

简介

1.1 简介

德州仪器公司TMS320C28X系列的IQmath库为C/C++程序员收集了高度优化和准确的数学函数库并精确地在 TMS320C28X 芯片上将浮点算法转换成固定点算法的运算代码。这些函数被经常用在密集的实时计算且运行速度和精度又是至关重要的程序里。使用这些函数你能拥有比使用同等标准的ANSI C语言编写的代码更快的运行速度。此外提供可以使用的高精密功能的TI IQmath 库能明显的缩短你 DSP 应用开发的时间。

安装IQMath

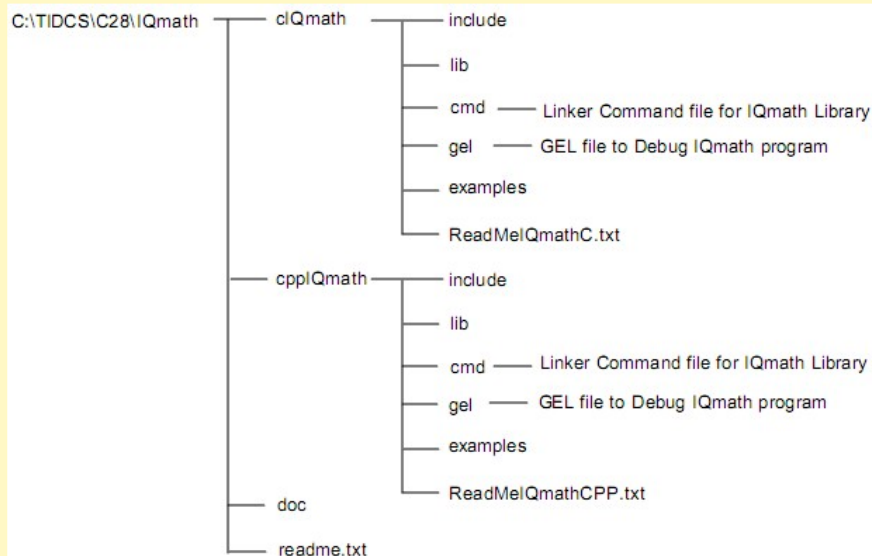
2.1 IQMath的内容

TI IQmath 库在 C/CPP程序中提供使用语法和它的5个组成部分。

- 1) IQmath标题文件IQmathLib.h
- 2) IQmath对象库包含所有的功能与搜寻表文件 IQmath.lib
- 3) 连接指令文件IQmath.cmd
- 4) IQmath调试文件GEL IQmath.gel
- 5) 程序实例

2.2 如何安装IQMath库

IQmath构建在一个以ZIP形式的自解压文件中，该ZIP文件会自动释放IQmath库， 自动的组建一个如下的索引结构，阅读README.TXT可以了解到释放的更多详情。



3.1 IQMath参数与数据类型

IQmath 输入/输出系统是典型的32位定点数系统，定点数的Q可在 Q1到Q30之间定义，我们也可以使用自定义的 IQ数据类型名，这更方便了开发者在应用程序中定义各种类型的IQmath数据类型。

```
typedef long _iq; /*fixed piont data type: GLAOBAL_Q format*/
typedef long _iq30; /*fixed piont data type: Q30 format */
typedef long _iq29; /*fixed piont data type: Q29 format */
typedef long _iq28; /*fixed piont data type: Q28 format */
typedef long _iq27; /*fixed piont data type: Q27 format */
typedef long _iq26; /*fixed piont data type: Q26 format */
typedef long _iq25; /*fixed piont data type: Q25 format */
typedef long _iq24; /*fixed piont data type: Q24 format */
typedef long _iq23; /*fixed piont data type: Q23 format */
typedef long _iq22; /*fixed piont data type: Q22 format */
typedef long _iq21; /*fixed piont data type: Q21 format */
typedef long _iq20; /*fixed piont data type: Q20 format */
typedef long _iq19; /*fixed piont data type: Q19 format */
typedef long _iq18; /*fixed piont data type: Q18 format */
typedef long _iq17; /*fixed piont data type: Q17 format */
typedef long _iq16; /*fixed piont data type: Q16 format */
typedef long _iq15; /*fixed piont data type: Q15 format */
typedef long _iq14; /*fixed piont data type: Q14 format */
typedef long _iq13; /*fixed piont data type: Q13 format */
typedef long _iq12; /*fixed piont data type: Q12 format */
typedef long _iq11; /*fixed piont data type: Q11 format */
typedef long _iq10; /*fixed piont data type: Q10 format */
typedef long _iq9; /*fixed piont data type: Q9 format */
typedef long _iq8; /*fixed piont data type: Q8 format */
typedef long _iq7; /*fixed piont data type: Q7 format */
typedef long _iq6; /*fixed piont data type: Q6 format */
```

3.2 IQmath数据类型：Range和Resolution

下表总结了不同Q格式的32位定点数的Range & Resolution。典型的Q函数支持Q1到Q30格式，但是对于IQNsin, IQNcos, IQNatan2, IQNatan2PU, Iqatan等不支持Q30格式，因为这些函数的输入和输出需要在 $-\pi$ 到 π 范围中改变。

| Data Type | Range | | Resolution/Precision |
|-----------|-------------|-------------------------|----------------------|
| | Min | Max | |
| 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 |

3.3 在C中调用一个IQMath函数

除了安装IQmath软件，在调用IQmath函数时还应：

- * 在文件中包含IQmath.h文件
- * 将你的代码与IQmath.h连接
- * 在程序存储器中用正确的CMD文件放置IQmath代码段
- * IQmath 表包含IQmath函数查询表，其在F2810/F2812的BOOTROM中。因此，该段在CMD文件中必须设置为NoLoad类型。这样可以不用将该段载入目标板中，而方便了用符号来参考查询表。

注意：

IQmath函数在IQmath段中，用于高精度计算的查询表放置在IQmath Tables段中。

IQmath Linker Command File (F28x device)

```
MEMORY
{
    PAGE 0:
    BOOTROM (RW) : origin = 0x3ff000, length = 0x000fc0
    RAMH0 (RW)   : origin = 0x3f8000, length = 0x002000
}

SECTIONS
{
    IQmathTables : load = BOOTROM, type = NOLOAD, PAGE = 0
    IQmath       : load = RAMH0, PAGE = 0
}
```

例如，下面的代码包含在IQmath.lib中，调用IQ25sin。

```
<!--
#include<IQmathLib.h> /* Header file for IQmath routine */
#define PI 3.14159 ;
_iq input, sin_out;
void main(void )
{
    input=_IQ29(0.25*PI); /* 0.25 ×π radians represented in Q29 format
    sin_out = _IQ29sin(input);
}
```

3.4 IQMath函数命名规则

每一个IQmath函数都包括，两种函数类型的handles,即

1. GLOBAL_Q 函数，在这种格式中用到输入 和输出
例如：

- _IQsin(A) /* High Precision SIN */
- _IQcos(A) /* High Precision COS */
- _IQrmppy(A,B) /* IQ multiply with rounding */

2.Q-格式特殊函数主要为了应用Q1到Q30数据格式

例如：

- _IQ29sin(A) /* High Precision SIN: input/output are in Q29
- _IQ28sin(A) /* High Precision SIN: input/output are in Q28
- _IQ27sin(A) /* High Precision SIN: input/output are in Q27
- _IQ26sin(A) /* High Precision SIN: input/output are in Q26
- _IQ25sin(A) /* High Precision SIN: input/output are in Q25
- _IQ24sin(A) /* High Precision SIN: input/output are in Q24

IQmath Function Naming Convention

GLOBAL_Q Function

_IQxxx(), Where “xxx” is the Function Name

Q Specific Function

_IQNxxx(), Where “xxx” is the Function Name &
“N” is the Q format of input/output

3.5选择GLOBAL_Q格式

数字精度和动态量程需要在不同的应用中会有很大的改变。IQmath库方便了在定点算法中的程序的应用，不用事先确定数字精度。这个允许系统工程师用不同的数字精度检查应用效果并最终确定数字的定点位置。如上3.2解释，高精度导致低的动态量程。因此，系统设计师在选用GLOBAL_Q格式之前必须平衡rang和resolution的关系。

例1：

默认GLOBAL_Q格式被设置位Q24。编辑IQmathlib.h头文件，修改该定点位置，可以选择Q1到Q29格式作为GLOBAL_Q格式。注意：修改该定点位置意味着所有的GLOBAL_Q函数将应用这种Q格式进行输入和输出，除非这种符号性质的定义在源代码中有更高级的声明。

IQmathLib.h : Selecting GLOBAL_Q format

```
#ifndef GLOBAL_Q
#define GLOBAL_Q 24 /* Q1 to Q29 */
#endif
```

3.6 在调试过程使用IQMath GEL 文件

IQmath GEL 文件包含GEL 功能，这些功能用于帮助在观察窗口中查看IQ 变量值的。而且IQ变量值也可以通过对话框进行设置。

步骤1：定义全局IQ变量” Global_Q”

在用户源文件夹一个文件里，必须定义以下全局变量：

```
long GlobalQ=GLOBAL_Q;
```

GEL 函数利用这个变量来决定当前全局IQ变量(GLOBAL_Q)的设置。

步骤2：加载GEL文件

加载GEL文件 “IQmath.gel” 到用户工程文件中。它将会自动加载一系列的GEL 函数使得可以在观察窗口中看到IQ变量值并且在GEL工具条中产生以下目录：

->IQ C支持

->IQ C++支持

步骤3：查看IQmath 变量

为了在观察窗口中查看一个IQ变量，你可以在观察窗口中简单地输入下面的命令，将会把IQ变量值转换为等价的浮点型数值。

对于C 变量而言是以以下形式出现的：

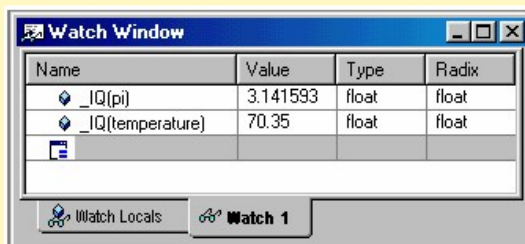
_IQ(变量名) ; 全局变量设置的IQ值（这里是N中的一个固定值）

_IQN(变量名) ; N=1 to 30 的IQ变量值（N是变量）

对于C 变量而言是以以下形式出现的：

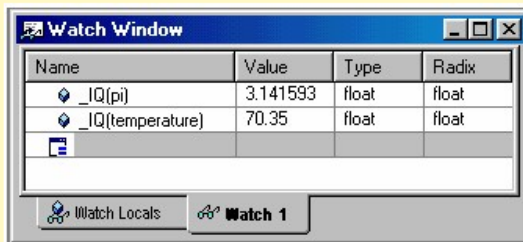
_IQ(变量名) ; 全局变量设置的IQ值（这里是N中的一个固定值）

_IQN(变量名) ; N=1 to 30 的IQ变量值（N是变量）



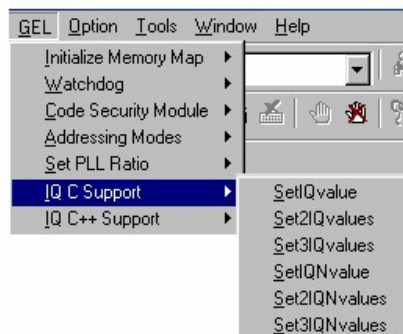
步骤4: 修改IQmath 变量

观察窗口不能对非本地类型的变量进行修改, 为了使能这个, 在GEL 工具栏下的子目录栏可以找到对对应变量使能转换的支持, 如下所示:



IQ C Support

- SetIQvalue ; GLOBAL_Q format
- Set2IQvalues
- Set3IQvalues
- SetIQNvalue ; IQN format
- Set2IQNvalues
- Set3IQNvalues



激活以上任何一个子目录将会产生一个对话框窗口, 用户可以在这个窗口中输入变量名和要设置的浮点数值。这个函数将会把浮点值自动的转化为相对应的IQ值了。

IQMath函数概述

IQmath库函数中包含如下几部分程序:

- 格式转换函数 : atoiQ, IQtoF, IQtoIQN 等。
- 算术函数 : IQmpy, IQdiv 等。
- 三角函数 : IQsin, IQcos, IQatan2 等。
- 数学函数 : IQsqrt, IQisqrt 等。
- 其它 : IQabs, IQsat 等。

各小节内容

4.1 常用的变量和字符

4.2 IQMath函数

4.1 常用的变量和字符

在各函数的描述说明中采用了如下一些变量或符号：

| | |
|------|---|
| QN | 16位定点Q格式数，其中N=1:15 |
| IQN | 32位定点Q格式数，其中N=1:31 |
| int | 16位数 |
| long | 32位数 |
| _iq | 数据类型定义等同与long，是一个用来表示全局q格式的32位数。为了在将来器件使用中提供便利，建议使用_iq而不使用long。 |
| _iqN | 数据类型定义等同与long，IQN是一个32位数，其中N=1:30 |
| A,B | IQmath函数或宏的输入操作数 |
| F | 浮点数输入:比如:-1.232,+22.433,0.4343,-0.32 |
| S | 浮点字符串:" +1.32" ," 0.232" ," -2.343" 等 |
| P | 正数最大值 |
| N | 负数最大值 |

IQMath函数

格式转换函数：

| 函数 | 说明 | IQ格式 |
|------------------------|------------------|------------|
| _iq _IQ(float F) | 浮点数转化为IQN数据类型 | Q=GLOBAL_Q |
| _iqN _IQN(float F) | | Q=1:30 |
| float _IQtoF(_iq A) | 转化IQN数据格式为浮点数 | Q=GLOBAL_Q |
| float _IQNtoF(_iqN A) | | Q=1:30 |
| _iq _atoiQ(char *S) | 转化字符串为IQN | Q=GLOBAL_Q |
| _iqN _atoiQN(char *S) | | Q=1:30 |
| long _IQint(_iq A) | 提取IQ值的整数部分 | Q=GLOBAL_Q |
| long _IQNint(_iqN A) | | Q=1:30 |
| _iq _IQfrac(_iq A) | 提取IQ值的小数部分 | Q=GLOBAL_Q |
| _iqN _IQNfrac(_iqN A) | | Q=1:30 |
| _iqN _IQtoIQN(_iq A) | 将IQ值转换为IQN值（32位） | Q=GLOBAL_Q |
| _iq _IQNtoIQ(_iqN A) | 将IQN值（32位）转换为IQ值 | Q=GLOBAL_Q |
| int _IQtoQN(_iq A) | 将IQ值转换为QN值（16位） | Q=GLOBAL_Q |
| _iq _QNtoIQ(int A) | 将QN值（16位）转换为IQ值 | Q=GLOBAL_Q |

算术函数

| 函数 | 说明 | IQ格式 |
|---|-----------------------------|----------------------|
| _iq _IQmpy(_iq A, _iq B) _iqN _IQNmpy(_iqN A, _iqN B) | Q乘法 (IQN*IQN) | Q=GLOBAL_Q Q=1:30 |
| _iq _IQrmpy(_iq A, _iq B) _iqN _IQNrmpy(_iqN A, _iqN B) | 进行四舍五入的IQ乘法 (IQN*IQN) | Q=GLOBAL_Q Q=1:30 |
| _iq _IQrsmPy(_iq A, _iq B) _iqN _IQNrsmPy(_iqN A, _iqN B) | 带四舍五入带饱和处理的IQ乘法(IQN *IQN) | Q=GLOBAL_Q Q=1:30 |
| _iq _IQmpyI32(_iq A, long B) _iqN _IQNmpyI32(_iqN A, long B) | IQ格式与长整型相乘 (IQN*LONG) | Q=GLOBAL_Q Q=1:30 |
| long _IQmpyI32int(_iq A, long B) long _IQNmpyI32int(_iqN A, long B) | 长整型与IQ格式数乘法, 返回整数部分 | Q=GLOBAL_Q Q=1:30 |
| long _IQmpyI32frac(_iq A, long B) long _IQNmpyI32frac(_iqN A, long B) | 长整型与IQ格式数乘法, 返回小数部分 | Q=GLOBAL_Q Q=1:30 |
| _iq _IQmpyIQX(_iqN1 A, N1, _iqN2 B, N2) _iqN _IQmpyIQX(_iqN1 A, N1, _iqN2 B, N2) | Q格式不同的两个数相乘 | Q=GLOBAL_Q Q=1:30 |
| _iq _IQdiv(_iq A, _iq B) _iqN _IQNdiv(_iqN A, _iqN B) | 定点除法 | Q=GLOBAL_Q Q=1:30 |

三角函数

| 函数 | 说明 | IQ格式 |
|---|---------------------------|----------------------|
| _iq _IQsin(_iq A) _iqN _IQNsin(_iqN A) | 定点正弦函数(输入单位:弧度) | Q=GLOBAL_Q Q=1:29 |
| _iq _IQsinPU(_iq A) _iqN _IQNsinPU(_iqN A) | 定点正弦函数 (输入:标么值) | Q=GLOBAL_Q Q=1:29 |
| _iq _IQcos(_iq A) _iqN _IQNcos(_iqN A) | 定点余弦函数 (输入单位:弧度) | Q=GLOBAL_Q Q=1:29 |
| _iq _IQcosPU(_iq A) _iqN _IQNcosPU(_iqN A) | 定点余弦函数 (输入:标么值) | Q=GLOBAL_Q Q=1:29 |
| _iq _IQatan2(_iq A, _iq B) _iqN _IQNatan2(_iqN A, _iqN B) | 第四象限定点反正切函数atan (输入单位:弧度) | Q=GLOBAL_Q Q=1:29 |
| _iq _IQatan2PU(_iq A, _iq B) _iqN _IQNatanPU(_iqN A, _iqN B) | 第四象限定点反正切函数atan(输入:标么值) | Q=GLOBAL_Q Q=1:29 |
| _iq _IQatan(_iq A, _iq B) _iqN _IQNatan(_iqN A, _iqN B) | 定点反正切函数atan (输入:标么值) | Q=GLOBAL_Q Q=1:29 |

| 数学函数 | | |
|-------------------------------|--------------------|------------|
| 函数 | 说明 | IQ格式 |
| _iq _IQsqrt(_iq A) | 定点平方根函数 | Q=GLOBAL_Q |
| _iqN _IQNsqrt(_iqN A) | | Q=1:30 |
| _iq _IQisqrt(_iq A) | 定点开方根的倒数 | Q=GLOBAL_Q |
| _iqN _IQNisqrt(_iqN A) | | Q=1:30 |
| _iq _IQmag(_iq A, _iq B) | 求模运算:sqrt(A^2+B^2) | Q=GLOBAL_Q |
| _iqN _IQNmag(_iqN A, _iqN B) | | Q=1:30 |

| 其它函数 | | |
|------------------------------------|-----------|------------|
| 函数 | 说明 | Q格式 |
| _iq _IQsat(_iq A, long P, long N) | IQ数值的限幅函数 | Q=GLOBAL_Q |
| _iq _IQabs(_iq A) | IQ数据的绝对值 | Q=GLOBAL_Q |

函数描述

5.1格式转换函数

| |
|--------------------------------|
| 5.1.1 IQN: 浮点数转化为IQN数据类型 |
| 5.1.2 IQNtoF: 转化IQN数据格式为浮点数 |
| 5.1.3 atoiIQN: 转化字符串为IQN |
| 5.1.4 IQNint: IQN数值的整数部分 |
| 5.1.5 IQNfrac: IQN数值的小数部分 |
| 5.1.6 IQtoIQN: 全局IQ格式转换为特定IQ格式 |
| 5.1.7 IQNtoIQ: 特定IQ格式转换为全局IQ格式 |
| 5.1.8 IQtoQN: 全局IQ格式转换为QN格式 |
| 5.1.9 QNtoIQ: QN格式转换为全局IQ格式 |

5.2算术函数

| |
|--|
| 5.2.1 IQNmpy: IQ乘法 (IQN*IQN) |
| 5.2.2 IQNrmppy: 进行四舍五入的IQ乘法 (IQN*IQN) |
| 5.2.3 IQNrsmppy: 带四舍五入带饱和处理的IQ乘法(IQN *IQN) |
| 5.2.4 IQNmpy32: IQ格式与长整型相乘 (IQN*LONG) |
| 5.2.5 IQNmpy32int: IQN与LONG相乘结果的整数部分 |
| 5.2.6 IQNmpy32frac: IQN与LONG相乘结果的小数部分 |
| 5.2.7 IQNmpyIQX: 乘法 (GLOBAL_Q=IQN1*IQN2) |
| 5.2.8 IQNdiv: 定点除法 |

5.3 三角函数

| |
|--|
| 5.3.1 IQNsin:定点正弦函数(输入单位:弧度) |
| 5.3.2 IQNsinPU:定点正弦函数 (输入:标么值) |
| 5.3.3 IQNcos:定点余弦函数 (输入单位:弧度) |
| 5.3.4 IQNcosPU:定点余弦函数 (输入:标么值) |
| 5.3.5 IQNatan2:第四象限定点反正切函数atan (输入单位:弧度) |
| 5.3.6 IQNatan2PU:第四象限定点反正切函数atan(输入:标么值) |
| 5.3.7 IQNatan:定点反正切函数atan (输入:标么值) |

5.4 数学函数

| |
|-------------------------|
| 5.4.1 IQNsqrt:定点平方根函数 |
| 5.4.2 IQNisqrt:定点开方根的倒数 |
| 5.4.3 IQNmag:IQ复数的幅值函数 |

5.5 其他函数

| |
|-------------------------|
| 5.5.1 IQNabs:IQ数值的绝对值函数 |
| 5.5.2 IQNsat:IQ数值的限幅函数 |

5.1.1 IQN：浮点数转化为IQ数据类型

| | |
|----|---|
| 描述 | 该C宏把浮点数常量或变量转化为相应的IQ值 |
| 声明 | 全局IQ函数 (IQ 格式=GLOBAL_Q) _iq_IQ(float F) |
| 输入 | 特定Q格式IQ函数 (IQ 格式=IQ1 到 IQ29) _IQN_IQN(float F) |
| 输出 | 浮点数常量或变量 全局IQ函数 (IQ 格式=GLOBAL_Q) GLOBAL_Q格式所对应的定点数 |
| 用法 | 特定Q格式IQ函数 (IQ 格式=IQ1 到 IQ29) 浮点数输入的IQN格式所对应的定点数 这个操作通常用来把浮点数常量或变量转化为相应的IQ值。 |



例1： 用IQmath 方法执行方程
浮点数方程： $Y = M \times 1.26 + 2.345$
IQmath方程(类型1)： $Y = _IQmpy(M, _IQ(1.26)) + _IQ(2.345)$
IQmath方程(类型2)： $Y = _IQ23mpy(M, _IQ23(1.26)) + _IQ23(2.345)$



例2: 把浮点数常量或变量转化为IQ数据类型

```
float x=3.343;
 IQ y1;
 IQ23 y2;
 IQmath (类型1): y1=_IQ(x)
 IQmath (类型2): y2=_IQ(x)
```



例3: 初始化全局变量或表

```
IQmath (类型1):
 IQ Array[4] = {_IQ(1.0),_IQ(2.5),_IQ(-0.2345),_IQ(0.0) }
 IQmath (类型2):
 IQ23 Array[4] = {_IQ23(1.0),_IQ23(2.5),_IQ23(-0.2345),_IQ23(0.0) }
```

5.1.2 IQNtoF:转化IQ数据格式为浮点数

| | |
|----|---|
| 描述 | 这个函数把一个IQ数转化为响应的IEEE 754格式浮点数 |
| 声明 | 全局IQ函数 (IQ 格式= GLOBAL_Q) float _IQtoF(_iq A) 特定Q格式IQ函数 (IQ 格式= IQ1 到 IQ30) float _IqNtoF(_iqN A) |
| 输入 | 全局IQ函数(IQ 格式=GLOBAL_Q) GLOBAL_Q格式的定点IQ数 特定Q格式IQ函数(IQ 格式=IQ1 到 IQ30) IQN格式的定点IQ数 |
| 输出 | 定点数输入相应的浮点数 |
| 用法 | 这个操作通常用在用户希望使用浮点数格式的操作或者把数据转化回浮点数来显示的情况下 |



例:

把IQ数的数组转化为相应的浮点数值

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

5.1.3 atoiQN:转化字符串为IQN

| | |
|------|--|
| 描述 | 这个函数把字符串转化为IQ数 |
| 声明 | 全局IQ函数（IQ 格式= GLOBAL_Q） <code>float _atoiQF(char *S)</code> 特定Q格式IQ函数（IQ 格式= IQ1 到 IQ30） <code>float __atoiQN(char *S)</code> |
| 输入 | 这个函数识别任意符号，和一个包含任意基数字母的数字字符串 |
| 输出数， | 有效输入字符串： “ 12.23456”，“-12.23456”，“0.23456”，“0.0”，“0”，“-89”， 字符串以第一个为识别的字母结尾并返回零。如果把字符串转化为大于给定Q值的最大、最小值的 返回值将限定在最大、最小值的范围内 |
| 用法 | 全局IQ函数（IQ 格式= GLOBAL_Q） GLOBAL_Q格式的输入字符串的相应定点数 特定Q格式IQ函数（IQ 格式= IQ1 到 IQ30） IQN格式的输入字符串的相应定点数 当处理用户输入或ASCII字符串时，这是一个有用的程序 |



例：
以下代码提示用户输入X的值：
`Char buffer[N];`
`_iq X;`
`Printf("Enter value X = ");`
`Gets(buffer);`
`X = _atoiQ(buffer); //IQ value (GLOBAL_Q)`

5.1.4 IQNint: IQN数值的整数部分

| | |
|--------|---|
| IQNint | IQN数值的整数部分 |
| 描述 | 该函数返回IQN数值的整数部分 |
| 声明 | 全局IQ函数 (IQ 格式=GLOBAL_Q) <code>long_IQint(_iq A)</code> |
| 输入 | 特定Q格式IQ函数 (IQ 格式=IQ1 到 IQ30) <code>long_IQNint(_iqN A)</code> 全局IQ函数 (IQ 格式=GLOBAL_Q) 用全局Q格式表示的定点IQ数值 特定Q格式IQ函数 (IQ 格式=IQ1 到 IQ30) 用IQN格式表示的定点IQ数值 |
| 输出 | IQ数值的整数部分 |
| 用法 | |



例1: 下面的例子分离出两个IQ数值的整数部分和小数部分。

```
_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
```



例2: 下面的例子说明了如何由整数部分和小数部分来得到IQ数值:

```
_iq Y;  
long Yint;  
_iq Yfrac;  
Y = _IQmpyl32(_IQ(1.0), Yint) + Yfrac;
```

5.1.5 IQNfrac: IQN数值的小数部分

| | |
|--------|---|
| IQfrac | IQN数值的小数部分 |
| 描述 | 该函数返回IQN数值的小数部分 |
| 声明 | 全局IQ函数（IQ 格式=GLOBAL_Q） _iq_IQfrac(_iq A) |
| | 特定Q格式IQ函数（IQ 格式=IQ1 到 IQ30） iqN_IQNfrac (_iqN A) |
| 输入 | 全局IQ函数（IQ 格式=GLOBAL_Q） 用全局Q格式表示的定点IQ数值 特定Q格式IQ函数（IQ 格式=IQ1 到 IQ30） 用IQN格式表示的定点IQ数值 |
| 输出 | IQ数值的小数部分 |
| 用法 | |

“

例1: 下面的例子分离出两个IQ数值的整数部分和小数部分。

```
_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
```

“

例2: 下面的例子说明了如何由整数部分和小数部分来得到IQ数值：

```
_iq Y;  
long Yint;  
_iq Yfrac;  
Y = _IQmpyl32(_IQ(1.0), Yint) + Yfrac;
```

5.1.6 IQtoIQN:全局IQ格式转换为特定IQ格式

| | |
|----|--|
| 描述 | 该宏将一个IQ数值从全局Q格式转换为特定IQ格式 |
| 声明 | <code>_iqN _IQtoIQN(_iq A)</code> |
| 输入 | 用全局Q格式表示的IQ数值 |
| 输出 | 用IQN格式表示的等值输入 |
| 用法 | 这个宏一般被用在下面的情况：在某一值IQ值进行计算时结果会出现溢出，这样就需要用不同的IQ值来进行转换，以避免溢出。 |



例1:

用Q26格式计算复数 $(x+jY)$ 的幅值

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

值Z、X、Y都是用Q26格式来表示的，但是结果会产生溢出，为了避免溢出，我们用Q=23来进行替代计算，在计算的最后我们再转换回去，如下所示：

```
_iq Z, Y, X; // GLOBAL_Q = 26
_iq23 temp;
temp = _IQ23sqrt( _IQ23mpy(_IQtoIQ23(X), _IQtoIQ23(X)) +
_IQ23mpy(_IQtoIQ23(Y), _IQtoIQ23(Y)) );
Y = _IQ23toIQ(temp);
```

5.1.7 IQNtoIQ:特定IQ格式转换为全局IQ格式

| | |
|----|------------------------------------|
| 描述 | 该宏将一个IQ数值从特定IQ格式转换为全局Q格式 |
| 声明 | <code>_iq _IQNtoIQ(_iqN A)</code> |
| 输入 | 用特定IQ格式表示的IQ数值 |
| 输出 | 用全局Q格式表示的等值输入 |
| 用法 | 该宏可以用于将用特定IQ格式的表示的IQ值转化成全局Q格式的值 |



例：下面的这个例子是计算一个IQ26格式的 $(X+jY)$ 的幅值

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

Z、X、Y为IQ26格式的值，但是这个式子在计算‘Z’时可能会产生溢出

为了防止产生溢出现象，可以在中间计算过程中用IQ23来表示数据X、Y，最后再将结果转换回到IQ26，函数定义过程如下：

```
_iq Z, Y, X; // GLOBAL_Q = 26
_iq23 temp;

temp = _IQ23sqrt( _IQ23mpy(_IQtoIQ23(X), _IQtoIQ23(X)) +
_IQ23mpy(_IQtoIQ23(Y), _IQtoIQ23(Y)) );

Y = _IQ23toIQ(temp);
```


5.1.8 IQtoQN:全局IQ格式转换为QN格式

| | |
|----|--|
| 描述 | 该宏可以将一个32位的全局格式转换为一个16位的QN格式的数。 |
| 声明 | int_IQtoQN(_iq A) |
| 输入 | 用全局Q格式表示的IQ数值 |
| 输出 | 输入的QN格式的等效值（16位的浮点数） |
| 用法 | 宏可以用于在那些输入输出的是16位的应用场合，但是在中间计算过程中采用IQ格式。 |



例1：输入序列为非GLOBAL_Q的乘加

$Y = X0 \cdot C0 + X1 \cdot C1 + X2 \cdot C2$ // X0,X1,X2为Q15格式 C0,C1,C2为GLOBAL_Q格式

可以通过Q15到IQ转换实习在中间计算过程中用IQ，最后的结果以Q15格式存贮。函数定义如下：

```
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);
```

5.1.9 QNtoIQ:QN格式转换为全局IQ格式

| | |
|----|---|
| 描述 | 该宏可以将一个16位的QN格式的数转换为32位的全局格式。 |
| 声明 | int_QNtoIQ (_iq A) |
| 输入 | 16位的QN格式的浮点数 |
| 输出 | 输入的QN格式的等效GLOBAL_Q值 |
| 用法 | 该宏可以用于在那些输入输出的是16位的应用场合，但是在中间计算过程中采用IQ格式。 |



例1：输入序列为非GLOBAL_Q的乘加

$Y = X0 \cdot C0 + X1 \cdot C1 + X2 \cdot C2$ // X0,X1,X2为Q15格式 C0,C1,C2为GLOBAL_Q格式

可以通过Q15到IQ转换实习在中间计算过程中用IQ，最后的结果以Q15格式存贮。函数定义如下：

```
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);
```

5.2.1 IQNmpy: IQ乘法 (IQN*IQN)

描述 C编译器实质上是使两个IQ值相乘，它并不表现出饱和以及四舍五入的格式。在大多数情况下，两个IQ值相乘不会超出IQ变量的范围。由于该操作使得运行周期最小并且代码最少，所以很常用。

声明 全局IQ格式（即IQ格式等于GLOBAL_Q时）
_iq_IQmpy(_iq A,_iq B)
特定Q格式IQ格式（即IQ格式等于IQ1 – IQ30之间的某个格式）
_iqN_IQNmpy(_iq NA,_iq NB)

输入 全局IQ格式（即IQ格式等于GLOBAL_Q时）
输入“A”和“B”为以GLOBAL_Q为格式的IQ数据
特定Q格式IQ格式（即IQ格式等于IQ1 – IQ30之间的某个格式）
输入IQ值“A”和“B”为IQN格式上的IQ数据

输出 全局IQ格式（即IQ格式等于GLOBAL_Q时）
乘法结果为GLOBAL_Q格式的数据
特定Q格式IQ格式（即IQ格式等于IQ1 – IQ30之间的某个格式）
乘法结果为IQN格式的数据

用法



例1：

下面的代码是在GLOBAL_Q格式下进行“Y= M*X+B”的计算，不带饱和和四舍五入：

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



例2：

下面的代码是在Q10格式下进行“Y= M*X+B”的计算，不带饱和和四舍五入，假设MXB是以IQ10格式：

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

5.2.3 IQNrsmpy:带四舍五入带饱和处理的IQ乘法(IQN *IQN)

| | |
|----|--|
| 描述 | 这个函数对两个IQ数据进行乘法运算，并进行四舍五入和饱和处理。当计算有可能超出IQ变量范围时，在存储结果之前对结果四舍五入和进行饱和处理，达到IQ变量范围的最大值 |
| 声明 | 全局IQ格式（即IQ格式等于GLOBAL_Q时） _iq_IQrsmpy(_iq A,_iq B) 特定Q格式IQ格式（即IQ格式等于IQ1~ IQ30之间的某个格式） _iqN_IQNrsmpy(_iq NA,_iq NB) |
| 输入 | 全局IQ格式（即IQ格式等于GLOBAL_Q时） 输入“A”和“B”为以GLOBAL_Q为格式的IQ数据 特定Q格式IQ格式（即IQ格式等于IQ1~ IQ30之间的某个格式） 输入IQ值“A”和“B”为IQN格式上的IQ数据 |
| 输出 | 全局IQ格式（即IQ格式等于GLOBAL_Q时） 乘法结果为GLOBAL_Q格式的数据 特定Q格式IQ格式（即IQ格式等于IQ1~ IQ30之间的某个格式） 乘法结果为IQN格式的数据 |
| 用法 | 我们假设我们所用的IQ26是GLOBAL_Q格式。这也就是说所取数值是在-32.0，32.0之间的（参照3.2节）。如果两个IQ变量相乘，那么所得结果就是[-1024,1024]这个操作可以确保当计算结果超出[-32，32]时使结果在+/-32处取值。 |



例1：

下面的代码是在GLOBAL_Q格式下进行“Y= M*X”的计算，带四舍五入，带饱和处理（假定GLOBAL_Q=IQ26）：

```
_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,输出已饱和，输出最大值
```



例2：

下面的代码是在Q26格式下进行“Y= M*X”的计算，带四舍五入，带饱和处理：

```
_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,输出已饱和，输出最小值
```

5.2.4 IQNmpyl32:IQ格式与长整型相乘 (IQN*LONG)

| | |
|----|---|
| 描述 | 用于IQ格式的数与长整型的数相乘 |
| 声明 | Global IQ 格式 (IQ 格式=GLOBAL_Q) _iq_IQmpyl32(_iq A,long B) 特定Q格式格式 (IQ格式=IQ1到IQ30) _iqN_IQNmpyl32(_iqN A,long B) |
| 输入 | Global IQ格式 (IQ 格式=GLOBAL_Q) A为GLOBAL_Q格式的IQ数, B为长整型数 特定Q格式格式 (IQ格式=IQ1到IQ30) A为IQN格式 (IQ1到IQ30) 的IQ数, B为长整型数 |
| 输出 | Global IQ格式 (IQ 格式=GLOBAL_Q) 乘法的结果是Global IQ格式 特定Q格式格式 (IQ格式=IQ1到IQ30) 乘法的结果是IQN格式 |
| 用法 | |



例1:

下面的代码用GLOBAL_Q格式计算"Y=5*X" (假设GLOBAL_Q=IQ26)

```
_iq Y, X;  
X=_IQ(5.1); // X=5.1 GLOBAL_Q格式  
Y= IQmpyl32(X,5); // Y= 25.5 GLOBAL_Q格式
```



例2:

下面的代码用IQ26格式计算"Y=5*X"

```
_iq26 Y, X;  
long M;  
M=5; // M=5  
X=_IQ26(5.1); // X=5.1 IQ26格式 ( 此处IQmath手册为IQ29, 应该是有误 )  
Y= _IQ26mpyl32(X,M); // Y=25.5 IQ26格式 ( 此处IQmath手册为IQ29, 应该是有误 )
```

5.2.5 IQNmpyl32int: IQN与LONG相乘结果的整数部分

| | |
|----|--|
| 描述 | 这一函数用于IQ格式的数与长整型的数相乘，返回为相乘结果的整数部分 |
| 声明 | Global IQ 函数（IQ 格式=GLOBAL_Q） long_IQmpyl32int(_iq A,long B) 特定Q格式函数（IQ格式=IQ1到IQ30） long_IQNmpyl32int(_iqN A,long B) |
| 输入 | Global IQ(IQ 格式=GLOBAL_Q) A为GLOBAL_Q格式的IQ数，B为长整型数 特定Q格式（IQ格式=IQ1到IQ30） A为IQN格式（IQ1到IQ30）的IQ数，B为长整型数 |
| 输出 | Global IQ(IQ 格式=GLOBAL_Q) 结果的整数部分（32位） 特定Q格式（IQ格式=IQ1到IQ30） 结果的整数部分（32位） |
| 用法 | |



例1：

将范围在[-1.0,1.0]IQ格式的数转换成0到1023的整数

```
_iq Output;  
long temp;  
short OutputDAC;  
temp = _IQmpyl32int(Output, 512); // 转换到+/- 512  
temp += 512; // 标定到0到1023  
if( temp > 1023 ) // 设定输出的上下限（饱和值）  
temp = 1023;  
if( temp < 0 )  
temp = 0;  
OutputDAC = (int)temp; // 输出整型数
```

注：因为是整数运算，运算结果是64位计算结果的整数部分，所以要避免出现溢出情况。

5.2.6 IQNmpyl32frac:IQN与LONG相乘结果的小数部分

| | |
|----|--|
| 描述 | 这一函数用于IQ格式的数与长整型的数相乘，返回为相乘结果的小数部分 |
| 声明 | Global IQ 函数（IQ 格式=GLOBAL_Q） long_IQmpyl32frac(_iq A,long B) 特定Q格式函数（IQ格式=IQ1到IQ30） long_IQNmpyl32frac(_iqN A,long B) |
| 输入 | Global IQ(IQ 格式=GLOBAL_Q) A为GLOBAL_Q格式的IQ数，B为长整型数 特定Q格式（IQ格式=IQ1到IQ30） A为IQN格式（IQ1到IQ30）的IQ数，B为长整型数 |
| 输出 | Global IQ(IQ 格式=GLOBAL_Q) 结果的小数部分（32位） 特定Q格式（IQ格式=IQ1到IQ30） 结果的小数部分（32位） |
| 用法 | |



例1：

下面的例子是提取乘法运算结果的小数部分（假设GLOBAL_Q=IQ26）

```
_iq X1= _IQ(2.5);  
_iq X2= _IQ26(-1.1);  
long M1=5, M2=9;  
_iq Y1frac, Y2frac;
```

```
Y1frac = IQmpyl32frac(X1, M1); // Y1frac = 0.5 GLOBAL_Q格式
```

```
Y2frac = IQ26mpyl32frac(X2, M2); // Y2frac = -0.9 GLOBAL_Q格式
```

5.2.7 IQNmpylQX:乘法 (GLOBAL_Q=IQN1*IQN2)

| | |
|----|--|
| 描述 | C编译器将两个不同IQ格式的IQ数进行相乘 |
| 声明 | Global IQ格式 (IQ 格式等于 GLOBAL_Q) _iq _IQmpylQX(_iqN1 A, int N1, _iqN2 B, int N2) 特定IQ格式 (IQ格式为IQ1到IQ30) _iqN _IQNmpylQX(_iqN1 A, int N1, _iqN2 B, int N2) |
| 输入 | 操作数“A”是“IQN1”格式的IQ数，而操作数“B”则是“IQN2”格式 |
| 输出 | 全局 IQ乘积(IQ 格式=GLOBAL_Q) 用GLOBAL_Q格式表示的乘法结果 特定Q格式IQ乘积 (IQ格式=IQ1到IQ30) 用IQN格式表示的乘法结果 |
| 用法 | 在我们进行不同IQ格式的值进行相乘的时候，这个操作是很有用的。 |



例1：

我们希望计算以下等式

$$Y=X0*C0+X1*C1+X2*C2$$

这里，

X0, X1, X2是IQ30格式 (范围从-2到+2)

C0, C1, C2是IQ28格式 (范围从-8到+8)

Y的最大范围将是-48到+48，因此我们应该用小于IQ25的IQ格式来存储结果

情况1：GLOBAL_Q=IQ25

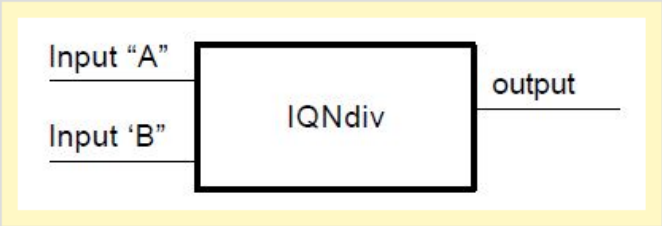
```
_iq30 X0, X1, X2;           //所有变量为IQ30格式
_iq28 C0, C1, C2;           //所有变量为IQ28格式
_iq Y;                       //结果 GLOBAL_Q = IQ25 格式
Y=_IQmpylQX(X0, 30, C0, 28);
Y+=_IQmpylQX(X1, 30, C1, 28);
Y+=_IQmpylQX(X02, 30, C2, 28);
```

情况2：特定IQ的计算

```
_iq30 X0, X1, X2;           //所有变量为IQ30格式
_iq28 C0, C1, C2;           //所有变量为IQ28格式
_iq25 Y;                     //结果 GLOBAL_Q = IQ25 格式
Y=_IQ25mpylQX(X0, 30, C0, 28);
Y+=_IQ25mpylQX(X1, 30, C1, 28);
Y+=_IQ25mpylQX(X02, 30, C2, 28);
```

5.2.8 IQNdiv:定点除法

描述 这个模块使用牛顿-拉斐森方法（Newton-Raphson technique）对两个IQN的数进行相除，并给出32bit的商（IQN格式）



高效性 C可调用的汇编语言（C cA）

模块属性 类型：与目标无关，与应用无关
目标设备： x28xx
C/C++接口文件：IQmathLib.h, IQmathCPP.h & IQmath.lib

| 项目 | C可调用的汇编函数 | 注释 |
|-------|-----------|--------------|
| 代码量 | 71 words | |
| 数据RAM | 0 words | |
| 多实例 | N/A | |
| 重入性 | YES | |
| 多调用 | YES | |
| 堆栈使用 | 2 words | 堆栈按2 words增加 |

精确度 = 20 log2 (231)-20 log2(7)
 = 28 bits

| | |
|----|---|
| 声明 | Global IQ格式（IQ 格式等于 GLOBAL_Q） _iq _IQdiv(_iq A, _iq B) 特定Q格式的IQ函数（IQ格式等于IQ1到IQ30） _iq _IQNdiv(_iqN A, _iq B) |
| 输入 | Global IQ函数（IQ格式等于GLOBAL_Q） 输入“A”，“B”是GLOBAL_Q格式所代表的定点数 特定Q格式的IQ函数（IQ格式等于IQ1到IQ30） 输入“A”和输入“B”是IQN格式的定点数（N=1：30） |
| 输出 | Global IQ函数（IQ格式等于GLOBAL_Q） 输出是GLOBAL_Q格式 特定Q格式的IQ函数（IQ格式等于IQ1到IQ30） 输出是IQN格式（N=1：30） |
| 用法 | |

“

例1：

以下的例子是计算 $1/1.5=0.666$ ，假设在IQmath头文件里，GLOBAL_Q被设为Q28格式。

```
#include<IQmathLib.h> /* IQ math 函数的头文件 */
```

```
_iq in1 out1;
```

```
_iq28 in2 out2;
```

```
void main(void)
```

```
{
```

```
in1 = _IQ(1.5);
```

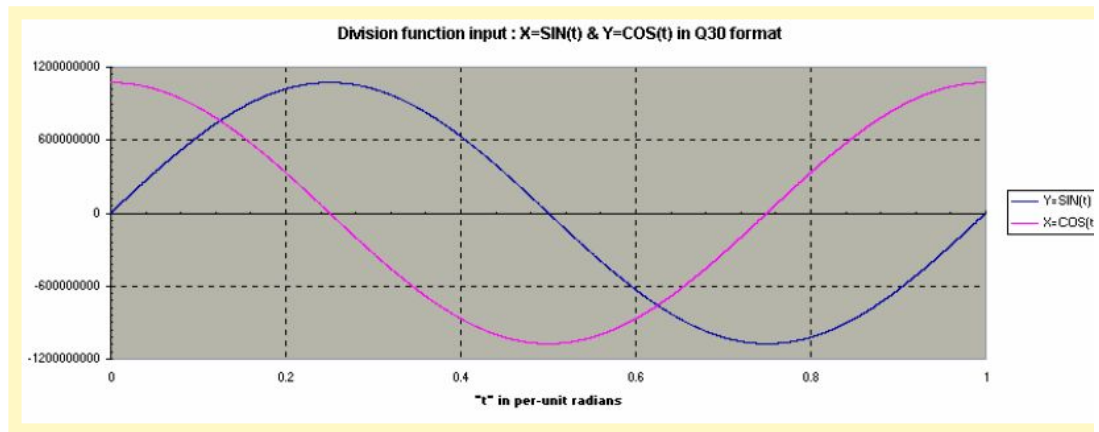
```
out1 = _IQdiv(_IQ(1.0), in1);
```

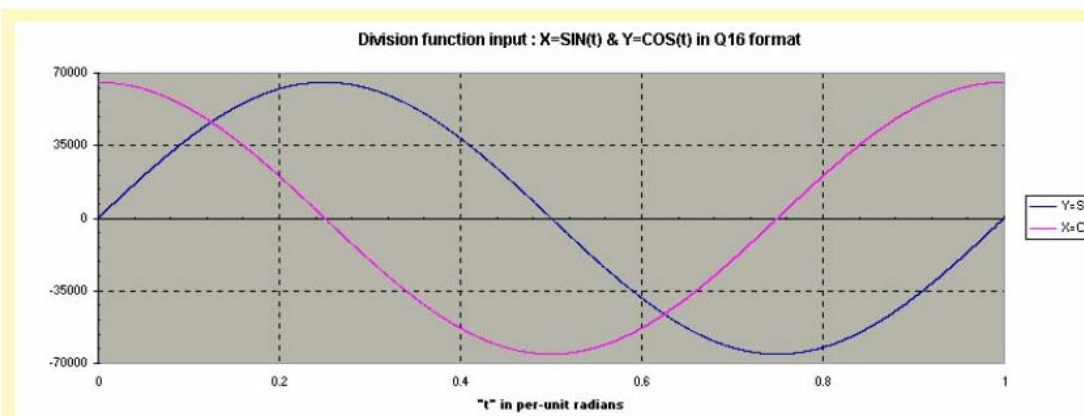
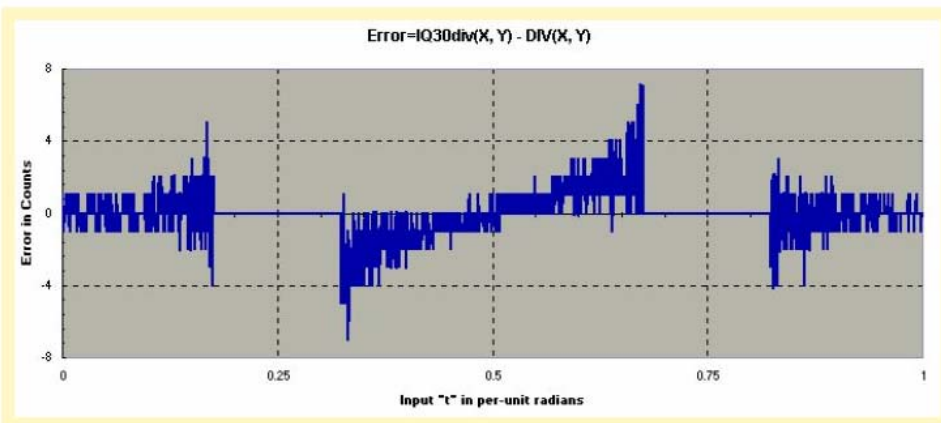
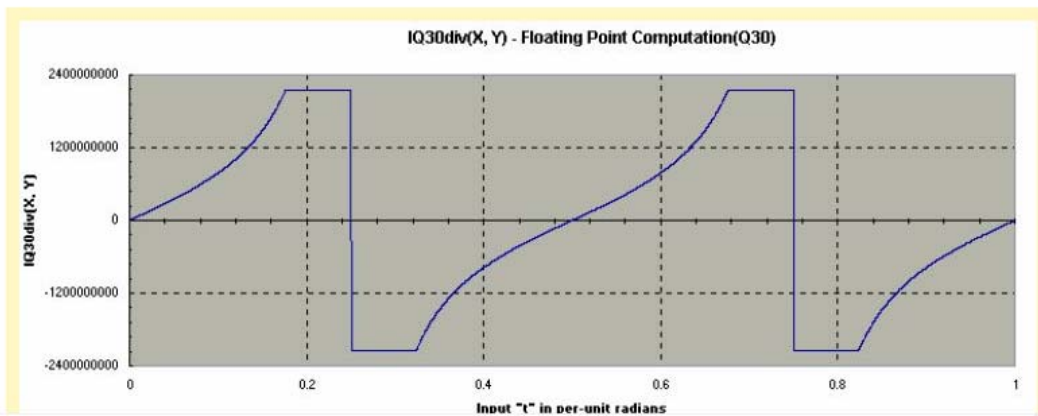
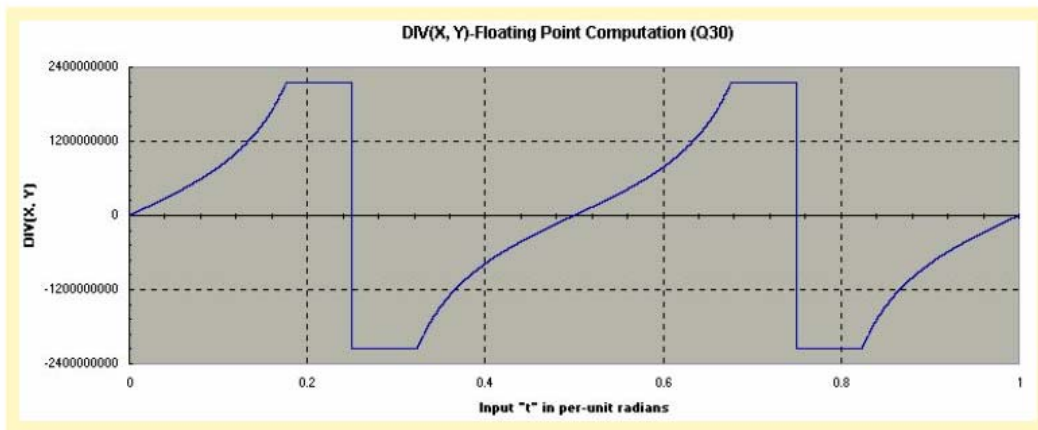
```
in2 = _IQ28(1.5);
```

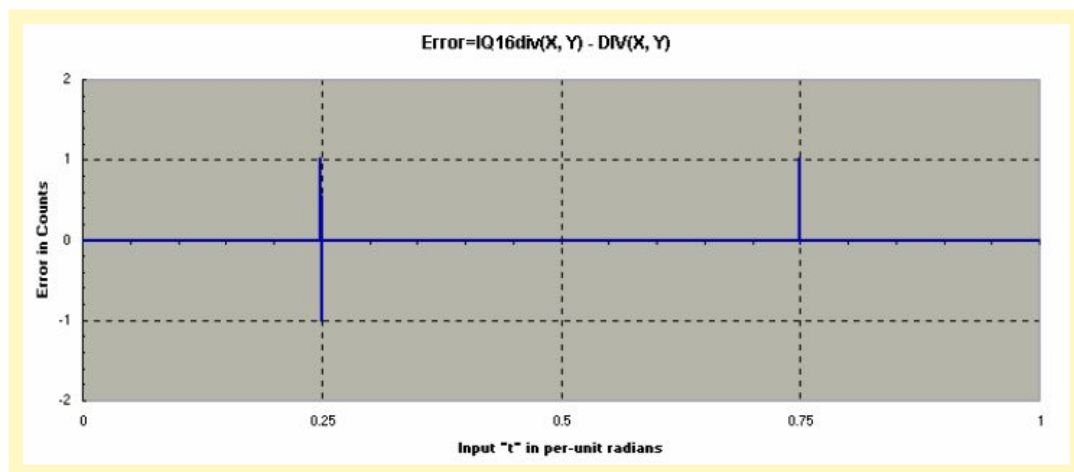
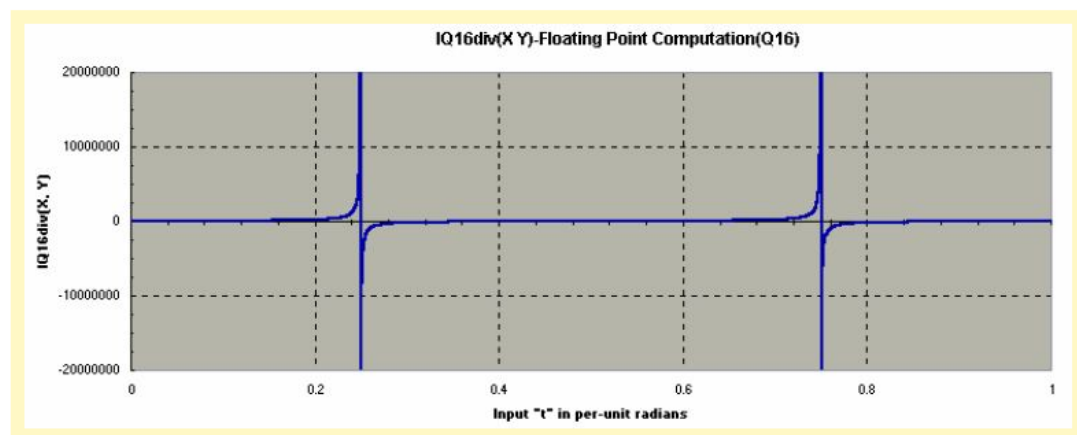
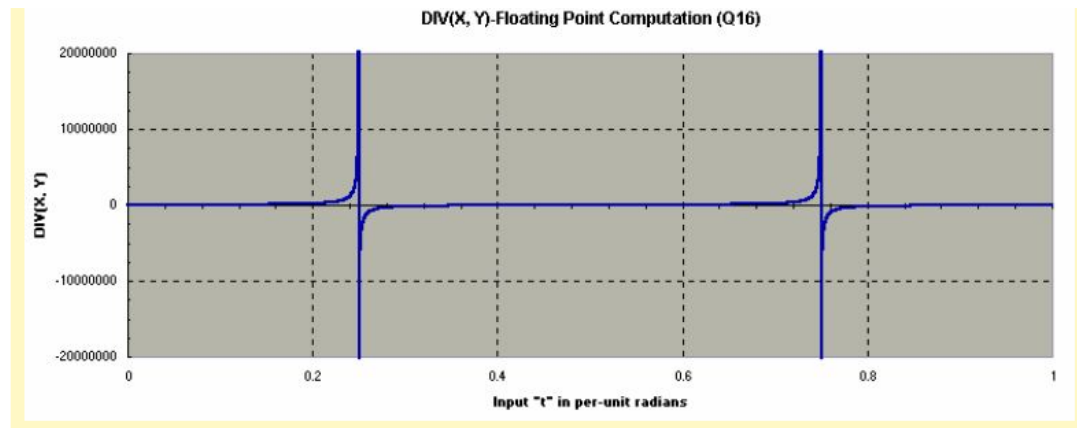
```
out2 = _IQ28div(_IQ28(1.0), in2);
```

```
}
```

定点除法 VS C浮点除法



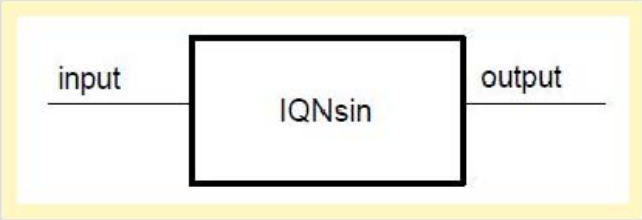




IQNsin:定点正弦函数(输入单位:弧度)

描述

这个模块使用查询表项目中的表查询和泰勒级数展开来计算输入（弧度范围）的正弦值。



高效性

C/C++可调用汇编

模块属性

类型：目标独立，应用独立
目标设备：x28xx
C/C++接口文件：IQmathLib.h, IQmathCPP.h & IQmath.lib

| 项目 | C可调用的汇编函数 | 注释 |
|-------|-----------|--------------|
| 代码量 | 49 words | |
| 数据RAM | 0 words | |
| 多实例 | N/A | |
| 重入性 | YES | |
| 多调用 | YES | |
| 堆栈使用 | 2 words | 堆栈按2 words增加 |

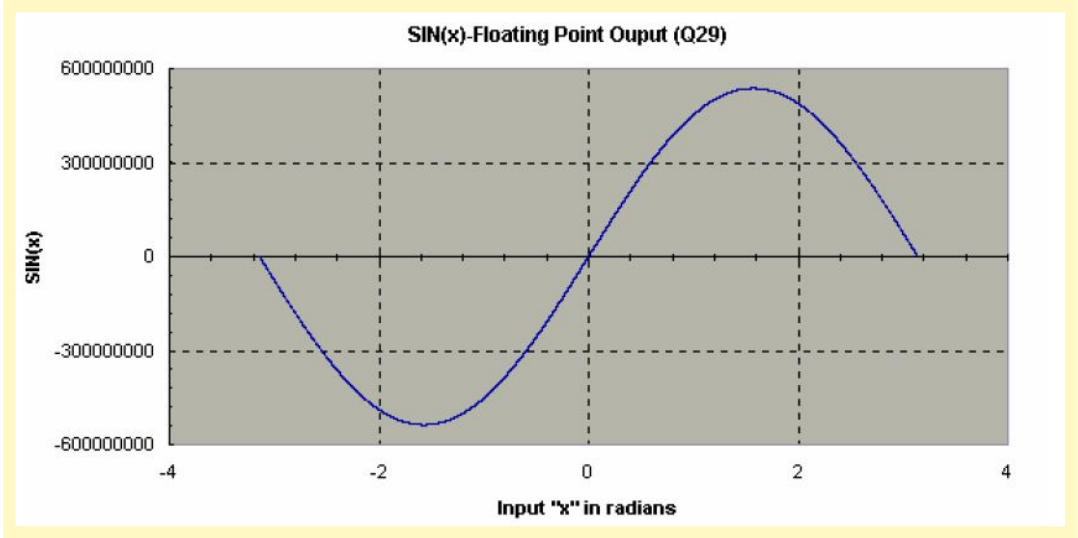
精确度

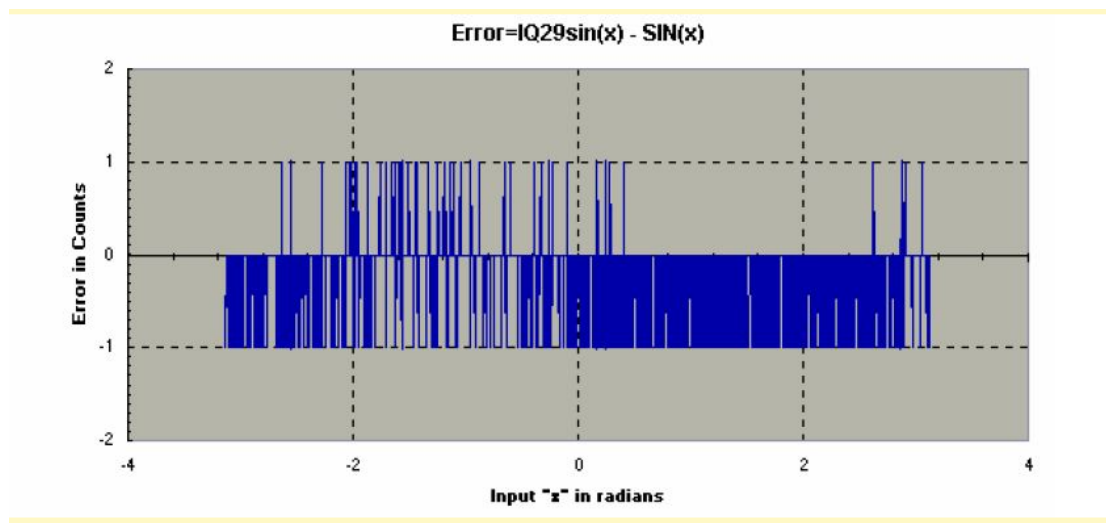
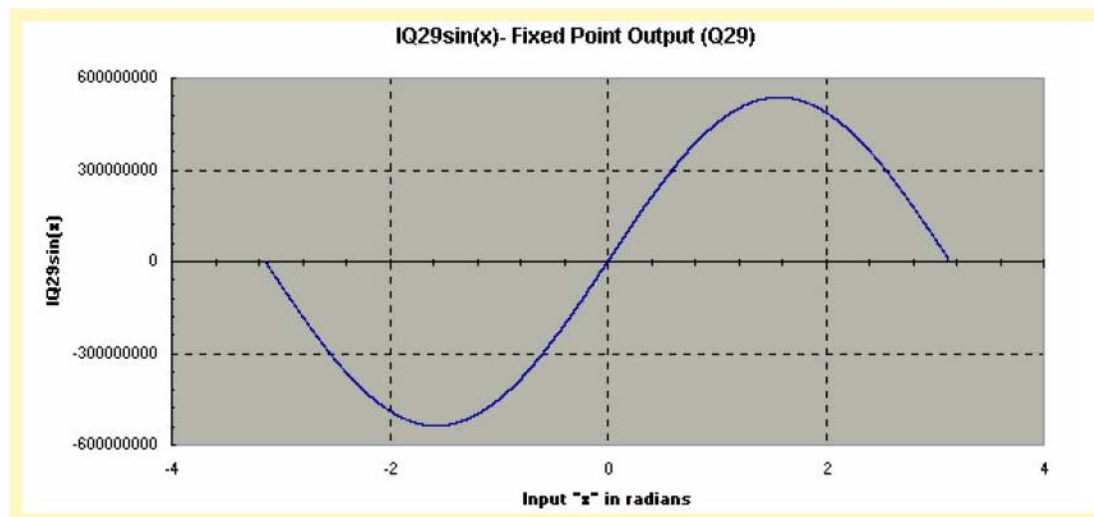
$$= 20\log_2(\pi \times 2^{29}) - 20\log_2(1)$$
$$= 30 \text{ bits}$$

| | |
|----|---|
| 声明 | 全局IQ函数（IQ格式=GLOBAL_Q） <code>_iq_lqsin(_iq A)</code> 特定的Q格式IQ函数（IQ格式=IQ1到IQ29） <code>_iqN_IQNsin(_iqN A)</code> |
| 输入 | 全局IQ函数（IQ格式=GLOBAL_Q） 输入参数是弧度范围和GLOBAL_Q格式范围的定点数 特定的Q格式IQ函数（IQ格式=IQ1到IQ29） 输入参数是弧度范围和IQN格式(N=1:29)范围的定点数 |
| 输出 | 全局IQ函数（IQ格式=GLOBAL_Q） 函数返回输入参数是GLOBAL_Q格式的定点数的正弦值 特定的Q格式IQ函数（IQ格式=IQ1到IQ29） 函数返回输入参数是IQN格式(N=1:29)的定点数的正弦值 |
| 用法 | |

“
例1：
 >假设GLOBAL_Q被设置为IQmath头文件中的Q29格式，由下面例子可得到 $\sin(0.25 \times \pi) = 0.707$ 。
`#include <IQmathLib.h> /*用于IQmath例行程序的头文件*/`
`#define PI 3.14156`
`_iq in1,out1;`
`_iq28 in2,out2;`
`void main(void)`
`{`
 `in1=_IQ(0.25*PI);` */*in1 = 0.25 × π × 2²⁹ = 19214854h π*/*
 `out1=_IQsin(in1);` */*out1 = sin(0.25 × π) × 2²⁹ = 16A09E66h π*/*
 `in2=_IQ29(0.25*PI);` */*in2 = 0.25 × π × 2²⁹ = 19214854h π*/*
 `out2=_IQsin(in2);` */*out2 = sin(0.25 × π) × 2²⁹ = 16A09E66h π*/*
`}`

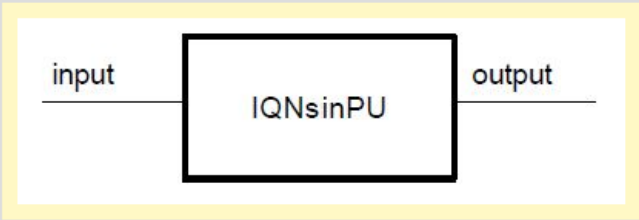
IQNsin函数和C浮点SIN函数比较：输入范围从-π到π





5.3.2 IQNsinPU:定点正弦函数（输入:标么值）

描述 该函数模块利用查表和泰勒级数展开的方法计算输入为单位弧度的正弦值。



高效性 C/C++可调用汇编

模块属性 类型：目标独立，应用独立
目标设备： x28xx
C/C++接口文件：IQmathLib.h, IQmathCPP.h & IQmath.lib

| 项目 | C可调用的汇编函数 | 注释 |
|-------|-----------|--------------|
| 代码量 | 41 words | |
| 数据RAM | 0 words | |
| 多实例 | N/A | |
| 重入性 | YES | |
| 多调用 | YES | |
| 堆栈使用 | 2 words | 堆栈按2 words增加 |

精确度 $= 20\log_2(1 \times 2^{-30}) - 20\log_2(1)$
= 30 bits

声明 全局IQ函数（IQ格式=GLOBAL_Q）
iq_IQsinPU(_iq A)
特定Q格式IQ函数（IQ格式=IQ1到IQ30）
iqN_IQNsinPU(_iqN A)

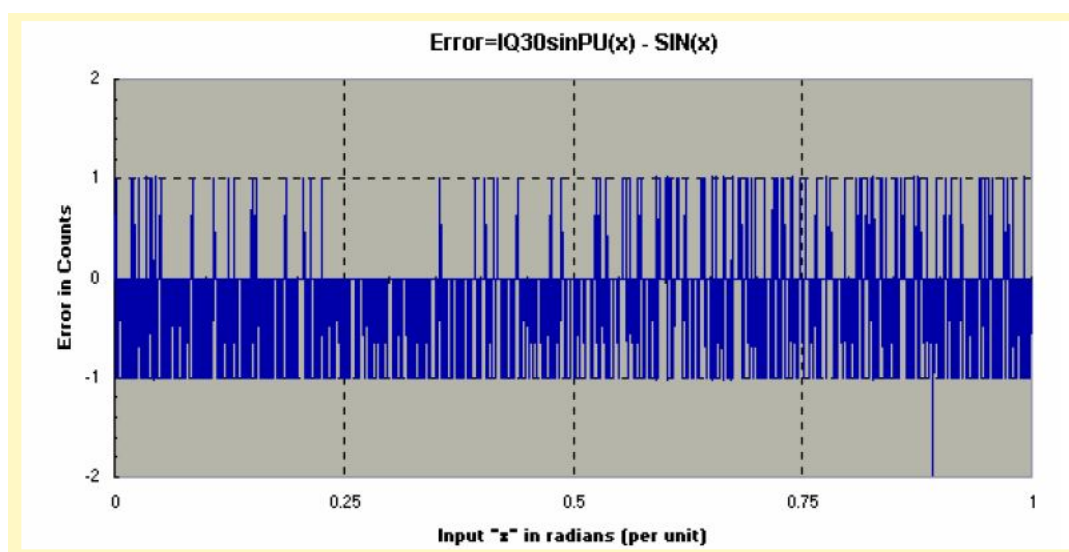
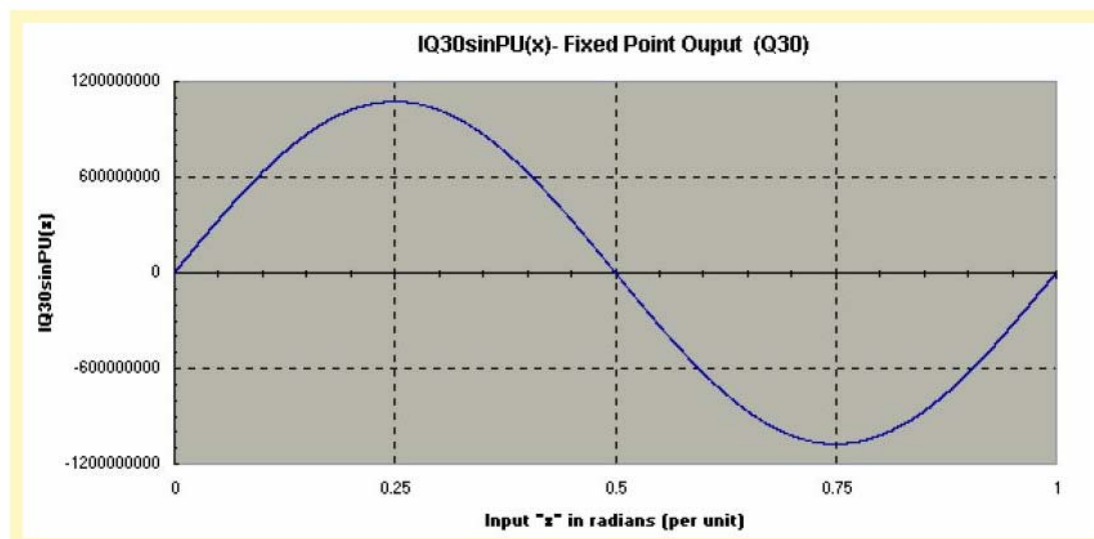
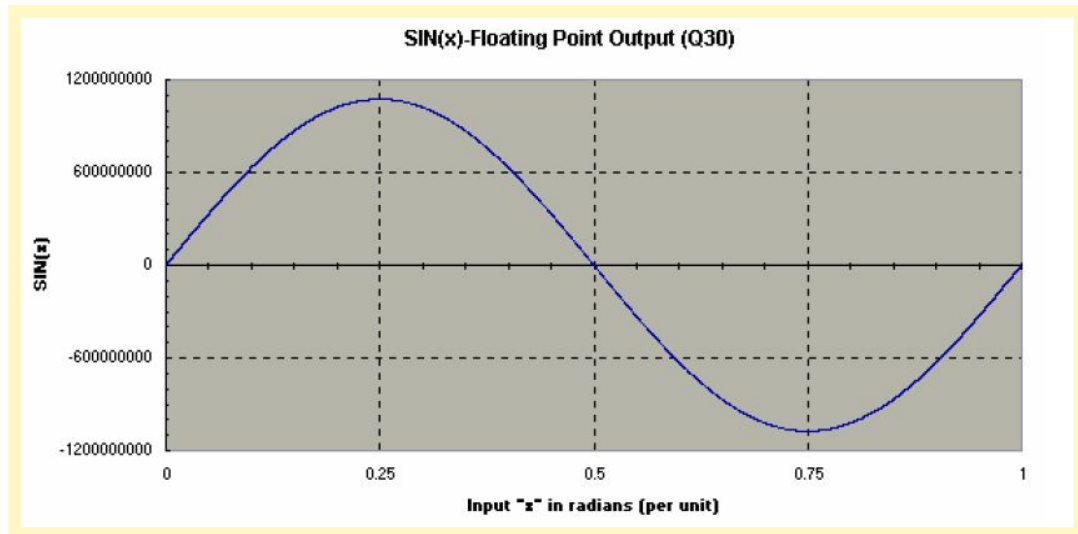
输入 全局IQ函数（IQ格式=GLOBAL_Q）
输入量是以全局Q格式定点数表示的标么值
特定的Q格式IQ函数（IQ格式=IQ1到IQ30）
输入量是以特定Q格式定点数表示的标么值

输出 全局IQ函数（IQ格式=GLOBAL_Q）
该函数模块返回一个对应于输入标么值的正弦值，它的数据格式为全局Q格式
特定的Q格式IQ函数（IQ格式=IQ1到IQ30）
该函数模块返回一个对应于输入标么值的正弦值，它的数据格式为特定Q格式表示的定点数（IQ1 ~ IQ30）

用法

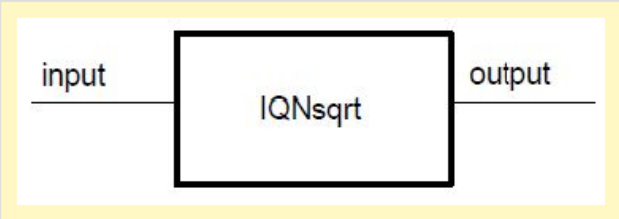
“
例1：
假设在IQmath库函数的头文件中全局Q格式定义为Q30，则下面的例子可得到式子 $\sin(0.25 \times \pi) = 0.707$ 。
#include<IQmathLib.h> /* IQmath 函数库的头文件 */
#define PI 3.14156
_iq in1, out1;
_iq30 in2, out2;
void main(void)
{
in1=_IQ(0.25*PI/PI); /* in1 = (0.25 × π / 2 π) × 2³⁰= 08000000h */
out1=_IQsinPU(in1); /* out1 = sin (0.25 × π) × 2³⁰= 2D413CCCh */
in2=_IQ30(0.25*PI/PI); /* in2 = (0.25 × π / 2 π) × 2³⁰= 08000000h */
out2=_IQ30sinPU(in2); /* out2= sin (0.25 × π) × 2³⁰= 2D413CCCh */
}

IQNsinPU函数和C语言浮点SIN函数的比较：输入量从0到2PI以单位弧度增加



5.4.1 IQNsqr:定点平方根函数

描述 这个函数用来计算平方根，使用查表和牛顿-拉斐逊逼近算法



高效性 C/C++可调用汇编

模块属性 类型：目标独立，应用独立
目标设备： x28xx
C/C++接口文件：IQmathLib.h, IQmathCPP.h & IQmath.lib

| 项目 | C可调用的汇编函数 | 注释 |
|-------|-----------|--------------|
| 代码量 | 66 words | |
| 数据RAM | 0 words | |
| 多实例 | N/A | |
| 重入性 | YES | |
| 多调用 | YES | |
| 堆栈使用 | 2 words | 堆栈按2 words增加 |

精确度
$$= 20\log_2\left(\frac{\pi}{2} \times 2^{31}\right) - 20\log_2(6)$$

= 29 bits

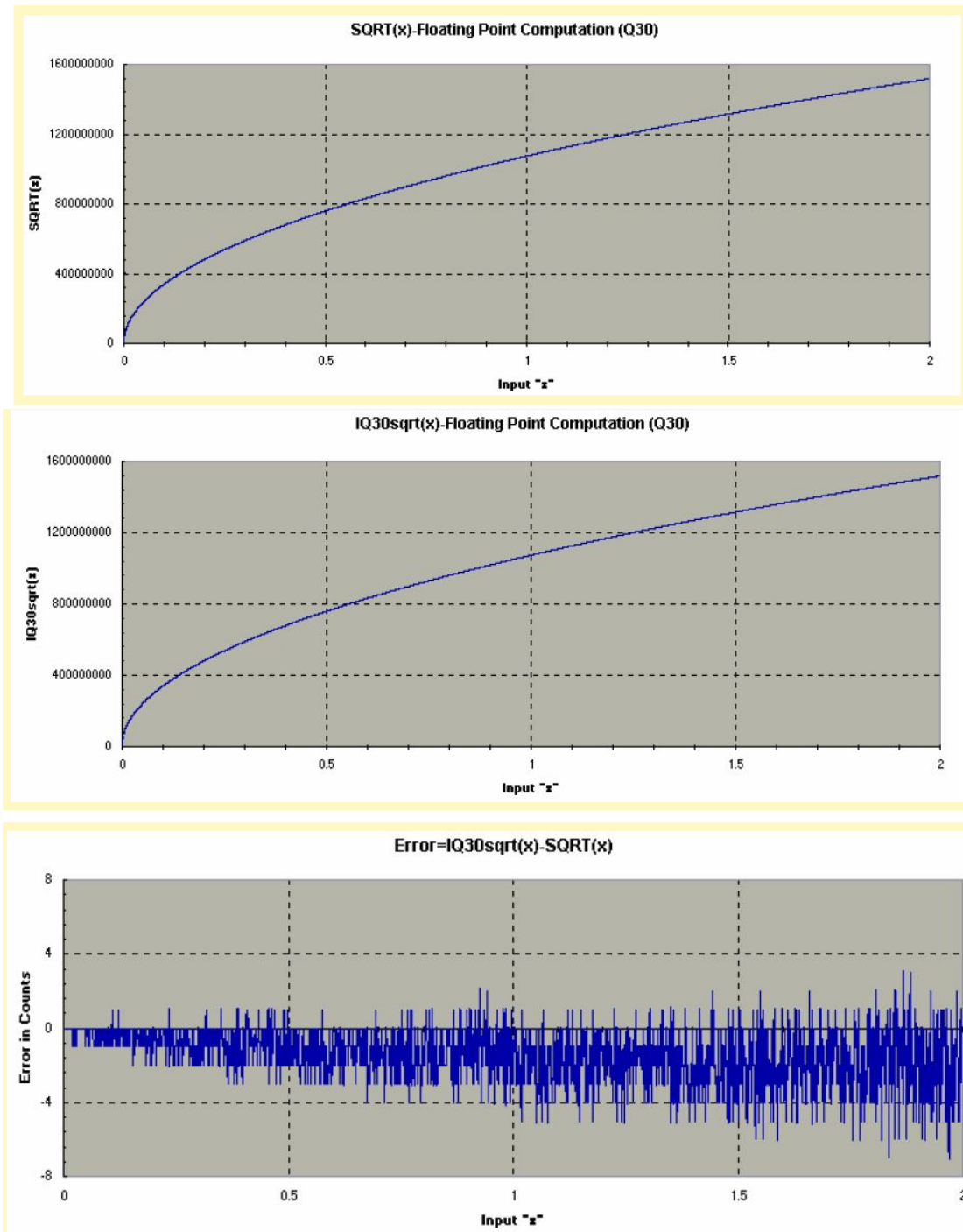


例1：

下面以计算 $\sqrt{1.8} = 1.34164$ 为例说明函数的使用，假设在IQmath头文件中GLOBAL_Q设置成Q30。
#include<IQmathLib.h>

```
_iq in1, out1;
_iq30 in2, out2;
void main(void)
{
    in1=_IQ(1.8);          /*in1 = 1.8 × 230 = 73333333h */
    out1=_IQsqr(in1);      /*out1 = √1.8 × 230 = 55DD7151h */
    in2=_IQ30(1.8);        /*in2 = 1.8 × 230 = 73333333h */
    out2=_IQ30sqr(in2);    /*out1 = √1.8 × 230 = 55DD7151h */
}
```

定点SQRT函数与C语言浮点SQRT比较



5.4.2 IQNisqrt:定点开方根的倒数

描述

该模块采用查找表与牛顿–Raphson逼近计算输入数的开方根

input

IQNisqrt

output

高效性

C/C++可调用汇编

模块属性

类型：目标独立，应用独立
目标设备：x28xx
C/C++接口文件：IQmathLib.h, IQmathCPP.h & IQmath.lib

| | | |
|-------|-----------|--------------|
| 项目 | C可调用的汇编函数 | 注释 |
| 代码量 | 69 words | |
| 数据RAM | 0 words | |
| 多实例 | N/A | |
| 重入性 | YES | |
| 多调用 | YES | |
| 堆栈使用 | 2 words | 堆栈按2 words增加 |

精度

= 20 log2 (2³¹) -20 log2 (5)
= 29 bits

声明

全局IQ函数（IQ格式=GLOBAL_Q）
_iq_IQisqrt(_iq A)
特定Q格式IQ函数（IQ格式=IQ1到IQ30）
_iqN_IQNisqrt(_iqN A)

输入

全局IQ函数（IQ格式=GLOBAL_Q）
输入参数是由GLOBAL_Q格式定点数
特定的Q格式IQ函数（IQ格式=IQ1到IQ30）
输入参数是由IQN特定的Q格式定点数

输出

全局IQ函数（IQ格式=GLOBAL_Q）
返回以GLOBAL_Q格式表示的输入开方根的倒数
特定的Q格式IQ函数（IQ格式=IQ1到IQ30）
返回以IQN（N=1:30）格式表示的输入开方根的倒数

用法

“

例1：

$$\frac{1}{\sqrt{1.8}}=0.74535$$

以下例程求 $\frac{1}{\sqrt{1.8}}$ ，假设在Iqmath头文件中定义GLOBAL_Q为30。

```
#include<IQmathLib.h>          /* Header file for IQ math routine */

_iq in1, out1;
_iq30 in2, out2;

void main(void )
{
    in1=_IQ(1.8);                /* in1= 1.8×230 = 73333333h */
    out1=_IQisqrt(in1);          /* out1=  $\frac{1}{\sqrt{1.8}} \times 2^{30} = 2FB3E99Eh$  */

    in2=_IQ30(1.8);              /* in2= 1.8×230 = 73333333h */
}
```

```

out2=_IQ30isqrt(in2);
}

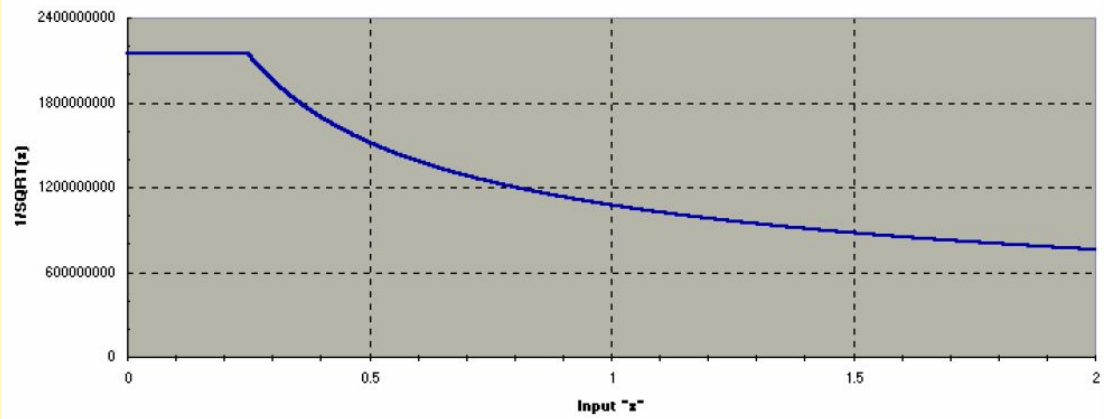
```

/ out2= $\frac{1}{\sqrt{1.8}} \times 2^{30} = 2FB3E99Eh$ */*

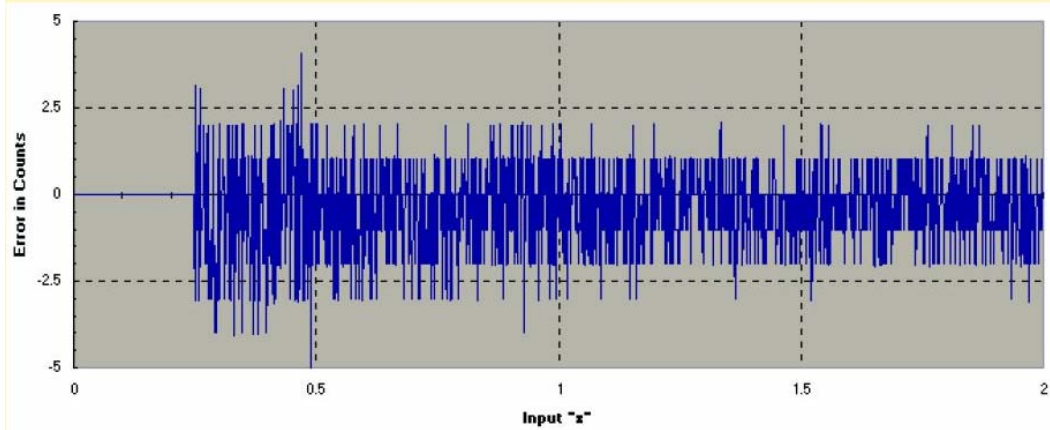
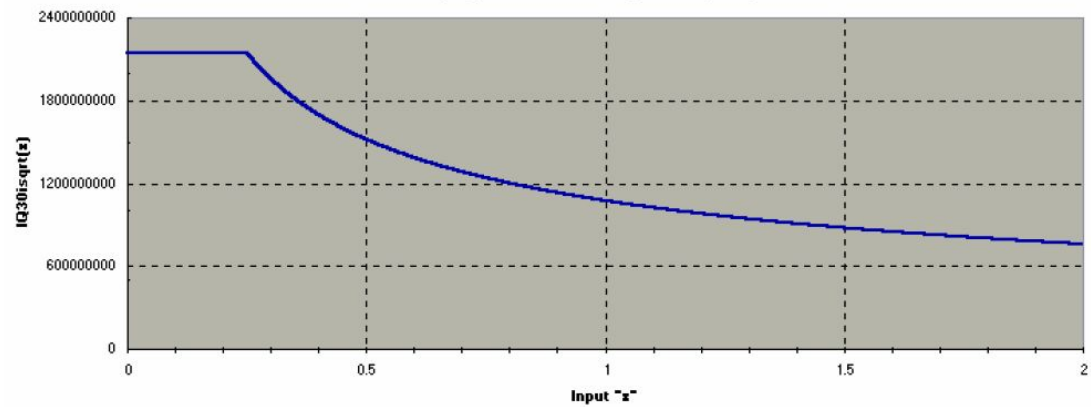
**/*

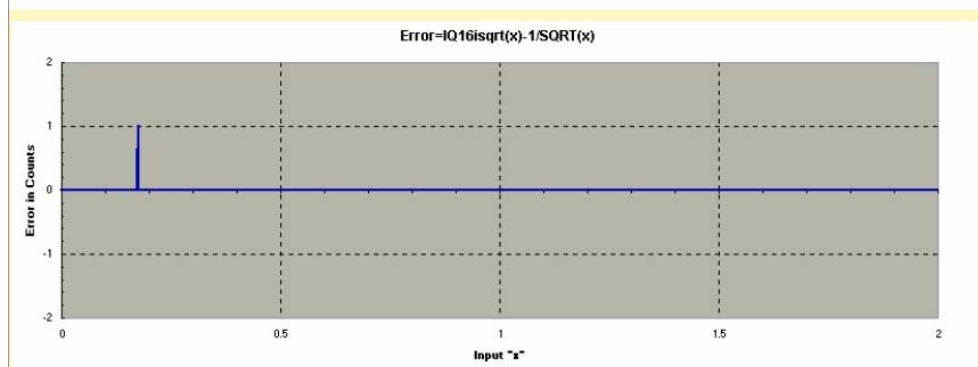
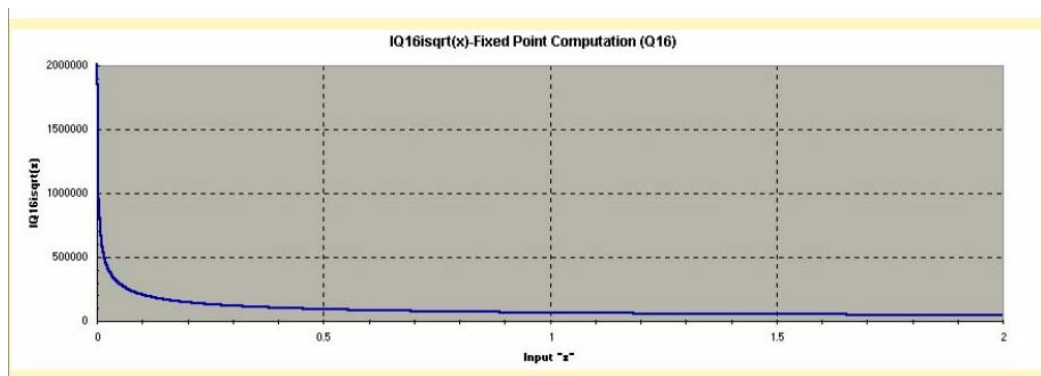
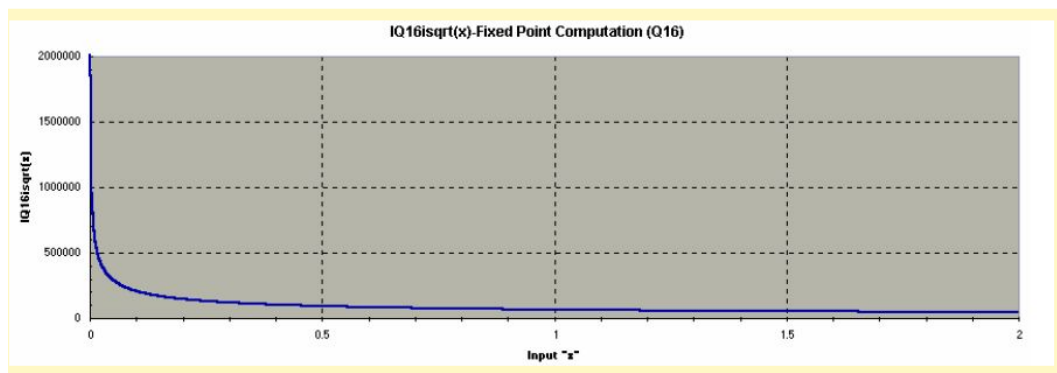
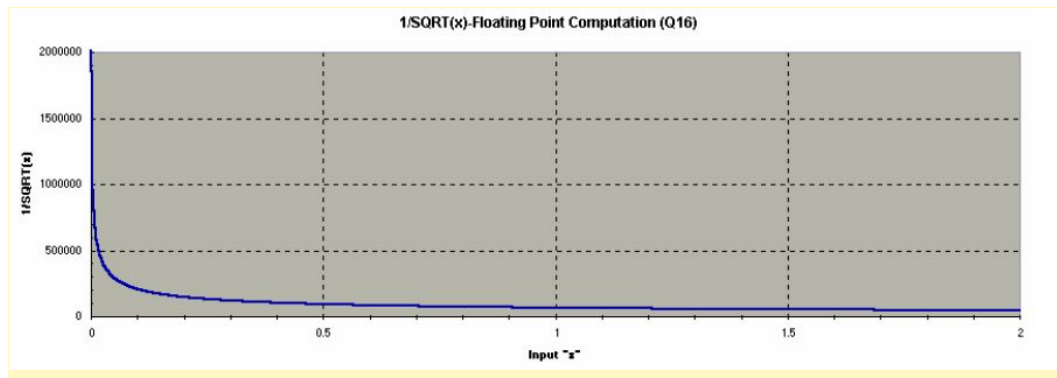
点ISORT函数与C语言浮点ISORT函数比较

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



IQ30isqrt(x)-Fixed Point Computation (Q30)





5.4.3 IQNmag:IQ复数的幅值函数

描述 函数计算两个正交向量的幅值，如下： $\text{Mag}=\sqrt{A^2+B^2}$ 。计算达到了更高的精确度，同时避免了使用“_IQsqrt”函数时遇到的溢出问题。



高效性 C/C++可调用汇编

模块属性 类型：目标独立，应用独立
目标设备：x28xx
C/C++接口文件：IQmathLib.h, IQmathCPP.h & IQmath.lib

| 项目 | C可调用的汇编函数 | 注释 |
|-------|-----------|--------------|
| 代码量 | 96 words | |
| 数据RAM | 0 words | |
| 多实例 | N/A | |
| 重入性 | YES | |
| 多调用 | YES | |
| 堆栈使用 | 2 words | 堆栈按2 words增加 |

精确度 29位（等同于SQRT函数）

声明 全局IQ函数（IQ格式=GLOBAL_Q）
_iq_IQmag(_iq A,_iq B)
特定Q格式IQ函数（IQ格式=IQ1到IQ30）
_iqN_iqNmag(_iqN A,_iqN B)

输入 全局IQ函数（IQ格式=GLOBAL_Q）
输入的A和B的值是GLOBAL_Q格式的定点数
特定的Q格式IQ函数（IQ格式=IQ1到IQ30）
输入的A和B的值是IQN格式（N=1:30）格式的定点数

输出 全局IQ函数（IQ格式=GLOBAL_Q）
用全局Q格式表示的输入向量的幅值
用特定的Q格式表示的输入向量的幅值（IQ格式=IQ1到IQ30）

用法



例1：

下面的代码是求复数的幅值（假设GLOBAL_Q=IQ28）

```
#include<IQmathLib.h>          /* IQmath路径的头文件*_iq real1, imag1, mag1;          //复数
=real1+j*imag1
_iq28 real2, imag2, mag2;      //复数=real2+j*imag2
void main(void)
{
    real1=_IQ(4,0);
    imag1=_IQ(4,0);
    mag1=_iqmag(real1, imag1);    //在IQ28格式下，mag=5.6568
    real2=_IQ28(7.0);
    imag2=_IQ28(7.0);
    mag2=_IQ28mag(real2, imag2);  //饱和至最大值！！！ mag=~8.0
}
```

IQNabs: IQ数值的绝对值函数

| | |
|----|--|
| 描述 | 此函数能够计算一个IQ数值的绝对值 |
| 声明 | 全局IQ函数（IQ 格式=GLOBAL_Q） _iq_IQabs(_iq A) 特定Q格式IQ函数（IQ 格式=IQ1 到 IQ30） _iqN_IQNabs(_iqN A) |
| 输入 | 全局IQ函数（IQ 格式=GLOBAL_Q） 全局Q格式表示的IQ数 特定Q格式IQ函数（IQ 格式=IQ1 到 IQ30） IQN格式表示的IQ数 |
| 输出 | 全局IQ函数（IQ 格式=GLOBAL_Q） 全局Q格式表示的输入向量绝对值 特定Q格式IQ函数（IQ 格式=IQ1 到 IQ30） IQN格式表示的输入向量绝对值 |
| 用法 | |



例1:

计算3个IQ数的绝对值之和（GLOBAL_Q=IQ28）

```
_iq xin1, xin2, xin3, xsum;  
_iq20 yin1, yin2, yin3, ysum;  
  
xsum=_IQabs(X0)+_IQabs(X1)+IQabs(X2);  
xsum=_IQ28abs(X0)+_IQ28abs(X1)+IQabs(X2);
```

5.5.2 IQNsat:IQ数值的限幅函数

| | |
|----|---|
| 描述 | 此函数能够将一个IQ值限幅在给定的正负极限，经常应用在计算溢出的场合 |
| 声明 | <code>_iq_IQsat(_iq A, long P, long N)</code> |
| 输入 | 全局IQ函数（IQ 格式=GLOBAL_Q） 全局Q格式表示的IQ数 |
| 输出 | 全局IQ函数（IQ 格式=GLOBAL_Q） 全局Q格式表示的输入绝对值 |
| 用法 | |

“

例1:

计算线性方程“ $Y=M \times X+B$ ”，以饱和方式。

所有变量均有GLOBAL_Q=26，然而，变化范围可能产生溢出，所以必须完成计算并将结果饱和化

为此，应采取中间操作使IQ=20，并在将结果转化回至适当的全局Q格式值之前将其饱和化

```
_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=_IQ20sat(temp,_IQtoIQ20(MAX_IQ_PQS), _IQtoIQ20(MAX_IQ_NEG));

Y=_IQ20toIQ(temp);
```