C28x IQmath Library

A Virtual Floating Point Engine

V1.64.00.00

October 1, 2020

IQMATH user's Guide

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI 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. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

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

Trademarks

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

Acronyms

C28x+FPU: Refers to devices with the C28x plus floating-point-unit.

IQmath: High Accuracy Mathematical Functions (32-bit implementation).

QMATH: Fixed Point Mathematical computation

Chapter 1. Introduction

1.1. Introduction

Texas Instruments TMS320C28x IQmath Library is collection of highly optimized and high precision mathematical functions for C/C++ programmers to seamlessly port a 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 and 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 the IQmath Library

2.1. IQmath Package Contents

The TI IQmath library can be used in both C and C++ programs and it consists of 5 parts:

1) The IQmath header files
   The header files include the definitions needed to interface with the IQmath library.
   a) C programs use IQmathLib.h
   b) C++ programs use both IQmathLib.h and IQmathCPP.h
2) The IQmath object library. The library contains all of the IQmath functions and look-up tables. There are six libraries provided:
   a) IQmath.lib: This is an indexed library and based on the options picked up in CCS 10.x the corresponding EABI or COFF library will be used in the project. This indexed library is created using the libraries IQmath_coff.lib and IQmath_eabi.lib.
   b) IQmath_fpu32.lib : This is an indexed library and based on the options picked up in CCS 10.x the corresponding EABI or COFF library will be used in the project. This library can be linked with code built with the --float_support=fpu32 switch. This indexed library is created using the libraries IQmath_fpu32_coff.lib and IQmath_fpu32_eabi.lib
   c) IQmath_coff.lib : This is the COFF version of the IQmath library. The library is built for running on C28x. This library is built without using —fpu_support = fpu32.
   d) IQmath_eabi.lib : This is the EABI version of the IQmath library. The library is built for running on C28x. This library is built without using —fpu_support = fpu32.
   e) IQmath_fpu32_coff.lib : This is the COFF version of the IQmath library. The library is built for running on C28x+FPU devices. This library is built using —fpu_support = fpu32.
   f) IQmath_fpu32_eabi.lib : This is the EABI version of the IQmath library. The library is built for running on C28x+FPU devices. This library is built using —fpu_support = fpu32.

Library Name	Library Format
IQmath.lib	Indexed library – for more information refer to http://www.ti.com/lit/ug/spru514r/spru514r.pdf – section 8.4.2.
IQmath_fpu32.lib	Indexed library – for more information refer to http://www.ti.com/lit/ug/spru514r/spru514r.pdf. - section 8.4.2.
IQmath_coff.lib	This is the COFF version of the IQmath library. The library is built for running on C28x
IQmath_eabi.lib	This is the EABI version of the IQmath library. The library is built for running on C28x.
IQmath_fpu32_coff.lib	This is the COFF version of the IQmath library. The library is built for running on C28x+FPU devices.
IQmath_fpu32_eabi.lib	This is the EABI version of the IQmath library. The library is built for running on C28x+FPU devices.

3) Example linker command files. The example linker command files allocate the sections used by the IQmath library. For some sections the location is device specific. For example, the tables used by the IQsin and IQcos functions are located within the boot ROM of the device.
4) Legacy IQmath GEL file. These gel functions are useful when using a version of Code Composer Studio that does not support IQ data types directly. The .gel file is not needed when using Code Composer Studio V3.3 or later.
5) Example program – example program files can be found in C2000Ware_X_XX_XX_XX\libraries\math\IQmath\c28\examples\C\source for C and in C2000Ware_X_XX_XX_XX\libraries\math\IQmath\c28\examples\CPP\source for C++.

2.2. How to Install the IQmath Library

The IQmath library is provided as part of the C2000WARE and can be found under the folder C2000Ware_X_XX_XX_XX\libraries\math\IQmath.

NOTE:

The directory structure of some earlier versions differed from what is shown below. It has been reorganized to reduce duplication of files. The <base> directory has been changed to correspond to C2000Ware.

<base> install directory is C:\ti\c2000\C2000Ware_X_XX_XX_XX\libraries\math\IQmath\c28

<base>\docs	Contains this file
<base>\include	The IQmath header files C code uses IQmathLib.h C++ code uses IQmathLib.h and IQmathCPP.h
<base>\lib	The IQmath library files. These are used by both C and C++ Refer to **Error! Reference source not found.**.
<base>\gel	Legacy GEL file for debug Refer to section 3.10.

The following example projects run on CCS V4 and newer:

<base>\examples\C	**C example: Refer to ReadMe_SampleC.txt**
<base>\examples\C\<device> <base>\examples\C\source	Each device family has its own project folder Shared source code for the C example
<base>\examples\Cpp	**C++ code example: Refer to ReadMe_SampleCpp.txt**
<base>\examples\Cpp\source <base>\examples\Cpp\<device>	Shared source code for the C++ example Each device family has its own project folder
<base>\examples\cmd	Linker command files used by the examples
<base>\examples\graph_properties	The files in this directory can be used to setup the watch window and graphs in CCS 4 and CCS 10.x.

Chapter 3. Using the 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 functions like IQNsin, IQNcos, IQNatan2, IQNatan2PU, IQatan do not support Q30 format, due to the fact that these functions input or output vary between $-\pi$ to π 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 an IQmath Function from C

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

- Include the IQmathLib.h include file
- Link your code with the IQmath object code library.
- The linker command file should be updated to properly access the IQmath lookup tables and place the IQmath code in the memory block you wish. These sections are described later in this chapter.
- Call the functions using the _iq and _iqN data types along with the C-code function definitions. For example, the following code contains a call to the ***IQ25sin*** routines in IQmath Library:

```
#include<IQmathLib.h>
#define PI  3.14159

_iq input, sin_out;
void main(void )
{
  /* 0.25 x PI radians represented in Q29 format      */
  input=_IQ29(0.25*PI);
  sin_out =_IQ29sin(input);
}
```

3.4. Calling an IQmath function from C++

In C++ the _iq type becomes the iq class. This allows for function overloading of operators such as multiply and divide. To access the library from C++ follow these steps:

- Include both the IQmathLib.h and the IQmathCPP.h header file as shown below:

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

- Link your code with the IQmath object code library.
- The linker command file should be updated to properly access the IQmath lookup tables and place the IQmath code in the memory block you wish. These sections are described later in this chapter.

- Call the functions using the iq and iqN classes along with the C++ code function definitions. In C++ the functions are called without the leading underscore and the math operations are overloaded For example:

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

The following example code contains a call to the ***IQ25sin*** routines in IQmath Library using C++ code:

```
extern "C" {
#include "IQmathLib.h"
}
#include "IQmathCPP.h"
#define PI  3.14159

iq input, sin_out;
void main(void )
{
  /* 0.25 x PI radians represented in Q29 format       */
  input = IQ29(0.25*PI);
  sin_out = IQ29sin(input);
}
```

3.5. IQmath Section and Lookup Tables

Some of the IQmath functions use look-up tables. Many of these tables are included in the boot ROM of 28x devices. Using the boot ROM copy will save you memory space and the time required to load the tables themselves. Note that the boot ROM on some devices is 1 wait state compared to SARAM that is usually 0 wait state. The look-up tables are usually not accessed often enough for this to cause an issue with performance. Therefore, in most cases you will want to use the boot ROM look-up tables.

The IQmath library uses the following lookup tables:

IQmathTables:
This section contains look-up tables for the most often used IQmath functions. This includes IQdiv, IQsin, IQcos, IQsqrt and IQatan2. The IQmathTables are available in the boot ROM of all 28x devices. Hence, this section should be identified as a "NOLOAD" type in the linker command file. This facilitates referencing look-up table symbols without actually loading the section into the target.

IQmathTablesRam:
This section contains initialized look-up tables used by the IQexp, IQasin and IQacos functions. If you do not call these functions, then there is no need to load this section. Some devices also include some of these tables in the boot ROM as shown in the following table.

Device	Assembly Section Name (i.e. .sect)	Boot ROM Location
281x	IQmathTables	0x3FF000
280x	IQmathTables	0x3FF000
2801x	IQmathTables	0x3FF000
2833x/2823x 2834x	IQmathTables IQmathTablesRam (IQNexpTable.obj only)	0x3FE000 0x3FEB50
2802x 2803x	IQmathTables IQmathTablesRam (IQNexpTable.obj) IQmathTablesRam (IQNasinTable.obj)	0x3FE000 0x3FEB50 0x3FEBDC
2806x	IQmathTables IQmathTablesRam (IQNexpTable.obj) IQmathTablesRam (IQNasinTable.obj)	0x3FDF00 0x3FEA50 0x3FEADC

If your device is not listed, refer to the boot ROM source code for your device to get more information – this can be found in C2000Ware_X_XX_XX_XX\libraries\boot_rom.

If a table is not included in the boot ROM of the device, then it must be loaded into the appropriate memory in the linker command file.

NOTE: IQmathTablesRam

The section name IQmathTablesRam may be a bit misleading but has remained due to legacy reasons.

This table contains initialized data tables for the IQexp, IQasin, and IQacos functions. During debug, these tables can be loaded directly into SARAM as shown in the linker file below.

During stand-alone operation, if these functions are used, then the table should be loaded into non-volatile memory (for example flash). If you want to access them in SARAM then copy them from flash to SARAM during initialization.

Assembly Section Name and obj file	Symbols / Initialized Tables Defined	
IQmathTables (IQmathTables.obj)	IQdivTable QdivTableEnd IQdivRoundSatTable IQsqrtTable IQsqrtTableEnd IQisqrtTable IQisqrtTableEnd IQsqrtRoundSatTable IQisqrtRoundSatTable	IQsinTable IQsinTableEnd IQcosTable IQcosTableEnd IQatan2Table IQatan2TableEnd IQatan2HalfPITable
IQmathTablesRam (IQNexpTable.obj)	IQexpTable IQexpTableMinMax IQexpTableMinMaxEnd IQexpTableCoeff IQexpTableCoeffEnd	
IQmathTablesRam (IQNasinTable.obj)	IQasinTable	

The following is an example of a linker file for the 281x devices. In this case only the IQmathTables section is in the boot ROM. If used, the IQmathTablesRam section must be loaded into the device along with the application itself.

IQmath Linker Command File Example:
281x Devices

```
MEMORY
{
PAGE 0:
   PRAMH0 (RW)     : origin = 0x3f8000, length = 0x001000
PAGE 1:
   IQTABLES (R)    : origin = 0x3FF000, length = 0x000b50
   DRAMH0 (RW)     : origin = 0x3f9000, length = 0x001000
}
SECTIONS
{
   IQmathTables     : load = IQTABLES, type = NOLOAD, PAGE = 1
   IQmathTablesRam : load = DRAMH0, PAGE = 1
   IQmath           : load = PRAMH0, PAGE = 0
}
```

The 2833x and 2823x boot ROM includes the IQNexpTable used by the IQNexp() and IQexp() functions. In this case the IQNexpTable can be placed into its own section and identified as type NOLOAD as shown in the next example. The remainder of the IQmathTablesRam section is allocated to SARAM.

IQmath Linker Command File Example:
2823x Device

```
MEMORY
{
PAGE 0:
   PRAML0 (RW)     : origin = 0x008000, length = 0x001000
PAGE 1:
   IQTABLES (R)    : origin = 0x3FE000, length = 0x000b50
   IQTABLES2 (R)   : origin = 0x3FEB50, length = 0x00008c
   DRAML1 (RW)     : origin = 0x009000, length = 0x001000
}
SECTIONS
{
   IQmathTables    : load = IQTABLES, type = NOLOAD, PAGE = 1
   IQmathTables2 > IQTABLES2, type = NOLOAD, PAGE = 1
   {
        IQmath.lib<IQNexpTable.obj> (IQmathTablesRam)
   }
   IQmathTablesRam : load = DRAML1, PAGE = 1
   IQmath           : load = PRAML0, PAGE = 0
}
```

Some devices, such as the 2802x.2803x and 2806x, include both IQNexpTable used by the IQNexp() and IQexp() functions as well as the IQasinTable used by the IQasin() and IQNasin() functions. In this case the IQNexpTable and IQasinTable can be placed into their own section and identified as type NOLOAD as shown in the next example.

IQmath Linker Command File Example:
2803x Device

```
MEMORY
{
PAGE 0:
   PRAML0 (RW)     : origin = 0x008000, length = 0x001000
PAGE 1:
   IQTABLES (R)    : origin = 0x3FE000, length = 0x000b50
   IQTABLES2 (R)   : origin = 0x3FEB50, length = 0x00008c
   IQTABLES3 (R)   : origin = 0x3FEBDC, length = 0x0000AA
   DRAML1 (RW)     : origin = 0x009000, length = 0x001000
}
SECTIONS
{
   IQmathTables    : load = IQTABLES, type = NOLOAD, PAGE = 1
   IQmathTables2 > IQTABLES2, type = NOLOAD, PAGE = 1
   {
       IQmath.lib<IQNexpTable.obj> (IQmathTablesRam)
   }
   IQmathTables3 > IQTABLES3, type = NOLOAD, PAGE = 1
   {
       IQmath.lib<IQNasinTable.obj> (IQmathTablesRam)
   }
   IQmath           : load = PRAML0, PAGE = 0
}
```

NOTE: Linker warning when including IQNexpTable.obj and/or IQNasinTable.obj in the linker file SECTIONS:

It should be noted that if the IQexp() function is not called in the application, then the allocation for IQNexpTable.obj will result in a linker warning.

Likewise if IQasin() or IQacos() are not called, then IQNasinTable.obj will result in a linker warning. This is because the table will not be included if it is not used. This warning will not cause any ill effects to the normal operation of the application.

3.6. Accessing IQmath Functions in the Boot ROM

Some devices contain selected IQmath functions within the boot ROM itself. You can use these copies to save both flash and RAM space.

Device	IQmath Functions in Boot ROM	Values of N
281x	None	N/A
280x	None	N/A
2801x	None	N/A
2833x, 2823x	None	N/A
2834x	None	N/A
2802x 2803x 2806x	IQNatan2 IQNdiv IQNmag IQNisqrt IQNsqrt IQNsin IQNcos	15, 20, 24, 29

To easily use the boot ROM copies of the functions in place of the existing functions, an IQmath boot ROM symbol library has been provided with the boot ROM source code. This source code is available C2000Ware location C2000Ware_X_XX_XX_XX\libraries\boot_rom.

You will link the IQmath Boot ROM symbol library before the normal IQmath library. The boot ROM symbol library replaces only the subset of functions in the boot ROM. Therefore, the standard IQmath library should be linked after the boot ROM symbol library.

The following pages give an overview of how to set the link order for the library in CCS V3.3 and CCS V4.

To confirm that the boot ROM copies of the functions are being used, check the generated .map file. Look for the symbols of the functions. For example, the address of the IQ24div, IQ24exp and IQ24sin symbols shown below are from the boot ROM area of the 2803x device. The remaining symbols are in RAM.

```
 Address       Symbol
0000821a   __IQ24atan2PU
003ff083   __IQ24div          <- Boot ROM
000082a2   __IQ24exp
003fef35   __IQ24mag          <- Boot ROM
000082df   __IQ24mpyI32int
003ff3c2   __IQ24sin          <- Boot ROM
```

The following instructions apply to Code Composer Studio V4

1. In the C/C++ perspective, right click on the project and add both the boot ROM Symbol Library (for example: 2802x_IQmath_BootROMSymbols.lib) and the IQmath library files to the project.

2. Right click on the project and select properties.

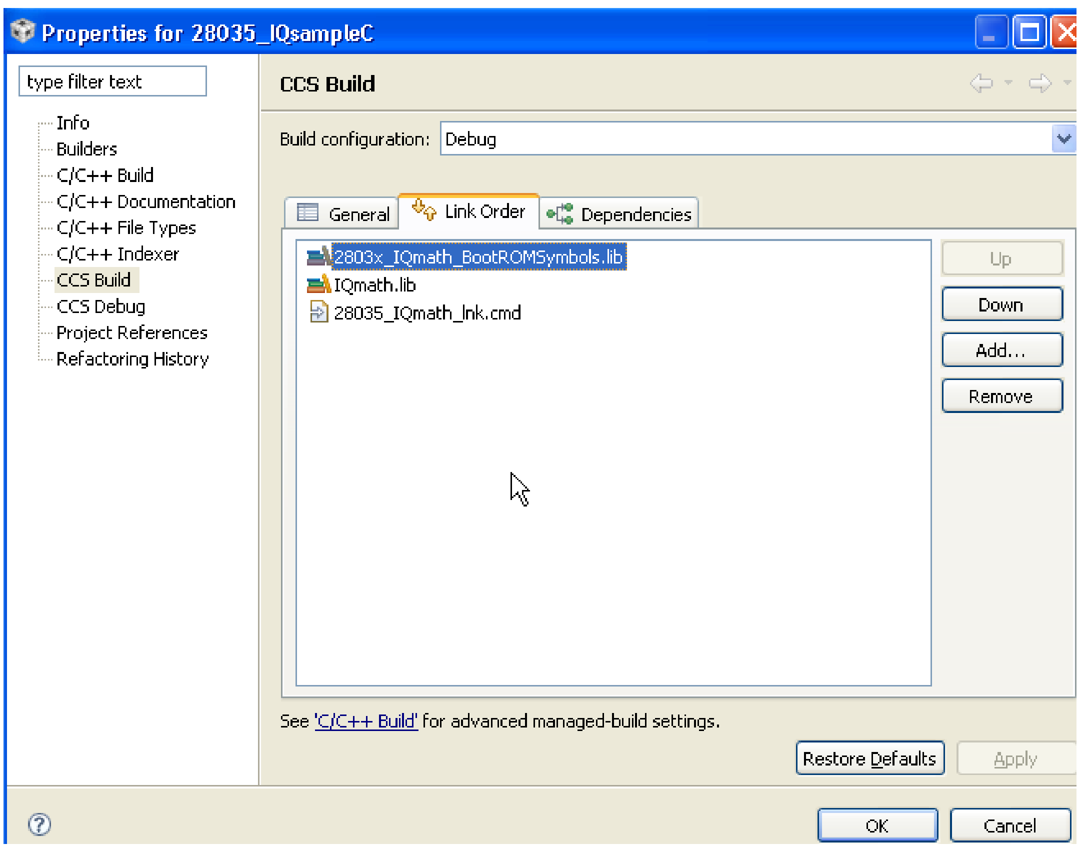
3. Select the CCS Build Options and the link order tab.

4. Add both the boot ROM symbol library and the IQmath library to the link order. Adjust the order if required such that the symbol library is linked in first as shown below.

5. Under the C/C++ Build options, go to the
   Tools Setting Tab -> C2000 Linker options -> File Search Path:

6. Check the options
   "Search libraries in priority order (-priority)"
   "Reread libraries; resolve backward references (-x)"

7. Click "ok".



The following instructions apply to Code Composer Studio V3.3

1. Add the boot ROM symbol library (for example: 2802x_IQmath_BootROMSymbols.lib) and the standard IQmath.lib library to the project using

   Project->Add Files to Project

   Once the libraries are added to the project they will appear in the link order tab of the build options.

2. Open the build options dialog box under Project->Build Options

3. Under the Linker->Advanced tab, select the –priority linker switch.

   This will force the linker to resolve symbols to the first library linked.

4. Under the Link Order tab, select the two libraries and add them to the link order.

   Use the up/down arrows to arrange them in the proper order. The first library listed will be linked first.

5. Under the Linker->Libraries dialog add the path to the 2802x_IQmath_BootROMSymbols.lib (or 2803x_IQmath_BootROMSymbols.lib) and the IQmath library in the search path.

   Do not include either libraries in the “Incl. Libraries” box. Doing so can cause problems when changing the link order since Code Composer uses both this field and the link order tab to determine which object files are linked first.

6. Save the project. (Project->Save).

3.7. Example Projects

Several simple examples can be found in the following directory:

<base>\examples\C	**C example: Refer to ReadMe_SampleC.txt**
<base>\examples\Cpp	**C++ code example: Refer to ReadMe_SampleCpp.txt**

These examples are intended to be a very simple introduction to IQmath. The examples do not configure the PLL of the device or any peripheral. They do, however, disable the watchdog.

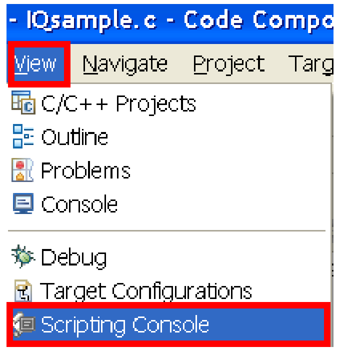
If an example for a particular device is not present, it does not mean the IQmath library will not work with that device. The library is compatible with all C28x based devices. All that is needed is an updated linker .cmd file to match the memory of the desired device.

To view the outputs follow the following instructions for CCS 3.x, CCS 4.x and CCS 10.x

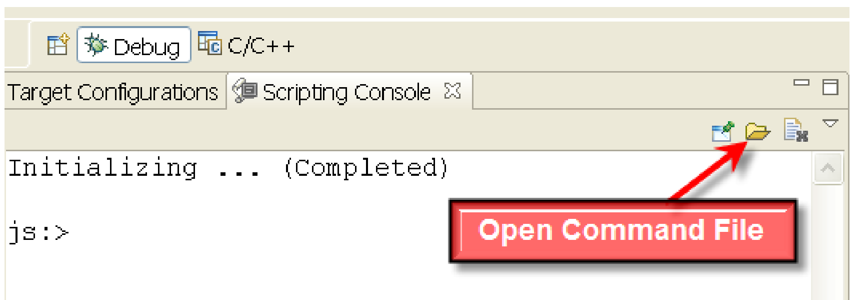
For CCS 3.x and CCS 4.x - After you execute the example, you can load the watch window and the graphs by using the scripts found in the <base>\examples\graph_properties directory.

1. Open the SetupDebugEnv.cmd file in an editor. Confirm that the path to the graphs matches your install location. Save the file.
2. In Code Composer Studio, open the scripting console: view->scripting console.

After you execute the example, you can load the watch window and the graphs by using the scripts found in the <base>\examples\graph_properties directory.



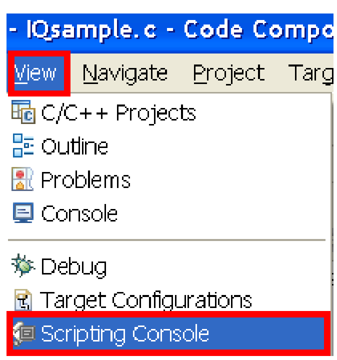
3. In the scripting console, select the “open command file” icon at the upper right.



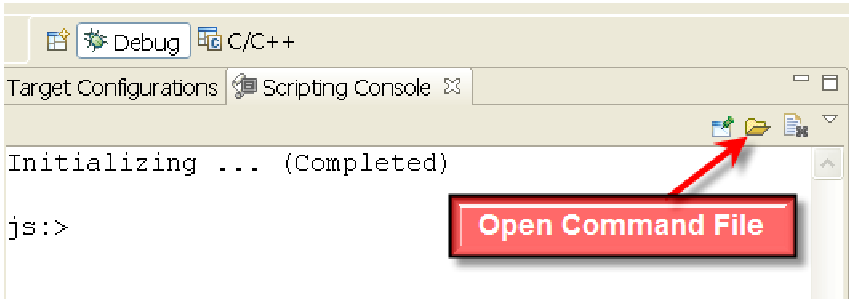
4. Browse to the file SetupDebugEnv.js and select it. This will run the script and load the graphs and watch window.

For CCS 10.x - After you execute the example, you can load the watch window and the graphs by using the scripts found in the <base>\examples\graph_properties directory.

1. Open the SetupDebugEnv_CCS9.cmd file in an editor. Confirm that the path to the graphs matches your install location.
2. In Code Composer Studio, open the scripting console: view->scripting console.



3. In the scripting console, select the “open command file” icon at the upper right.



4. Browse to the file SetupDebugEnv_CCS9.cmd and select it. This will run the script and load the graphs and watch window.

3.8. IQmath Naming Conventions

Each IQmath function comes in two types as shown below:

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

  C-Code Examples:

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

  C++ Code Examples:

  - IQsin(A) /* High Precision SIN */
  - IQcos(A) /* High Precision COS */
  - IQrmpy(A,B) /* IQ multiply with rounding */
  - A*B /* IQ multiply */

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

  C-Code 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 */

  C++ Code 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 C Function Naming Convention
GLOBAL_Q Function **_IQ**xxx( ), Where "xxx" is the Function Name Q Specific Function **_IQN**xxx( ), Where "xxx" is the Function Name & "**N**" is the Q format of input/output

3.9. Selecting the 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. 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>
```

3.10. Using the IQmath GEL file for Debugging

This section contains legacy information and is not required for CCS V3.3 or later.

This information is only useful when using an older version of Code Composer Studio prior to CCS V3.3 that does not support IQ data types directly.

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 "GlobalIQ" 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

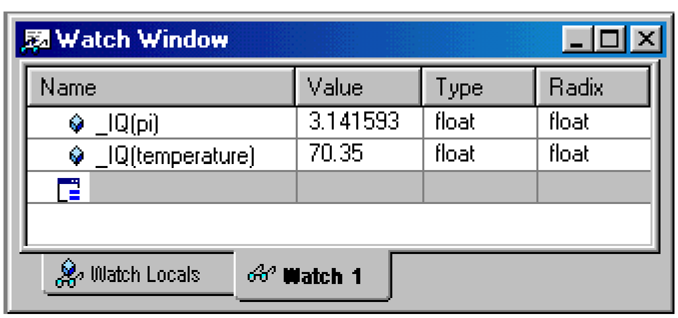
Step 3: Viewing IQmath variable
To view a variable in the watch window, simply type the following commands in the watch window. They will convert the specified "VarName" in IQ format to the equivalent floating-point value:

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

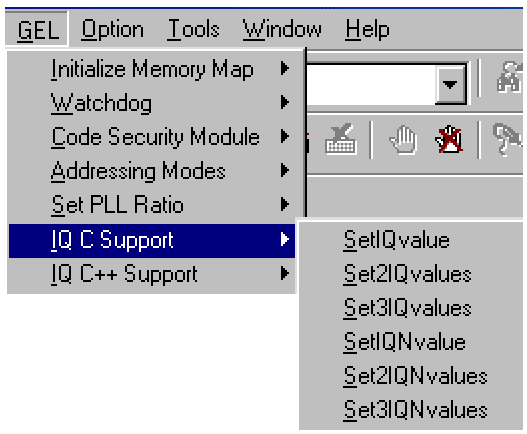


Step 4: Modifying IQmath variable

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:

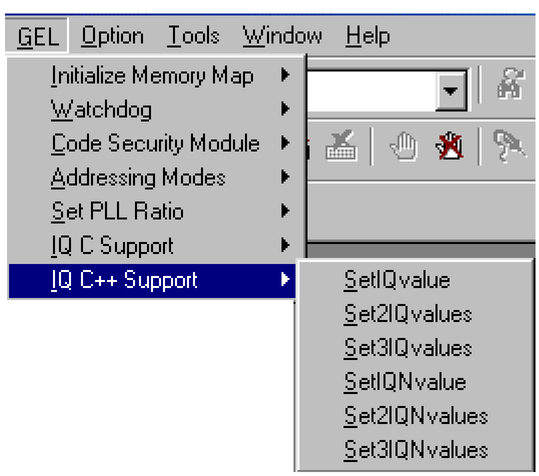
IQ C Support

- 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.

3.11. Converting an IQmath Application to Floating-Point

To convert an IQmath application to floating point, follow these steps:

1. In the IQmath header file, select FLOAT_MATH. The header file will convert all IQmath function calls to their floating-point equivalent.

2. When writing a floating-point number into a device register, you need to convert the floating-point number to an integer. Likewise when reading a value from a register it will need to be converted to float. In both cases, this is done by multiplying the number by a conversion factor.

For example to convert a floating-point number to IQ15, multiply by 32768.0.

```
#if MATH_TYPE == IQ_MATH
    PwmReg = (int16)_IQtoIQ15(Var1);
#else // MATH_TYPE is FLOAT_MATH
    PwmReg = (int16)(32768.0L*Var1);
#endif
```

Likewise, to convert from an IQ15 value to a floating-point value, multiply by 1/32768.0 or 0.000030518.0.

Note: The integer range is restricted to 24-bits for a 32-bit floating-point value.

3. If your device has the on-chip floating-point processing unit (C28x+FPU), then you can take advantage of the on-chip floating point unit by doing the following:

   - Use C28x codegen tools version 5.0.2 or later.
   - Tell the compiler it can generate native C28x floating-point code. To do this, use the –v28 --float_support=fpu32 compiler switches.

     In Code Composer Studio V4: the float_support switch is under the C/C++ build options->Tools Settings Tab -> Runtime Model Operations:

     In Code Composer Studio V3.3: the float_support switch is on the advanced tab of the compiler options.

   - Use the correct run-time support library for native 32-bit floating-point. For C code this is rts2800_fpu32.lib. For C++ code with exception handling, use rts2800_fpu32_eh.lib.
   - Use the C28x FPU Fast RTS library (SPRC664) to get a performance boost from math functions such as sin, cos, div, sqrt, and atan. The Fast RTS library should be linked in before the normal run-time support library.

3.12. The IQmath C-Calling Convention

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

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	_iq is the C-code 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. For C++ code the iq class is used instead.
_iqN	_iqN is the C-code data type definition equating a long, a 32-bit value representing a IQN number, where N=1:30 For C++ code, the iqN class is used instead.
iq	C++ iq class for handling the _iq data type.
iqN	C++ iqN class for handling the _iqN data type.
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 Function Overview

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
int _IQtoa(char *S, const * format, long x) int _IQNtoa(char *S, const * format, long x)	IQ to ASCII string	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

Shift to Multiply or Divide by Powers of 2

Added to the IQmathLib.h file as of V15c

Functions	Description	IQ format
_iq _IQmpy2(_Iq A)	Multiply by 2 by using a left shift by 1	
_iq _IQmpy4(_Iq A)	Multiply by 4 by using a left shift by 2	
_iq _IQmpy8(_Iq A)	Multiply by 8 by using a left shift by 3	
_iq _IQmpy16(_Iq A)	Multiply by 16 by using a left shift by 4	
_iq _IQmpy32(_Iq A)	Multiply by 32 by using a left shift by 5	
_iq _IQmpy64(_Iq A)	Multiply by 64 by using a left shift by 6	
_iq _IQdiv2(_Iq A)	Division by 2 by using a right shift by 1	
_iq _IQdiv4(_Iq A)	Division by 4 by using a right shift by 2	
_iq _IQdiv8(_Iq A)	Division by 8 by using a right shift by 3	
_iq _IQdiv16(_Iq A)	Division by 16 by using a right shift by 4	
_iq _IQdiv32(_Iq A)	Division by 32 by using a right shift by 5	
_iq _IQdiv64(_Iq A)	Division by 64 by using a right shift by 6	

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

Trigonometric Functions:

Functions	Description	IQ format
_iq _IQasin(_iq A) _iqN _IQNasin(_iqN A)	High precision ASIN (output in radians)	Q=GLOBAL_Q Q=1:29
_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 _IQacos(_iq A) _iqN _IQNacos(_iqN A)	High precision ACOS (output in radians)	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) _iqN _IQNatan(_iqN A)	Arctangent	Q=GLOBAL_Q Q=1:29

Mathematical Functions:

Functions	Description	IQ format
_iq _IQexp(_iq A) _iqN _IQNexp(_iqN A)	High precision e raised to the A power	Q=GLOBAL_Q Q=1:30
_iq _IQlog(_iq A) _iqN _IQNlog(_iqN A)	IQ Natural Logarithmic Math Function	Q=GLOBAL_Q Q=1:29
_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

4.3. C28x IQmath Library Benchmarks

Function Name	IQ Format	Execution Cycles	Accuracy (in bits)	Program Memory (words)	Input format	Output format	Remarks
Trigonometric Functions							
IQNasin	1-29	154		82 words	IQN	IQN	Note A
IQNsin	1-29	46	30 bits	49 words	IQN	IQN	
IQNsinPU	1-30	40	30 bits	41 words	IQN	IQN	
IQNacos	1-29	170		93 words	IQN	IQN	Note A
IQNcos	1-29	44	30 bits	47 words	IQN	IQN	
IQNcosPU	1-30	38	29 bits	39 words	IQN	IQN	
IQNatan2	1-29	109	26 bits	123 words	IQN	IQN	
IQNatan2PU	1-29	117	27 bits	136 words	IQN	IQN	
IQatan	1-29	109	25 bits	123 words	IQN	IQN	
Mathematical Functions							
IQNexp	1-30	190		61 words	IQN	IQN	Note A
IQNlog	1-29	148		157 words	IQN	IQN	
IQNsqrt	1-30	63	29 bits	66 words	IQN	IQN	
IQNisqrt	1-30	64	29 bits	69 words	IQN	IQN	
IQNmag	1-30	86	29 bits	96 words	IQN	IQN	
Arithmetic Functions							
IQNmpy	1-30	~ 6	32 bits	NA	IQN*IQN	IQN	INTRINSIC
IQNrmpy	1-30	17	32 bits	13 words	IQN*IQN	IQN	
IQNrsmpy	1-30	21	32 bits	21 words	IQN*IQN	IQN	
IQNmpyI32	1-30	~ 4	32 bits	NA	IQN*long	IQN	C-MACRO
IQNmpyI32int	1-30	22	32 bits	16 words	IQN*long	long	
IQNmpyI32frac	1-30	24	32 bits	20 words	IQN*long	IQN	
IQNmpyIQX		~ 7	32 bits	NA	IQN*IQN	IQN	INTRINSIC
IQNdiv	1-30	63	28 bits	71 words	IQN/IQN	IQN	
Format Conversion Utilities							
IQN	1-30	NA	N/A	NA	Float	IQN	C-MACRO
IQNtoF	1-30	22	N/A	20 words	IQN	Float	
IQNtoa	1-30	N/A	N/A	210 words	IQN	string	
atoIQN	1-30	N/A	N/A	143 words	char *	IQN	
IQNint	1-30	14	32 bits	8 words	IQN	long	
IQNfrac	1-30	17	32 bits	12 words	IQN	IQN	
IQtoIQN	1-30	~4	N/A	N/A	GLOBAL_Q	IQN	C-MACRO
IQNtoIQ	1-30	~4	N/A	N/A	IQN	GLOBAL_Q	C-MACRO
IQtoQN	1-15	~4	N/A	N/A	GLOBAL_Q	QN	C-MACRO
QNtoIQ	1-15	~4	N/A	N/A	QN	GLOBAL_Q	C-MACRO
Miscellaneous							
IQsat	1-30	~7	N/A	N/A	IQN	IQN	INTRINSIC
IQNabs	1-30	~2	N/A	N/A	IQN	IQN	INTRINSIC

Notes:

A. IQNexp, IQNasin and IQNacos use look-up tables in the IQmathTablesRam section. Refer to Section 3.3.

B. 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.

C. 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.

D. Accuracy should always be verified and tested within the end application.

Chapter 5. Function Descriptions

5.1. Conversion Utilities

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)**

```
C      _iq _IQ(float F)
C++     iq  IQ(float F)
```

Q format specific IQ Macro (IQ format = IQ1 to IQ29)

```
C      _iqN _IQN(float F)
C++     iq   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: Implement an equation in IQmath

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

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

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

Example 2: Convert a floating-point variable to an IQ data type.

```
#include "IQmathLib.h"
float x=3.343;
_iq y1;
_iq23 y2

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

Example 3: Initialize global variables or tables

```
#include "IQmathLib.h"

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

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

IQNtoF

Float to IQN data type

Description This function converts an IQ number to equivalent floating-point value in IEEE 754 format.

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

```
C      float _IQtoF(_iq A)
C++    float  IQtoF(const iq &A)
```

Q format specific IQ function (IQ format = IQ1 to IQ30)

```
C      float _IQNtoF(_iqN A)
C++    float  IQNtoF(const 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:

Convert an array of IQ numbers to their 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 an IQ number.

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

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

Q format specific IQ function (IQ format = IQ1 to IQ30)

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

IQNtoa — *IQN to String*

Description This function converts an IQ number to a string.

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

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

Q format specific IQ function (IQ format = IQ1 to IQ30)

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

Input

string: output string

format: conversion format. Must be of the form "%xx.yyf" with xx and yy at most 2 characters in length.
For Example: "%10.12f", "%2.4f", "%11.6f"
The maximum supported integer field width (xx) is 11 (including any negative sign). This captures the full integer range for I2Q30 to I31Q1 numbers.

x: Global IQ function: input value in IQ format
Q format specific function: input value in IQN format

Output The output string is returned in the location pointed to by the “string” argument.

If you are using MATH_TYPE set to IQ_MATH, then the return integer value is an error code with the following possible values:

0 = no error
1 = width too small to hold integer characters
2 = illegal format specified

If you are using MATH_TYPE set to FLOAT_MATH, then sprintf() is called and the return integer value is the number of characters written.

Usage

- Any leading zeros are not printed for the integer part. Hence, the format specifies the maximum width of the integer field. The field may be smaller.
- The output string is terminated with the null character.
- The integer width in "format" includes the negative sign for negative numbers, e.g. -12.3456 is "%3.5f"
- The decimal width in "format" includes the decimal point. For example: -12.3456 is "%3.5f"
- "string" must be large enough to hold the output (including the negative sign, and the terminating null character). The program does not check for overrun. Memory corruption will occur if "string" is too small.
- A non-zero return value indicates that the output string is invalid.

Example:

```
char buffer[30];

_iq   x1 = _IQ(1.125);
_iq1  x2 = _IQ1(-6789546.3);
_iq14 x3 = _IQ14(-432.6778);
_iq30 x4 = _IQ30(1.127860L);
int error;

// Global_Q
   error = _IQtoa(buffer, "%10.10f", x1);

// IQ1
   error = _IQ1toa(buffer, "%8.2f", x2);

// IQ14
   error = _IQ14toa(buffer, "%6.6f", x3);

// IQ30
   error = _IQ30toa(buffer, "%11.12f", x4);
```

IQNint	***Integer part of IQN number***

Description This function returns the integer portion of IQ number.

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

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

Q format specific IQ function (IQ format = IQ1 to IQ30)

```
C      long _IQNint(_iqN A)
C++    long  IQNint(const 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:**

Extract the integer and fractional part of an 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:

Build an IQ number from an integer and fractional part

```
_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)**

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

Q format specific IQ function (IQ format = N = 1 to 30)

```
C      _iqN _IQNfrac(_iqN A)
C++     iqN  IQNfrac(const 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:**

Extract the integer and fractional part of an IQ number

```
_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:

Build an IQ number from an integer and fractional part.

```
_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

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

The 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:

```
#include "IQmathLib.h"
_iq Z, Y, X;            // GLOBAL_Q = 26
_iq23 temp;

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

Y = _IQ23toIQ(temp);
```

IQNtoIQ

IQN number to GLOBAL_Q

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

Declaration

```
C      _iq _IQNtoIQ(_iqN A)
C++     iq  IQNtoIQ(const 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:

```
#include "IQmathLib.h"
_iq Z, Y, X;            // GLOBAL_Q = 26
_iq23 temp;

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

Y = _IQ23toIQ(temp);
```

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

```
C      int _IQtoQN(_iq A)
C++    int  IQtoQN(const 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

```
C       _iq _QNtoIQ(int A)
C++      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);
```

5.2. Shift to Multiply or Divide by Powers of 2

IQmpy2, 4, 8..64	*Right Shift to Multiply by 2*

Description This #define macro in the IQmathLib.h can be used to perform a left shift in order to multipy a number by a power of 2. Macros are provided for multiply by 2, 4, 8 16, 32 and 64.

When using FLOAT_MATH the corresponding multiply will be applied.

Input Input A is an IQ number.

Output Output is an IQ number.

Usage **Example 1;**

Compute `Y = 2*X = X << 1`

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

Compute `Y = 16*X = X << 4`

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

IQdiv2, 4, 8..64 — *Right Shift to Multiply by 2*

Description This #define macro in the IQmathLib.h can be used to perform a right shift in order to multipy a number by a power of 2. Macros are provided for divide by 2, 4, 8 16, 32 and 64.

When using FLOAT_MATH the corresponding multiply (i.e. .5, .25 etc) will be applied.

Input Input A is an IQ number.

Output Output is an IQ number.

Usage **Example 1;**

Compute `Y = 2*X = X << 1`

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

Compute `Y = 16*X = X << 4`

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

5.3. Arithmetic Operations

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)

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

Q format specific IQ intrinsic (IQ format = IQ1 to IQ30)

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

Input

Global IQ intrinsic (IQ format = GLOBAL_Q)
Inputs A and B are IQ numbers in GLOBAL_Q format

Q format specific IQ intrinsic (IQ format = IQ1 to IQ30)
Inputs A and B are IQ numbers in IQN format

Output

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;

Compute `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:

Compute `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)

```
C      _iq _IQrmpy(_iq A, _iq B)
C++    _iq _IQrmpy(const _iq &A, const _iq &B)
```

Q format specific IQ function (IQ format = IQ1 to IQ30)

```
C      _iqN _IQNrmpy(_iqN A, _iqN B)
C++    _iqN _IQNrmpy(const _iqN &A, const _iqN &B)
```

Input

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

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:

Compute `Y = M*X + B` in GLOBAL_Q format with rounding but without saturation.

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

Example 2:

Compute `Y = M*X + B` in IQ10 format with rounding but without 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)

```
C      _iq _IQrsmpy(_iq A, _iq B)
C++    iq  IQrsmpy(iq &A, iq &B)
```

Q format specific IQ function (IQ format = IQ1 to IQ30)

```
C      _iqN _IQNrsmpy(_iqN A, _iqN B)
C++    iqN  IQNrsmpy(const iqN &A, const iqN &B)
```

Input

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

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 approximately [-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:

Compute Y = M*X in the GLOBAL_Q format with rounding and saturation. Assume GLOBAL_Q=IQ26 in the IQmath header file.

```
_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:

Compute 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, 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)**

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

Q format specific IQ Macro (IQ format = IQ1 to IQ30)

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

Input **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 **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:**

Compute Y = 5*X in the GLOBAL_Q format. Assume GLOBAL_Q =IQ26 in the IQmath header file.

```
_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:

Compute 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)**

```
C     long _IQmpyI32int(_iq A, long B)
C++   long  IQmpyI32int(const iq &A, long B)
```

Q format specific IQ function (IQ format = IQ1 to IQ30)

```
C     long _IQNmpyI32int(_iqN A, long B)
C++   long  IQNmpyI32int(const iqN &A, long B)
```

Input **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 **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;

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

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)

```
C      _iq _IQmpyI32frac(_iq A, long B)
C++     iq  IQmpyI32frac(const iq &y, long x)
```

Q format specific IQ function (IQ format = IQ1 to IQ30)

```
C      _iqN _IQNmpyI32frac( _iqN A, long B)
C++     iqN  IQNmpyI32frac(const iqN &A, long B)
```

Input

Global IQ function (IQ format = GLOBAL_Q)
Operand A is an IQ number in GLOBAL_Q format and B is a 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

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:

The following example extracts the fractional part of result after multiplication (Assuming GLOBAL_Q=IQ26)

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

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

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)**

```
C      _iq _IQmpyIQX(_iqN1 A, int N1, _iqN2 B, int N2)
C++     iq  IQmpyIQX(iqN1 A, int N1, iqN2 B, int N2)
```

Q format specific IQ Intrinsic (IQ format = IQ1 to IQ30)

```
C      _iqN _IQNmpyIQX(_iqN1 A, int N1, _iqN2 B, int N2)
C++     iqN  IQNmpyIQX(iqN1 A, int N1, iqN2 B, int N2)
```

Input Operand “A” is an IQ number in “IQN1” format and operand “B” is in “IQN2” format.

Output **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:

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)

The maximum range of Y will be -48 to +48. Therefore, we will store the result in an 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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

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

Q format specific IQ function (IQ format = IQ1 to IQ30)

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

Input

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

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)

Output is:
- 0x7FFFFFFF on positive overflow
- 0x80000000 on negative overflow
- 0x7FFFFFFF if Denominator is 0

Accuracy

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

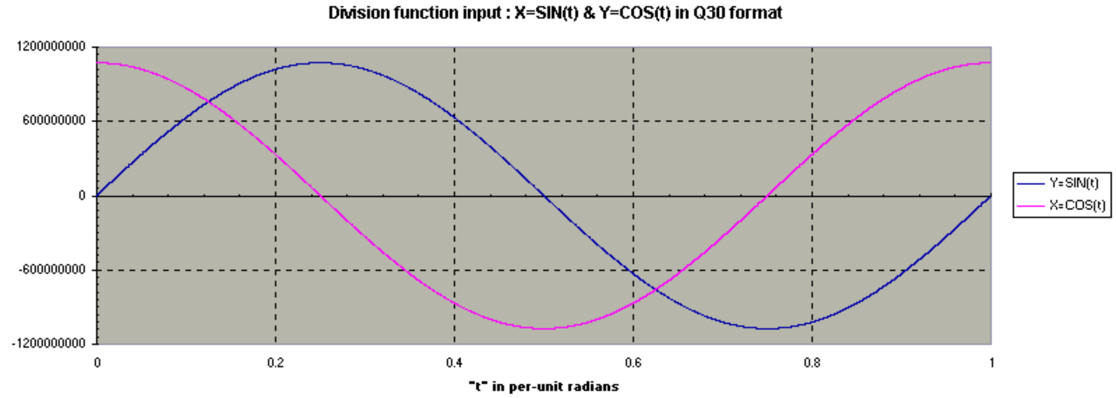
Usage

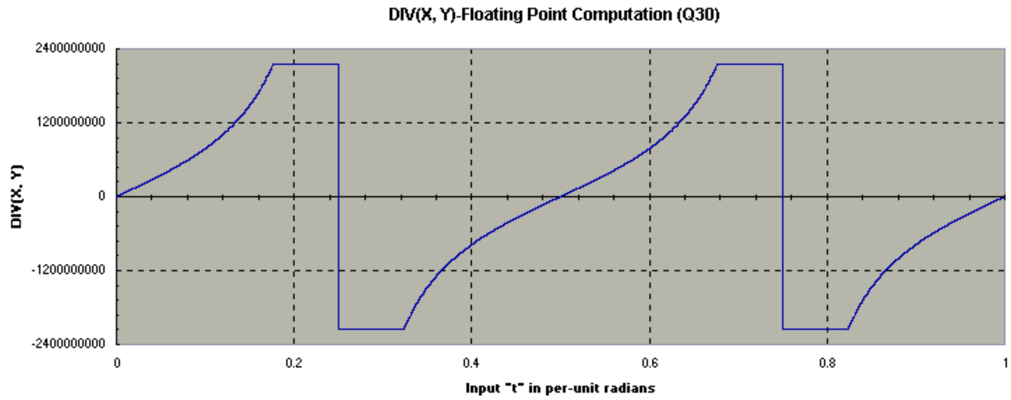
Example:

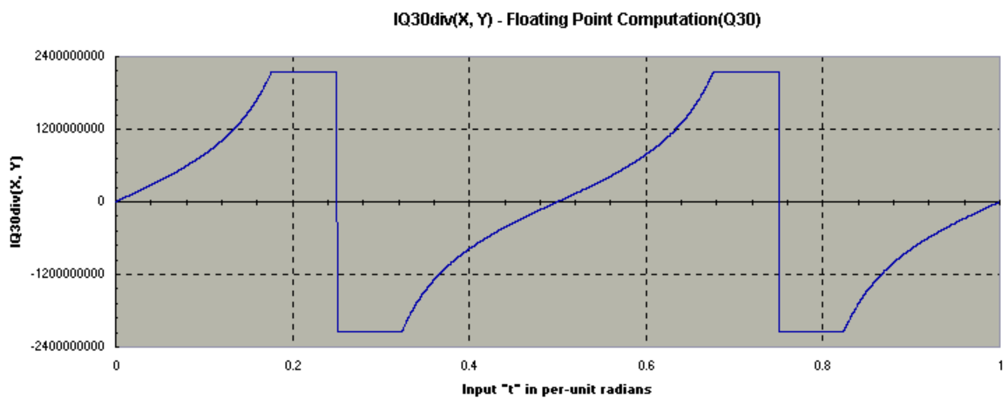
The following example obtains $^1\!/_{1.5}$=0.666 assuming that GLOBAL_Q is set to Q28 format in the IQmath header file.

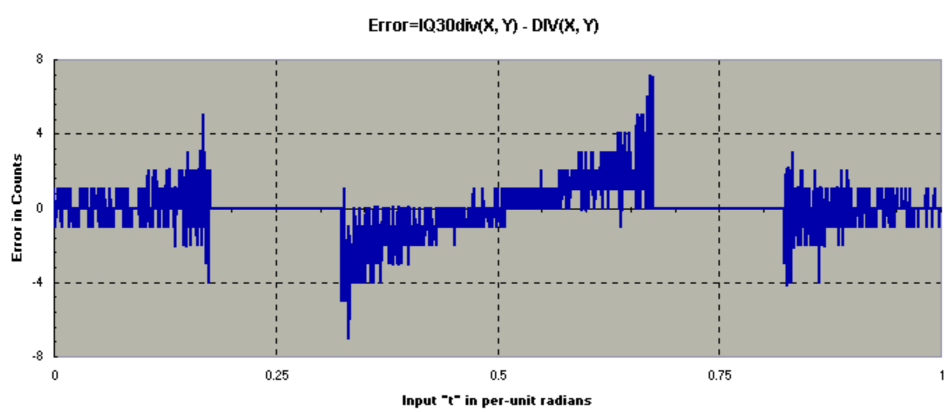
```
#include “IQmathLib.h”
_iq in1 out1;
_iq28 in2 out2;

void main(void )
{
   in1  = _IQ(1.5);
   out1 = _IQdiv(_IQ(1.0), in1);

   in2  = _IQ28(1.5);
   out2 = _IQ28div(_IQ28(1.0), in2);
}
```

Fixed Point division vs. C Float division

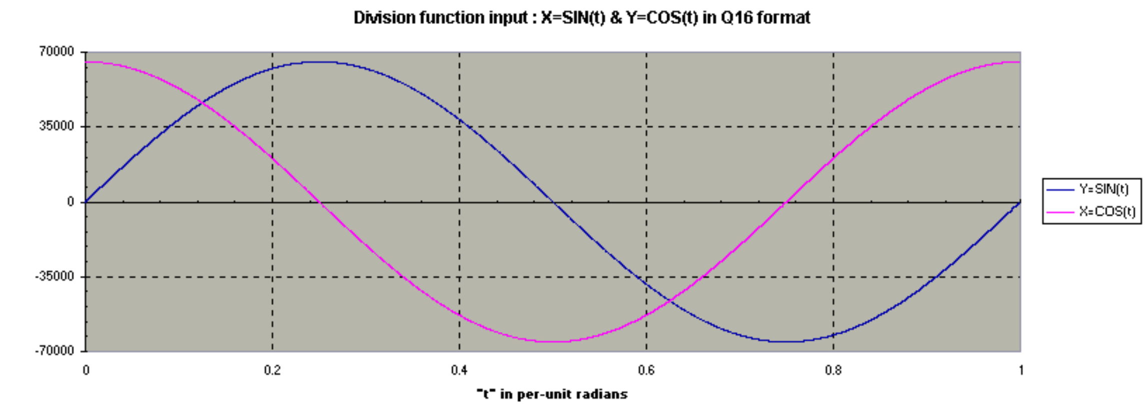


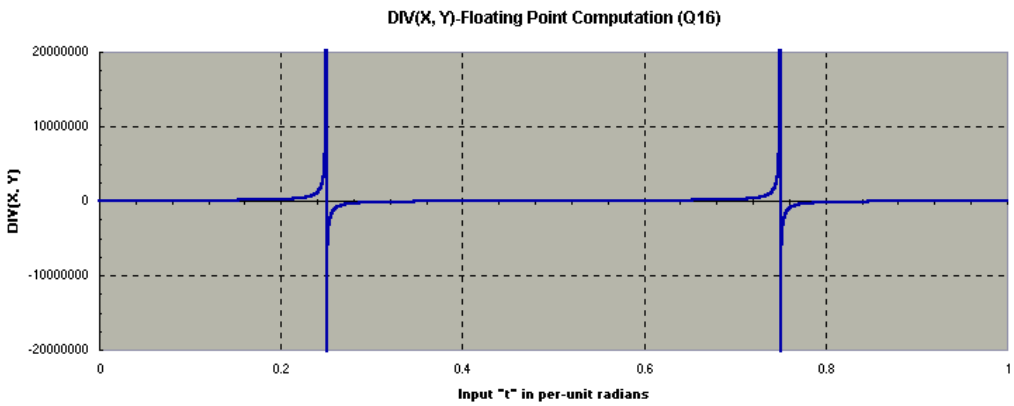


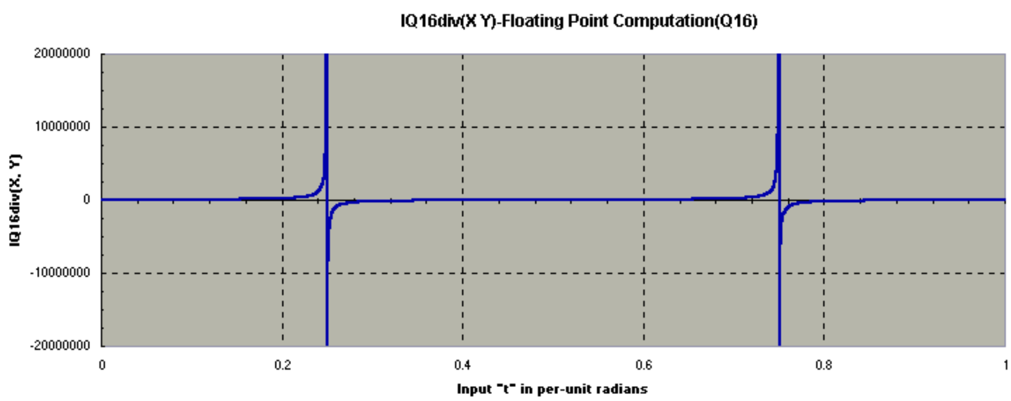


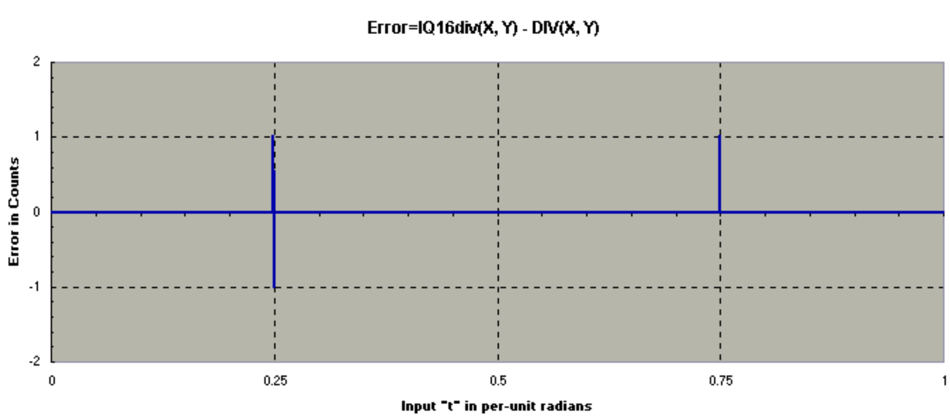


Fixed Point division vs. C Float division









5.4. Trigonometric Functions

IQNasin	***Fixed point ASIN (radians)***

Description This module computes the inverse sine of the input and returns the result in radians.

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

```
C      _iq _IQasin(_iq A)
C++     iq  IQasin(const iq &A)
```

Q format specific IQ function (IQ format = IQ1 to IQ29)

```
C      _iqN _IQNasin(_iqN A)
C++     iqN  IQNasin(const iqN &A)
```

Input **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 **Global IQ function (IQ format = GLOBAL_Q)**
This function returns the inverse 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 inverse sine of the input argument as fixed-point number in IQN format (N=1:29)

Example The following example obtains the result of the equation `asin(0.70710678) = (.25 x pi)`. It assumes that GLOBAL_Q is set to Q29 format in the IQmath header file.

```
#include “IQmathLib.h”
#define   PI 3.14156
_iq   in1, out1;
_iq29 in2, out2;

void main(void )
{
// in1  = in2  =  0.70710678L x 2^29 = 0x16A09E60
// out1 = out2 = asin(0.70710678) = 0x1921FB4A

   in1  =_IQ(0.70710678L);
   out1 =_IQasin(in1);

   in2  =_IQ29(0.70710678L)
   out2 =_IQ29asin(in2);
}
```

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.

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

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

Q format specific IQ function (IQ format = IQ1 to IQ29)

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

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

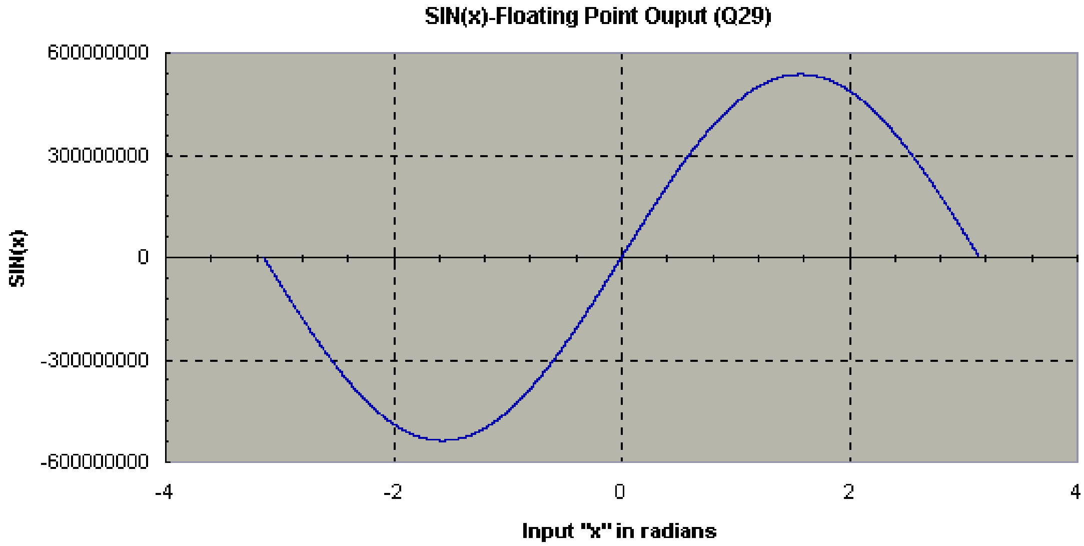
Q format specific IQ function (IQ format = IQ1 to IQ29)
This function returns the sine of the input argument as fixed-point number in IQN format (N=1:29)

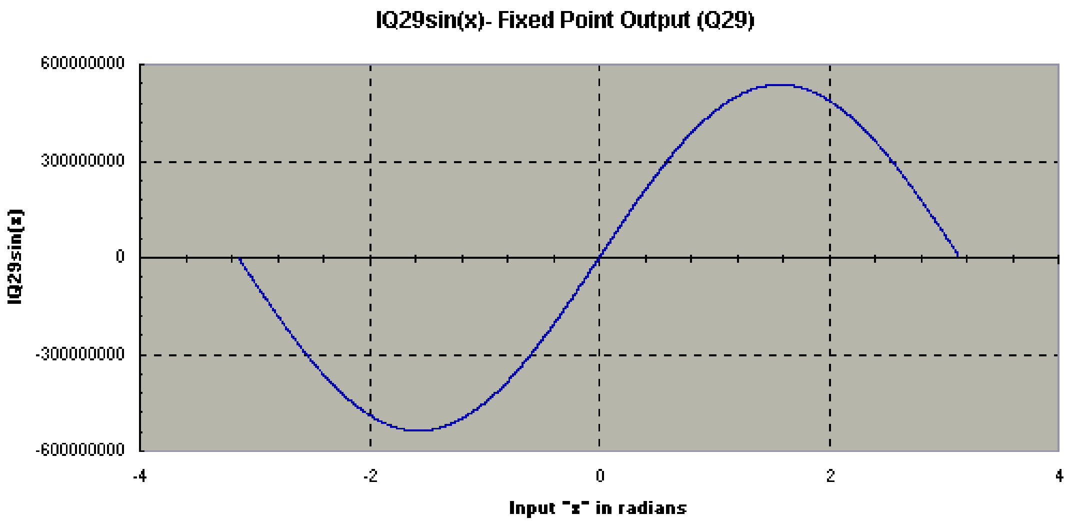
Accuracy $= 20\log_2\left(\pi \times 2^{29}\right) - 20\log_2(1)$ = 30 bits

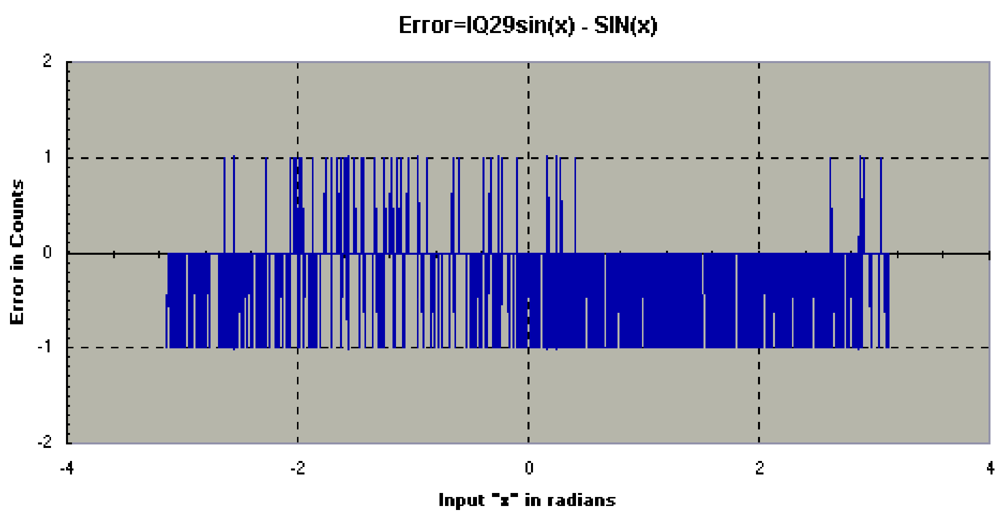
Example The following example obtains sin (0.25 x pi) = 0.707 assuming that GLOBAL_Q is set to Q29 format in the IQmath header file.

```
#include "IQmathLib.h"
#define   PI 3.14156
_iq in1, out1;
_iq28 in2, out2;

void main(void ){
// in1  = 0.25 x pi x 2^29 = 0x1921FB54
// out1 = sin(0.25 x pi)x 2^29 = 0x16A09E66
// in2  = 0x25 x pi x 2^29 = 0x1921FB54
// out2 = sin(0.25 x pi)x 2^29 = 0x16A09E66

   in1=_IQ(0.25*PI);
   out1=_IQsin(in1)
   in2=_IQ29(0.25*PI)
   out2=_IQ29sin(in2);
}
```

IQNsin Function vs. C Float SIN: Input varies from $-\pi$ to π







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.

Declaration
Global IQ function (IQ format = GLOBAL_Q)

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

Q format specific IQ function (IQ format = IQ1 to IQ30)

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

Input
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
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)

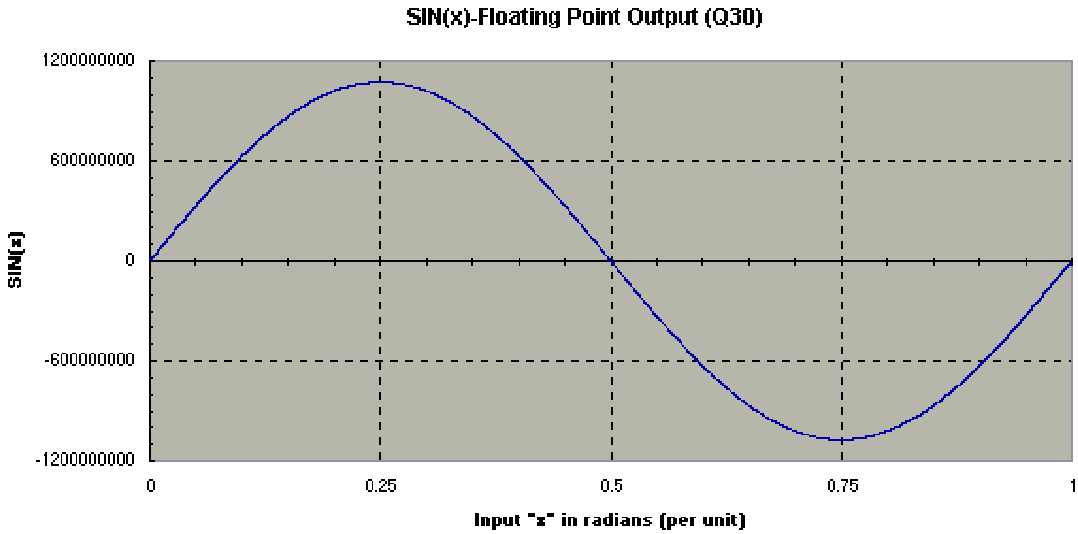
Accuracy = $20\log_2\left(1\times 2^{30}\right)-20\log_2\left(1\right)$ = 30 bits

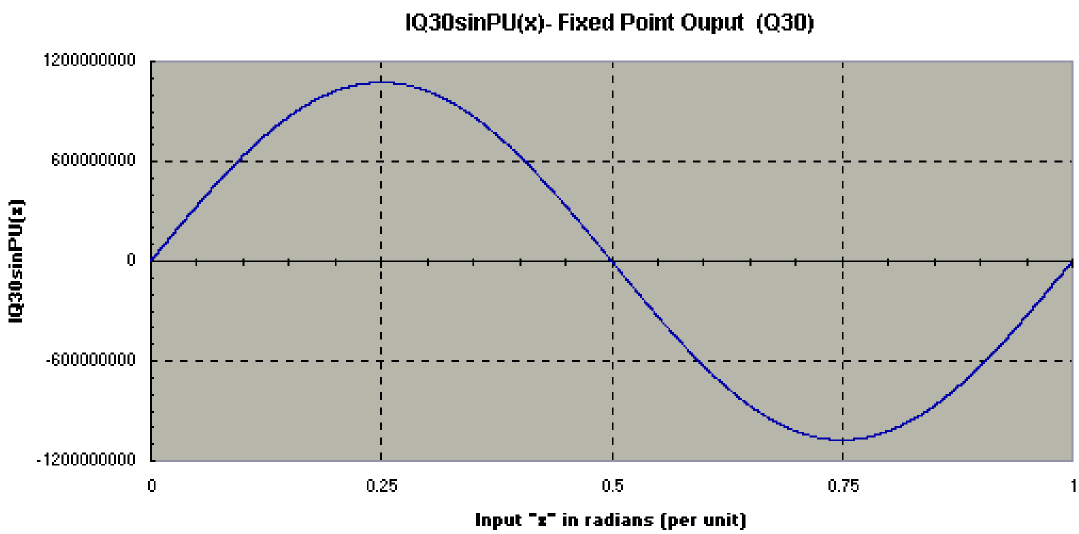
Usage Example:

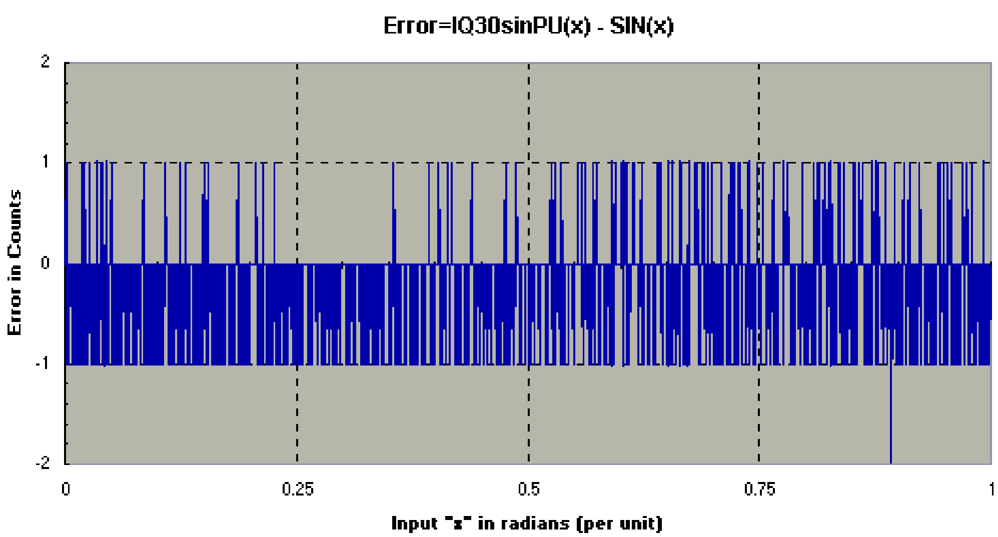
The following example obtains the $\sin\left(0.25\times\pi\right)$=0.707 assuming that GLOBAL_Q is set to Q30 format in the IQmath header file.

```
#include “IQmathLib.h”
#define   PI 3.14156
_iq   in1, out1;
_iq30 in2, out2;

void main(void ){
// in1  = in2 =  (0.25 x PI)/(2PI) x 2^30
//            = (.25/2) x 2^30 = 0x08000000 or .125
// out1 = out2 = sinPU(0.25/2) x 2^30 = 0x2D413CCC or .707

   in1  =_IQ(0.25L/2.0L);
   out1 =_IQsinPU(in1)
   in2  =_IQ30(0.25*PI/PI);
   out2 =_IQ30sinPU(in2);
}
```

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







IQNacos	*Fixed point ACOS (radians)*

Description This module computes the inverse cosine of the input and returns the result in radians.

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

```
C      _iq _IQacos(_iq A)
C++     iq _IQacos( const iq & A)
```

Q format specific IQ function (IQ format = N = 1 to 29)

```
C      _iqN _IQNacos(_iqN A)
C++    _iq N _IQNacos(const iqN &A)
```

Input **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 **Global IQ function (IQ format = GLOBAL_Q)**

This function returns the inverse 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 inverse cosine of the input argument as fixed-point number in IQN format (N=1:29)

Example The following example obtains the result of $\mathrm{asin}\,(0.70710678) = (0.25 \times \pi)$ assuming that GLOBAL_Q is set to Q29 format in the IQmath header file.

```
#include “IQmathLib.h”
 _iq in1, out1;
_iq29 in2, out2;

void main(void )
{
// in1  = 0x70710678 x 2^29 = 0x16A09E60
// out1 = acos(0.70710678L) x 2^29 = 0x1921FB5E
// in2  = 0x70710678 x 2^29 = 0x16A09E60
// out2 = acos(0.70710678L) x 2^29 = 0x1921FB5E

   in1  =_IQ(0.70710678L);
   out1 =_IQacos(in1);
   in2  =_IQ29(0.70710678L)
   out2 =_IQ29acos(in2);

}
```

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.

Declaration

Global IQ function (IQ format = GLOBAL_Q)

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

Q format specific IQ function (IQ format = IQ1 to IQ29)

```
C      _iqN _IQNcos(_iqN A)
C++     iqN  IQNcos(const iqN &A)
```

Input

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

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)

Accuracy

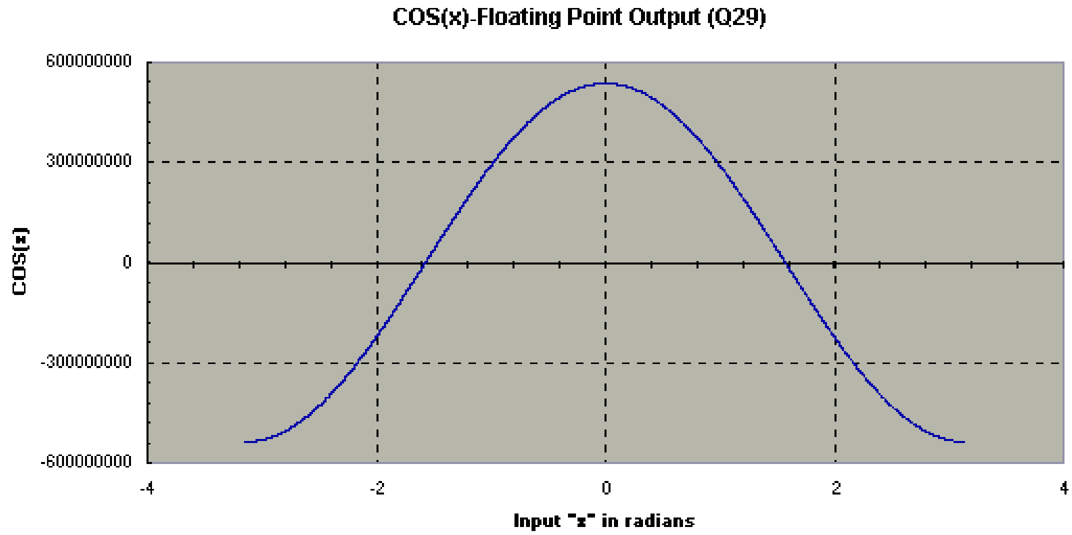
$= 20\log_2\left(\pi \times 2^{29}\right) - 20\log_2(2)$ = 30 bits

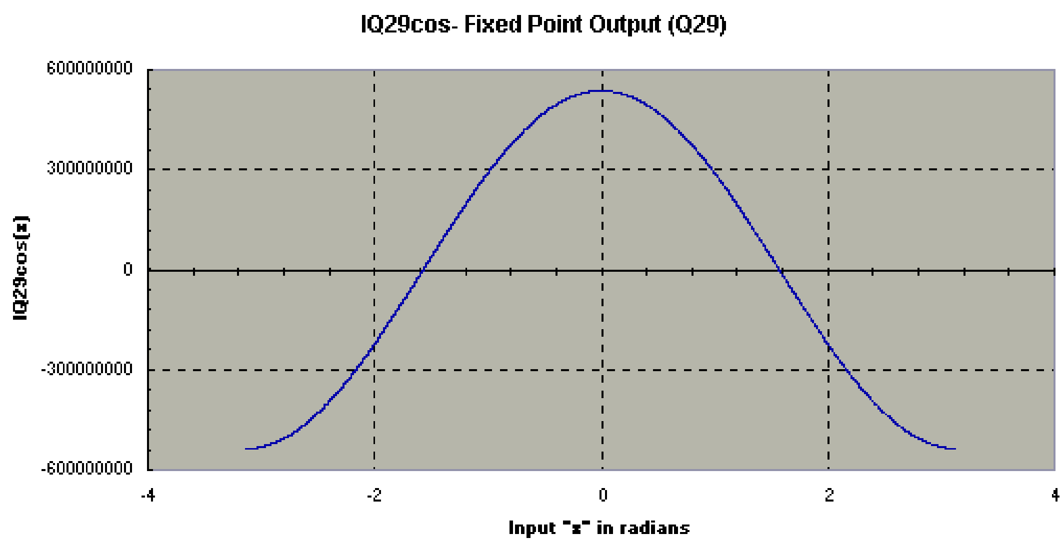
Example

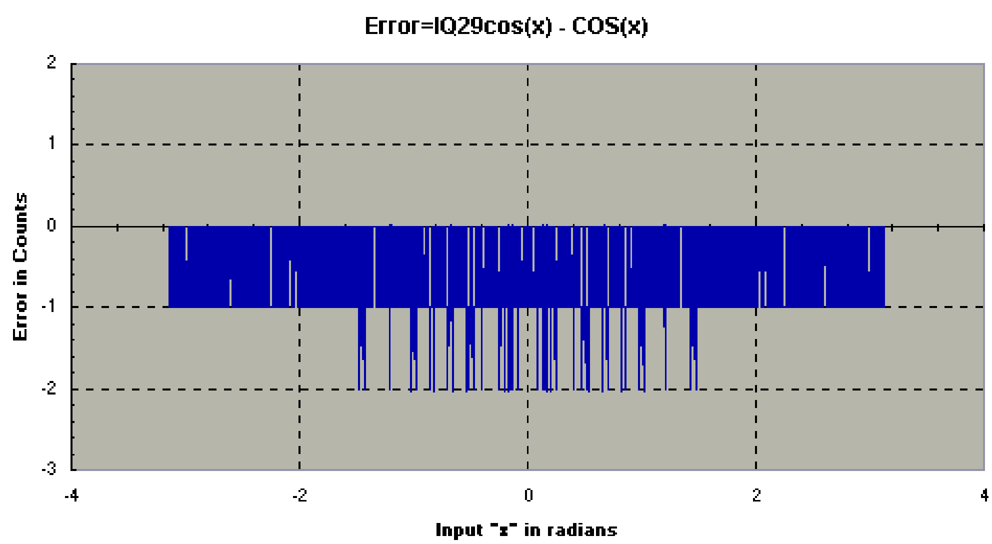
The following example obtains the $\cos(0.25 \times \pi)$=0.707 assuming that GLOBAL_Q is set to Q29 format in the IQmath header file.

```
#include “IQmathLib.h”
#define   PI 3.14156

_iq in1, out1;
_iq29 in2 out2;

void main(void ){
// in   = 0.25 x PI x 2^29 = 0x1921FB54
// out1 = cos(.25 x PI) x 2^29 = 0x16A09E66
// in   = 0.25 x PI x 2^29 = 0x1921FB54
// out1 = cos(.25 x PI) x 2^29 = 0x16A09E66

   in1  =_IQ(0.25*PI);
   out1 =_IQcos(in1);
   in2  =_IQ29(0.25*PI);
   out2 =_IQ29cos(in2);
}
```

Fixed Point COS Function vs. C Float COS: Input varies from $-\pi$ to π







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.

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

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

Q format specific IQ function (IQ format = IQ1 to IQ30)

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

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

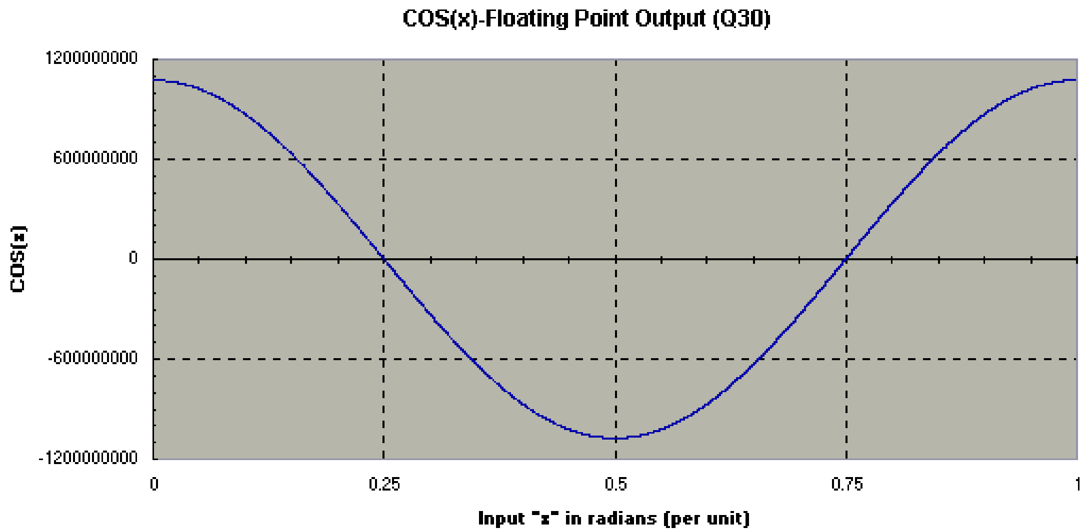
Q format specific IQ function (IQ format = IQ1 to IQ30)
This function returns the sine of the input argument as fixed-point number in IQN format (N=1:30)

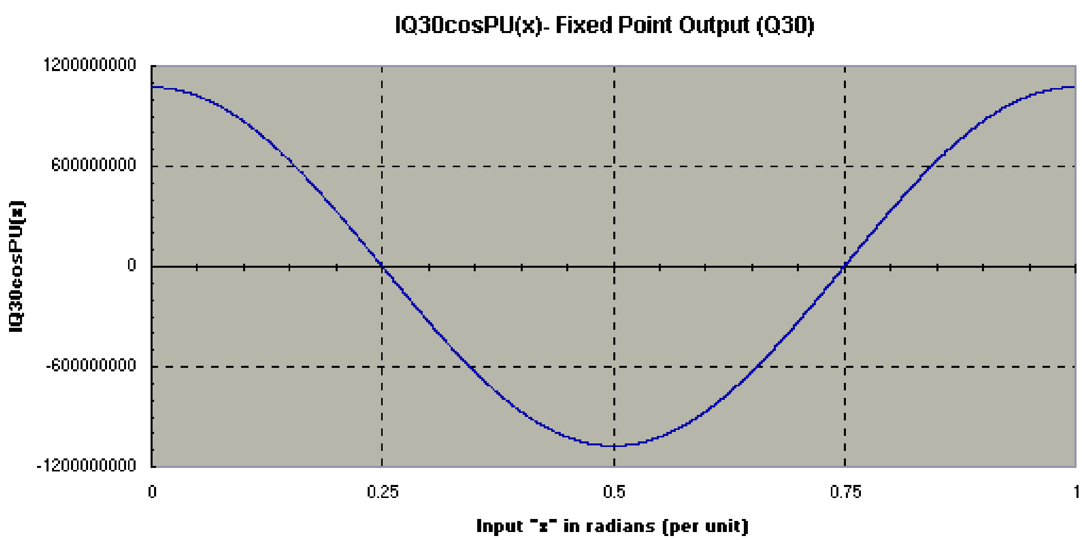
Accuracy $= 20\log_2\left(1\times 2^{30}\right) - 20\log_2(2)$ = 29 bits

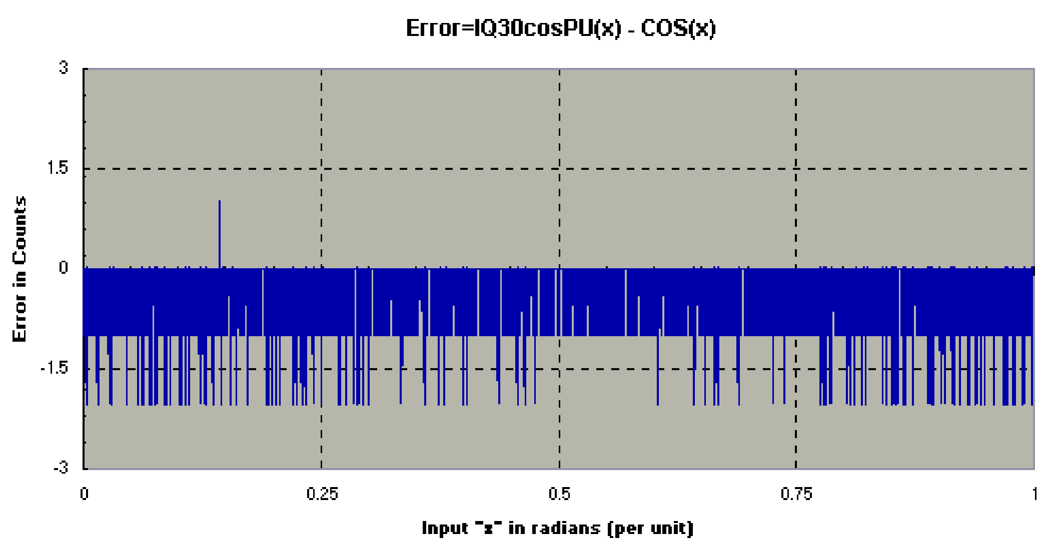
Usage **Example:** The following sample code obtains the $\cos(0.25\times\pi)$=0.707 assuming that GLOBAL_Q is set to Q30 format in the IQmath header file.

```
#include “IQmathLib.h”
#define  PI 3.14156

_iq in1, out1;
_iq30 in2, out2

void main(void ){
// in1  = in2  = (0.25 x PI)/(2PI) x 2^30
//              = (.025/2.0) x 2^30 = 0x08000000 or .125
// out1 = out2 = cosPU(0.25/2.0) x 2^30
               = 0x2D413CCC or .707

  in1  =_IQ(0.25L/2.0L);
  out1 =_IQcosPU(in1)

  in2  =_IQ30(0.25L/2.0L);
  out2 =_IQ30cosPU(in2);
}
```

Fixed Point COS Function vs. C Float COS: Input varies from 0 to 2π in per unit







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 $-\pi$ to π

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

```
C      _iq _IQatan2(_iq A, _iq B)
C++     iq  IQatan2(const iq &A, const iq &B)
```

Q format specific IQ function (IQ format = IQ1 to IQ29)

```
C      _iqN _IQNatan2(_iqN A, _iqN B)
C++     iqN  IQNatan2(const iqN &A, const iqN &B)
```

Input **Global IQ function (IQ format = GLOBAL_Q)**
Inputs A and B are fixed-point numbers represented in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ29)
Inputs A and B are fixed-point numbers in IQN format (N=1:29)

Output **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 $[-\pi,+\pi]$

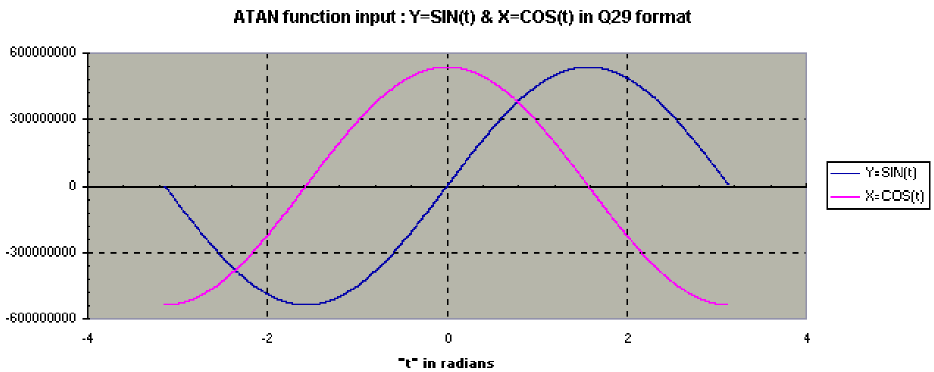
Q format specific IQ function (IQ format = IQ1 to IQ29)
This function returns the inverse tangent of the input argument as fixed-point number in IQN format (N=1:29). The output contains the angle in radians between $[-\pi,+\pi]$

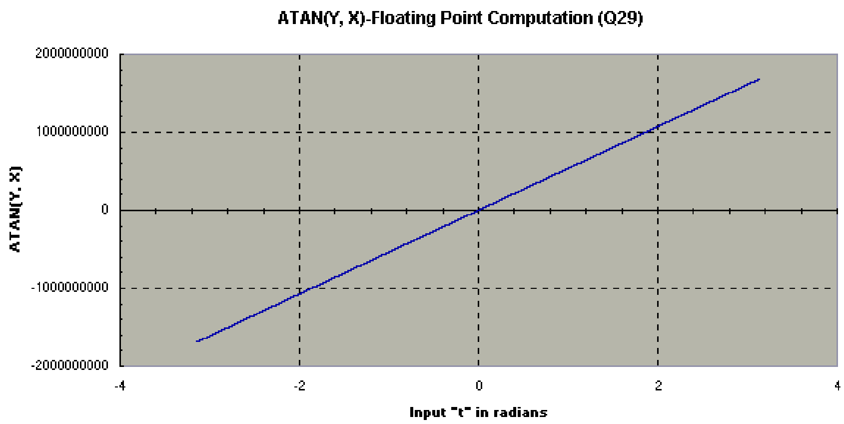
Accuracy = $20\log_2(\pi \times 2^{29}) - 20\log_2(32)$ = 26 bits

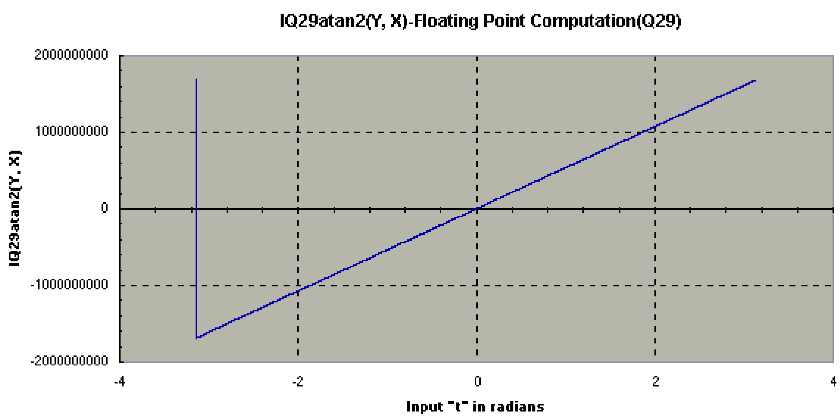
Usage The following example obtains $\tan^{-1}(\sin(\pi/5), \cos(\pi/5)) = \pi/5$, assuming that the GLOBAL_Q is set to Q29 format in the IQmath header file.

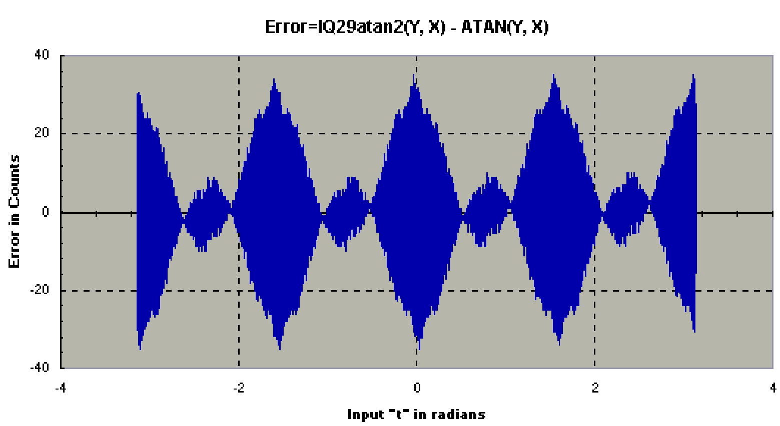
```
#include “IQmathLib.h”
#define    PI 3.14156L
_iq   xin1, yin1, out1;
_iq29 xin2, yin2, out2;
void main(void ){
 // xin1 = xin2 = cos(PI/5) x 2^29 = 0x19E37FA8
 // yin1 = yin2 = sin(PI/5) x 2^29 = 0x12CF17EF
 // out1 = out2 = PI/5 x 2^29 = 0x141B21C3

    xin1  = _IQcos(_IQ(PI/5.0L));
    yin1  = _IQsin(_IQ(PI/5.0L));
    out1  = _IQatan2(yin1,xin1);

    xin2  = _IQ29cos(_IQ29(PI/5.0L));
    yin2  = _IQ29sin(_IQ29(PI/5.0L));
    out2  = _IQ29atan2(yin1,xin1);
}
```

Fixed Point ARCTAN Function vs. C Float ARCTAN









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

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

```
C      _iq _IQatan2PU(_iq A, _iq B)
C++    _iq _IQatan2PU(const iq &A, const iq &B)
```

Q format specific IQ function (IQ format = IQ1 to IQ29)

```
C      _iqN _IQNatan2PU(_iqN A, _iqN B)
C++    _iqN _IQNatan2PU(const iqN &A, const iqN &B)
```

Input **Global IQ function (IQ format = GLOBAL_Q)**
Inputs A and B are fixed-point number represented in GLOBAL_Q format.

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

Output **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 (2π 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 (2π radians).

Accuracy = $20\log_2\left(1\times 2^{29}\right)-20\log_2\left(6\right)$ = 27 bits

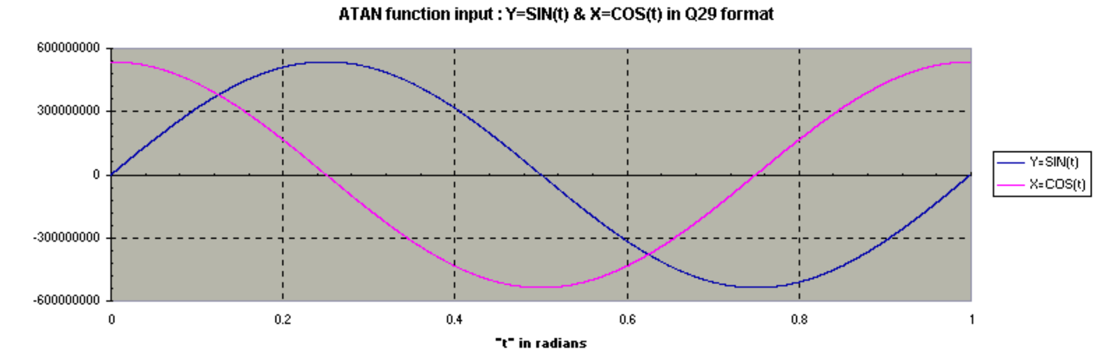
Usage The atan2PU for the floating point math is defined such that it incurs two calls to the atan2 function. The user is advised to use atan2 and do the conversion to PU in his own code for most efficient code.
temp = atan2(A,B)*(1.0/6.283185307);
if(temp<0.0) temp=temp+1; // where temp will have the PU value

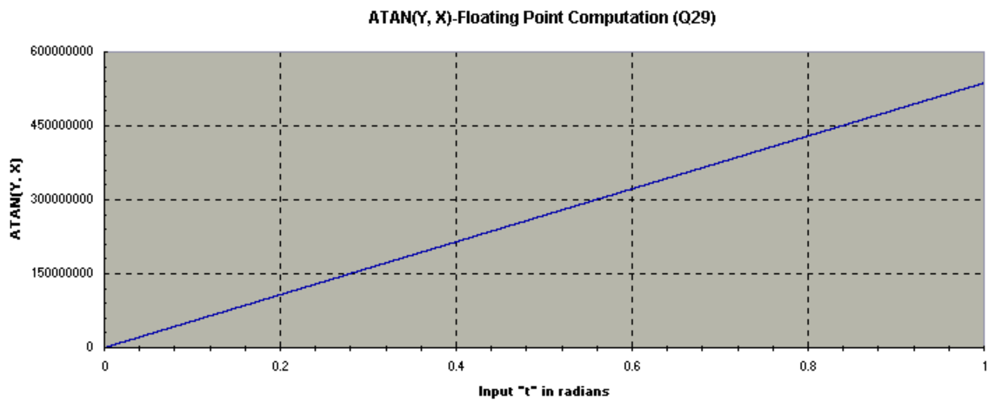
The following sample code obtains $\tan^{-1}\left(\sin\left(\pi/5\right),\cos\left(\pi/5\right)\right)=\pi/5$, assuming that GLOBAL_Q is set to Q29 format in the IQmath header file.

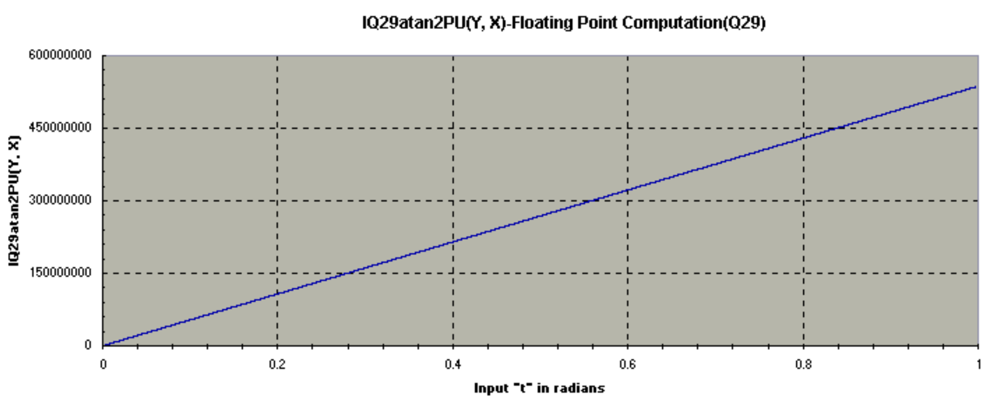
```
#include "IQmathLib.h"
#define    PI 3.14156L
_iq   xin1, yin1, out1;
_iq29 xin2, yin2, out2;
void main(void ){
 // xin1 = xin2 = cos(PI/5) x 2^29 = 0x19E37FA8
 // yin1 = yin2 = sin(PI/5) x 2^29 = 0x12CF17EF
 // out1 = out2 = (PI/5)/(2PI) x 2^29 = 0x03333104

    xin1 = _IQcos(_IQ(PI/5.0L));
    yin1 = _IQsin(_IQ(PI/5.0L));
    out1 = _IQatan2PU(yin1,xin1);
```

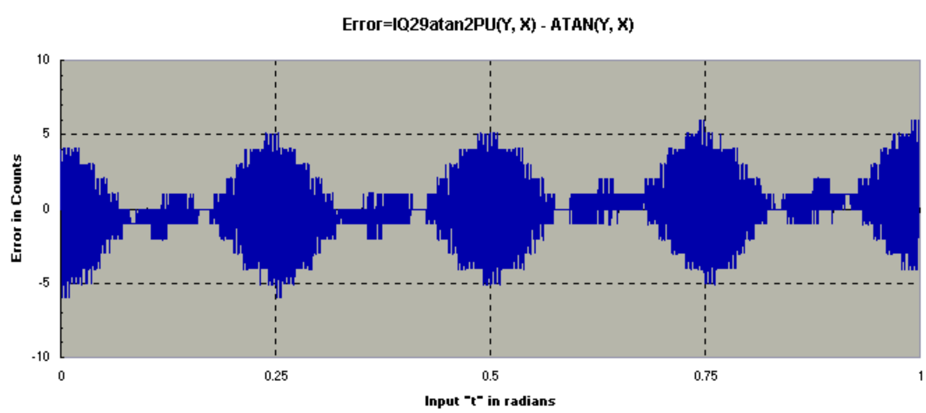
```
    xin2 = _IQ29cos(_IQ29(PI/5.0L));
    yin2 = _IQ29sin(_IQ29(PI/5.0L));
    out2 = _IQ29atan2PU(yin1,xin1);
}
```

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 $\pi/2$.

Declaration **Global IQ Macro (IQ format = GLOBAL_Q)**

```
C      _iq _IQatan(_iq A)
C++    _iq _IQatan(const _iq &A)
```

Q format specific IQ Macro (IQ format = IQ1 to IQ29)

```
C      _iqN  _IQNatan(_iqN A)
C++    _iqN  _IQNatan(const _iqN &A)
```

Input **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 **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 $\left[-\pi/2, +\pi/2\right]$

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

Usage 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"
_iq in1, out1;
_iq29 in2, out2;

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

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

5.5. Mathematical Utilities

IQNexp	***Fixed point Exponential***

Description This module computes the exponential of a value A.

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

```
C       _iq _IQexp(_iq A)
C++     _iq _IQexp(const iq &A)
```

Q format specific IQ function (IQ format = IQ1 to IQ30)

```
C       _iqN _IQNexp(_iqN A)
C++     _iqN _IQNexp(const iqN &A)
```

Input **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 **Global IQ function (IQ format = GLOBAL_Q)**
Exponential value of the input in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Exponential value of the input in IQN format (N=1:30).

Example Calculate $e(1.8)$= 6.0496474, assuming that GLOBAL_Q is set to Q24 format in the IQmath header file.

```
#include "IQmathLib.h"
_iq in1, out1;
_iq30 in2, out2;

void main(void )
{
// in1  = in2  = 1.8 x 2^24 = 0x01CCCCCC
// out1 = out2 = exp(1.8) x 2^24 = 0x060CB5AA

   in1  = _IQ(1.8);
   out1 = _IQexp(x);

   in2  = _IQ24(1.8);
   out2 = _IQ24exp(x);
}
```

IQNlog

Fixed point Natural Log

Description This module computes the natural log of A.

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

```
C      _iq _IQlog(_iq A)
C++    _iq _IQlog(const _iq &A)
```

Q format specific IQ function (IQ format = IQ1 to IQ30)

```
C      _iqN _IQNlog(_iqN A)
C++    _iqN _IQNlog(const _iqN &A)
```

Input **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=1:29).

Output **Global IQ function (IQ format = GLOBAL_Q)**
Exponential value of the input in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Exponential value of the input in IQN format (N=1:29).

Example Calculate log(6.0496474) = 1.8, assuming that GLOBAL_Q is set to Q24 format in the IQmath header file.

```
#include "IQmathLib.h"
_iq in1, out1;
_iq30 in2, out2;

void main(void )
{
// in1  = in2  = 6.0496474 x 2^24 = 0x060CB5B0
// out1 = out2 = log(6.0496474) x 2^24 = 0x01CCCCCC

   in1  = _IQ(1.8);
   out1 = _IQlog(x);

   in2  = _IQ24(1.8);
   out2 = _IQ24log(x);
}
```

IQNsqrt

Fixed point Square-root

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

```
C      _iq _IQsqrt(_iq A)
C++     iq  IQsqrt(const iq &A)
```

Q format specific IQ function (IQ format = IQ1 to IQ30)

```
C      _iqN _IQNsqrt(_iqN A)
C++     iqN  IQNsqrt(const iqN &A)
```

Input

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

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

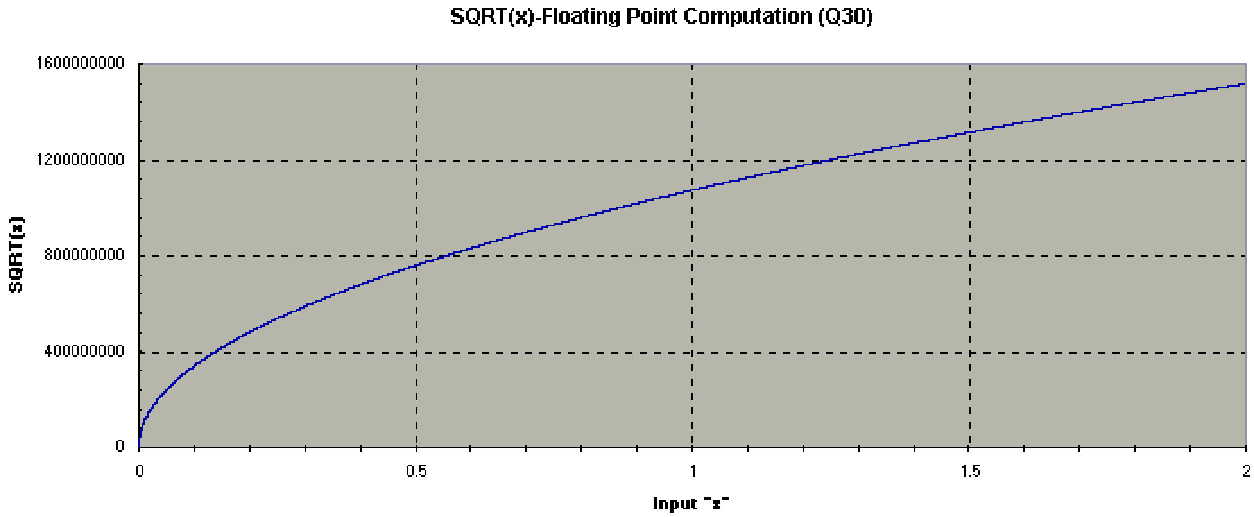
Q format specific IQ function (IQ format = IQ1 to IQ30)
Square root of input in IQN format (N=1:30)

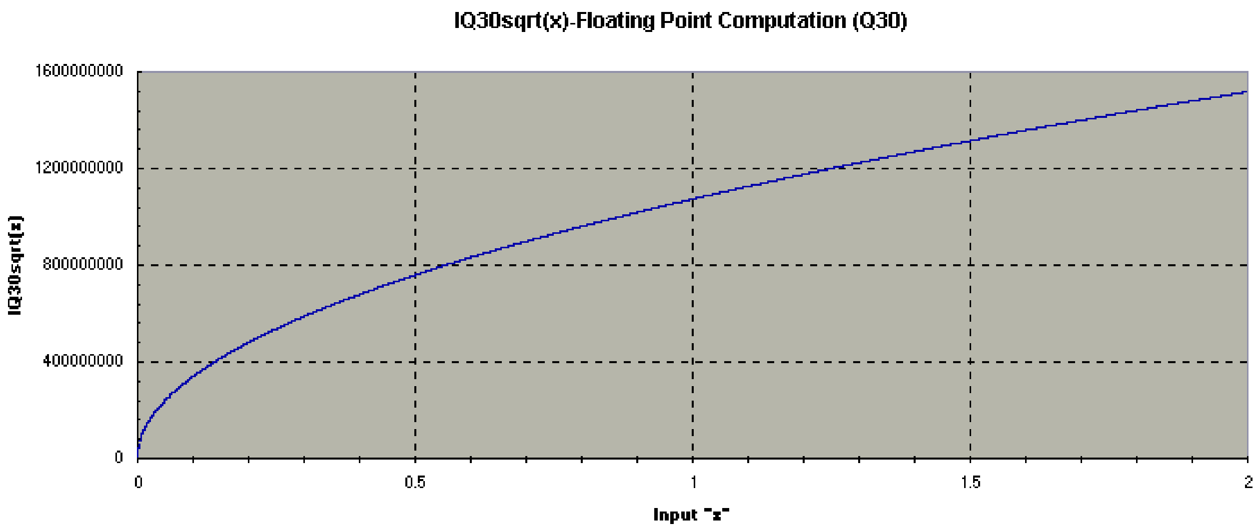
Accuracy $= 20\log_2\left(2^{31}\right) - 20\log_2\left(6\right)$ = 29 bits

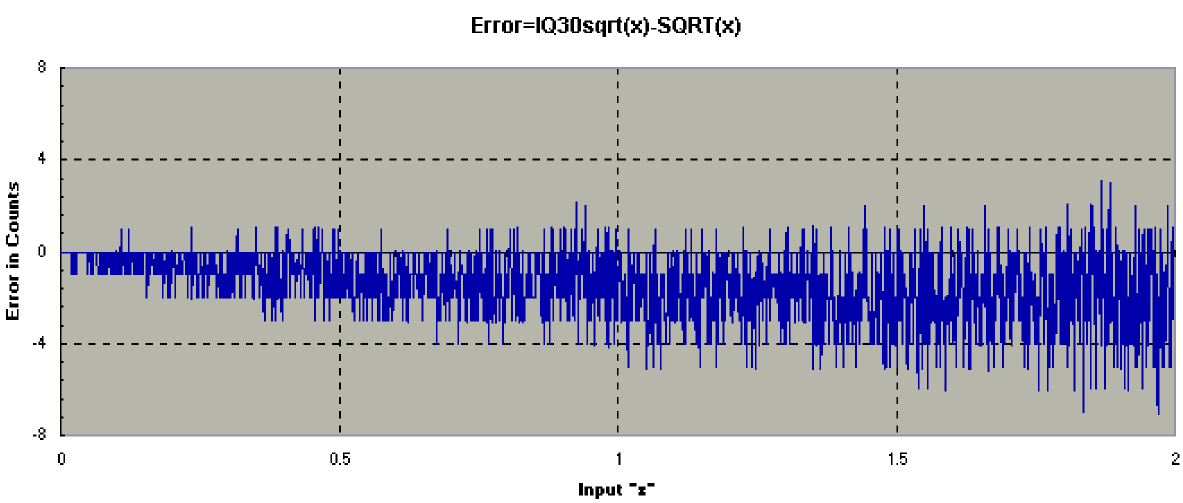
Usage Calculate $\sqrt{1.8}$ = 1.34164, assuming that GLOBAL_Q is set to Q30 format in IQmath header file.

```
#include "IQmathLib.h"
_iq in1, out1;
_iq30 in2, out2;

void main(void )
{
// in1  = in2  = 1.8 x 2^30 = 0x73333333
// out1 = out2 = sqrt(1.8) x 2^30 = 0x55DD7151

  in1  = _IQ(1.8);
  out1 = _IQsqrt(x);

  in2  = _IQ30(1.8);
  out2 = _IQ30sqrt(x);
}
```

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.

Declaration **Global IQ function (IQ format = GLOBAL_Q)**

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

Q format specific IQ function (IQ format = IQ1 to IQ30)

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

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

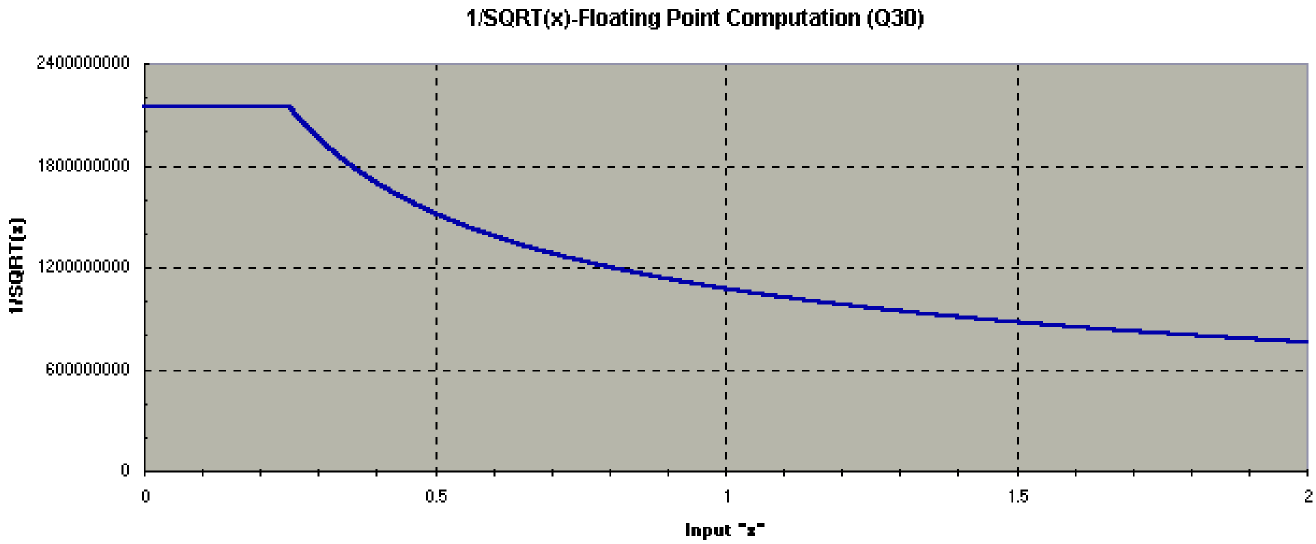
Q format specific IQ function (IQ format = IQ1 to IQ30)
Inverse square root of input expressed in IQN format (N=1:30)

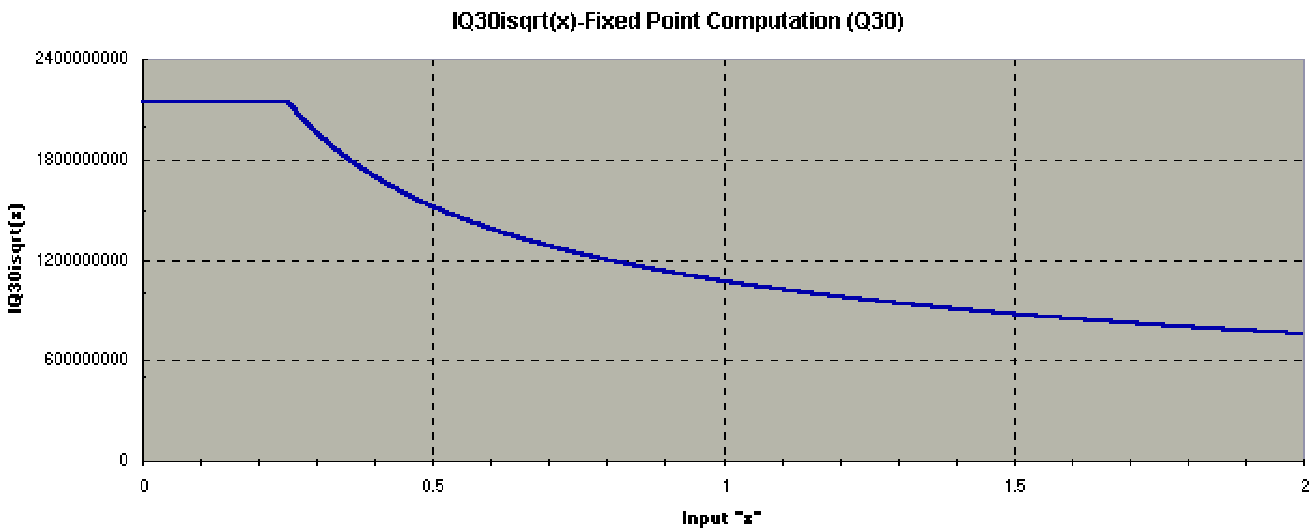
Accuracy = $20\log_2\left(2^{31}\right) - 20\log_2\left(5\right)$ = 29 bits

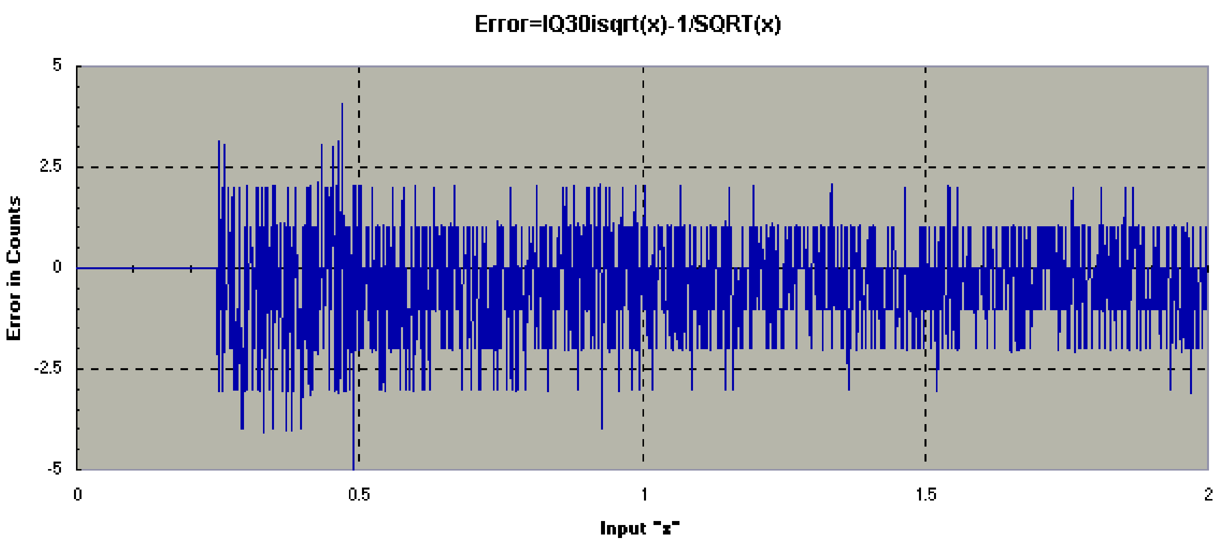
Usage Calculate $1/\sqrt{1.8}$ =0.74535 assuming that GLOBAL_Q is set to Q30 format in the IQmath header file.

```
#include "IQmathLib.h"

_iq in1, out1;
_iq30 in2, out2;

void main(void )
{

// in1  = in2  = 1.8 x 2^30 = 0x73333333
// out1 = out2 = 1/sqrt(1.8) x 2^30 = 0x2FB3E99E

   in1  =_IQ(1.8);
   out1 =_IQisqrt(in1);

   in2  =_IQ30(1.8);
   out2 =_IQ30isqrt(in2);
}
```

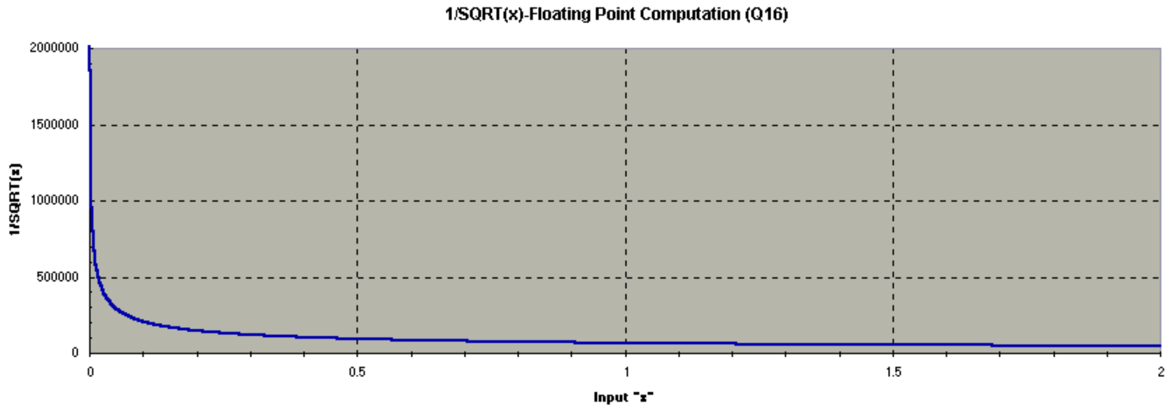
Fixed Point inverse SQRT Function vs. C Float inverse SQRT

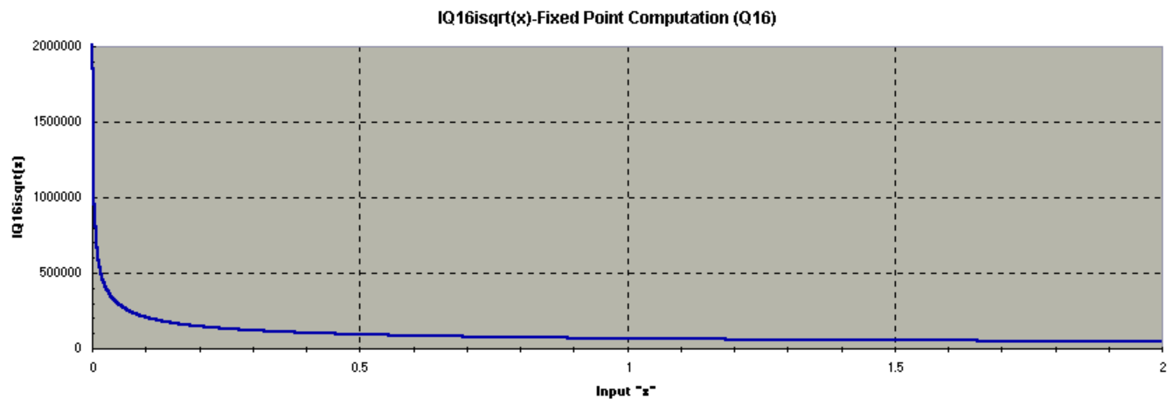


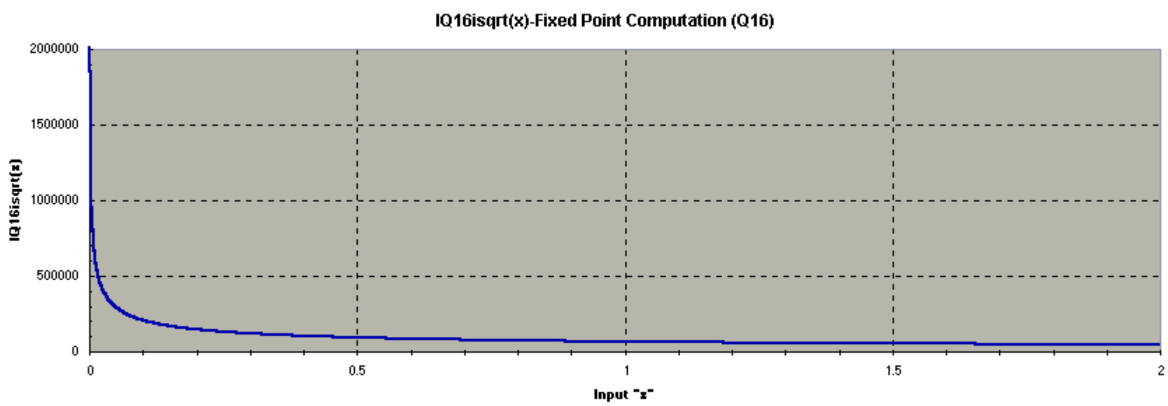


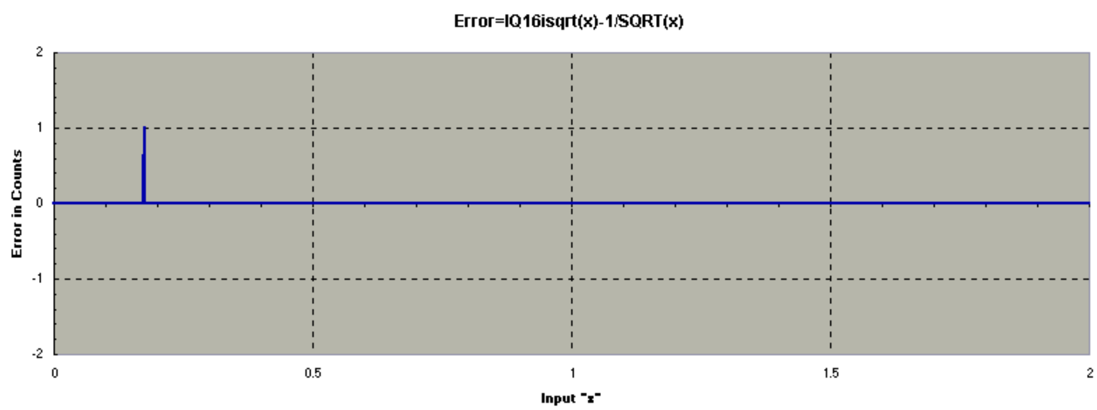


Fixed Point inverse SQRT Function vs. C Float inverse SQRT









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.

Declaration

Global IQ function (IQ format = GLOBAL_Q)

```
C       _iq _IQmag(_iq A, _iq B)
C++      iq  IQmag(const iq &A, const iq &B)
```

Q format specific IQ function (IQ format = IQ1 to IQ30)

```
C       _iqN _IQNmag(_iqN A, _iqN B)
C++      iqN  IQNmag(const iqN &A, const iqN &B)
```

Input

Global IQ function (IQ format = GLOBAL_Q)
Inputs A and B are IQ numbers represented in GLOBAL_Q format.

Q format specific IQ function (IQ format = IQ1 to IQ30)
Inputs A and B are IQ numbers represented in IQN format

Output

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

Accuracy

29-bits (Same as SQRT function)

Usage

Example:

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

```
#include “IQmathLib.h”
// Complex number = real1 + j*imag1
// Complex number = real2 + j*imag2

_iq real1, imag1, mag1;
_iq28 real2, imag2, mag2;

void main(void )
{

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

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

}
```

5.6. Miscellaneous Utilities

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)**

```
C      _iq _IQabs(_iq A)
C++     iq  IQabs(const iq &A)
```

Q format specific IQ function (IQ format = IQ1 to IQ30)

```
C      _iqN _IQNabs(_iqN A)
C++     iqN  IQNabs(const iqN &A)
```

Input **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 **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 assuming GLOBAL_Q=IQ28 in the IQmath header file.

```
#include "IQmathLib.h"

void main(void)
{
    _iq xin1, xin2, xin3, xsum;
    _iq20 yin1, yin2, yin3, ysum;

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

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

Output Format Global IQ function (IQ format = GLOBAL_Q)
Saturated output 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:

```
#include "IQmathLib.h

void main(void)
{
  _iq Y, M, X, B;        // GLOBAL_Q = 26 (+/- 32 range)
  _iq20 temp;            // IQ = 20 (+/- 2048 range)

  temp = _IQ20mpy(_IQtoIQ20(M), _IQtoIQ20(X)) +
            _IQtoIQ20(B);
  temp = _IQsat(temp, _IQtoIQ20(MAX_IQ_POS),
            _IQtoIQ20(MAX_IQ_NEG));

  Y = _IQ20toIQ(temp);
}
```

Chapter 6. Revision History

Version	Date	Comment
V1.64.00.00	October 1, 2020	Migrated to compiler 20.2.1 and CCS 10.x
V1.61.00.00	May 24, 2019	Added IQmath eabi library and IQmath_fpu32 eabi library. Added index library IQmath.lib and IQmath_fpu.lib. Added support for CCS 9.x and examples on F2837xd and F2838. Fixed bug in _IQmpyI32frac(A,B) in FLOAT_MATH. Fixed bug in __IQNtoa to report error for 0x80000000.
V1.60.01.00	December 15, 2016	Updated directory details for C2000Ware
V1.6.0	August 31, 2011	Added IQNlog function Added example for the 2806x devices Added graph_properties folder. The files in this folder can be used to populate the Code Composer watch window and graphs through the scripting console. *View->scripting console*

V1.5c	June 6, 2010	Updates made to only the IQmathLib.h file. No changes were made to the library code itself.
		
		• Added left shift and right shift #defines for multiplying and dividing by power of 2 in IQ_MATH:
		
		`#define _IQmpy2(A)          ((A)<<1)`
		`#define _IQmpy4(A)          ((A)<<2)`
		`. . .`
		`#define _IQmpy64(A)         ((A)<<6)`
		
		`#define _IQdiv2(A)          ((A)>>1)`
		`#define _IQmpy4(A)          ((A)>>2)`
		`. . .`
		`#define _IQdiv64(A)         ((A)>>6)`
		
		• Added corresponding Multiply/Divide in FLOAT_MATH:
		
		`#define _IQmpy2(A)        ((A)*2.0)`
		`#define _IQmpy4(A)        ((A)*4.0)`
		`. . .`
		`#define _IQmpy64(A)       ((A)*64.0)`
		
		• FLOAT_MATH: Corrected the #defines for conversion from IQ to Q15
		  o Removed the "L" from the constant so the compiler will not call the 64-bit floating point routine in the RTS library.
		  o Removed the cast of the float to long before the multiply.
		• FLOAT_MATH: Modified IQdiv and IQNdiv macros such that the arguments are cast to float before the division. This will allow the macros to be used with

		integer types, and not just _iq types, properly. • Examples: Fixed some issues with the projects being portable under CCS 4. • Fixed the 28335 project so it links in the fpu32 versions of the rts library and IQmath. • Fixed typos in this document.
V1.5b	January 12, 2010	Minor release to add examples and build information for Code Composer Studio V4. Changed the install directory to fit with ControlSuite.
V1.5a	June 1, 2009	Rebuilt the IQmath.lib with large model enabled for the function IQNtoa. No other functions were changed. Updated documentation to include 2802x and 2803x devices. Added examples for Piccolo 28027 and 28035.
V1.5	July 8, 2008	Added IQNtoa. Header IQmathLib.h and IQmathCPP.h files were updated to fix typos and missing information. Refer to the header files themselves for more information. Added a version of the library built with the compiler switch --float_support=fpu32. This enables mixing the IQmath with the native floating support capabilities of C28x+FPU. Added information describing how to convert between IQmath and float math. Added the IQexp, IQasin, IQacos, and IQNtoa function information to this document. Added C++ information to this document. Previously this information was only in a readme.txt file. Added information regarding locations of tables stored in the boot ROM of different devices. Noted the IQmath .gel file is most useful when using a legacy debugger that does not support IQmath. Added examples for 2808, 28335 and 28235. Updated the example flow to better match the header file and peripheral examples also provided by TI. General documentation cleanup and improvements. Install is now under a version specific directory. Changed the directory structure to reduce the number of duplicated files.

V1.4f	March 10, 2005	Fixed bug in IQexp.
V1.4e	June 17, 2004	Added IQexp, IQasin, IQacos.
V1.4d	March 30, 2003	Previous Web Release