IQmath Library

A Virtual Floating Point Engine

Module user's Guide

C28x Foundation Software

 Version 1.4.1

Revision History

Version	Date	Comment
V1.4.1	June 24, 2002	Original Draft Release
V1.4d	March 30, 2003	Corrected error in IQNfrac function description

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible or liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that products or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.
www.ti.com/sc/docs/stdterms.htm

Mailing Address:
Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Trademarks

TMS320 is the trademark of Texas Instruments Incorporated.
All other trademark mentioned herein are property of their respective companies

Acronyms

xDAIS : eXpress DSP Algorithm Interface Standard
IALG : Algorithm interface defines a framework independent interface for the creation of algorithm instance objects
STB : Software Test Bench
IQmath: High Accuracy Mathematical Functions (32-bit implementation).
QMATH: Fixed Point Mathematical computation
CcA : C-Callable Assembly
FIR : Finite Impulse Response Filter
IIR : Infinite Impulse Response Filter
FFT : Fast Fourier Transform

Contents

C28x IQmath LIBRARY BENCHMARKS

Function Name	IQ Format	Execution Cycles	Accuracy (in bits)	Program Memory	Input format	Output format	Remarks
Trigonometric Functions							
IQNsin	N=1 to 29	46	30 bits	49 words	IQN	IQN	
IQNsinPU	N=1 to 30	40	30 bits	41 words	IQN	IQN	
IQNcos	N=1 to 29	44	30 bits	47 words	IQN	IQN	
IQNcosPU	N=1 to 30	38	29 bits	39 words	IQN	IQN	
IQNatan2	N=1 to 29	109	26 bits	123 words	IQN	IQN	
IQNatan2PU	N=1 to 29	117	27 bits	136 words	IQN	IQN	
IQatan	N=1 to 29	109	25 bits	123 words	IQN	IQN	
Mathematical Functions							
IQNsqrt	N=1 to 30	63	29 bits	66 words	IQN	IQN	
IQNisqrt	N=1 to 30	64	29 bits	69 words	IQN	IQN	
IQNmag	N=1 to 30	86	29 bits	96 words	IQN	IQN	
Arithmetic Functions							
IQNmpy	N=1 to 30	~ 6 cycles	32 bits	NA	IQN*IQN	IQN	INTRINSIC
IQNrmpy	N=1 to 30	17	32 bits	13 words	IQN*IQN	IQN	
IQNrsmpy	N=1 to 30	21	32 bits	21 words	IQN*IQN	IQN	
IQNmpyI32	N=1 to 30	~ 4 cycles	32 bits	NA	IQN*long	IQN	C-MACRO
IQNmpyI32int	N=1 to 30	22	32 bits	16 words	IQN*long	long	
IQNmpyI32frac	N=1 to 30	24	32 bits	20 words	IQN*long	IQN	
IQNmpyIQX		~ 7 cycles	32 bits	NA	IQN1*IQN2	IQN	INTRINSIC
IQNdiv	N=1 to 30	63	28 bits	71 words	IQN/IQN	IQN	
Format Conversion Utilities							
IQN	N=1 to 30	NA	N/A	NA	Float	IQN	C-MACRO
IQNtoF	N=1 to 30	22	N/A	20 words	IQN	Float	
atoIQN	N=1 to 30	N/A	N/A	143 words	char *	IQN	
IQNint	N=1 to 30	14	32 bits	8	IQN	long	
IQNfrac	N=1 to 30	17	32 bits	12	IQN	IQN	
IQtoIQN	N=1 to 30	~4 cycles	N/A	N/A	GLOBAL_Q	IQN	C-MACRO
IQNtoIQ	N=1 to 30	~4 cycles	N/A	N/A	IQN	GLOBAL_Q	C-MACRO
IQtoQN	N=1 to 15	~4 cycles	N/A	N/A	GLOBAL_Q	QN	C-MACRO
QNtoIQ	N=1 to 15	~4 cycles	N/A	N/A	QN	GLOBAL_Q	C-MACRO
Miscellaneous							
IQsat	N=1 to 30	~7 cycles	N/A	N/A	IQN	IQN	INTRINSIC
IQNabs	N=1 to 30	~2 cycles	N/A	N/A	IQN	IQN	INTRINSIC

Notes:

- Execution cycles & Program memory usage mentioned in the Table assumes IQ24 format.
  - Execution cycles may vary by few cycles for some other IQ format.
  - Program memory may vary by few words for some other IQ format.
- Execution Cycles mentioned in the table includes the CALL and RETURN (LCR + LRETR) and it assumes that the IQmath table is loaded in internal memory.

Chapter 1: Introduction

1.1. Introduction

Texas Instruments TMS320C28x IQmath Library is collection of highly optimized and high precision mathematical Function Library for C/C++ programmers to seamlessly port the floating-point algorithm into fixed point code on TMS320C28x devices. These routines are typically used in computationally intensive real-time applications where optimal execution speed & high accuracy is critical. By using these routines you can achieve execution speeds considerable faster than equivalent code written in standard ANSI C language. In addition, by providing ready-to-use high precision functions, TI IQmath library can shorten significantly your DSP application development time.

Chapter 2: Installing IQmath Library

2.1 IQmath Content

The TI IQmath library offers usage in C/CPP program and it consists of 5 parts:

1) IQmath header file:
IQmathLib.h

2) IQmath object library containing all function & look-up tables
IQmath.lib

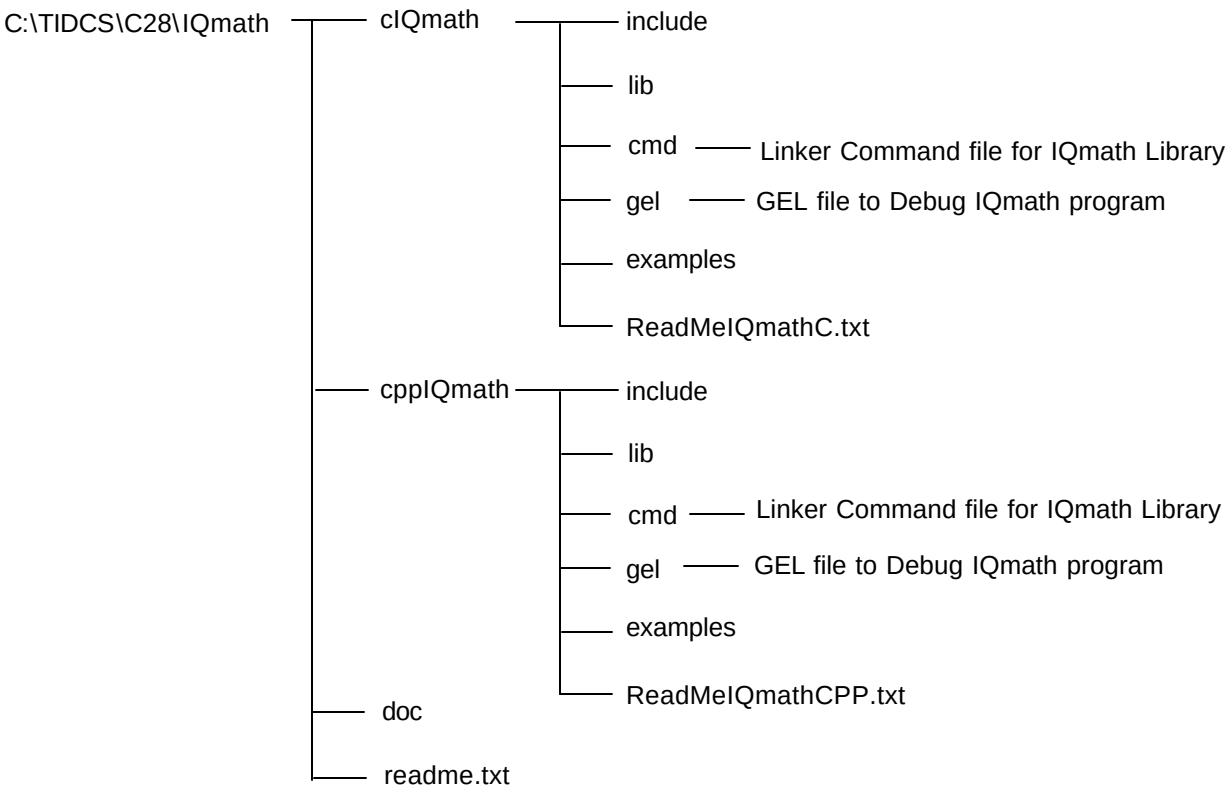
3) Linker Command File
IQmath.cmd

4) IQmath GEL file for debugging
IQmath.gel

5) Example programs

2.2 How to Install IQmath Library

IQmath library is distributed in the form of an self-extracting ZIP file. The zip file automatically restores the IQmath library individual components in the directory structure shown below. Read README.TXT File for Specific Details of Release



Chapter 3: Using IQmath Library

3.1. IQmath Arguments and Data Types

Input/output of the IQmath functions are typically 32-bit fixed-point numbers and the Q format of the fixed-point number can vary from Q1 to Q30.

We have used typedefs to create aliases for IQ data types. This facilitates the user to define the variable of IQmath data type in the application program.

```
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       */
```

3.2. IQmath Data type: Range & Resolution

Following table summarizes the Range & Resolution of 32-bit fixed-point number for different Q format representation. Typically IQmath function supports Q1 to Q30 format, nevertheless some function like IQNsin, IQNcos, IQNatan2, IQNatan2PU, IQatan does not support Q30 format, due to the fact that these functions input or output need to vary between $-p$ to p radians.

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. Calling a IQmath Function from C

In addition to installing the IQmath software, to include a IQmath function in your code you have to:

- Include the *IQmathLib.h* include file
- Link your code with the IQmath object code library, *IQmath.lib*.
- Use a correct linker command file to place "IQmath" section in program memory
- The section "IQmathTables" contains look-up tables for IQmath functions and it is available in the BOOTROM of F2810/F2812 devices. Hence, this section must be of set to "NOLOAD" type in the linker command. This facilitates referencing look-up table symbols, without actually loading the section into the target.

Note:
IQmath functions are assembled in ***"IQmath"*** *section & the look-up tables used to perform high precision computation are placed in* ***"IQmathTables"*** *section.*

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
}
```

For example, the following code contains a call to the ***IQ25sin*** routines in IQmath Library:

```
#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×p radians represented in Q29 format      */
     sin_out =_IQ29sin(input);
}
```

3.4. IQmath Function Naming Convention

Each IQmath function provides, two types of function handles, viz.,

- GLOBAL_Q function, that takes input/output in GLOBAL_Q format

  Examples:

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

- Q-format specific functions to cater to Q1 to Q30 data format.

  Examples:

  - _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. Selecting GLOBAL_Q format

Numerical precision and dynamic range requirement will vary considerably from one application to other. IQmath Library facilitates the application programming in fixed-point arithmetic, without fixing the numerical precision up-front. This allows the system engineer to check the application performance with different numerical precision and finally fix the numerical resolution. As explained in section 3.2, higher the precision results in lower dynamic range. Hence, the system designer must trade-off between the range and resolution before choosing the GLOBAL_Q format.

CASE I:

Default GLOBAL_Q format is set to Q24. Edit "IQmathLib.h" header file to modify this value as required, user can choose from Q1 to Q29 as GLOBAL_Q format. Note that modifying this value means that all the GLOBAL_Q functions will use this Q format for input/output, unless this symbolic definition is overridden in the source code.

IQmathLib.h : Selecting GLOBAL_Q format

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

CASE II :

A complete system consists of various modules. Some modules may require different precision, then the rest of the system. In such situation, we need to over-ride the GLOBAL_Q defined in the “IQmathLib.h” file and use the local Q format.

This can be easily done by defining the GLOBAL_Q constant in the source file of the module before the include statement.

MODULE6.C : Selecting Local Q format

```
#define  GLOBAL_Q      27      /*      Set the Local Q value      */
#include <IQmathLib.h>
```

3.6. Using IQmath GEL file for De-bugging

IQmath GEL file contains GEL functions that helps to view IQ variables in watch window and allows the setting of IQ variable values via dialogue boxes.

Step 1: Define “GlobalQ” variable

In one of the user source file, the following global variable must be defined:

long GlobalQ = GLOBAL_Q;

This variable is used by the GEL functions to determine the current GLOBAL_Q setting.

Step 2: Load GEL file

Load the "IQmath.gel" file into the user project. This will automatically load a set of GEL functions for displaying IQ variables in the watch window and create the following menus under the GEL toolbar

- IQ C Support
- IQ C++ Support

Step 3: Viewing IQmath variable

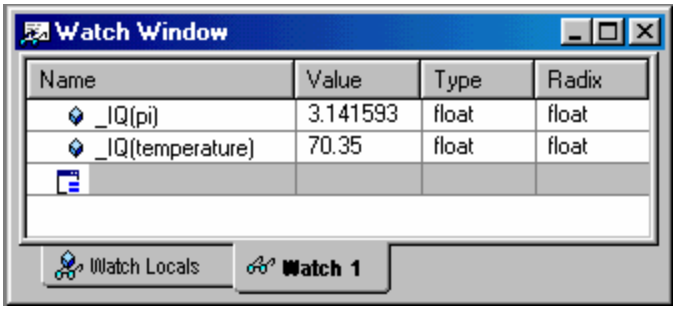
To view a variable in the watch window, simply type the following commands in the watch window. They will convert the specified "VarName" in IQ format to the equivalent floating-point value:

For C variables:

```
_IQ(VarName)        ; GLOBAL_Q value
_IQN(VarName)       ; N = 1 to 30
```

For C++ variables:

```
IQ(VarName)         ; GLOBAL_Q value
IQN(VarName)        ; N = 1 to 30
```

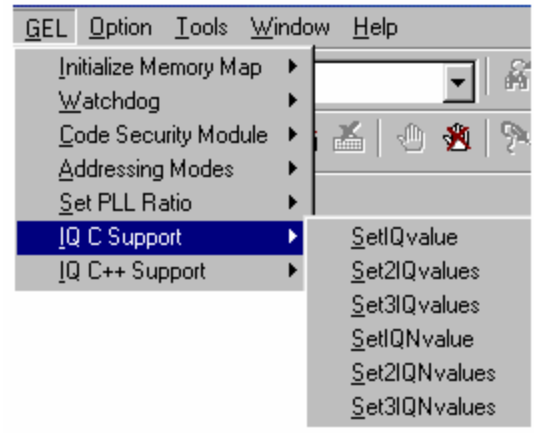


Step 4: Modifying IQmath variable

The watch window does not allow the modification of variables that are not of native type. To facilitate this, the following GEL operations can be found under the GEL toolbar:

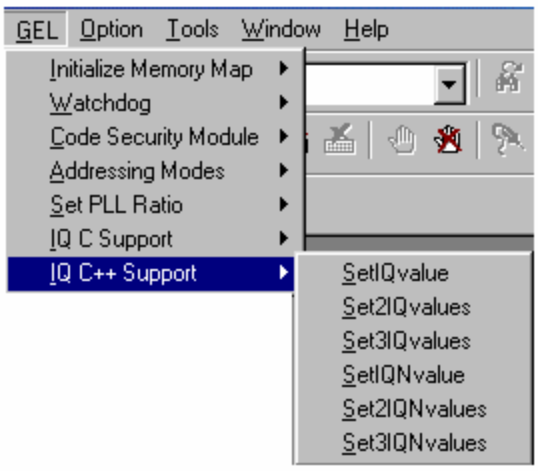
IQ C Support

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



IQ C++ Support

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



Invoking one of the above GEL operations will bring up a dialogue box window, which the user can enter the variable name and the floating-point value to set. The function will convert the float value to the appropriate IQ value.

Chapter 4: Function Summary

The routines included within the IQmath library are organized as follows

- Format conversion utilities : atoIQ, IQtoF, IQtoIQN etc.
- Arithmetic Functions : IQmpy, IQdiv etc.
- Trigonometric Functions : IQsin, IQcos, IQatan2 etc.
- Mathematical functions : IQsqrt, IQisqrt etc.
- Miscellaneous : IQabs, IQsat etc

4.1 Arguments and Conventions Used

The following convention has been followed when describing the arguments for each individual function:

QN	16-bit fixed point Q number, where N=1:15
IQN	32-bit fixed point Q number, where N=1:31
int	16-bit number
long	32-bit number
_iq	Data type definition equating a long, a 32-bit value representing a GLOBAL_Q number. Usage of _iq instead of long is recommended to increase future portability across devices.
_iqN	Data type definition equating a long, a 32-bit value representing a IQN number, where N=1:30
A, B	Input operand to IQmath function or Macro
F	Floating point input : Ex: -1.232, +22.433, 0.4343, -0.32
S	Floating point string: “+1.32”, “0.232”, “-2.343” etc
P	Positive Saturation value
N	Negative Saturation value

4.2. IQmath Functions

Format conversion Utilities:

Functions	Description	IQ format
_iq _IQ(float F) _iqN _IQN(float F)	Converts float to IQ value	Q=GLOBAL_Q Q=1:30
float _IQtoF(_iq A) float _IQNtoF(_iqN A)	IQ to Floating point	Q=GLOBAL_Q Q=1:30
_iq _atoIQ(char *S) _iqN _atoIQN(char *S)	Float ASCII string to IQ	Q=GLOBAL_Q Q=1:30
long _IQint(_iq A) long _IQNint(_iqN A)	extract integer portion of IQ	Q=GLOBAL_Q Q=1:30
_iq _IQfrac(_iq A) _iqN _IQNfrac(_iqN A)	extract fractional portion of IQ	Q=GLOBAL_Q Q=1:30
_iqN _IQtoIQN(_iq A)	Convert IQ number to IQN number (32-bit)	Q=GLOBAL_Q ,
_iq _IQNtoIQ(_iqN A)	Convert IQN (32-bit) number to IQ number	Q=GLOBAL_Q
int _IQtoQN(_iq A)	Convert IQ number to QN number (16-bit)	Q=GLOBAL_Q ,
_iq _QNtoIQ(int A)	Convert QN (16-bit) number to IQ number	Q=GLOBAL_Q

Arithmetic Operations:

Functions	Description	IQ format
_iq _IQmpy(_iq A, _iq B) _iqN _IQNmpy(_iqN A, _iqN B)	IQ Multiplication	Q=GLOBAL_Q Q=1:30
_iq _IQrmpy(_iq A, _iq B) _iqN _IQNrmpy(_iqN A, _iqN B)	IQ Multiplication with rounding	Q=GLOBAL_Q Q=1:30
_iq _IQrsmpy(_iq A, _iq B) _iqN _IQNrsmpy(_iqN A, _iqN B)	IQ multiplication with rounding & saturation	Q=GLOBAL_Q Q=1:30
_iq _IQmpyI32(_iq A, long B) _iqN _IQNmpyI32(_iqN A, long B)	Multiply IQ with "long" integer	Q=GLOBAL_Q Q=1:30
long _IQmpyI32int(_iq A, long B) long _IQNmpyI32int(_iqN A, long B)	Multiply IQ with "long", return integer part	Q=GLOBAL_Q Q=1:30
long _IQmpyI32frac(_iq A, long B) long _IQNmpyI32frac(_iqN A, long B)	Multiply IQ with "long", return fraction part	Q=GLOBAL_Q Q=1:30
_iq _IQmpyIQX(_iqN1 A, N1, _iqN2 B, N2) _iqN _IQmpyIQX(_iqN1 A, N1, _iqN2 B, N2)	Multiply two 2-different IQ number	Q=GLOBAL_Q Q=1:30
_iq _IQdiv(_iq A, _iq B) _iqN _IQNdiv(_iqN A, _iqN B)	Fixed point division	Q=GLOBAL_Q Q=1:30

Trignometric Functions:

Functions	Description	IQ format
_iq _IQsin(_iq A) _iqN _IQNsin(_iqN A)	High precision SIN (Input in radians)	Q=GLOBAL_Q Q=1:29
_iq _IQsinPU(_iq A) _iqN _IQNsinPU(_iqN A)	High precision SIN (input in per-unit)	Q=GLOBAL_Q Q=1:30
_iq _IQcos(_iq A) _iqN _IQNcos(_iqN A)	High precision COS (Input in radians)	Q=GLOBAL_Q Q=1:29
_iq _IQcosPU(_iq A) _iqN _IQNcosPU(_iqN A)	High precision COS (input in per-unit)	Q=GLOBAL_Q Q=1:30
_iq _IQatan2(_iq A, _iq B) _iqN _IQNatan2(_iqN A, _iqN B)	4-quadrant ATAN (output in radians)	Q=GLOBAL_Q Q=1:29
_iq _IQatan2PU(_iq A, _iq B) _iqN _IQNatanPU(_iqN A, _iqN B)	4-quadrant ATAN (output in per-unit)	Q=GLOBAL_Q Q=1:29
_iq _IQatan(_iq A, _iq B) _iqN _IQNatan(_iqN A, _iqN B)	Arctangent	Q=GLOBAL_Q Q=1:29

Mathematical Functions:

Functions	Description	IQ format
_iq _IQsqrt(_iq A) _iqN _IQNsqrt(_iqN A)	High precision square root	Q=GLOBAL_Q Q=1:30
_iq _IQisqrt(_iq A) _iqN _IQNisqrt(_iqN A)	High precision inverse square root	Q=GLOBAL_Q Q=1:30
_iq _IQmag(_iq A, _iq B) _iqN _IQNmag(_iqN A, _iqN B)	Magnitude Square: sqrt(A^2 + B^2)	Q=GLOBAL_Q Q=1:30

Miscellaneous

Functions	Description	Q format
_iq _IQsat(_iq A, long P, long N)	Saturate the IQ number	Q=GLOBAL_Q
_iq _IQabs(_iq A)	Absolute value of IQ number	Q=GLOBAL_Q

Chapter 5: Function Description

_IQN	*Float to IQN data type*

Description This C-Macro converts a floating-point constant or variable to the equivalent IQ value.

Declaration

Global IQ Macro (IQ format = GLOBAL_Q)
_iq _IQ(float F)

Q format specific IQ Macro (IQ format = IQ1 to IQ29)
_IQN _IQN(float F)

Input Floating point variable or constant

Output

Global IQ Macro (IQ format = GLOBAL_Q)
Fixed point equivalent of floating-point input in GLOBAL_Q format

Q format specific IQ Macro (IQ format = IQ1 to IQ29)
Fixed point equivalent of floating-point input in IQN format

Usage This operation is typically used to convert a floating-point constant or variable to the equivalent IQ value.

Example 1: Implementing equation in IQmath way

Floating point equation: Y = M*1.26 + 2.345
IQmath equation (Type 1): Y =_IQmpy(M, _IQ(1.26)) + _IQ(2.345)
IQmath equation (Type 2): Y =_IQ23mpy(M, _IQ23(1.26)) +_IQ23(2.345)

Example 2: Converting Floating point variable to IQ data type

```
float x=3.343;
_iq y1;
_iq23 y2

IQmath (Type 1):    y1=_IQ(x)
IQmath (Type 2):    y2=_IQ23(x)
```

Example 3: Initialing Global variables or Tables

IQmath (Type 1):

```
_iq Array[4] = {_IQ(1.0), _IQ(2.5) _IQ(-0.2345), _IQ(0.0) }
```

IQmath (Type 2):

```
_iq23 Array[4] = {_IQ23(1.0), _IQ23(2.5) _IQ23(-0.2345), _IQ23(0.0) }
```

IQNtoF	*Float to IQN data type*

Description

This function converts a IQ number to equivalent floating point value in IEEE 754 format.

Declaration

Global IQ function (IQ format = GLOBAL_Q)
float _IQtoF(_iq A)

Q format specific IQ function (IQ format = IQ1 to IQ30)
float _IQNtoF(_iqN A)

Input

Global IQ function (IQ format = GLOBAL_Q)
Fixed point IQ number in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Fixed point IQ number in IQN format.

Output

Floating point equivalent of fixed-point input.

Usage

This operation is typically used in cases where the user may wish to perform some operations in floating-point format or convert data back to floating-point for display purposes.

Example:
Converting array of IQ numbers to the equivalent floating-point values

```
_iq DataIQ[N];
float DataF[N];

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

atoIQN	*String to IQN*

Description This function converts a string to IQ number.

Declaration

Global IQ function (IQ format = GLOBAL_Q)
float _atoIQ(char *S)

Q format specific IQ function (IQ format = IQ1 to IQ30)
float _atoIQN(char *S)

Input This function recognizes (in order) an optional sign, a string of digits optionally containing a radix character.

Valid Input strings:
“12.23456”, “-12.23456”, “0.2345”, “0.0” , “0” , “127” , “-89”

Output The first unrecognized character ends the string and returns zero. If the input string converts to a number greater then the max/min values for the given Q value, then the returned value will be limited to the min/max values

Global IQ function (IQ format = GLOBAL_Q)
Fixed point equivalent of input string in GLOBAL_Q format

Q format specific IQ function (IQ format = IQ1 to IQ29)
Fixed point equivalent of input string in IQN format

Usage This is useful for programs that need to process user input or ASCII strings.

Example:
The following code prompts the user to enter the value X:

```
char buffer[N];
_iq X;

printf("Enter value X = ");
gets(buffer);
X = _atoIQ(buffer);                    // IQ value (GLOBAL_Q)
```

IQNint	*Integer part of IQN number*

Description This function returns the integer portion of IQ number.

Declaration

Global IQ function (IQ format = GLOBAL_Q)
long _IQint(_iq A)

Q format specific IQ function (IQ format = IQ1 to IQ30)
long _IQNint(_iqN A)

Input

Global IQ function (IQ format = GLOBAL_Q)
Fixed point IQ number in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Fixed point IQ number in IQN format.

Output Integer part of the IQ number

Usage

Example 1: Extracting Integer & fractional part of IQ number

Following example extracts the integer & fractional part of two IQ number

```
_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
```

Example 2: Building IQ number from integer & Fractional part

Following example shows how to rebuild the IQ value from the integer and fractional portions:

```
_iq Y;
long Yint;
_iq Yfrac;

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

IQNfrac	*Fractional part of IQN number*

Description This function returns the fractional portion of IQ number.

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQfrac(_iq A)

Q format specific IQ function (IQ format = IQ1 to IQ30)
iqN _IQNfrac (_iqN A)

Input

Global IQ function (IQ format = GLOBAL_Q)
Fixed point IQ number in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Fixed point IQ number in IQN format.

Output Fractional part of the IQ number

Usage

Example 1: Extracting Integer & fractional part of IQ number

Following example extracts integer & fractional part of two IQ numbers

```
_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
```

Example 2: Building IQ number from integer & Fractional part

Following example shows how to rebuild the IQ value from the integer and fractional portions:

```
_iq Y;
long Yint;
_iq Yfrac;

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

IQtoIQN	*GLOBAL_Q number to IQN*

Description This Macro converts an IQ number in GLOBAL_Q format to the specified IQ format.

Declaration _iqN _IQtoIQN(_iq A)

Input IQ number in GLOBAL_Q format

Output Equivalent value of input in IQN format

Usage This macro may be used in cases where a calculation may temporarily overflow the IQ value resolution and hence require a different IQ value to be used for the intermediate operations.

Example:
Following example calculates the magnitude of complex number (X+jY) in Q26 format:

Z = sqrt(X^2 + Y^2)

The values Z, X, Y are given as GLOBAL_Q = 26, but the equation itself may generate an overflow.

To guard against this, the intermediate calculations will be performed using Q = 23 and the value converted back at the end as shown below:

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

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

Y = _IQ23toIQ(temp);
```

IQNtoIQ	*IQN number to GLOBAL_Q*

Description This Macro converts an IQ number in IQN format to the GLOBAL_Q format.

Declaration _iq _IQNtoIQ(_iqN A)

Input IQ number in IQN format

Output Equivalent value of input in GLOBAL_Q format

Usage This macro may be used in cases where the result of the calculation performed in different IQ resolution to be converted to GLOBAL_Q format.

Example:
Following example calculates the magnitude of complex number (X+jY) in Q26 format:

Z = sqrt(X^2 + Y^2)

The values Z, X, Y are given as GLOBAL_Q = 26, but the equation itself may generate an overflow.

To guard against this, the intermediate calculations will be performed using Q = 23 and the value converted back at the end as shown below:

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

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

Y = _IQ23toIQ(temp);
```

IQtoQN	*GLOBAL_Q number to QN*

Description This Macro converts a 32-bit number in GLOBAL_Q format to 16-bit number in QN format.

Declaration int _IQtoQN(_iq A)

Input IQ number in GLOBAL_Q format

Output Equivalent value of input in QN format (16-bit fixed point number)

Usage This macro may be used in cases where the input and output data is 16-bits, but the intermediate operations are operated using IQ data types.

Example:
Sum of product computation using the input sequence that is not in GLOBAL_Q format:

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

We can convert the Q15 values to IQ, perform the intermediate sums using IQ and then store the result back as Q15:

```
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);
```

QNtoIQ	*QN number to GLOBAL_Q*

Description This Macro converts a 16-bit number in QN format to 32-bit number in GLOBAL_Q format.

Declaration _iq _QNtoIQ(int A)

Input 16-bit fixed point number in QN format

Output Equivalent value of input in GLOBAL_Q format

Usage This macro may be used in cases where the input and output data is 16-bits, but the intermediate operations are operated using IQ data types.

Example:
Sum of product computation using the input sequence that is not in GLOBAL_Q format:

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

We can convert the Q15 values to IQ, perform the intermediate sums using IQ and then store the result back as Q15:

```
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);
```

IQNmpy	*IQ Multiplication(IQN*IQN)*

Description

This “C” compiler intrinsic multiplies two IQ number. It does not perform saturation and rounding. In most cases, the multiplication of two IQ variables will not exceed the range of the IQ variable. This operation takes the least amount of cycles and code size and should be used most often.

Declaration

Global IQ intrinsic (IQ format = GLOBAL_Q)
_iq _IQmpy(_iq A, _iq B)

Q format specific IQ intrinsic (IQ format = IQ1 to IQ30)
_iqN _IQNmpy(_iqN A, _iqN B)

Input Format

Global IQ intrinsic (IQ format = GLOBAL_Q)
Input “A” & “B” are IQ number in GLOBAL_Q format

Q format specific IQ intrinsic (IQ format = IQ1 to IQ30)
Input “A” & “B” are IQ number in IQN format

Output Format

Global IQ intrinsic (IQ format = GLOBAL_Q)
Result of multiplication in GLOBAL_Q format

Q format specific IQ intrinsic (IQ format = IQ1 to IQ30)
Result of multiplication in IQN format.

Usage

Example 1:
Following code computes "Y = M*X + B" in GLOBAL_Q format with no rounding or saturation:

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

Example 2:
Following code computes "Y = M*X + B" in IQ10 format with no rounding or saturation, assuming M, X, B are represented in IQ10 format:

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

IQNrmpy	*IQ Multiplication with rounding (IQN*IQN)*

Description

This function multiplies two IQ number and rounds the result. In cases where absolute accuracy is necessary, this operation performs the IQ multiply and rounds the result before storing back as an IQ number. This gives an additional 1/2 LSBit of accuracy.

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQrmpy(_iq A, _iq B)

Q format specific IQ function (IQ format = IQ1 to IQ30)
_iqN _IQNrmpy(_iqN A, _iqN B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input “A” & “B” are IQ number in GLOBAL_Q format

Q format specific IQ function (IQ format = IQ1 to IQ30)
Input “A” & “B” are IQ number in IQN format

Output Format

Global IQ function (IQ format = GLOBAL_Q)
Result of multiplication in GLOBAL_Q format

Q format specific IQ function (IQ format = IQ1 to IQ30)
Result of multiplication in IQN format.

Usage

Example 1:
Following code computes "Y = M*X + B" in GLOBAL_Q format with rounding but no saturation:

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

Example 2:
Following code computes "Y = M*X + B" in IQ10 format with rounding but no saturation:

```
_iq10 Y, M, X, B;
Y = _IQ10rmpy(M,X) + B;
```

IQNrsmpy	*IQ Multiplication with rounding & saturation (IQN*IQN)*

Description

This function multiplies two IQ number with rounding and saturation. In cases where the calculation may possibly exceed the range of the IQ variable, then this operation will round and then saturate the result to the maximum IQ value range before storing.

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQrsmpy(_iq A, _iq B)

Q format specific IQ function (IQ format = IQ1 to IQ30)
_iqN _IQNrsmpy(_iqN A, _iqN B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input “A” & “B” are IQ number in GLOBAL_Q format

Q format specific IQ function (IQ format = IQ1 to IQ30)
Input “A” & “B” are IQ number in IQN format

Output Format

Global IQ function (IQ format = GLOBAL_Q)
Result of multiplication in GLOBAL_Q format

Q format specific IQ function (IQ format = IQ1 to IQ30)
Result of multiplication in IQN format.

Usage

Let us assume that we use IQ26 are GLOBAL_Q format. This means that the range of the numbers is appx [-32.0, 32.0] (Refer section 3.2). If two IQ variables are multiplied together, then the maximum range of the result is [-1024, 1024]. This operation would make sure that the result is saturated to +/- 32 in cases where the result exceeds this.

Example 1:
Following code computes "Y = M*X" in GLOBAL_Q format with rounding and saturation (Assuming 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, output is Saturated to MAX
```

Example 2:
Following code computes "Y = M*X" in IQ26 format with rounding and saturation:

```
_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, output is Saturated to MIN
```

IQNmpyI32	*Multiplication (IQN*LONG)*

Description This macro multiplies an IQ number with a long integer.

Declaration

Global IQ Macro (IQ format = GLOBAL_Q)
_iq _IQmpyI32(_iq A, long B)

Q format specific IQ Macro (IQ format = IQ1 to IQ30)
_iqN _IQNmpyI32(_iqN A, long B)

Input Format

Global IQ Macro (IQ format = GLOBAL_Q)
Operand “A” is an IQ number in GLOBAL_Q format and “B” is the long integer.

Q format specific IQ Macro (IQ format = IQ1 to IQ30)
Operand “A” is an IQ number in IQN format and “B” is the long integer.

Output Format

Global IQ Macro (IQ format = GLOBAL_Q)
Result of multiplication in GLOBAL_Q format

Q format specific IQ Macro (IQ format = IQ1 to IQ30)
Result of multiplication in IQN format.

Usage

Example 1:
Following code computes "Y = 5*X" in GLOBAL_Q format (assuming 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
```

Example 2:
Following code computes "Y = 5*X" in IQ26 format

```
_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
```

IQNmpyI32int	*Integer portion of (IQN*LONG)*

Description

This function multiplies an IQ number with a long integer and returns the integer part of the result.

Declaration

Global IQ function (IQ format = GLOBAL_Q)
long _IQmpyI32int(_iq A, long B)

Q format specific IQ function (IQ format = IQ1 to IQ30)
long _IQNmpyI32int(_iqN A, long B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Operand “A” is an IQ number in GLOBAL_Q format and “B” is the long integer.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Operand “A” is an IQ number in IQN format and “B” is the long integer.

Output Format

Global IQ function (IQ format = GLOBAL_Q)
Integer part of the result (32-bit)

Q format specific IQ function (IQ format = IQ1 to IQ30)
Integer part of the result (32-bit)

Usage

Example 1:
Convert an IQ value in the range [- 1.0, +1.0] to a DAC value with the range [0 to 1023]:

```
_iq Output;
long temp;
short OutputDAC;

temp = _IQmpyI32int(Output, 512);      // value converted to +/- 512
temp += 512;                           // value scaled to 0 to 1023

if( temp > 1023 )                      // saturate within range of DAC
temp = 1023;

if( temp < 0 )
temp = 0;

OutputDAC = (int )temp;                // output to DAC value
```

Note: The integer operation performs the multiply and calculates the integer portion from the resulting 64-bit calculation. Hence it avoids any overflow conditions.

IQNmpyI32frac	*Fractional Part of (IQN*LONG)*

Description

This function multiplies an IQ number with a long integer and returns the fractional part of the result.

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQmpyI32frac(_iq A, long B)

Q format specific IQ function (IQ format = IQ1 to IQ30)
_iqN _IQNmpyI32frac(_iqN A, long B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Operand “A” is an IQ number in GLOBAL_Q format and “B” is the long integer.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Operand “A” is an IQ number in IQN format and “B” is the long integer.

Output Format

Global IQ function (IQ format = GLOBAL_Q)
Fractional part of the result (32-bit)

Q format specific IQ function (IQ format = IQ1 to IQ30)
Fractional part of the result (32-bit)

Usage

Example 1:
Following example extracts the fractional part of result after multiplication (Assuming GLOBAL_Q=IQ26)

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

Y1frac = IQmpyI32frac(X1, M1);          // Y1frac = 0.5 in GLOBAL_Q
Y2frac = IQ26mpyI32frac(X2, M2);        // Y2frac = -0.9 in GLOBAL_Q
```

IQNmpyIQX	*Multiplication (GLOBAL_Q=IQN1*IQN2)*

Description

This "C" compiler intrinsic multiplies two IQ number that are represented in different IQ format

Declaration

Global IQ Intrinsic (IQ format = GLOBAL_Q)
_iq _IQmpyIQX(_iqN1 A, int N1, _iqN2 B, int N2)

Q format specific IQ Intrinsic (IQ format = IQ1 to IQ30)
_iqN _IQNmpyIQX(_iqN1 A, int N1, _iqN2 B, int N2)

Input Format

Operand "A" is an IQ number in "IQN1" format and operand "B" is in "IQN2" format.

Output Format

Global IQ Intrinsic (IQ format = GLOBAL_Q)
Result of the multiplication in GLOBAL_Q format

Q format specific IQ Intrinsic (IQ format = IQ1 to IQ30)
Result of the multiplication in IQN format

Usage

This operation is useful when we wish to multiply values of different IQ.

Example:
We wish to calculate the following equation:
Y = X0*C0 + X1*C1 + X2*C2
Where,
X0, X1, X2 values are in IQ30 format (Range -2 to +2)
C0, C1, C2 values are in IQ28 format (Range –8 to +8)

Maximum range of Y will be -48 to +48, Hence we should store the result in IQ format that is less then IQ25.

Case 1: GLOBAL_Q=IQ25

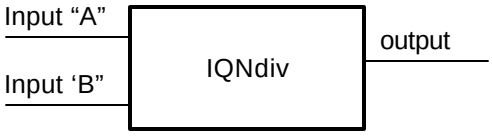
```
_iq30 X0, X1, X2;                    // All values IQ30
_iq28 C0, C1, C2;                    // All values IQ28
_iq Y;                               // Result GLOBAL_Q = IQ25

Y = _IQmpyIQX(X0, 30, C0, 28)
Y += _IQmpyIQX(X1, 30, C1, 28)
Y += _IQmpyIQX(X2, 30, C2, 28)
```

Case 2: IQ Specific computation

```
_iq30 X0, X1, X2;                    // All values IQ30
_iq28 C0, C1, C2;                    // All values IQ28
_iq25 Y;                             // Result GLOBAL_Q = IQ25

Y = _IQ25mpyIQX(X0, 30, C0, 28)
Y += _IQ25mpyIQX(X1, 30, C1, 28)
Y += _IQ25mpyIQX(X2, 30, C2, 28)
```

IQNdiv	*Fixed point division*

Description This module divides two IQN number and provide 32-bit quotient (IQN format) using Newton-Raphson technique



Availability C-Callable Assembly (CcA)

Module Properties **Type:** Target Independent, Application Independent

Target Devices: x28xx

C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

Item	C-Callable ASM	Comments
Code Size	71 words	
Data RAM	0 words	
Multiple instances	N/A	
Reentrancy	Yes	
Multiple Invocation	Yes	
Stack usage	2 words	Stack grows by 2 words

Accuracy $= 20\log_2\left(2^{31}\right) - 20\log_2(7)$
= 28 bits

C/C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQdiv(_iq A, _iq B)

Q format specific IQ function (IQ format = IQ1 to IQ30)
_iqN _IQNdiv(_iqN A, iq B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input “A” & “B” are fixed-point number represented in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Input ‘A’ & ‘B’ are fixed-point number in IQN format (N=1:30)

Output Format

Global IQ function (IQ format = GLOBAL_Q)
Output in GLOBAL_Q format.

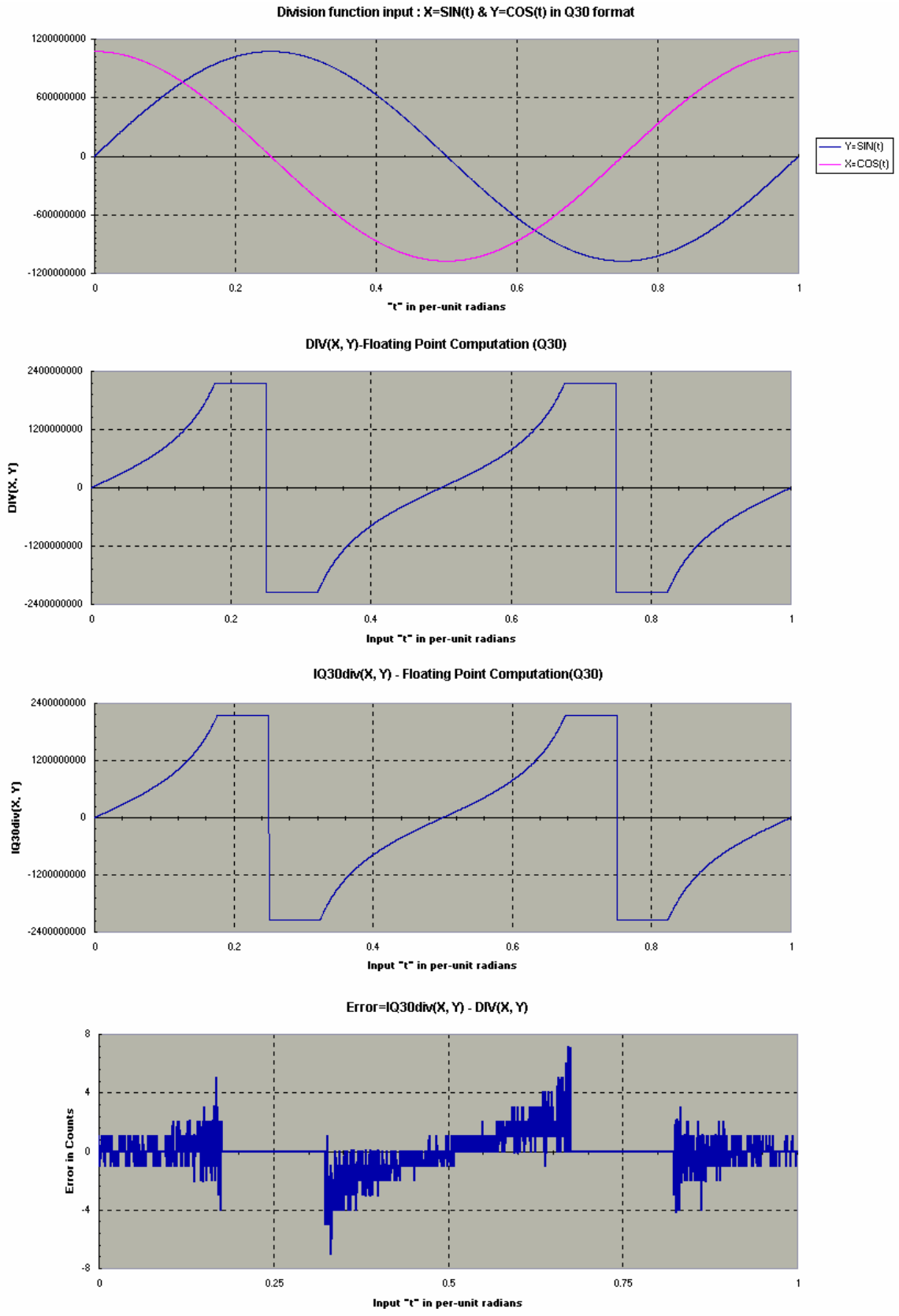
Q format specific IQ function (IQ format = IQ1 to IQ30)
Output in IQN format (N=1:30)

Example

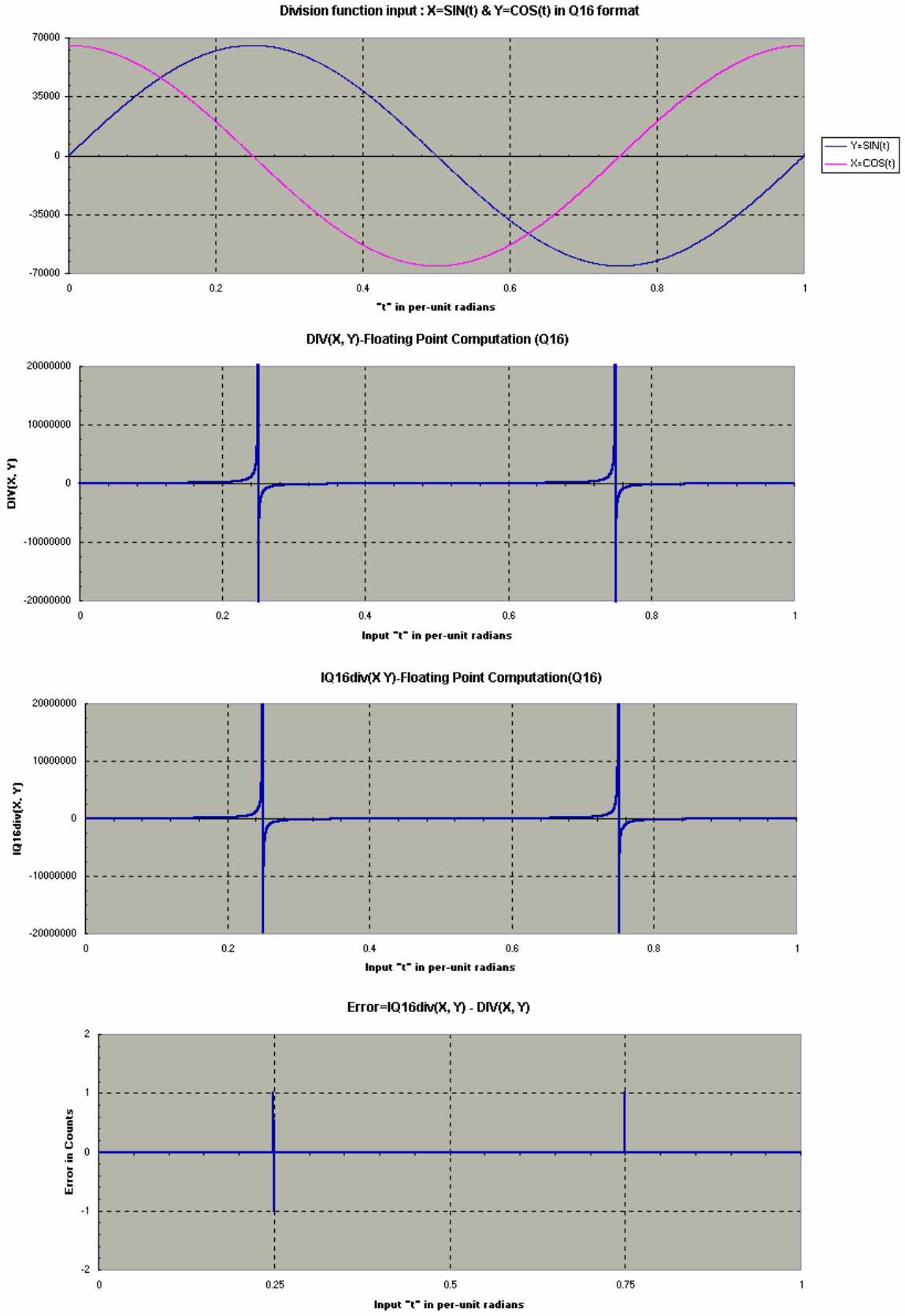
The following example obtains $1/15$ =0.666 assuming that GLOBAL_Q is set to Q28 format in IQmath header file.

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

_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);
}
```

Fixed Point division vs C Float division



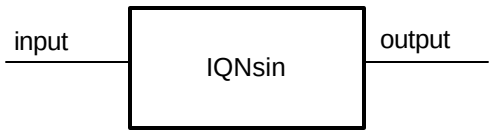
Fixed Point division vs C Float division



IQNsin	*Fixed point SIN (radians)*

Description

This module computes the sine value of the input (in radians) using table look-up and Taylor series expansion between the look-up table entries.



Availability

C/C++ Callable Assembly

Module Properties

Type: Target Independent, Application Independent

Target Devices: x28xx

C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

Item	C-Callable ASM	Comments
Code Size	49 words	
Data RAM	0 words	
Multiple instances	N/A	
Reentrancy	Yes	
Multiple Invocation	Yes	
Stack usage	2 words	Stack grows by 2 words

Accuracy

$= 20\log_2\left(p \times 2^{29}\right) - 20\log_2(1)$

= 30 bits

C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQsin(_iq A)

Q format specific IQ function (IQ format = IQ1 to IQ29)
_iqN _IQNsin(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input argument is in radians and represented as fixed-point number in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ29)
Input argument is in radians and represented as fixed-point number in IQN format (N=1:29).

Output Format

Global IQ function (IQ format = GLOBAL_Q)
This function returns the sine of the input argument as fixed-point number in GLOBAL_Q format.

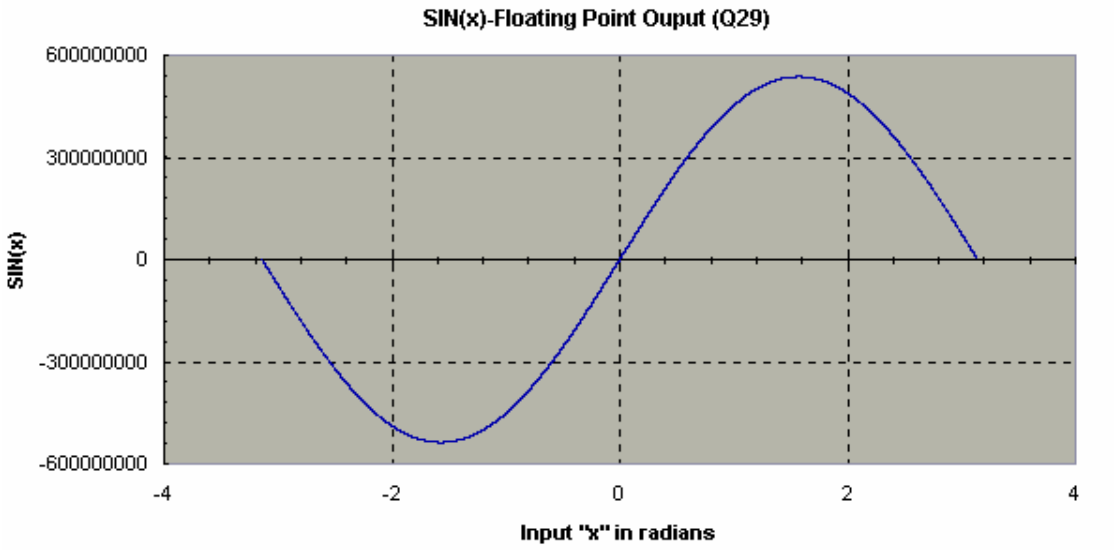
Q format specific IQ function (IQ format = IQ1 to IQ29)
This function returns the sine of the input argument as fixed-point number in IQN format (N=1:29)

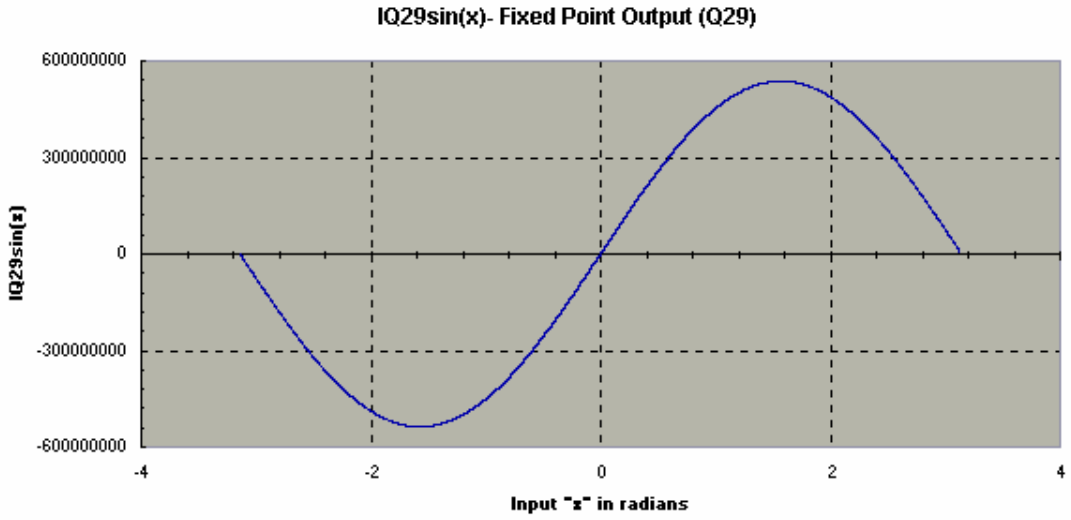
Example

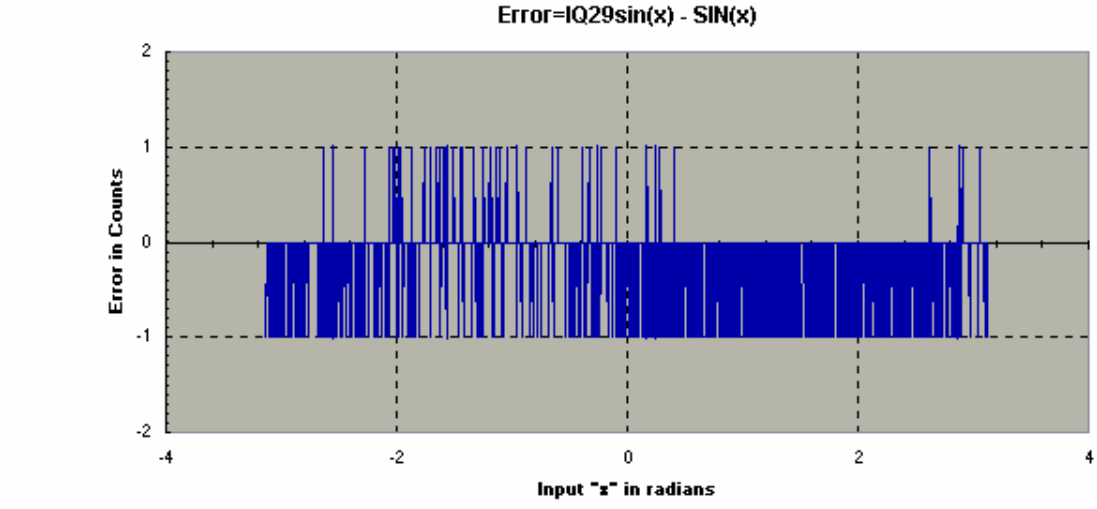
The following example obtains the $\sin(0.25\times p)$=0.707 assuming that GLOBAL_Q is set to Q29 format in the IQmath header file.

```
#include<IQmathLib.h>          /* Header file for IQmath routine                    */
#define   PI     3.14156

_iq in1, out1;
_iq28 in2, out2;

void main(void )
{
 in1=_IQ(0.25*PI);             /* in1= 0.25×p×2^29 = 1921FB54h                     */
 out1=_IQsin(in1)              /*  out1= sin(0.25×p)×2^29 = 16A09E66h              */

 in2=_IQ29(0.25*PI)            /* in2= 0.25×p×2^29 = 1921FB54h                     */
 out2=_IQ29sin(in2);           /*  out2= sin(0.25×p)×2^29 = 16A09E66h              */
}
```

IQNsin Function vs C Float SIN: Input varies from $-p$ to p



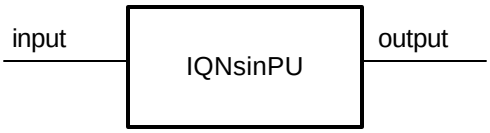




IQNsinPU	Fixed point SIN (radians in per unit)

Description

This module computes the sine value of the input (in per-unit radians) using table look-up and Taylor series expansion between the look-up table entries.



Availability

C/C++ Callable Assembly

Module Properties

Type: Target Independent, Application Independent

Target Devices: x28xx

C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

Item	C-Callable ASM	Comments
Code Size	41 words	
Data RAM	0 words	
Multiple instances	N/A	
Reentrancy	Yes	
Multiple Invocation	Yes	
Stack usage	2 words	Stack grows by 2 words

Accuracy

$= 20\log_2\left(1\times 2^{30}\right) - 20\log_2(1)$

= 30 bits

C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQsinPU(_iq A)

Q format specific IQ function (IQ format = IQ1 to IQ30)
_iqN _IQNsinPU(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input argument is in per-unit radians and represented as fixed-point number in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Input argument is in per-unit radians and represented as fixed-point number in IQN format (N=1:30).

Output Format

Global IQ function (IQ format = GLOBAL_Q)
This function returns the sine of the input argument as fixed-point number in GLOBAL_Q format.

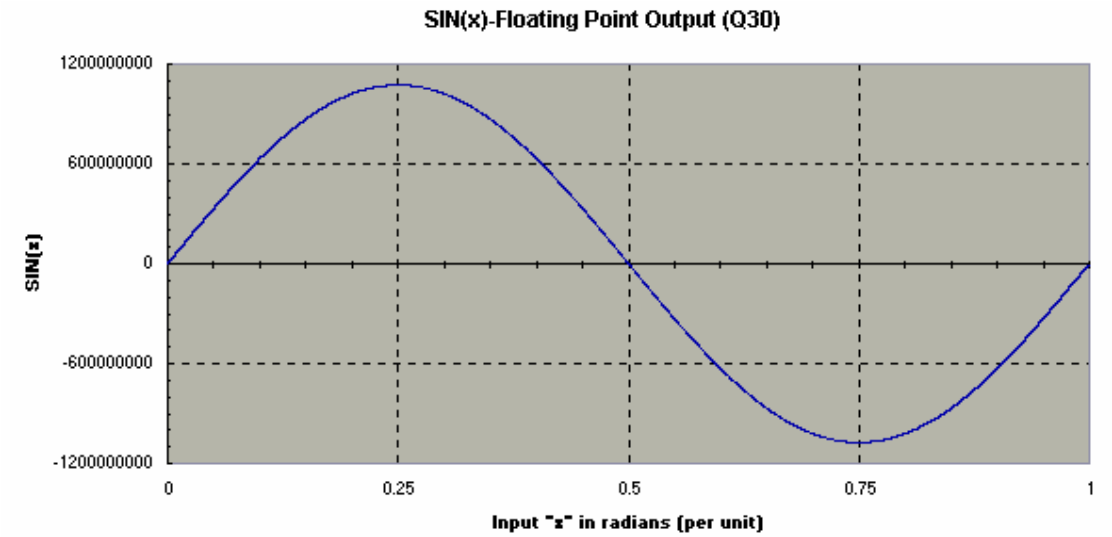
Q format specific IQ function (IQ format = IQ1 to IQ30)
This function returns the sine of the input argument as fixed-point number in IQN format (N=1:30)

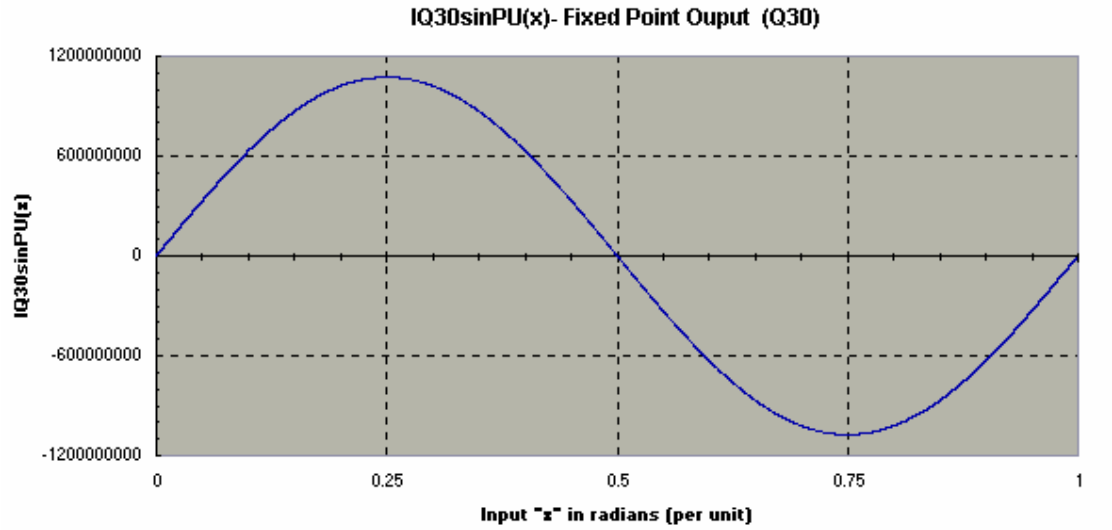
Example

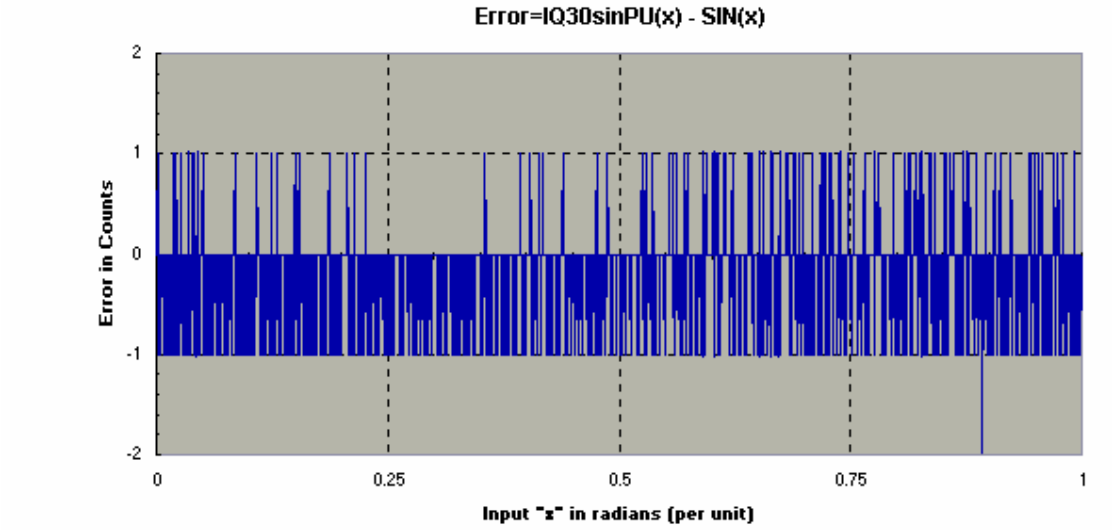
The following example obtains the $\sin(0.25\times p)$=0.707 assuming that GLOBAL_Q is set to Q30 format in the IQmath header file.

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

#define   PI     3.14156

_iq in1, out1;
_iq30 in2, out2;

void main(void )
{
 in1=_IQ(0.25*PI/PI);          /* in1 = 0.25×p/2p × 2^30 = 08000000h              */
 out1=_IQsinPU(in1)            /* out1= sin(0.25×p)× 2^30 = 2D413CCCh             */

 in2=_IQ30(0.25*PI/PI);        /* in2 = 0.25×p/2p × 2^30 = 08000000h              */
 out2=_IQ30sinPU(in2);         /* out2= sin(0.25×p)× 2^30 = 2D413CCCh             */
}
```

IQNsinPU Function vs C Float SIN: Input varies from 0 to $2p$ in per unit representation



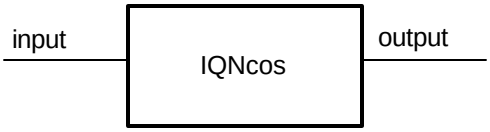




IQNcos	Fixed point COS (radians)

Description

This module computes the cosine value of the input (in radians) using table look-up and Taylor series expansion between the look up table entries.



Availability

C/C++ Callable Assembly

Module Properties

Type: Target Independent, Application Independent

Target Devices: x28xx

C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

Item	C-Callable ASM	Comments
Code Size	47 words	
Data RAM	0 words	
Multiple instances	N/A	
Reentrancy	Yes	
Multiple Invocation	Yes	
Stack usage	2 words	Stack grows by 2 words

Accuracy

$= 20\log_2\left(p \times 2^{29}\right) - 20\log_2(2)$

= 30 bits

C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQcos(_iq A)

Q format specific IQ function (IQ format = IQ1 to IQ29)
_iqN _IQNcos(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input argument is in radians and represented as fixed-point number in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ29)
Input argument is in radians and represented as fixed-point number in IQN format (N=1:29).

Output Format

Global IQ function (IQ format = GLOBAL_Q)
This function returns the cosine of the input argument as fixed-point number in GLOBAL_Q format.

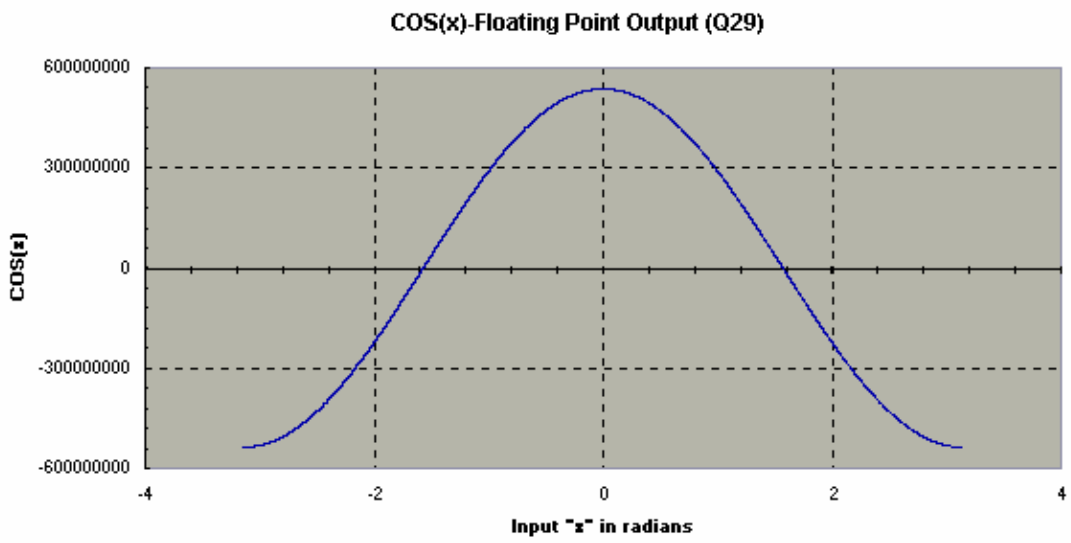
Q format specific IQ function (IQ format = IQ1 to IQ29)
This function returns the cosine of the input argument as fixed-point number in IQN format (N=1:29)

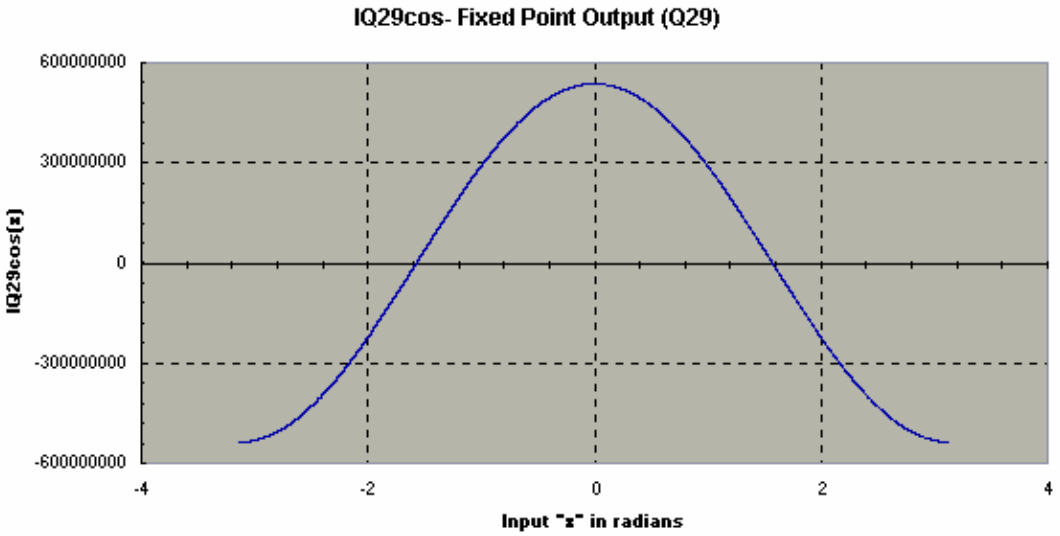
Example

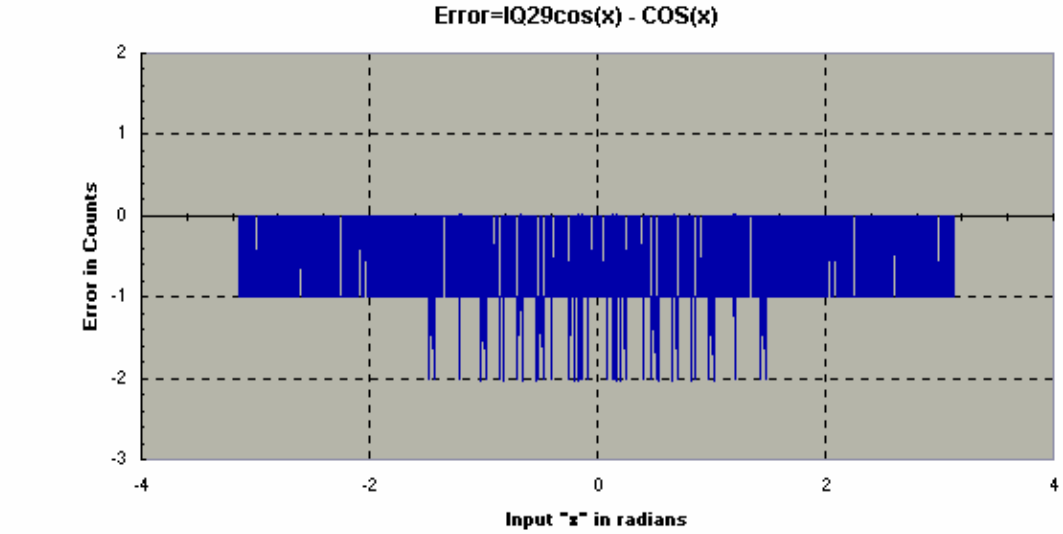
The following example obtains the $\cos(0.25\times p)$=0.707 assuming that GLOBAL_Q is set to Q29 format in the IQmath header file.

```
#include<IQmathLib.h>          /* Header file for IQmath routine
        */
#define   PI     3.14156

_iq in1, out1;
_iq29 in2 out2;

void main(void )
{
 in1=_IQ(0.25*PI);             /* in= 0.25×p×2^29 = 1921FB54h      */
 out1=_IQcos(in1);             /* out1= cos(0.25×p)×2^29 = 16A09E66h */

 in2=_IQ29(0.25*PI);           /* in2= 0.25×p×2^29 = 1921FB54h     */
 out2=_IQ29cos(in2);           /* out2= cos(0.25×p)×2^29 = 16A09E66h */
}
```

Fixed Point COS Function vs C Float COS: Input varies from $-p$ to p



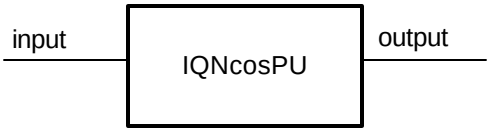




IQNcosPU	*Fixed point COS (radians in per unit)*

Description

This module computes the cosine value of the input (in per-unit radians) using table look-up and Taylor series expansion between the look up table entries.



Availability

C/C++ Callable Assembly

Module Properties

Type: Target Independent, Application Independent

Target Devices: x28xx

C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

Item	C-Callable ASM	Comments
Code Size	39 words	
Data RAM	0 words	
Multiple instances	N/A	
Reentrancy	Yes	
Multiple Invocation	Yes	
Stack usage	2 words	Stack grows by 2 words

Accuracy

$= 20\log_2\left(1\times 2^{30}\right) - 20\log_2(2)$

= 29 bits

C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQcosPU(_iq A)

Q format specific IQ function (IQ format = IQ1 to IQ30)
_iqN _IQNcosPU(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input argument is in per-unit radians and represented as fixed-point number in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Input argument is in per-unit radians and represented as fixed-point number in IQN format (N=1:30).

Output Format

Global IQ function (IQ format = GLOBAL_Q)
This function returns the sine of the input argument as fixed-point number in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
This function returns the sine of the input argument as fixed-point number in IQN format (N=1:30)

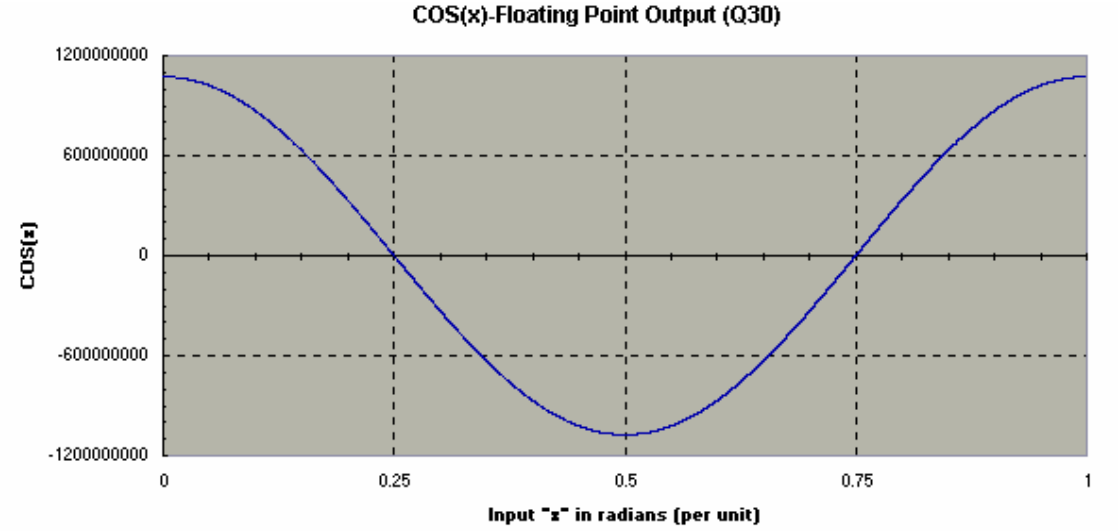
Example

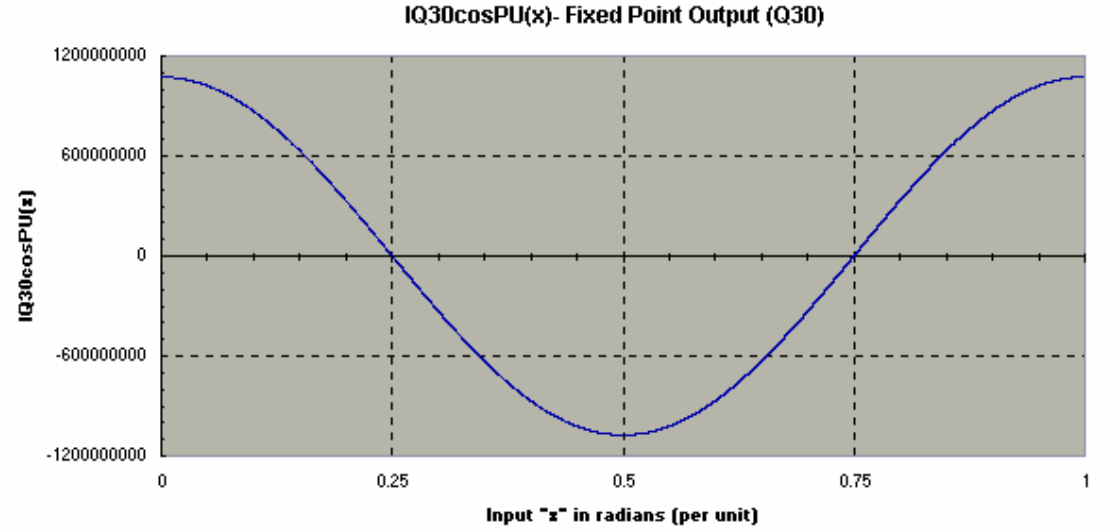
The following sample code obtains the $\cos(0.25\times p)$=0.707 assuming that GLOBAL_Q is set to Q30 format in the IQmath header file.

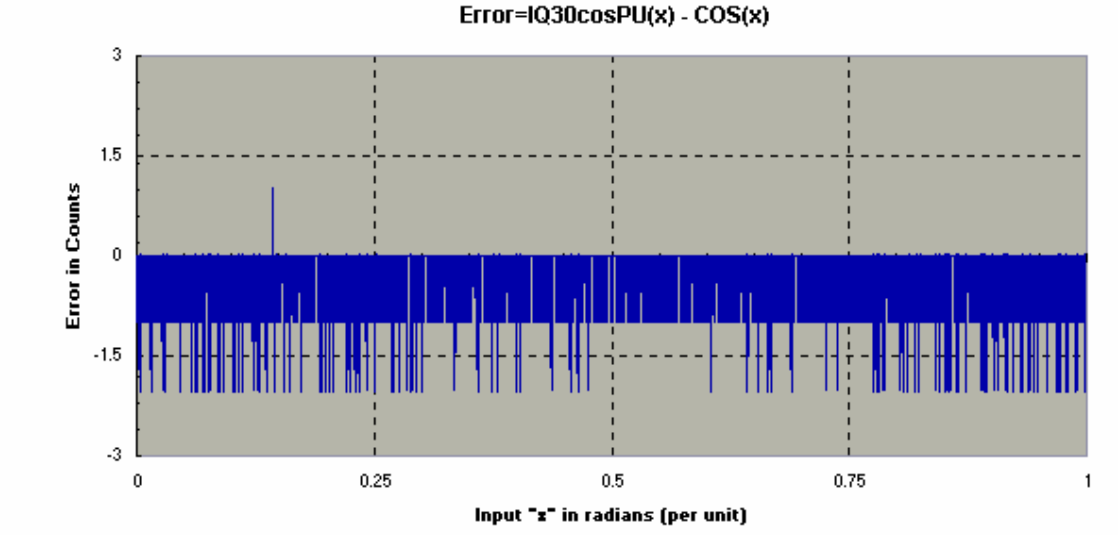
Sample Code

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

#define    PI      3.14156

_iq in1, out1;
_iq30 in2, out2

void main(void )
{
 in1=_IQ(0.25*PI/PI);          /* in1 = 0.25×p/2p × 2^30 = 08000000h        */
 out1=_IQcosPU(in1)            /* out1== cos(0.25×p)×2^30 = 2D413CCCh       */

 in2=_IQ30(0.25*PI/PI);        /* in2 = 0.25×p/2p × 2^30 = 08000000h        */
 out2=_IQ30cosPU(in2);         /* out2== cos(0.25×p)×2^30 = 2D413CCCh       */
}
```

Fixed Point COS Function vs C Float COS: Input varies from 0 to $2p$ in per unit representation



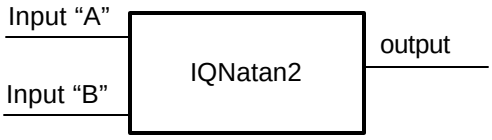




IQNatan2	*Fixed point 4-quadrant ATAN (in radians)*

Description

This module computes 4-quadrant arctangent. Output of this module is in radians that varies from $-p$ to p



Availability

C/C++ Callable Assembly

Module Properties

Type: Target Independent, Application Independent

Target Devices: x28xx

C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

Item	C-Callable ASM	Comments
Code Size	123 words	
Data RAM	0 words	
Multiple instances	N/A	
Reentrancy	Yes	
Multiple Invocation	Yes	
Stack usage	2 words	Stack grows by 2 words

Accuracy

$= 20\log_2\left(p \times 2^{29}\right) - 20\log_2(32)$

= 26 bits

C/C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQatan2(_iq A, _iq B)

Q format specific IQ function (IQ format = IQ1 to IQ29)
_iqN _IQNatan2(_iqN A, _iqN B), where the Q format "N" can vary from 1 to 29

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input "A" & "B" are fixed-point number represented in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ29)
Input 'A' & 'B' are fixed-point number in IQN format (N=1:29)

Output Format

Global IQ function (IQ format = GLOBAL_Q)
This function returns the inverse tangent of the input argument as fixed-point number in GLOBAL_Q format. The output contains the angle in radians between $[-\pi, +\pi]$

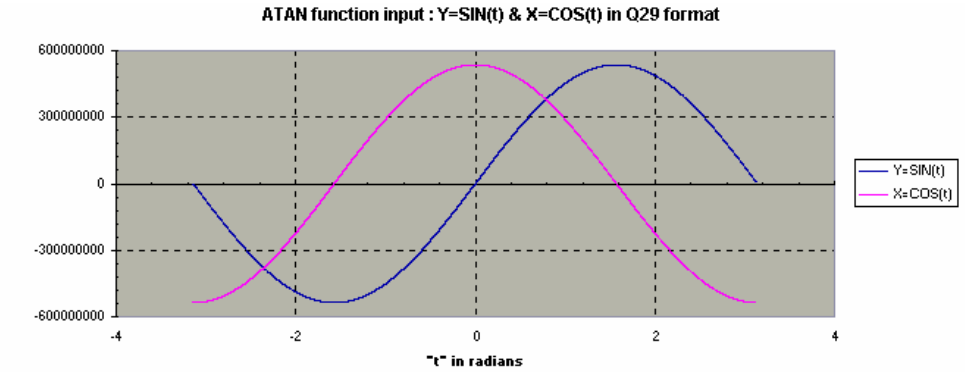
Q format specific IQ function (IQ format = IQ1 to IQ29)
This function returns the inverse tangent of the input argument as fixed-point number in IQN format (N=1:29). The output contains the angle in radians between $[-\pi, +\pi]$

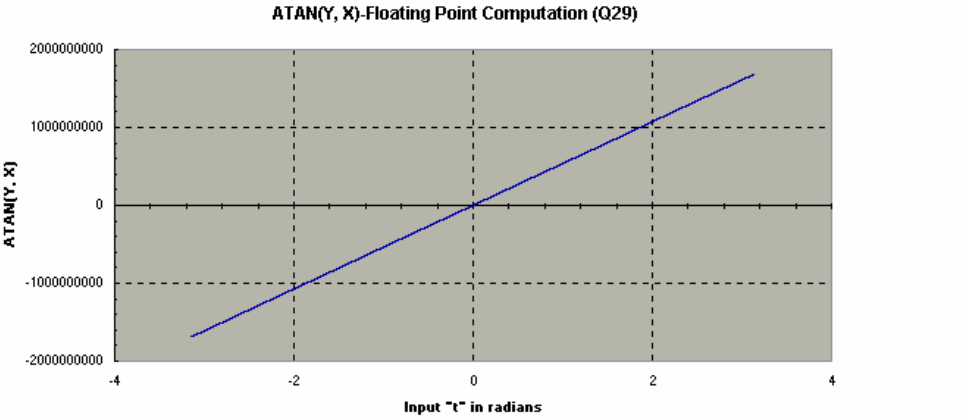
Example

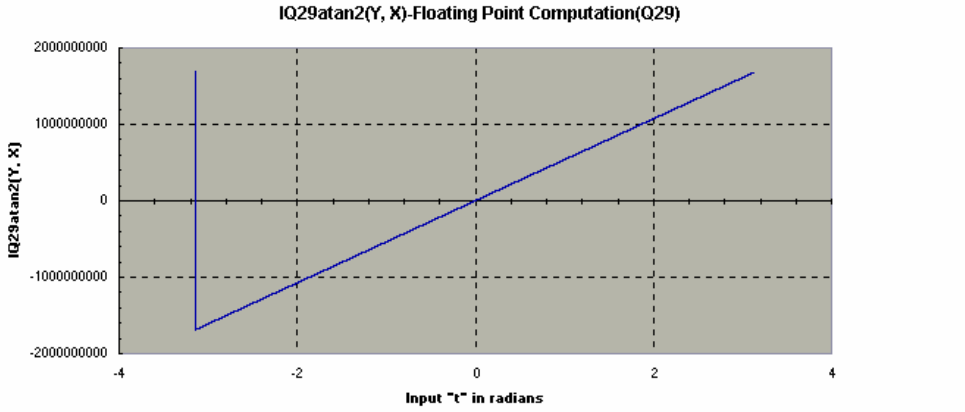
The following example obtains $\tan^{-1}\left(\sin\left(\pi/5\right), \cos\left(\pi/5\right)\right) = \pi/5$, assuming that GLOBAL_Q is set to Q29 format in the IQmath header file.

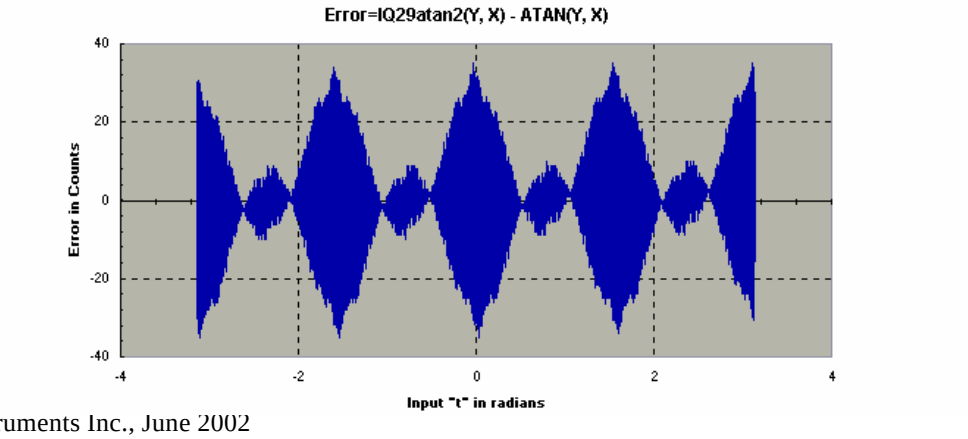
```
#include<IQmathLib.h>  /* Header file for IQ math routine                    */

#define    PI     3.14156

_iq xin1, yin1, out1;
_iq29 xin2, yin2, out2;

void main(void )
{
  xin1=_IQ(0.809)              /* xin1= cos(π/5)×2^29 = 19E3779Bh            */
  yin1=_IQ(0.5877)             /* yin1= sin(π/5)×2^29 = 12CF2304h            */
  out1=_IQatan2(yin1,xin1);    /* out1= π/5×2^29 = 141B2F76h                 */

  xin2=_IQ29(0.809)            /* xin1= cos(π/5)×2^29 = 19E3779Bh            */
  yin2=_IQ29(0.5877)           /* yin1= sin(π/5)×2^29 = 12CF2304h            */
  out2=_IQ29atan2(yin2,xin2);  /* out2= π/5×2^29 = 141B2F76h                 */
}
```

Fixed Point ARCTAN Function vs C Float ARCTAN





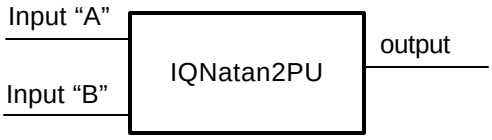




IQNatan2PU	*Fixed point 4-quadrant ATAN (in per unit)*

Description

This module computes 4-quadrant arctangent. Output of this module is in per unit radians that varies from 0 (0 radians) to 1 ($2p$ radians).



Availability

C/C++ Callable Assembly

Module Properties

Type: Target Independent, Application Independent

Target Devices: x28xx

C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

Item	C-Callable ASM	Comments
Code Size	136 words	
Data RAM	0 words	
Multiple instances	N/A	
Reentrancy	Yes	
Multiple Invocation	Yes	
Stack usage	2 words	Stack grows by 2 words

Accuracy

$= 20\log_2\left(1\times 2^{29}\right) - 20\log_2(6)$

= 27 bits

C/C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQatan2PU(_iq A, _iq B)

Q format specific IQ function (IQ format = IQ1 to IQ29)
_iqN _IQNatan2PU(_iqN A, _iqN B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input 'A' & 'B' are fixed-point number represented in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ29)
Input 'A' & 'B' are fixed-point number in IQN format (N=1:29)

Output Format

Global IQ function (IQ format = GLOBAL_Q)
This function returns the inverse tangent of the input argument as fixed-point number in GLOBAL_Q format. The output contains the angle in per unit radians that varies from 0 (0 radians) to 1 ($2p$ radians).

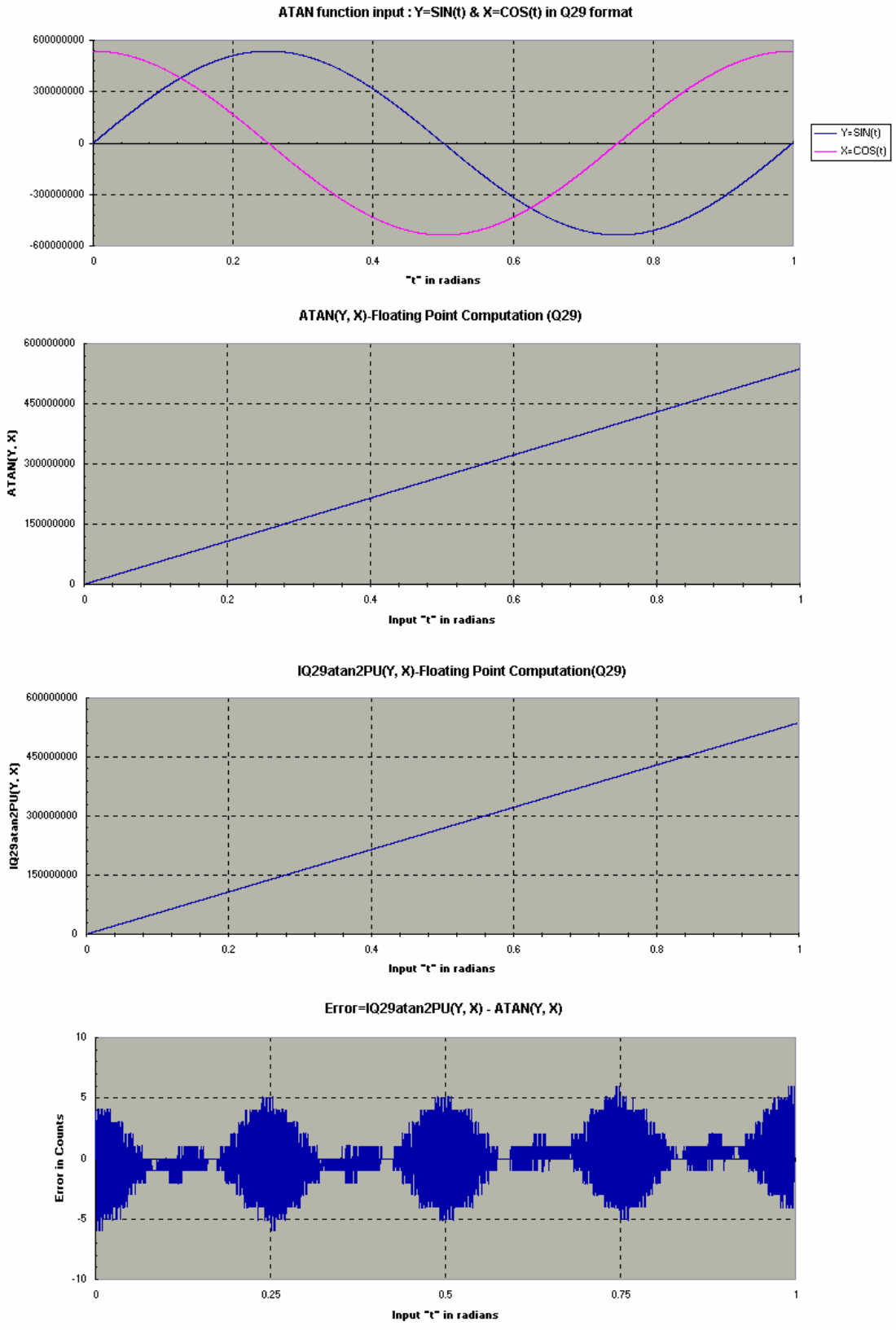
Q format specific IQ function (IQ format = IQ1 to IQ29)
This function returns the inverse tangent of the input argument as fixed-point number in IQN format (N=1:29). The output contains the angle in per unit radians that varies from 0 (0 radians) to 1 ($2p$ radians).

Example

The following sample code obtains $\tan^{-1}\left(\sin\left(p/5\right),\cos\left(p/5\right)\right)=p/5$, assuming that GLOBAL_Q is set to Q29 format in the IQmath header file.

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

_iq xin1, yin1, out1;
_iq29 xin2, yin2, out2;

void main(void )
{
  xin1=_IQ(0.809)                     /* xin1= cos(p/5)×2^29 = 19E3779Bh        */
  yin1=_IQ(0.5877)                    /* yin1= sin(p/5)×2^29 = 12CF2304h        */
  out1=_IQatan2PU(yin1,xin1);         /* ou1 = (p/5)/2p × 2^29 = 03333333h      */

  xin2=_IQ29(0.809)                   /* xin2= cos(p/5)×2^29 = 19E3779Bh        */
  yin2=_IQ29(0.5877)                  /* yin2= sin(p/5)×2^29 = 12CF2304h        */
  out2=_IQ29atan2PU(yin2,xin2)        /* ou2 = (p/5)/2p × 2^29 = 03333333h      */
}
```

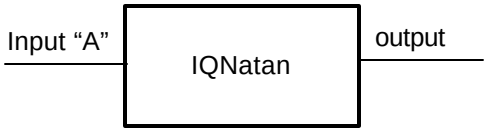
Fixed Point ARCTAN Function vs C Float ARCTAN



IQNatan	*Fixed point ATAN (in radians)*

Description

This module computes arctangent. Output of this module is in radians that vary from $-\pi/2$ to $\pi/2$.



Availability

C/C++ Callable Assembly

Module Properties

Type: Target Independent, Application Independent

Target Devices: x28xx

C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

Item	C-Callable ASM	Comments
Code Size	123 words	
Data RAM	0 words	
Multiple instances	N/A	
Reentrancy	Yes	
Multiple Invocation	Yes	
Stack usage	2 words	Stack grows by 2 words

Accuracy

$$= 20\log_2\left(\frac{p}{2}\times 2^{29}\right) - 20\log_2(2)$$

= 25 bits

C/C-Callable ASM Interface

Declaration

Global IQ Macro (IQ format = GLOBAL_Q)
#define _IQatan(A) _IQatan2(A , _IQ(1.0))

Q format specific IQ Macro (IQ format = IQ1 to IQ29)
#define _IQNatan(A) _IQNatan2(A , _IQN(1.0))

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input argument is a fixed-point number in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ29)
Input argument is a fixed-point number in IQN format (N=1:30)

Output Format

Global IQ function (IQ format = GLOBAL_Q)
This function returns the inverse tangent of the input argument as fixed-point number in GLOBAL_Q format. The output contains the angle in radians between $\left[-\pi/2, +\pi/2\right]$

Q format specific IQ function (IQ format = IQ1 to IQ29)
This function returns the inverse tangent of the input argument as fixed-point number in IQN format (N=1:29). The output contains the angle in radians between $\left[-\pi/2, +\pi/2\right]$

Example

The following example obtains $\tan^{-1}(1) = \pi/4$, assuming that GLOBAL_Q is set to Q29 format in the IQmath header file.

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

_iq in1, out1;
_iq29 in2, out2;

void main(void )
{
  in1=_IQ(1.0);
  out1=_IQatan(in1);

  in2=_IQ29(1.0);
  out2=_IQ29atan(in2)
}
```

IQNsqrt	*Fixed point Square-root*

Description

This module computes the square root of the input using table lookup and Newton-Raphson approximation.

Availability

C-Callable Assembly (CcA)

Module Properties

Type: Target Independent, Application Independent

Target Devices: x28xx

C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

Item	C-Callable ASM	Comments
Code Size	66 words	
Data RAM	0 words	
Multiple instances	N/A	
Reentrancy	Yes	
Multiple Invocation	Yes	
Stack usage	2 words	Stack grows by 2 words

Accuracy

$= 20\log_2\left(2^{31}\right) - 20\log_2(6)$

=29 bits

C/C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQsqrt(_iq A)

Q format specific IQ function (IQ format = IQ1 to IQ30)
_iqN _IQNsqrt(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input argument is a fixed-point number in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Input argument is a fixed-point number in IQN format (N=1:30)

Output Format

Global IQ function (IQ format = GLOBAL_Q)
Square root of input in GLOBAL_Q format.

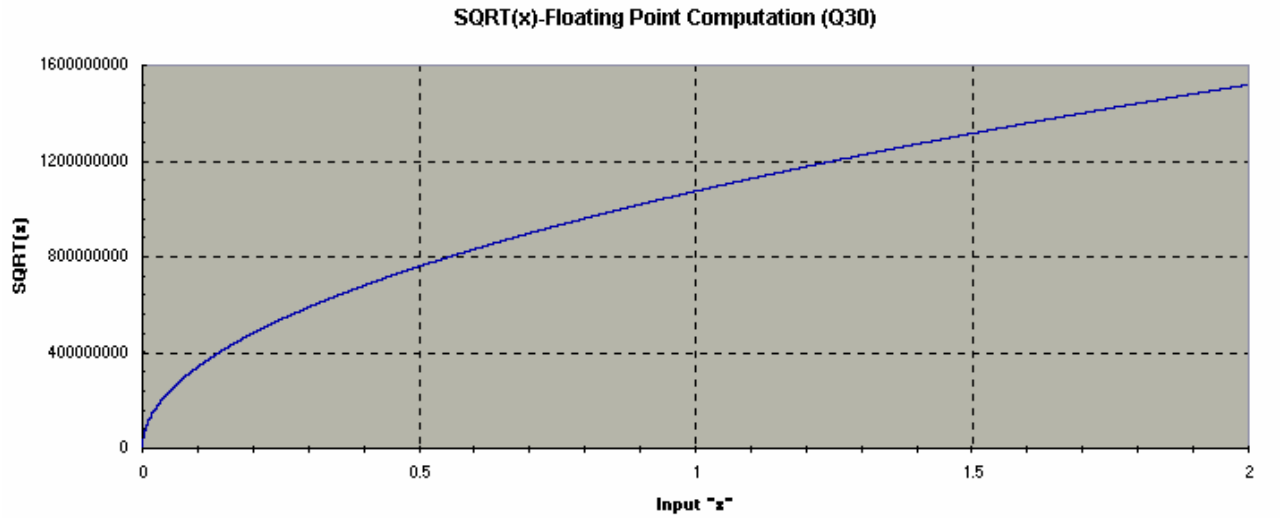
Q format specific IQ function (IQ format = IQ1 to IQ30)
Square root of input in IQN format (N=1:30)

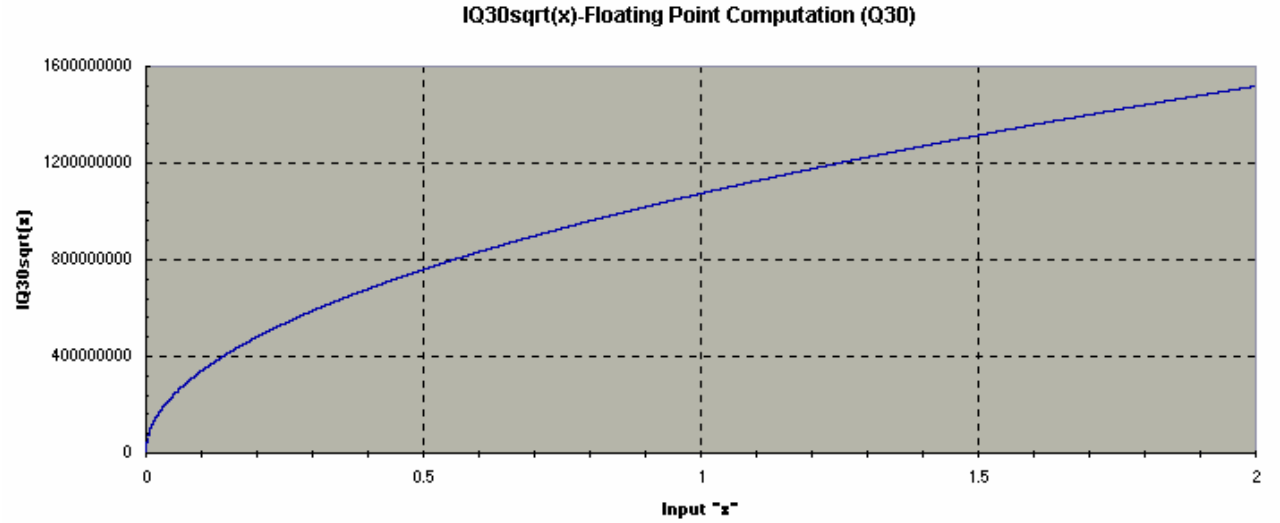
Example

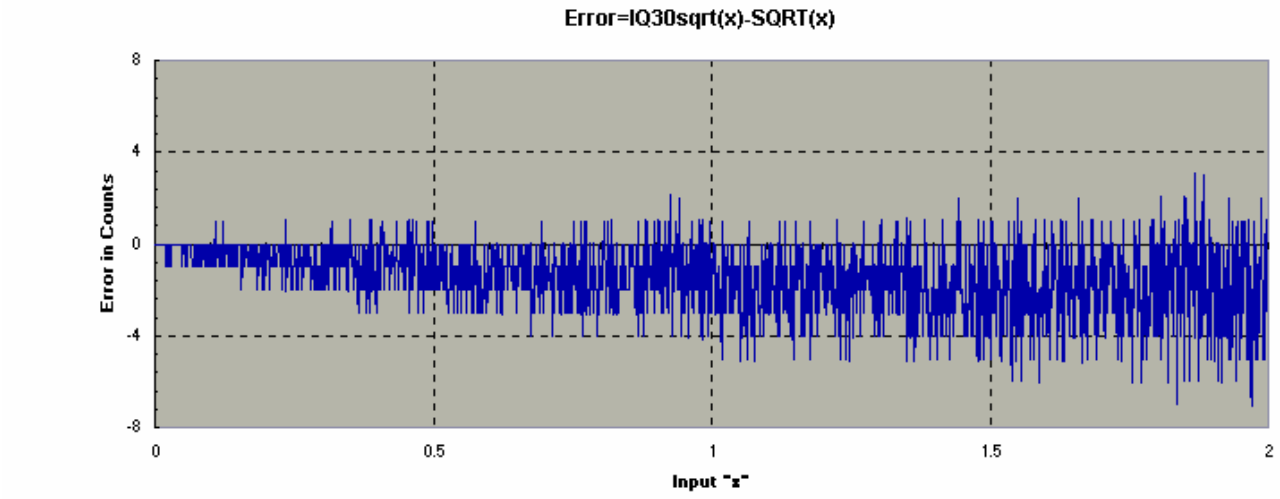
The following example obtains $\sqrt{1.8}$ = 1.34164, assuming that GLOBAL_Q is set to Q30 format in IQmath header file.

```
#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× 2^30 = 73333333h                       */
  out1=_IQsqrt(x);             /* out1= √1.8 ×2^30 = 55DD7151h                     */

  in2=_IQ30(1.8);              /* in2= 1.8× 2^30 = 73333333h                       */
  out2=_IQ30sqrt(x);           /* out2= √1.8 ×2^30 = 55DD7151h                     */
}
```

Fixed Point SQRT Function vs C Float SQRT







IQNisqrt	*Fixed point Inverse Square-root*

Description

This module computes the inverse square root of the input using table lookup and Newton-Raphson approximation.

Availability

C-Callable Assembly (CcA)

Module Properties

Type: Target Independent, Application Independent

Target Devices: x28xx

C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

Item	C-Callable ASM	Comments
Code Size	69 words	
Data RAM	0 words	
Multiple instances	N/A	
Reentrancy	Yes	
Multiple Invocation	Yes	
Stack usage	2 words	Stack grows by 2 words

Accuracy

$= 20\log_2\left(2^{31}\right) - 20\log_2(5)$

=29 bits

C/C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQisqrt(_iq A)

Q format specific IQ function (IQ format = IQ1 to IQ30)
_iqN _IQNisqrt(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input argument is a fixed-point number in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Input argument is a fixed-point number in IQN format (N=1:30)

Output Format

Global IQ function (IQ format = GLOBAL_Q)
Inverse square-root of input in GLOBAL_Q format.

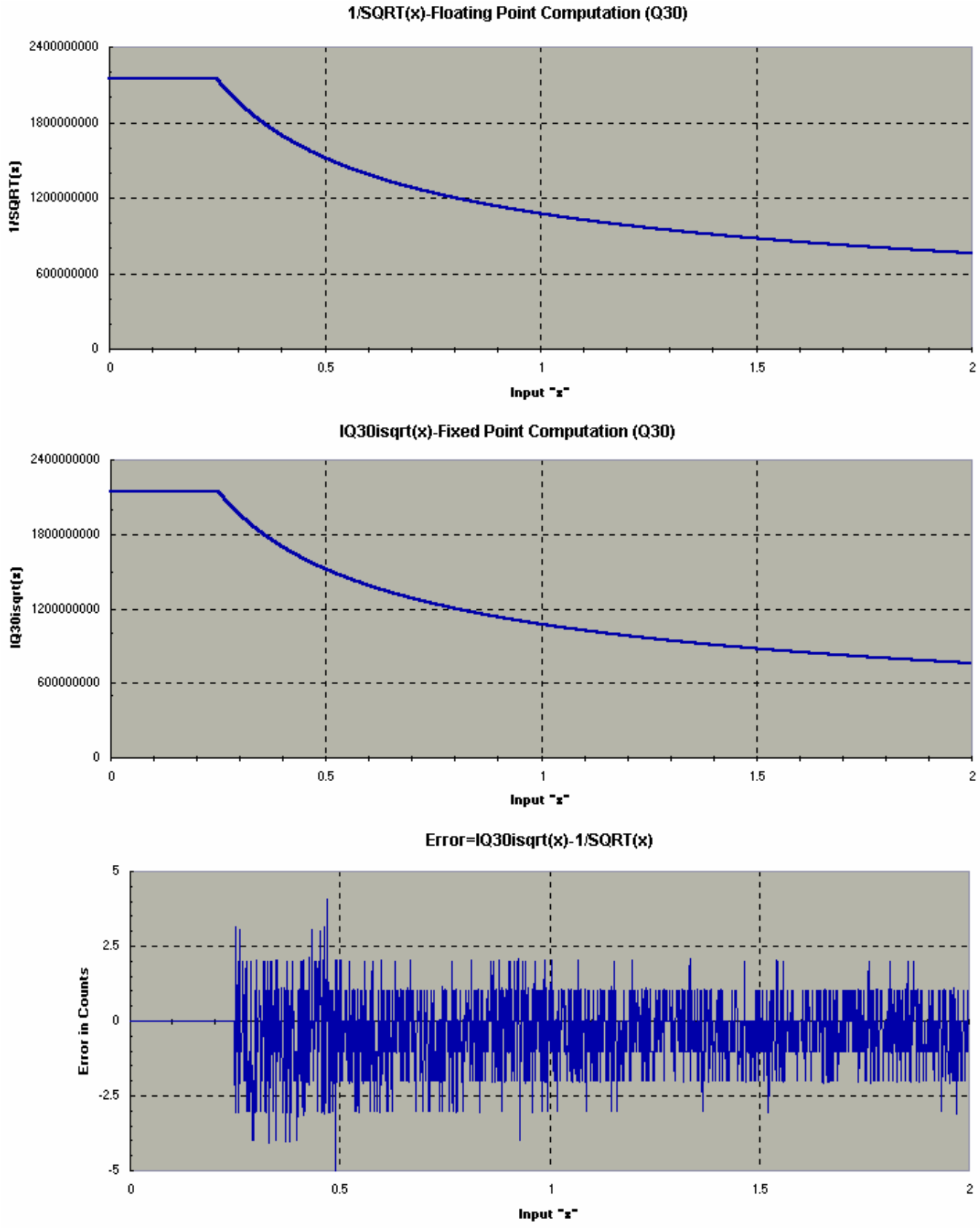
Q format specific IQ function (IQ format = IQ1 to IQ30)
Inverse square root of input in IQN format (N=1:30)

Example

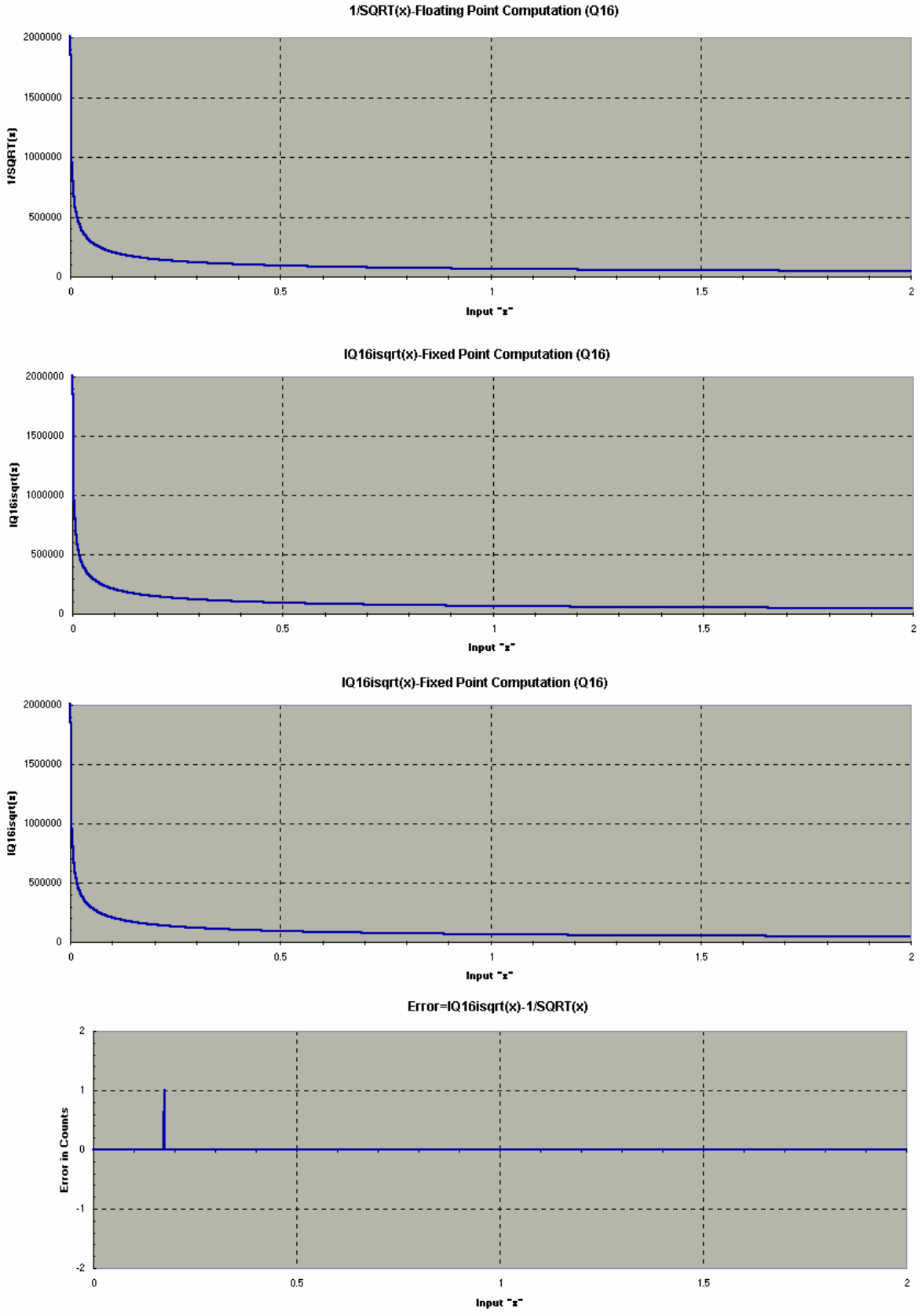
The following example obtains $1/\sqrt{1.8}$=0.74535 assuming that GLOBAL_Q is set to Q30 format in IQmath header file.

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

_iq in1, out1;
_iq30 in2, out2;

void main(void )
{
  in1=_IQ(1.8);                /* in1= 1.8×2^30 = 73333333h                        */
  out1=_IQisqrt(in1);          /* out1= 1/√1.8×2^30 = 2FB3E99Eh                    */

  in2=_IQ30(1.8);              /* in2= 1.8×2^30 = 73333333h                        */
  out2=_IQ30isqrt(in2);        /* out2= 1/√1.8×2^30 = 2FB3E99Eh                    */
}
```

Fixed Point inverse SQRT Function vs C Float inverse SQRT



Fixed Point inverse SQRT Function vs C Float inverse SQRT



IQNmag	*Magnitude of IQ complex number*

Description

This function calculates the magnitude of two orthogonal vectors as follows: Mag = sqrt(A^2 + B^2). This operation achieves better accuracy and avoids overflow problems that may be encountered by using the "_IQsqrt" function.

Availability

C-Callable Assembly (CcA)

Module Properties

Type: Target Independent, Application Independent

Target Devices: x28xx

C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

Item	C-Callable ASM	Comments
Code Size	96 words	
Data RAM	0 words	
Multiple instances	N/A	
Reentrancy	Yes	
Multiple Invocation	Yes	
Stack usage	2 words	Stack grows by 2 words

Accuracy

29-bits (Same as SQRT function)

C/C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQmag(_iq A, _iq B)

Q format specific IQ function (IQ format = IQ1 to IQ30)
_iqN _IQNmag(_iqN A, _iqN B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
Input argument “A” & “B” are IQ number represented in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Input argument “A” & “B” are IQ number represented in IQN format

Output Format

Global IQ function (IQ format = GLOBAL_Q)
Magnitude of the input vector in GLOBAL_Q format

Q format specific IQ function (IQ format = IQ1 to IQ30)
Magnitude of the input vector in IQN format

Example
The following sample code obtains the magnitude of the complex number (Assuming GLOBAL_Q=IQ28)

Sample Code

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

_iq real1, imag1, mag1;                  // Complex number = real1 + j*imag1
_iq28 real2, imag2, mag2;                // Complex number = real2 + j*imag2

void main(void )
{
  real1=_IQ(4.0);
  imag1=_IQ(4.0);
  mag1=_IQmag(real1, imag1);             // mag1=5.6568 in IQ28 format

  real2=_IQ28(7.0);
  imag2=_IQ28(7.0);
  mag2=_IQ28mag(real2, imag2);           // mag2=~8.0, saturated to MAX value (IQ28) !!!
}
```

IQNabs	*Absolute value of IQ number*

Description

This intrinsic calculates the absolute value of an IQ number:

Declaration

Global IQ function (IQ format = GLOBAL_Q)
_iq _IQabs(_iq A)

Q format specific IQ function (IQ format = IQ1 to IQ30)
_iqN _IQNabs(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)
IQ number in GLOBAL_Q format

Q format specific IQ function (IQ format = IQ1 to IQ30)
IQ number in IQN format

Output Format

Global IQ function (IQ format = GLOBAL_Q)
Absolute value of input in GLOBAL_Q format

Q format specific IQ function (IQ format = IQ1 to IQ30)
Absolute value of input in IQN format

Usage

Example:
Calculate the absolute sum of three IQ numbers (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) + _IQ28abs(X2);
```

IQsat	*Saturate the IQ number*

Description This intrinsic saturates an IQ value to the given Positive and Negative limits. This operation is useful in areas where there is potential for overflow in a calculation.

Declaration _iq _IQsat(_iq A, long P, long N)

Input Format **Global IQ function (IQ format = GLOBAL_Q)**
IQ number in GLOBAL_Q format

Output Format **Global IQ function (IQ format = GLOBAL_Q)**
Absolute value of input in GLOBAL_Q format

Usage **Example:**
Calculate the linear equation "Y = M*X + B", with saturation.

All variables are GLOBAL_Q = 26. However, there is a possibility that the variable ranges may cause overflow, so we must perform the calculation and saturate the result.

To do this, we perform the intermediate operations using IQ = 20 and then saturate before converting the result back to the appropriate GLOBAL_Q value:

```
_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);
```

Appendix A: IQmath C-Calling Convention

All the IQmath function strictly adheres to C28x C-Calling convention. To understand the C28x C-Calling convention, Please refer Chapter 7 (Run-time Environment) of TMC320C28x Optimizing C/C++ Compiler User's Guide (SPRU514).