象限反向正切值，输出单位为弧度，范围  $[-\pi \sim \pi]$ 。

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

C        \_iq        \_IQatan2(\_iq A, \_iq B)

C++     iq        IQatan2(const iq &A, const iq &B)

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

C        \_iqN       \_IQNatan2(\_iqN A, \_iqN B)

C++     iqN        IQNatan2(const iqN &A, const iqN &B)

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式定点浮点数. A 为正弦值, B 为余弦值

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

IQN 格式定点浮点数. A 为正弦值, B 为余弦值

**输出** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式反向正切值. 输出为弧度, 范围:  $[-\pi, +\pi]$

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

IQN 格式反向正切值. 输出为弧度, 范围:  $[-\pi, +\pi]$

**精度**  $= 20\log_2(\pi \times 2^{29}) - 20\log_2(32) = 26 \text{ bits}$

**举例**

获取  $\text{atan}(\sin(\pi/5), \cos(\pi/5)) = \pi/5$ , 假定 IQmath 头文件配置 GLOBAL\_Q = Q29.

```
#include "IQmathLib.h"
```

```
#define PI 3.14159265161
```

```
_iq xin1, yin1, out1;
```

```
_iq29 xin2, yin2, out2;
```

```
void main(void){
```

```
    // xin1 = xin2 = cos(PI/5) x 2^29 = 0x19E3779C
```

```
    // yin1 = yin2 = sin(PI/5) x 2^29 = 0x12CF2303
```

```
    // out1 = out2 = PI/5 x 2^29 = 0x141B2F74
```

```
    xin1 = _IQcos(_IQ(PI/5.0L));
```

```
    yin1 = _IQsin(_IQ(PI/5.0L));
```

```
    out1 = _IQatan2(yin1, xin1);
```

```
    xin2 = _IQ29cos(_IQ29(PI/5.0L));
```

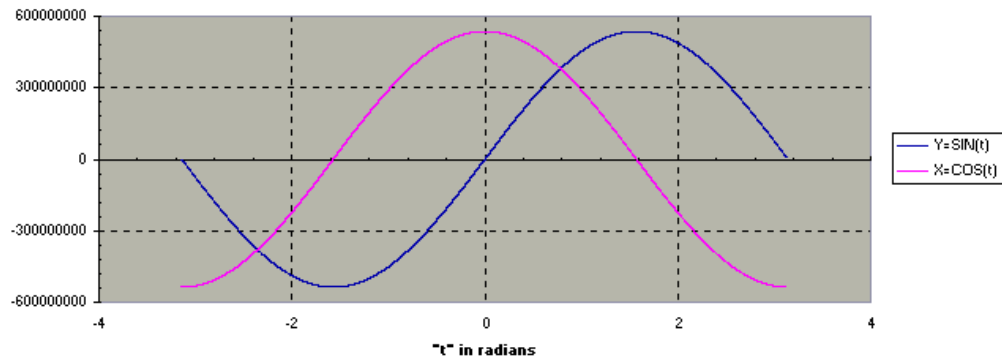
```
    yin2 = _IQ29sin(_IQ29(PI/5.0L));
```

```
    out2 = _IQ29atan2(yin1, xin1);
```

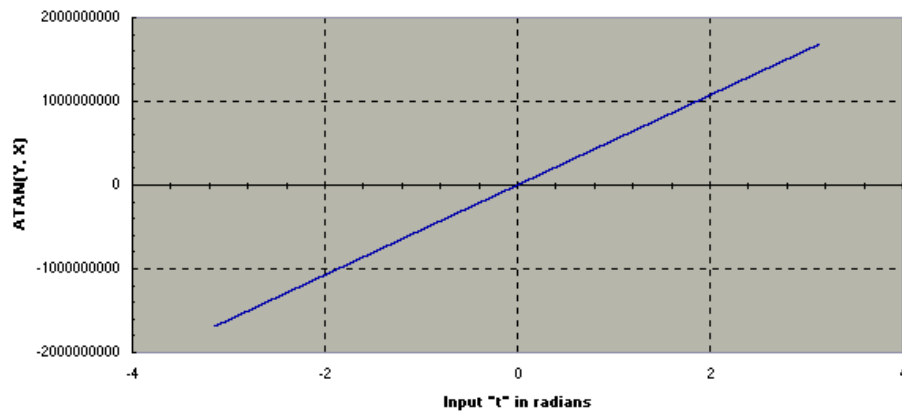
```
}
```

定点浮点 ATAN 函数对比 C 语言浮点 ATAN:

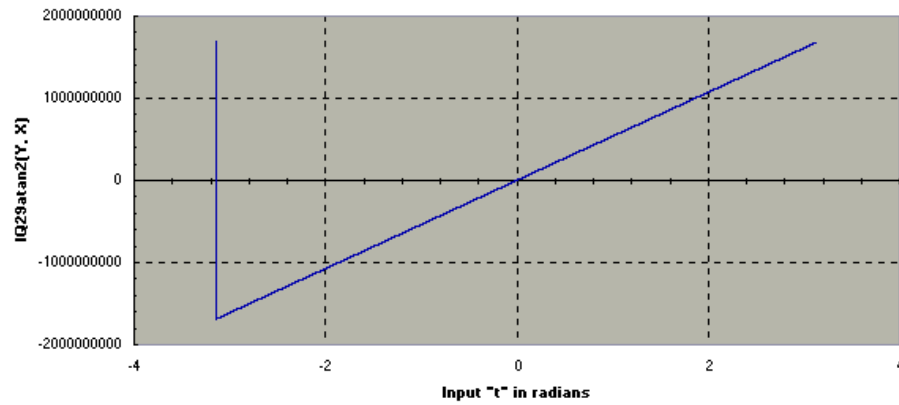
ATAN function input : Y=SIN(t) & X=COS(t) in Q29 format



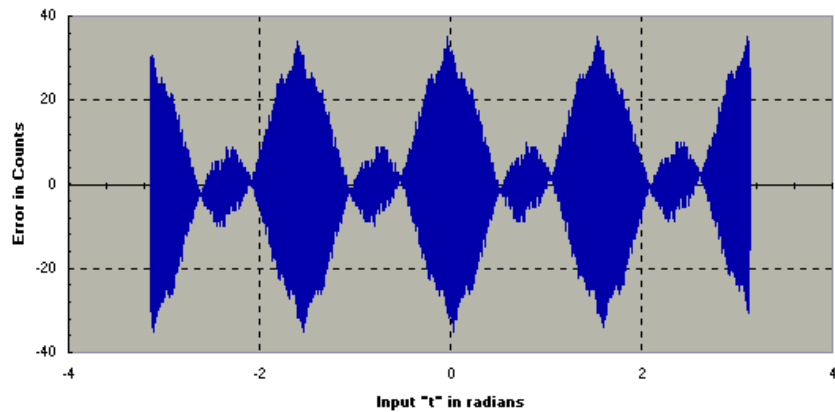
ATAN(Y, X)-Floating Point Computation (Q29)



IQ29atan2(Y, X)-Floating Point Computation(Q29)



Error=IQ29atan2(Y, X) - ATAN(Y, X)



### 3.3.8 定点浮点表达的反切值\_IQatan2PU \_IQNatan2PU

**描述** 计算四象限反向正切值，输出单位为弧度占比，范围  $0 \sim 2\pi$  对应【0~1】。

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

C        \_iq        \_IQatan2PU(\_iq A, \_iq B)

C++     iq        IQatan2PU(const iq &A, const iq &B)

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

C        \_iqN        \_IQNatan2PU(\_iqN A, \_iqN B)

C++     iqN        IQNatan2PU(const iqN &A, const iqN &B)

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式定点浮点数. A 为正弦值, B 为余弦值

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

IQN 格式定点浮点数. A 为正弦值, B 为余弦值

**输出** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式反向正切值. 单位为弧度占比, 即  $0 \sim 2\pi$  对应【0~1】

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

IQN 格式反向正切值. 单位为弧度占比, 即  $0 \sim 2\pi$  对应【0~1】

**精度**  $= 20\log_2(1 \times 2^{29}) - 20\log_2(6) = 27 \text{ bits}$

**举例**

获取  $\text{atan}(\sin(\pi/5), \cos(\pi/5)) = \pi/5$ , 假定 IQmath 头文件配置 GLOBAL\_Q = Q29.

```
#include "IQmathLib.h"
```

```
#define PI 3.14159265161
```

```
_iq xin1, yin1, out1;
```

```
_iq29 xin2, yin2, out2;
```

```
void main(void ) {
```

```
    // xin1 = xin2 = cos(PI/5) x 2^29 = 0x19E3779C
```

```
    // yin1 = yin2 = sin(PI/5) x 2^29 = 0x12CF2303
```

```
    // out1 = out2 = (PI/5)/(2PI) x 2^29 = 0x03333333
```

```
    xin1 = _IQcos(_IQ(PI/5.0L));
```

```
    yin1 = _IQsin(_IQ(PI/5.0L));
```

```
    out1 = _IQatan2PU(yin1, xin1);
```

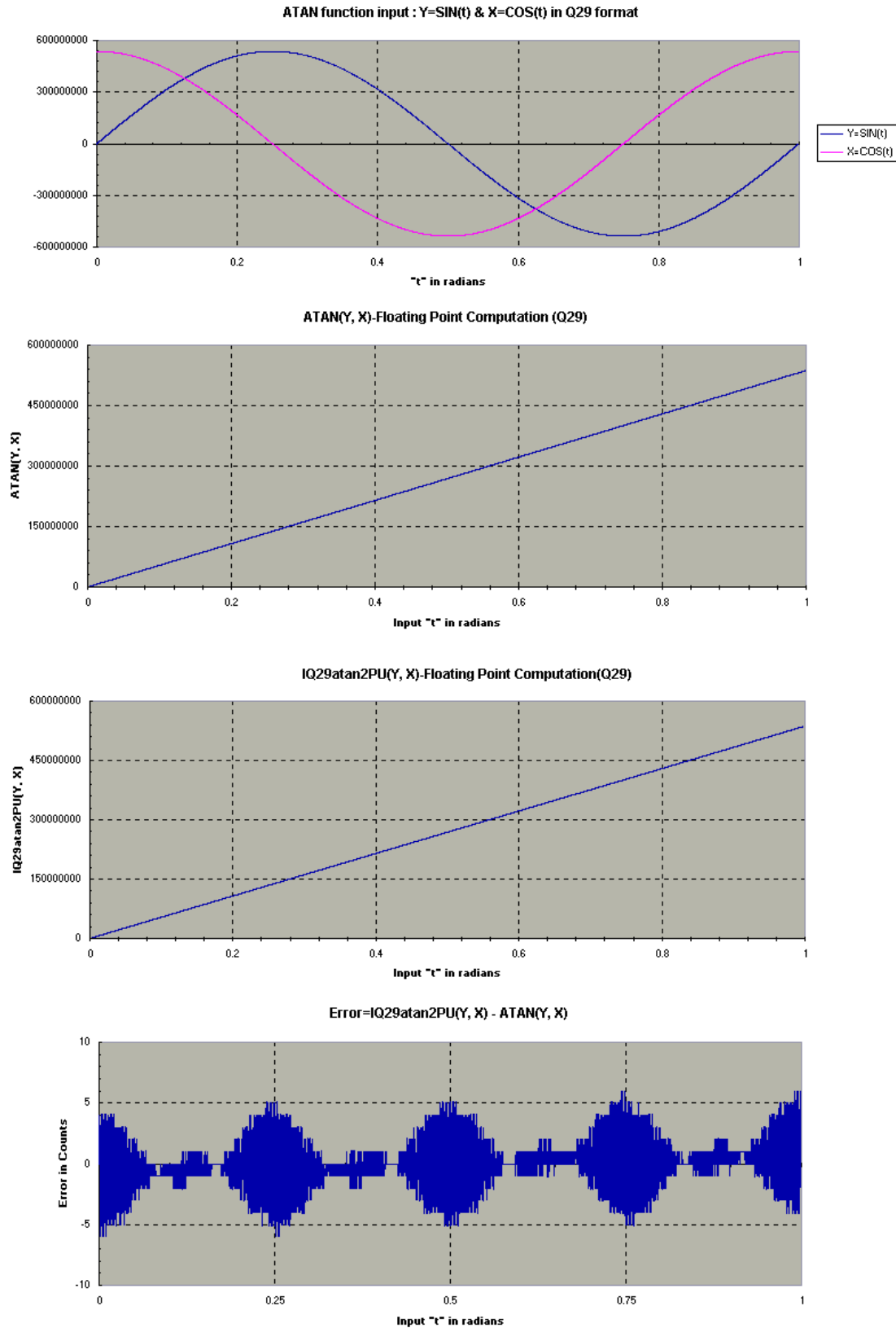
```
    xin2 = _IQ29cos(_IQ29(PI/5.0L));
```

```
    yin2 = _IQ29sin(_IQ29(PI/5.0L));
```

```
    out2 = _IQ29atan2PU(yin1, xin1);
```

```
}
```

定点浮点反向正切函数对比 C 语言浮点反向正切:





### 3.3.9 定点浮点表达的反切值\_IQatan\_IQNatan

**描述** 计算反向正切值. 输出为弧度, 范围为:  $(-\pi/2 \text{ 到 } \pi/2)$ 。该实现宏定义为:

```
#define _IQXXatan(A) _IQXXatan2(A, _IQXX(1.0))
```

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

```
C _iq _IQatan(_iq A)
```

```
C++ iq IQatan(const iq &A)
```

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

```
C _iqN _IQNatan(_iqN A)
```

```
C++ iqN IQNatan(const iqN &A)
```

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式定点浮点的输入正切值

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

IQN 格式定点浮点的输入正切值

**输出** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式反向正切值的定点浮点值. 范围见描述说明

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

IQN 格式反向正切值的定点浮点值. 范围见描述说明

$$= 20 \log_2 \left( \frac{\pi}{2} \times 2^{29} \right) - 20 \log_2 (2) = 25 \text{ bits}$$

**精度**

**举例**

获取  $\text{atan}(1) = \pi/4$ , 假定 IQmath 头文件配置 GLOBAL\_Q = Q29.

```
#include "IQmathLib.h"
```

```
_iq in1, out1;
```

```
_iq29 in2, out2;
```

```
void main(void )
```

```
{
```

```
    in1 = _IQ(1.0L);
```

```
    out1 = _IQatan(in1);
```

```
    in2 = _IQ29(1.0L);
```

```
    out2 = _IQ29atan(in2)
```

```
}
```

## 3.4 数学实现 exp log sqrt mag

### 3.4.1 计算 e 的定点浮点指数 IQexp IQNexp

**描述** 计算 e 的定点浮点数的指数.

声明	全局 IQ 函数(IQ 格式= GLOBAL_Q)
	C <code>_iq_IQexp(_iq A)</code>
	C++ <code>iq_IQexp(const iq &amp;A)</code>
	Q 格式函数 (IQ 格式= IQ1 ~IQ30)
	C <code>_iqN_IQNexp(_iqN A)</code>
	C++ <code>iqN_IQNexp(const iqN &amp;A)</code>
输入	全局 IQ 函数(IQ 格式= GLOBAL_Q)
	GLOBAL_Q 格式定点浮点输入参数
	Q 格式函数 (IQ 格式= IQ1 ~IQ30)
	IQN 格式定点浮点输入参数(N=1:30)
输出	全局 IQ 函数(IQ 格式= GLOBAL_Q)
	GLOBAL_Q 格式指数定点浮点值
	Q 格式函数 (IQ 格式= IQ1 ~IQ30)
	IQN 格式指数定点浮点值. (N=1:30)
举例	计算 $e(1.8) = 6.0496474$ , 假定 GLOBAL_Q 设定为 Q24 格式.
	<pre>#include "IQmathLib.h" _iq in1, out1; _iq24 in2, out2; void main(void ) {     // in1 = in2 = 1.8 x 2^24 = 0x01CCCCCC     // out1 = out2 = exp(1.8) x 2^24 = 0x060CB5AD     in1 = _IQ(1.8);     out1 = _IQexp(in1);     in2 = _IQ24(1.8);     out2 = _IQ24exp(in2); }</pre>

### 3.4.2 计算 2 的定点浮点指数 IQexp2 IQNexp2

描述 计算 2 的定点浮点数的指数. 类 exp, 但底数为 2.

声明	全局 IQ 函数(IQ 格式= GLOBAL_Q)
	C <code>_iq_IQexp2(_iq A)</code>
	C++ <code>iq_IQexp2(const iq &amp;A)</code>
	Q 格式函数 (IQ 格式= IQ1 ~IQ30)
	C <code>_iqN_IQNexp2(_iqN A)</code>
	C++ <code>iqN_IQNexp2(const iqN &amp;A)</code>
输入	全局 IQ 函数(IQ 格式= GLOBAL_Q)
	GLOBAL_Q 格式定点浮点输入参数
	Q 格式函数 (IQ 格式= IQ1 ~IQ30)
	IQN 格式定点浮点输入参数(N=1:30)

输出 全局 IQ 函数(IQ 格式= GLOBAL\_Q)  
GLOBAL\_Q 格式指数定点浮点值  
Q 格式函数 (IQ 格式= IQ1 ~IQ30)  
IQN 格式指数定点浮点值. (N=1:30)

### 3.4.3 计算底数 e 的定点浮点真数的对数\_IQlog \_IQNlog

描述 计算 IQ 格式定点浮点底数为 e 的对数，无理数  $e=2.718281828\cdots$ ，数学简写为  $\ln N$ 。

声明 全局 IQ 函数(IQ 格式= GLOBAL\_Q)  
C `_iq _IQlog(_iq A)`  
C++ `iq IQlog(const iq &A)`  
Q 格式函数 (IQ 格式= IQ1 ~IQ30)  
C `_iqN _IQNlog(_iqN A)`  
C++ `iqN IQNlog(const iqN &A)`

输入 IQ 格式定点浮点数

输出 IQ 格式输入数的 e 的对数结果 IQ 格式值

举例 计算输入值的对数  $\log(6.0496474) = 1.8$ ，如果结果超出 IQ 格式表达的范围，结果进行饱和处理，假定 IQmath 头文件配置 GLOBAL\_Q = Q24

```
#include "IQmathLib.h"
_iq in1, out1;
_iq30 in2, out2;
void main(void )
{
    // in1 = in2 = 6.0496474 x 2^24 = 0x060CB5B1
    // out1 = out2 = log(6.0496474) x 2^24 = 0x01CD053C, 实际1.80086112
    in1 = _IQ(6.0496474);
    out1 = _IQlog(in1);
    in2 = _IQ24(6.0496474);
    out2 = _IQ24log(in2);
}
```

### 3.4.4 定点浮点表达的平方根\_IQsqrt \_IQNsqrt

描述 使用查表与牛顿迭代近似法计算平方根。

声明 全局 IQ 函数(IQ 格式= GLOBAL\_Q)  
C `_iq _IQsqrt(_iq A)`  
C++ `iq IQsqrt(const iq &A)`  
Q 格式函数 (IQ 格式= IQ1 ~ IQ30)  
C `_iqN _IQNsqrt(_iqN A)`  
C++ `iqN IQNsqrt(const iqN &A)`

---

**输入**     全局 IQ 函数(IQ 格式= GLOBAL\_Q)  
             GLOBAL\_Q 格式定点浮点数  
             Q 格式函数 (IQ 格式= IQ1 ~ IQ30)  
             IQN 格式定点浮点数

**输出**     全局 IQ 函数(IQ 格式= GLOBAL\_Q)  
             输入的平方根值 GLOBAL\_Q 格式定点浮点值  
             Q 格式函数 (IQ 格式= IQ1 ~ IQ30)  
             输入的平方根值 IQN 格式定点浮点值

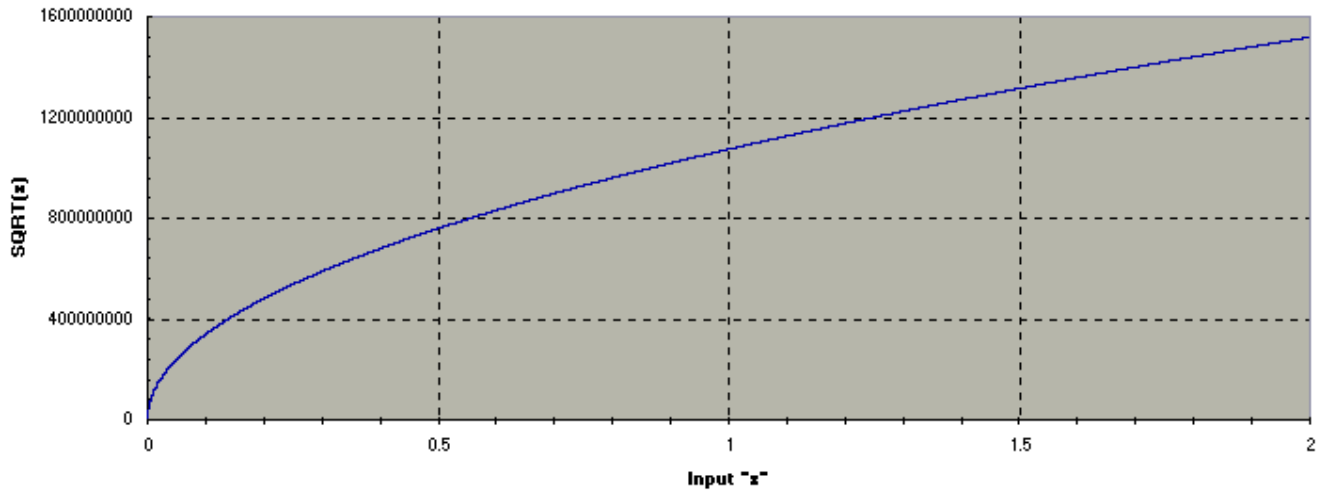
**精度**      $= 20\log_2(2^{31}) - 20\log_2(6) = 29 \text{ bits}$

**举例**     计算 $\sqrt{1.8} = 1.34164$ , 假定 IQmath 头文件配置 GLOBAL\_Q = Q30.

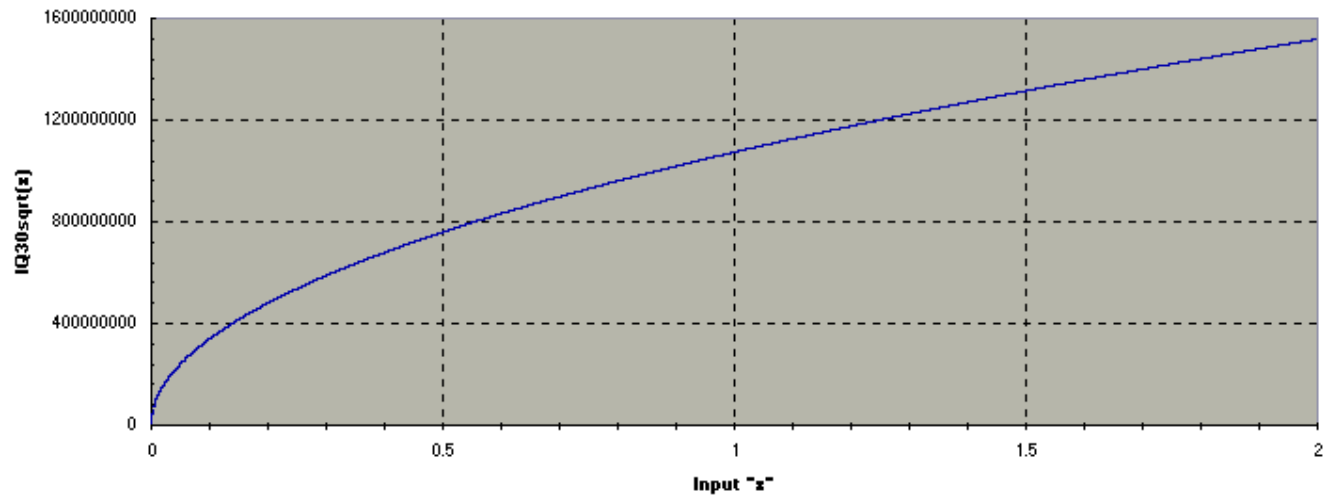
```
#include "IQmathLib.h"
_iq in1, out1;
_iq30 in2, out2;
void main(void )
{
    // in1 = in2 = 1.8 x 2^30 = 0x73333333
    // out1 = out2 = sqrt(1.8) x 2^30 = 0x55DD7151
    in1 = _IQ(1.8);
    out1 = _IQsqrt(in1);
    in2 = _IQ30(1.8);
    out2 = _IQ30sqrt(in2);
}
```

定点浮点 SQRT 函数 对比 C 语言浮点 SQRT:

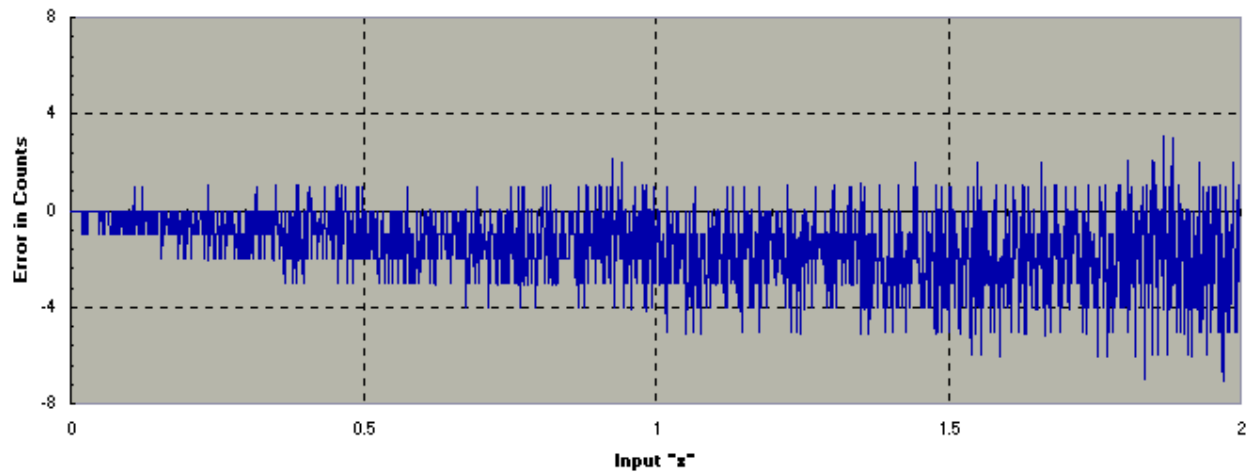
SQRT(x)-Floating Point Computation (Q30)



IQ30sqrt(x)-Floating Point Computation (Q30)



Error=IQ30sqrt(x)-SQRT(x)



### 3.4.5 定点浮点表达的平方根倒数\_IQisqrt\_IQNisqrt

**描述** 使用查表与牛顿迭代近似法计算平方根的倒数。

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

```
C      _iq      _IQisqrt(_iq A)
C++    iq      IQisqrt(const iq &A)
```

**Q 格式函数** (IQ 格式= IQ1 ~ IQ30)

```
C      _iqN      _IQNisqrt(_iqN A)
C++    iqN      IQNisqrt(const iqN &A)
```

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式定点浮点数

**Q 格式函数** (IQ 格式= IQ1 ~ IQ30)

IQN 格式定点浮点数

**输出** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

输入的平方根倒数 GLOBAL\_Q 格式定点浮点值

**Q 格式函数** (IQ 格式= IQ1 ~ IQ30)

输入的平方根倒数 IQN 格式定点浮点值

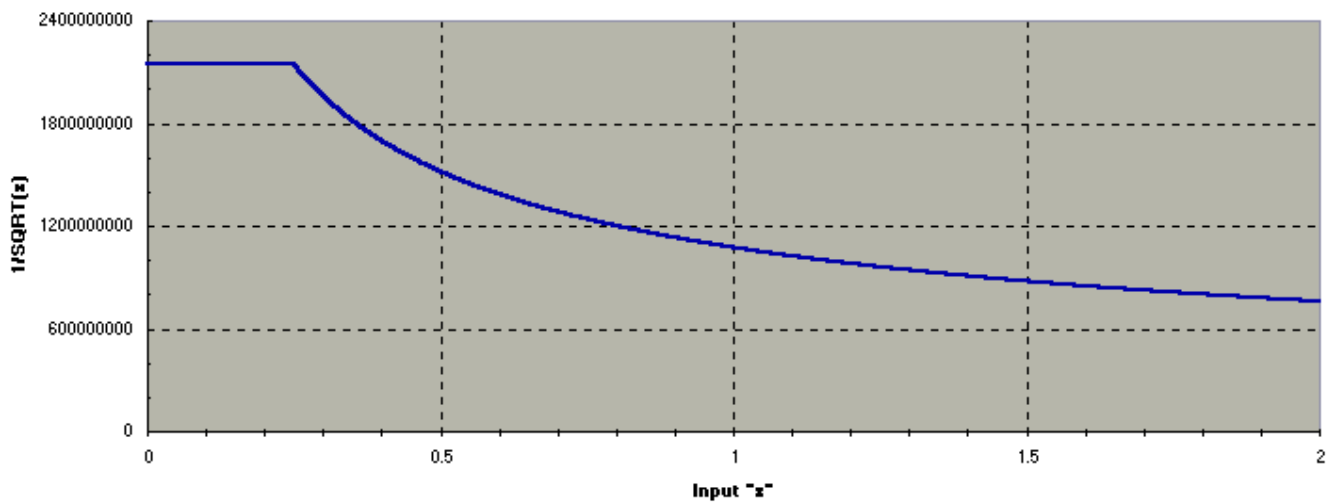
**精度**  $= 20\log_2(2^{31}) - 20\log_2(5) = 29 \text{ bits}$

**举例** 计算  $\frac{1}{\sqrt{1.8}} = 0.74535$ , 假定 IQmath 头文件配置 GLOBAL\_Q = Q30.

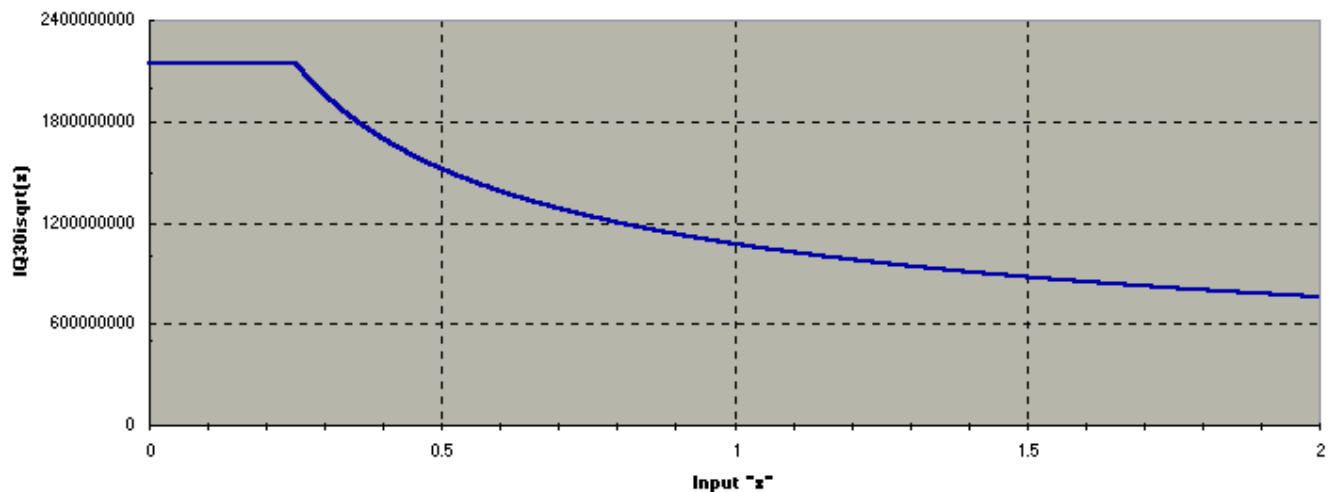
```
#include "IQmathLib.h"
_iq in1, out1;
_iq30 in2, out2;
void main(void )
{
    // in1 = in2 = 1.8 x 2^30 = 0x73333333
    // out1 = out2 = 1/sqrt(1.8) x 2^30 = 0x2FB3E99F
    in1 = _IQ(1.8);
    out1 = _IQisqrt(in1);
    in2 = _IQ30(1.8);
    out2 = _IQ30isqrt(in2);
}
```

定点浮点倒数 SQRT 函数对比 C 语言浮点倒数 SQRT:

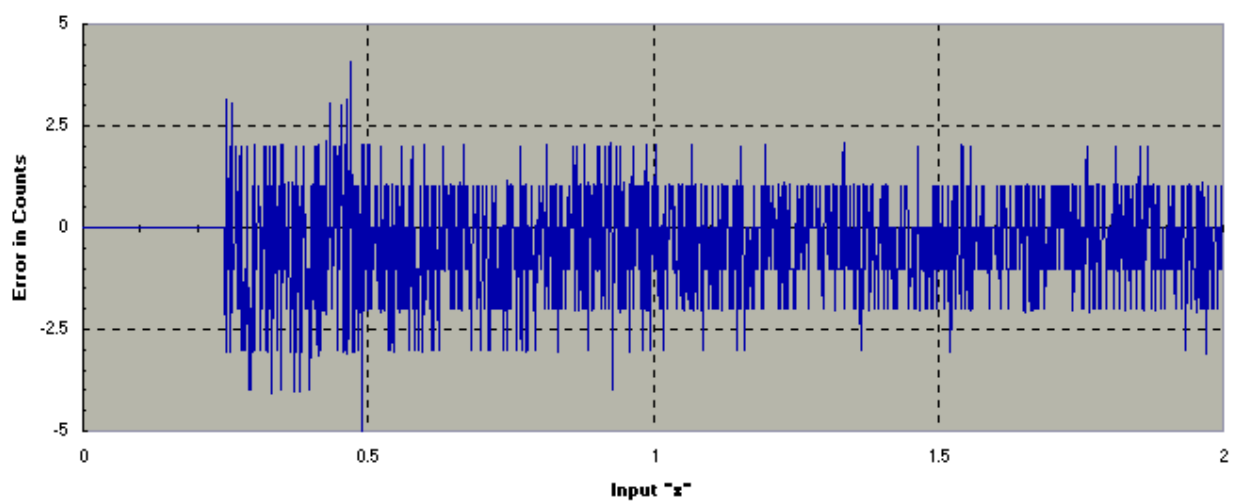
1/SQRT(x)-Floating Point Computation (Q30)



IQ30isqrt(x)-Fixed Point Computation (Q30)

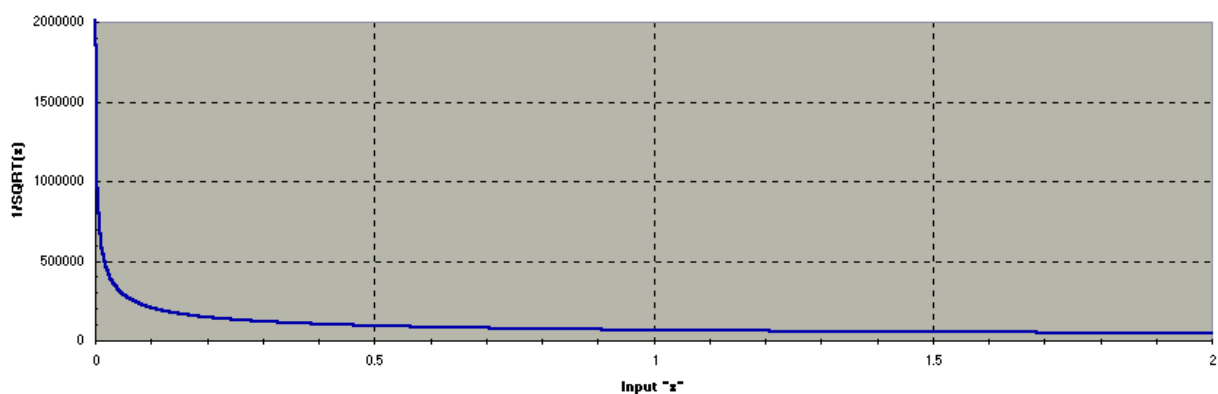


Error=IQ30isqrt(x)-1/SQRT(x)

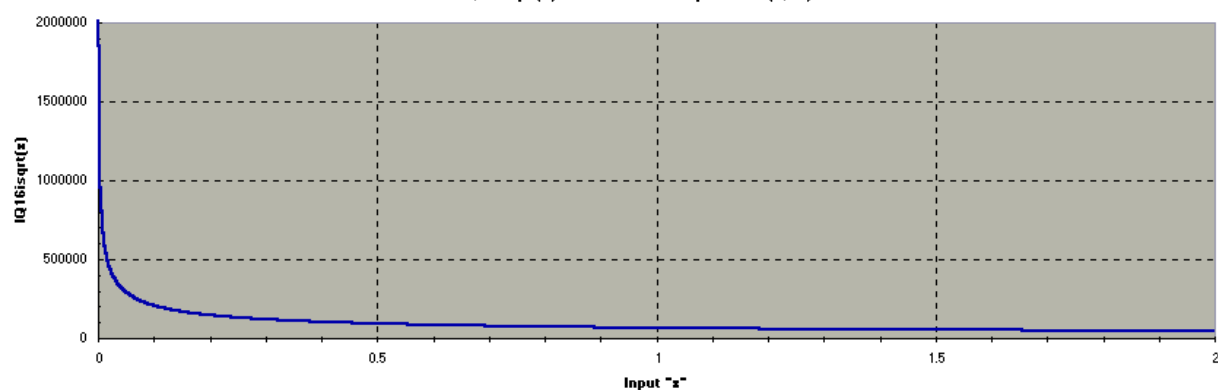


定点浮点倒数 SQRT 函数对比 C 语言浮点倒数 SQRT:

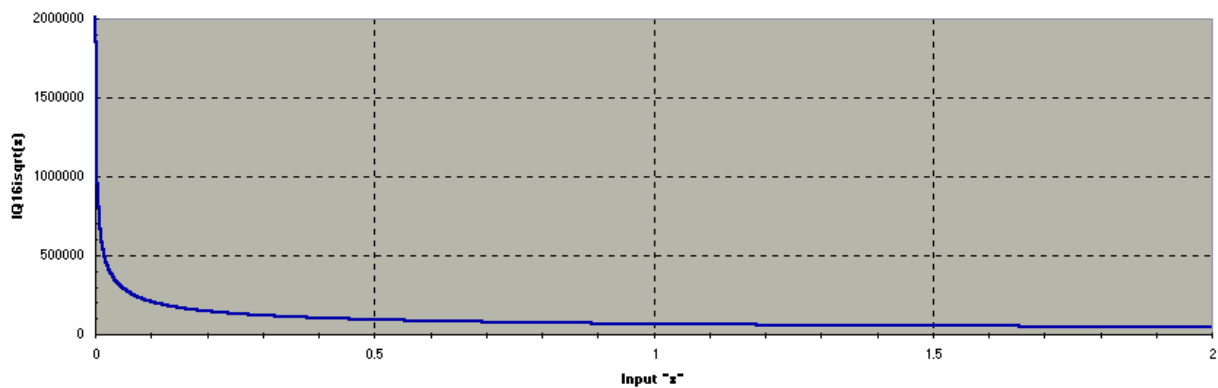
1/SQRT(x)-Floating Point Computation (Q16)



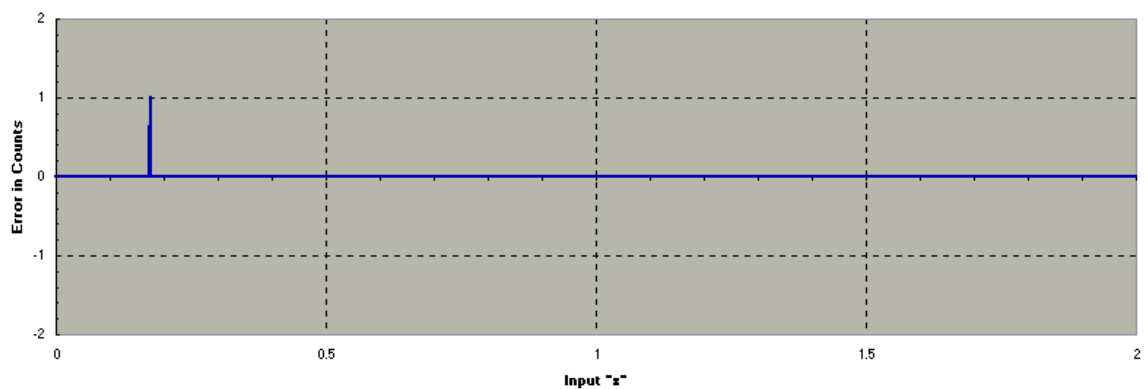
IQ16isqrt(x)-Fixed Point Computation (Q16)



IQ16isqrt(x)-Fixed Point Computation (Q16)



Error=IQ16isqrt(x)-1/SQRT(x)





### 3.4.6 定点浮点平方和的平方根 \_IQmag \_IQNmag

**描述** 计算实现形式：  $\text{Mag} = \sqrt{A^2 + B^2}$ 。与直接调用 “\_IQsqrt” 相比，该方法具有更高的精度，并避免使用\_IQsqrt 计算过程中的可能溢出。

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

C        \_iq        \_IQmag(\_iq A, \_iq B)

C++      iq        IQmag(const iq &A, const iq &B)

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

C        \_iqN        \_IQNmag(\_iqN A, \_iqN B)

C++      iqN        IQNmag(const iqN &A, const iqN &B)

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

参数 A 和 B 是 GLOBAL\_Q 格式定点浮点数

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

参数 A 和 B 是 IQN 格式定点浮点数

**输出** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

输入的平方和的平方根值 GLOBAL\_Q 格式定点浮点值

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

输入的平方和的平方根值 IQN 格式定点浮点值

**精度**  $= 20\log_2(2^{31}) - 20\log_2(6) = 29 \text{ bits}$

**举例：**

计算复数用平方和的平方差，假定 IQmath 头文件配置 GLOBAL\_Q = Q28

```
#include "IQmathLib.h"

// Complex number = real1 + j*imag1
// Complex number = real2 + j*imag2
_iq real1, imag1, mag1;
_iq28 real2, imag2, mag2;
void main(void )
{
    // mag1 = 5.65685415 in IQ28 format 0x5A82799A
    real1 = _IQ(4.0);
    imag1 = _IQ(4.0);
    mag1 = _IQmag(real1, imag1);
    // mag2 =~8.0, saturated to MAX value (IQ28) 0x7FFFFFFF, 即 7.99999952!!!
    real2 = _IQ28(7.0);
    imag2 = _IQ28(7.0);
    mag2 = _IQ28mag(real2, imag2);
}
```

## 3.5 其他实现

### 3.5.1 定点浮点数绝对值\_IQabs\_IQNabs

**描述** 该模块计算定点浮点数的绝对值. 该方法宏定义在 Iqmathlib 头文件中. 即定点浮点下  $((A) < 0) ? -(A) : (A)$ , 绝对浮点下调用浮点绝对值方法 fabs(A)

**声明** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

C        \_iq        \_IQabs(\_iq A)

C++     iq        IQabs(const iq &A)

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

C        \_iqN        \_IQNabs(\_iqN A)

C++     iqN        IQNabs(const iqN &A)

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式定点浮点值

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

IQN 格式定点浮点数值

**输出** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

GLOBAL\_Q 格式定点浮点输入的 GLOBAL\_Q 绝格式对值

**Q 格式函数 (IQ 格式= IQ1 ~ IQ30)**

IQN 格式定点浮点输出的 IQN 格式绝对值

**举例:**

假定 Iqmath 头文件下 GLOBAL\_Q=IQ28 下计算 3 个定点浮点的绝对值加和。

```
#include "IQmathLib.h"
```

```
void main(void)
```

```
{
```

```
  _iq xin1, xin2, xin3, xsum;
```

```
  _iq20 yin1, yin2, yin3, ysum;
```

```
  xsum = _IQabs(X0) + _IQabs(X1) + _IQabs(X2);
```

```
  xsum = _IQ28abs(X0) + _IQ28abs(X1) + _IQ28abs(X2);
```

```
}
```

### 3.5.2 饱和处理定点浮点值到给定的正与负值限定 \_IQsat

**描述** 该模块饱和给定定点浮点值到指定的正最大值和负最小值, 在计算过程可能异常时该方法使用。 该方法宏定义在 Iqmathlib 头文件中. 即

```
#defien _IQsat(A, Pos, Neg)    (((A) > (Pos)) ? (Pos) : (((A) < (Neg)) ? (Neg) : (A)))
```

**声明**    \_iq        \_IQsat(\_iq A, long P, long N)

**输入** 全局 IQ 函数(IQ 格式= GLOBAL\_Q)

全局格式定点浮点数，正限定值，负限定值。

输出 GLOBAL\_Q 格式饱和结果输出

举例：

使用饱和计算“ $Y = M * X + B$ ”接近结果。所有变量是全局格式 GLOBAL\_Q = 26. 因为变量范围可能溢出，必须执行计算与饱和处理结果。为实现此目的，使用 IQ=20 的中间处理，饱和和转换到合适的 GLOBAL\_Q 定点浮点值。

```
#include "IQmathLib.h"

void main(void)
{
    _iq Y, M, X, B; // GLOBAL_Q = 26 (+/- 32 range)
    _iq20 temp; // IQ = 20 (+/- 2048 range)
    temp = _IQ20mpy(_IQtoIQ20(M), _IQtoIQ20(X)) + _IQtoIQ20(B);
    temp = _IQsat(temp, _IQtoIQ20(MAX_IQ_POS), _IQtoIQ20(MAX_IQ_NEG));
    Y = _IQ20toIQ(temp);
}
```

注：这里 MAX\_IQ\_POS 对应 0x7FFFFFFF 的 2147483647 和 MAX\_IQ\_NEG 对应 0x80000000 的 -2147483647。通过 \_IQtoIQ20 实现到 [-2048: 2047.999 999 046]。

## 4. 历史记录

版本	时间	内容
V1.0	2020-7-30	初版实现 Iqmath 库