C28x IQmath Library

A Virtual Floating Point Engine

V1.5c

June 6, 2010

Module user’s Guide (SPRC990)

C28x Foundation Software

©Texas Instruments Inc., 2002-2010

1
IMPORTANT NOTICE

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

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

Customers are responsible for their applications using TI components.

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

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

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

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

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

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

Copyright ©2002, Texas Instruments Incorporated
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 ............................................................................................................ 6
1.1. Introduction .................................................................................................................. 6

Chapter 2. Installing the IQmath Library ............................................................................. 7
2.1. IQmath Package Contents .......................................................................................... 7
2.2. How to Install the IQmath Library ............................................................................ 8

Chapter 3. Using the IQmath Library .................................................................................. 9
3.1. IQmath Arguments and Data Types ........................................................................... 9
3.2. IQmath Data type: Range & Resolution ..................................................................... 10
3.3. Calling an IQmath Function from C .......................................................................... 11
3.4. Calling an IQmath function from C++ ....................................................................... 11
3.5. IQmath Section and Lookup Tables ......................................................................... 12
3.6. Accessing IQmath Functions in the Boot ROM ....................................................... 16
3.7. IQmath Naming Conventions ................................................................................... 19
3.8. Selecting the GLOBAL_Q format ........................................................................... 20
3.9. Using the IQmath GEL file for Debugging .............................................................. 21
3.10. Converting an IQmath Application to Floating-Point ........................................... 23
3.11. The IQmath C-Calling Convention ......................................................................... 23

Chapter 4. Libraries Available ............................................................................................ 24

Chapter 5. Function Summary .............................................................................................. 25
5.1. Arguments and Conventions Used .......................................................................... 25
5.2. IQmath Function Overview ..................................................................................... 26
   Format conversion Utilities .......................................................................................... 26
   Shift to Multiply or Divide by Powers of 2 ................................................................. 26
   Arithmetic Operations ............................................................................................... 27
   Trigonometric Functions: .......................................................................................... 27
   Mathematical Functions: .......................................................................................... 28
   Miscellaneous ............................................................................................................. 28
5.3. C28x IQmath Library Benchmarks ........................................................................... 29

Chapter 6. Function Descriptions ......................................................................................... 31
6.1. Conversion Utilities .................................................................................................... 31
   IQN.............................................................................................................................. 31
   IQN.............................................................................................................................. 31
   IQNtoF ...................................................................................................................... 33
   atofIQN ................................................................................................................... 34
   IQNtoa ...................................................................................................................... 35
   IQNinf ....................................................................................................................... 37
   IQNfrac ..................................................................................................................... 38
   IQtolQN ................................................................................................................... 39
   IQNtolQ ................................................................................................................... 40
Chapter 1. Introduction

1.1. Introduction

Texas Instruments TMS320C28x IQmath Library is a 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 considerably 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.
   C programs use IQmathLib.h
   a) 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 two builds of the library:
   a) IQmath.lib: This build of the library can be linked with code built for fixed-point.
   b) IQmath_f32.lib: This build of the library can be linked with code built with the --float_support=fpu32 switch. This can be useful for mixing IQmath with native floating-point code on devices with the C28x+FPU.
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 Code Composer Studio V4.
5) Example programs
2.2. How to Install the IQmath Library

The IQmath library is distributed in the form of a self-extracting ZIP file. By default, the install restores the IQmath library individual components in the directory structure shown below.

NOTE:

The directory structure of previous versions differed from what is shown below. For this release it has been reorganized to reduce duplication of files.

The <base> directory has been changed to correspond to ControlSUITE.

<base> install directory is C:\ti\controlSUITE\libs\math\IQmath
<base>\doc 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 Chapter 4.
<base>\gel Legacy GEL file for debug
  Refer to section 3.9.

The following example project run on CCS V4:

<base>\examples_ccsv4\C C example: Refer to ReadMe_SampleC.txt
<base>\examples_ccsv4\C\<device> Each device family has its own project folder
  Shared source code for the C example
<base>\examples_ccsv4\C\source
<base>\examples_ccsv4\Cpp C++ code example: Refer to ReadMe_SampleCpp.txt
<base>\examples_ccsv4\Cpp\source Shared source code for the C++ example
<base>\examples_ccsv4\Cpp\source Each device family has its own project folder
<base>\examples\cmd Linker command files used by the examples

The following example project runs on CCS V3.3:

<base>\examples\C C example: Refer to ReadMe_SampleC.txt
<base>\examples\C\projects CCS projects for the C example
<base>\examples\C\source Source code for the C example
<base>\examples\Cpp C++ code example: Refer to ReadMe_SampleCpp.txt
<base>\examples\Cpp\projects CCS projects for the C++ example
<base>\examples\Cpp\source Source code for the C++ example
<base>\examples\cmd Linker command files used by the examples
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.

```c
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 $\pi$ radians.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Range Min</th>
<th>Range Max</th>
<th>Resolution/Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>_iq30</td>
<td>-2</td>
<td>1.999 999 999</td>
<td>0.000 000 001</td>
</tr>
<tr>
<td>_iq29</td>
<td>-4</td>
<td>3.999 999 998</td>
<td>0.000 000 002</td>
</tr>
<tr>
<td>_iq28</td>
<td>-8</td>
<td>7.999 999 996</td>
<td>0.000 000 004</td>
</tr>
<tr>
<td>_iq27</td>
<td>-16</td>
<td>15.999 999 993</td>
<td>0.000 000 007</td>
</tr>
<tr>
<td>_iq26</td>
<td>-32</td>
<td>31.999 999 985</td>
<td>0.000 000 015</td>
</tr>
<tr>
<td>_iq25</td>
<td>-64</td>
<td>63.999 999 970</td>
<td>0.000 000 030</td>
</tr>
<tr>
<td>_iq24</td>
<td>-128</td>
<td>127.999 999 940</td>
<td>0.000 000 060</td>
</tr>
<tr>
<td>_iq23</td>
<td>-256</td>
<td>255.999 999 981</td>
<td>0.000 000 119</td>
</tr>
<tr>
<td>_iq22</td>
<td>-512</td>
<td>511.999 999 762</td>
<td>0.000 000 238</td>
</tr>
<tr>
<td>_iq21</td>
<td>-1024</td>
<td>1023.999 999 523</td>
<td>0.000 000 477</td>
</tr>
<tr>
<td>_iq20</td>
<td>-2048</td>
<td>2047.999 999 046</td>
<td>0.000 000 954</td>
</tr>
<tr>
<td>_iq19</td>
<td>-4096</td>
<td>4095.999 998 093</td>
<td>0.000 001 907</td>
</tr>
<tr>
<td>_iq18</td>
<td>-8192</td>
<td>8191.999 996 185</td>
<td>0.000 003 815</td>
</tr>
<tr>
<td>_iq17</td>
<td>-16384</td>
<td>16383.999 992 371</td>
<td>0.000 007 629</td>
</tr>
<tr>
<td>_iq16</td>
<td>-32768</td>
<td>32767.999 984 741</td>
<td>0.000 015 259</td>
</tr>
<tr>
<td>_iq15</td>
<td>-65536</td>
<td>65535.999 969 482</td>
<td>0.000 030 518</td>
</tr>
<tr>
<td>_iq14</td>
<td>-131072</td>
<td>131071.999 938 965</td>
<td>0.000 061 035</td>
</tr>
<tr>
<td>_iq13</td>
<td>-262144</td>
<td>262143.999 877 930</td>
<td>0.000 122 070</td>
</tr>
<tr>
<td>_iq12</td>
<td>-524288</td>
<td>524287.999 755 859</td>
<td>0.000 244 141</td>
</tr>
<tr>
<td>_iq11</td>
<td>-1048576</td>
<td>1048575.999 511 719</td>
<td>0.000 488 281</td>
</tr>
<tr>
<td>_iq10</td>
<td>-2097152</td>
<td>2097151.999 023 437</td>
<td>0.000 976 563</td>
</tr>
<tr>
<td>_iq9</td>
<td>-4194304</td>
<td>4194303.998 046 875</td>
<td>0.001 953 125</td>
</tr>
<tr>
<td>_iq8</td>
<td>-8388608</td>
<td>8388607.996 093 750</td>
<td>0.003 906 250</td>
</tr>
<tr>
<td>_iq7</td>
<td>-16777216</td>
<td>16777215.992 187 500</td>
<td>0.007 812 500</td>
</tr>
<tr>
<td>_iq6</td>
<td>-33554432</td>
<td>33554431.984 375 000</td>
<td>0.015 625 000</td>
</tr>
<tr>
<td>_iq5</td>
<td>-67108864</td>
<td>67108863.968 750 000</td>
<td>0.031 250 000</td>
</tr>
<tr>
<td>_iq4</td>
<td>-134217728</td>
<td>134217727.937 500 000</td>
<td>0.062 500 000</td>
</tr>
<tr>
<td>_iq3</td>
<td>-268435456</td>
<td>268435455.875 000 000</td>
<td>0.125 000 000</td>
</tr>
<tr>
<td>_iq2</td>
<td>-536870912</td>
<td>536870911.750 000 000</td>
<td>0.250 000 000</td>
</tr>
<tr>
<td>_iq1</td>
<td>-1073741824</td>
<td>1073741823.500 000 000</td>
<td>0.500 000 000</td>
</tr>
</tbody>
</table>
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. This will normally be the IQmath.lib library. If you have a C28x with floating-point device and wish to mix IQmath and native floating-point, then use the IQmath_fpu32.lib. This file contains all of the IQmath functions in object format. The library is constructed such that ONLY the functions that are used are included as part of the project. This minimizes code size.
- 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:

```c
#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:

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

- Link your code with the IQmath object code library. This will normally be the IQmath.lib library. If you have a C28x with floating-point device and wish to mix IQmath and native floating-point, then use the IQmath_fpu32.lib. Notice the same library is used by both C and C++ code. This file contains all of the IQmath functions in object format. The library is constructed such that ONLY the functions that are used are included as part of the project. This minimizes code size.
- 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 \texttt{iq} and \texttt{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:

<table>
<thead>
<tr>
<th>C Code</th>
<th>C++ Code</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>_iq, _iqN</td>
<td>iq, iqN</td>
<td>IQ Data Types</td>
</tr>
<tr>
<td>_IQ(A), _IQN(A)</td>
<td>IQ(A), IQN(A)</td>
<td>Convert float to IQ</td>
</tr>
<tr>
<td>_IQdiv(A,B)</td>
<td>A/B</td>
<td>Division</td>
</tr>
</tbody>
</table>

The following example code contains a call to the \texttt{IQ25sin} routines in IQmath Library using C++ code:

```c
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 \texttt{IQdiv}, \texttt{IQsin}, \texttt{IQcos}, \texttt{IQsqrt} and \texttt{IQatan2}. The \texttt{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 \texttt{IQexp}, \texttt{IQasin} and \texttt{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

<table>
<thead>
<tr>
<th>Device</th>
<th>Assembly Section Name (i.e., .sect)</th>
<th>Boot ROM Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>281x</td>
<td>IQmathTables</td>
<td>0x3FF000</td>
</tr>
<tr>
<td>280x</td>
<td>IQmathTables</td>
<td>0x3FF000</td>
</tr>
<tr>
<td>2801x</td>
<td>IQmathTables</td>
<td>0x3FF000</td>
</tr>
<tr>
<td>2833x/2823x 2834x</td>
<td>IQmathTables</td>
<td>0x3FE000</td>
</tr>
<tr>
<td></td>
<td>IQmathTablesRam (IQNexpTable.obj only)</td>
<td>0x3FEB50</td>
</tr>
<tr>
<td>2802x</td>
<td>IQmathTables</td>
<td>0x3FE000</td>
</tr>
<tr>
<td>2803x</td>
<td>IQmathTables</td>
<td>0x3FEB50</td>
</tr>
<tr>
<td></td>
<td>IQmathTablesRam (IQNexpTable.obj)</td>
<td>0x3FEBDC</td>
</tr>
<tr>
<td></td>
<td>IQmathTablesRam (IQNasinTable.obj)</td>
<td></td>
</tr>
</tbody>
</table>

If your device is not listed, refer to the boot ROM reference guide and the boot ROM source code for your device to get more information.

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. 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

<table>
<thead>
<tr>
<th>Assembly Section Name and obj file</th>
<th>Symbols / Initialized Tables Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQmathTables (IQmathTables.obj)</td>
<td>IQdivTable IQdivTableEnd IQdivRoundSatTable</td>
</tr>
<tr>
<td></td>
<td>IQsqrtTable IQsqrtTableEnd IQsqrtTableCoeff</td>
</tr>
<tr>
<td></td>
<td>IQexpTable IQexpTableMinMax IQexpTableCoeff</td>
</tr>
<tr>
<td>IQmathTablesRam (IQNexpTable.obj)</td>
<td>IQcosTable IQcosTableEnd</td>
</tr>
<tr>
<td></td>
<td>IQatan2Table IQatan2TableEnd IQatan2HalfPITable</td>
</tr>
<tr>
<td>IQmathTablesRam (IQNasinTable.obj)</td>
<td>IQasinTable</td>
</tr>
</tbody>
</table>
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.

```
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.

```
MEMORY
{
    PAGE 0:
        PRAML0 (RW) : origin = 0x008000, length = 0x001000
    PAGE 1:
        IQTABLES (R) : origin = ... (IQmathTablesRam)
        DRAML1 (RW) : origin = 0x009000, length = 0x001000
}
SECTIONS
{
    IQmathTables : load = IQTABLES, type = NOLOAD, PAGE = 1
    IQmathTables2 : load = IQTABLES2, type = NOLOAD, PAGE = 1
    IQmath : load = PRAML0, PAGE = 0
    {IQmath.lib<IQNexpTable.obj> (IQmathTablesRam)
    }
    IQmathTablesRam : load = DRAML1, PAGE = 1
    IQmath : load = PRAML0, PAGE = 0
}
```
Some devices, such as the 2802x and 2803x, 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 = 0x3FE50, 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:** 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. If your device is not listed in the table below, then refer to the boot ROM guide for more information.

<table>
<thead>
<tr>
<th>Device</th>
<th>IQmath Functions in Boot ROM</th>
<th>Values of N</th>
</tr>
</thead>
<tbody>
<tr>
<td>281x</td>
<td>none</td>
<td>N/A</td>
</tr>
<tr>
<td>280x</td>
<td>none</td>
<td>N/A</td>
</tr>
<tr>
<td>2801x</td>
<td>none</td>
<td>N/A</td>
</tr>
<tr>
<td>2833x, 2823x</td>
<td>none</td>
<td>N/A</td>
</tr>
<tr>
<td>2834x</td>
<td>none</td>
<td>N/A</td>
</tr>
<tr>
<td>2802x, 2803x</td>
<td>IQNatan2, IQNdiv, IQNmag, IQNisqrt, IQNsqrt, IQNcos</td>
<td>15, 20, 24, 29</td>
</tr>
</tbody>
</table>

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 for download from the TI website with the boot ROM reference guide.

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.

<table>
<thead>
<tr>
<th>Address</th>
<th>Symbol</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000821a</td>
<td>__IQ24atan2PU</td>
<td></td>
</tr>
<tr>
<td>003ff083</td>
<td>__IQ24div</td>
<td>&lt;- Boot ROM</td>
</tr>
<tr>
<td>000082a2</td>
<td>__IQ24exp</td>
<td></td>
</tr>
<tr>
<td>003fef35</td>
<td>__IQ24mag</td>
<td>&lt;- Boot ROM</td>
</tr>
<tr>
<td>000082df</td>
<td>__IQ24mpyI32int</td>
<td></td>
</tr>
<tr>
<td>003ff3c2</td>
<td>__IQ24sin</td>
<td>&lt;- Boot ROM</td>
</tr>
</tbody>
</table>
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”.

![Properties for 28035_I3sampleC](image)
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. 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 */
3.8. 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**

```c
#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**

```c
#define GLOBAL_Q 27 /* Set the Local Q value */
#include <IQmathLib.h>
```
3.9. Using the IQmath GEL file for Debugging

This section contains legacy information and is not required for CCS V3.3 or CCS V4.

This information is only useful when using an older version of Code Composer Studio 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 “GlobalQ” variable**
In one of the user source file, the following global variable must be defined:

```c
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.

3.10. 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.

3.11. 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. Libraries Available

This chapter lists the builds available for the IQmath source.

- **IQmath.lib**

  The IQmath.lib library has been built for use on standard C28x fixed-point devices. This library can be used with both C and C++ code.

- **IQmath_fpu32.lib**

  The IQmath_fpu32.lib library is useful for users who want to use both IQmath functions as well as native C28x floating-point code within their project. This library can be used with both C and C++ code.

  The IQmath_fpu32.lib library has been built with the following compiler flags:

  ```
  -v28 --float_support=fpu32
  ```

  IQmath_fpu32.lib still contains the same IQmath functions as IQmath.lib. It does not contain native floating-point functions. The float_support option is available in the C28x compiler version 5.0.0 and later.
Chapter 5. Function Summary

The routines included within the IQmath library are organized as follows:

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

5.1. Arguments and Conventions Used

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QN</td>
<td>16-bit fixed point Q number, where N=1:15</td>
</tr>
<tr>
<td>IQN</td>
<td>32-bit fixed point Q number, where N=1:31</td>
</tr>
<tr>
<td>int</td>
<td>16-bit number</td>
</tr>
<tr>
<td>long</td>
<td>32-bit number</td>
</tr>
<tr>
<td>_iq</td>
<td>_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.</td>
</tr>
<tr>
<td>_iqN</td>
<td>_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.</td>
</tr>
<tr>
<td>iq</td>
<td>C++ iq class for handling the _iq data type.</td>
</tr>
<tr>
<td>iqN</td>
<td>C++ iqN class for handling the _iqN data type.</td>
</tr>
<tr>
<td>A, B</td>
<td>Input operand to IQmath function or Macro</td>
</tr>
<tr>
<td>F</td>
<td>Floating point input: Ex: -1.232, +22.433, 0.4343, -0.32</td>
</tr>
<tr>
<td>S</td>
<td>Floating point string: &quot;+1.32&quot;, &quot;0.232&quot;, &quot;-2.343&quot; etc</td>
</tr>
<tr>
<td>P</td>
<td>Positive Saturation value</td>
</tr>
<tr>
<td>N</td>
<td>Negative Saturation value</td>
</tr>
</tbody>
</table>
## 5.2. IQmath Function Overview

### Format conversion Utilities

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>IQ format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_iq _IQ( float F)</td>
<td>Converts float to IQ value</td>
<td>Q=GLOBAL_Q Q=1:30</td>
</tr>
<tr>
<td>_iqN _IQN( float F)</td>
<td></td>
<td>Q=GLOBAL_Q Q=1:30</td>
</tr>
<tr>
<td>float _IQtoF( _iq A)</td>
<td></td>
<td>Q=GLOBAL_Q Q=1:30</td>
</tr>
<tr>
<td>float _IQNtoF( _iqN A)</td>
<td></td>
<td>Q=GLOBAL_Q Q=1:30</td>
</tr>
<tr>
<td>_iq _atoiQ( char *S)</td>
<td>Float ASCII string to IQ</td>
<td>Q=GLOBAL_Q Q=1:30</td>
</tr>
<tr>
<td>_iqN _atoiQN( char *S)</td>
<td></td>
<td>Q=GLOBAL_Q Q=1:30</td>
</tr>
<tr>
<td>int _IQtoa( char *S, const * format, long x)</td>
<td>IQ to ASCII string</td>
<td>Q=GLOBAL_Q Q=1:30</td>
</tr>
<tr>
<td>int _IQNtoa( char *S, const * format, long x)</td>
<td></td>
<td>Q=GLOBAL_Q Q=1:30</td>
</tr>
<tr>
<td>long _IQint( _iq A)</td>
<td>extract integer portion of IQ</td>
<td>Q=GLOBAL_Q Q=1:30</td>
</tr>
<tr>
<td>long _IQNint( _iqN A)</td>
<td></td>
<td>Q=GLOBAL_Q Q=1:30</td>
</tr>
<tr>
<td>_iq _IQfrac( _iq A)</td>
<td>extract fractional portion of IQ</td>
<td>Q=GLOBAL_Q Q=1:30</td>
</tr>
<tr>
<td>_iqN _IQNfrac( _iqN A)</td>
<td></td>
<td>Q=GLOBAL_Q Q=1:30</td>
</tr>
<tr>
<td>_iqN _IQtoIQN( _iq A)</td>
<td>Convert IQ number to IQN number (32-bit)</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iqN _IQNtoIQ( _iqN A)</td>
<td></td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>int _IQtoQN( _iq A)</td>
<td>Convert IQ number to QN number (16-bit)</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>int _IQdivQ( int A)</td>
<td>Convert QN (16-bit) number to IQ number</td>
<td>Q=GLOBAL_Q</td>
</tr>
</tbody>
</table>

### Shift to Multiply or Divide by Powers of 2

Added to the IQmathLib.h file as of V15c

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>IQ format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_iq _IQmpy2(_iq A)</td>
<td>Multiply by 2 by using a left shift by 1</td>
<td></td>
</tr>
<tr>
<td>_iq _IQmpy4(_iq A)</td>
<td>Multiply by 4 by using a left shift by 2</td>
<td></td>
</tr>
<tr>
<td>_iq _IQmpy8(_iq A)</td>
<td>Multiply by 8 by using a left shift by 3</td>
<td></td>
</tr>
<tr>
<td>_iq _IQmpy16(_iq A)</td>
<td>Multiply by 16 by using a left shift by 4</td>
<td></td>
</tr>
<tr>
<td>_iq _IQmpy32(_iq A)</td>
<td>Multiply by 32 by using a left shift by 5</td>
<td></td>
</tr>
<tr>
<td>_iq _IQmpy64(_iq A)</td>
<td>Multiply by 64 by using a left shift by 6</td>
<td></td>
</tr>
<tr>
<td>_iq _IQdiv2(_iq A)</td>
<td>Division by 2 by using a right shift by 1</td>
<td></td>
</tr>
<tr>
<td>_iq _IQdiv4(_iq A)</td>
<td>Division by 4 by using a right shift by 2</td>
<td></td>
</tr>
<tr>
<td>_iq _IQdiv8(_iq A)</td>
<td>Division by 8 by using a right shift by 3</td>
<td></td>
</tr>
<tr>
<td>_iq _IQdiv16(_iq A)</td>
<td>Division by 16 by using a right shift by 4</td>
<td></td>
</tr>
<tr>
<td>_iq _IQdiv32(_iq A)</td>
<td>Division by 32 by using a right shift by 5</td>
<td></td>
</tr>
<tr>
<td>_iq _IQdiv64(_iq A)</td>
<td>Division by 64 by using a right shift by 6</td>
<td></td>
</tr>
</tbody>
</table>
### Arithmetic Operations

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>IQ format</th>
</tr>
</thead>
</table>
| _iq  _IQmpy(_iq A, _iq B) | IQ Multiplication | Q=GLOBAL_Q
| _iqN _IQNmpy(_iqN A, _iqN B) | Q=1:30 |
| _iq  _IQmpy(_iq A, _iq B) | IQ Multiplication with rounding | Q=GLOBAL_Q
| _iqN _IQNmpy(_iqN A, _iqN B) | Q=1:30 |
| _iq  _IQrsmpy(_iq A, _iq B) | IQ multiplication with rounding & saturation | Q=GLOBAL_Q
| _iqN _IQNrsmpy(_iqN A, _iqN B) | Q=1:30 |
| _iq  _IQmpy32(_iq A, long B) | Multiply IQ with "long" integer | Q=GLOBAL_Q
| _iqN _IQNmpy32(_iqN A, long B) | Q=1:30 |
| long _IQmpy32int(_iq A, long B) | Multiply IQ with "long", return integer part | Q=GLOBAL_Q
| long _IQNmpy32int(_iqN A, long B) | Q=1:30 |
| long _IQmpy32frac(_iq A, long B) | Multiply IQ with "long", return fraction part | Q=GLOBAL_Q
| long _IQNmpy32frac(_iqN A, long B) | Q=1:30 |
| _iq  _IQmpyIQX(_iqN1 A, N1, _iqN2 B, N2) | Multiply two 2-different IQ number | Q=GLOBAL_Q
| _iqN _IQNmpyIQX(_iqN1 A, N1, _iqN2 B, N2) | Q=1:30 |
| _iq  _IQdiv(_iq A, _iq B) | Fixed point division | Q=GLOBAL_Q
| _iqN _IQNdiv(_iqN A, _iqN B) | Q=1:30 |

### Trigonometric Functions:

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>IQ format</th>
</tr>
</thead>
</table>
| _iq  _IQasin(_iq A) | High precision ASIN (output in radians) | Q=GLOBAL_Q
| _iqN _IQNasin(_iqN A) | Q=1:29 |
| _iq  _IQsin(_iq A) | High precision SIN (input in radians) | Q=GLOBAL_Q
| _iqN _IQNsin(_iqN A) | Q=1:29 |
| _iq  _IQsinPU(_iq A) | High precision SIN (input in per-unit) | Q=GLOBAL_Q
| _iqN _IQNsinPU(_iqN A) | Q=1:30 |
| _iq  _IQacos(_iq A) | High precision ACOS (output in radians) | Q=GLOBAL_Q
| _iqN _IQNacos(_iqN A) | Q=1:30 |
| _iq  _IQcos(_iq A) | High precision COS (input in radians) | Q=GLOBAL_Q
| _iqN _IQNcos(_iqN A) | Q=1:30 |
| _iq  _IQcosPU(_iq A) | High precision COS (input in per-unit) | Q=GLOBAL_Q
| _iqN _IQNcosPU(_iqN A) | Q=1:29 |
| _iq  _IQatan2(_iq A, _iq B) | 4-quadrant ATAN (output in radians) | Q=GLOBAL_Q
| _iqN _IQNatan2(_iqN A, _iqN B) | Q=1:30 |
| _iq  _IQatan2PU(_iq A, _iq B) | 4-quadrant ATAN (output in per-unit) | Q=GLOBAL_Q
| _iqN _IQNatan2PU(_iqN A, _iqN B) | Q=1:29 |
| _iq  _IQatan(_iq A, _iq B) | Arctangent | Q=GLOBAL_Q
| _iqN _IQNatan(_iqN A, _iqN B) | Q=1:29 |
Mathematical Functions:

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>IQ format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_iq _IQexp(_iq A)</td>
<td>High precision e raised to the A power</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iqN _IQNexp(_iqN A)</td>
<td>High precision e raised to the A power</td>
<td>Q=1:29</td>
</tr>
<tr>
<td>_iq _IQsqrt(_iq A)</td>
<td>High precision square root</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iqN _IQNsqrt(_iqN A)</td>
<td>High precision square root</td>
<td>Q=1:30</td>
</tr>
<tr>
<td>_iq _IQisqrt(_iq A)</td>
<td>High precision inverse square root</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iqN _IQNisqrt(_iqN A)</td>
<td>High precision inverse square root</td>
<td>Q=1:30</td>
</tr>
<tr>
<td>_iq _IQmag(_iq A, _iq B)</td>
<td>Magnitude Square: sqrt(A^2 + B^2)</td>
<td>Q=GLOBAL_Q</td>
</tr>
</tbody>
</table>

Miscellaneous

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Q format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_iq _IQsat(_iq A, long P, long N)</td>
<td>Saturate the IQ number</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iq _IQabs(_iq A)</td>
<td>Absolute value of IQ number</td>
<td>Q=GLOBAL_Q</td>
</tr>
</tbody>
</table>
### 5.3. C28x IQmath Library Benchmarks

<table>
<thead>
<tr>
<th>Function Name</th>
<th>IQ Format</th>
<th>Execution Cycles</th>
<th>Accuracy (in bits)</th>
<th>Program Memory (words)</th>
<th>Input format</th>
<th>Output format</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQNasin</td>
<td>1-29</td>
<td>154</td>
<td>82 words</td>
<td>IQN</td>
<td>IQN</td>
<td>Note A</td>
<td></td>
</tr>
<tr>
<td>IQNsin</td>
<td>1-29</td>
<td>46</td>
<td>30 bits</td>
<td>49 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNsincPU</td>
<td>1-30</td>
<td>40</td>
<td>30 bits</td>
<td>41 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNacscos</td>
<td>1-29</td>
<td>170</td>
<td>93 words</td>
<td>IQN</td>
<td>IQN</td>
<td>Note A</td>
<td></td>
</tr>
<tr>
<td>IQNcos</td>
<td>1-29</td>
<td>44</td>
<td>30 bits</td>
<td>47 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNcoscPU</td>
<td>1-30</td>
<td>38</td>
<td>29 bits</td>
<td>39 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNatan2</td>
<td>1-29</td>
<td>109</td>
<td>26 bits</td>
<td>123 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNatan2PU</td>
<td>1-29</td>
<td>117</td>
<td>27 bits</td>
<td>136 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQtan</td>
<td>1-29</td>
<td>109</td>
<td>25 bits</td>
<td>123 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNexp</td>
<td>1-29</td>
<td>190</td>
<td>61 words</td>
<td>IQN</td>
<td>IQN</td>
<td>Note A</td>
<td></td>
</tr>
<tr>
<td>IQNsqr</td>
<td>1-30</td>
<td>63</td>
<td>29 bits</td>
<td>66 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNsqr2</td>
<td>1-30</td>
<td>64</td>
<td>29 bits</td>
<td>69 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNmag</td>
<td>1-30</td>
<td>86</td>
<td>29 bits</td>
<td>96 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNmipy</td>
<td>1-30</td>
<td>~ 6</td>
<td>32 bits</td>
<td>NA</td>
<td>IQN*IQN</td>
<td>IQN</td>
<td>INTRINSIC</td>
</tr>
<tr>
<td>IQNmipy2</td>
<td>1-30</td>
<td>17</td>
<td>32 bits</td>
<td>13 words</td>
<td>IQN*IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNmipy3</td>
<td>1-30</td>
<td>21</td>
<td>32 bits</td>
<td>21 words</td>
<td>IQN*IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNmipy32</td>
<td>1-30</td>
<td>~ 4</td>
<td>32 bits</td>
<td>NA</td>
<td>IQN*long</td>
<td>IQN</td>
<td>C-MACRO</td>
</tr>
<tr>
<td>IQNmipy32int</td>
<td>1-30</td>
<td>22</td>
<td>32 bits</td>
<td>16 words</td>
<td>IQN*long</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>IQNmipy32frac</td>
<td>1-30</td>
<td>24</td>
<td>32 bits</td>
<td>20 words</td>
<td>IQN*long</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNmipyIQX</td>
<td>~ 7</td>
<td>32 bits</td>
<td>NA</td>
<td>IQN*IQN</td>
<td>IQN</td>
<td>INTRINSIC</td>
<td></td>
</tr>
<tr>
<td>IQNdiv</td>
<td>1-30</td>
<td>63</td>
<td>28 bits</td>
<td>71 words</td>
<td>IQN/IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQN</td>
<td>1-30</td>
<td>NA</td>
<td>N/A</td>
<td>NA</td>
<td>Float</td>
<td>IQN</td>
<td>C-MACRO</td>
</tr>
<tr>
<td>IQNtoF</td>
<td>1-30</td>
<td>22</td>
<td>N/A</td>
<td>20 words</td>
<td>IQN</td>
<td>Float</td>
<td></td>
</tr>
<tr>
<td>IQNtoa</td>
<td>1-30</td>
<td>N/A</td>
<td>N/A</td>
<td>210 words</td>
<td>IQN</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>atoiIQN</td>
<td>1-30</td>
<td>N/A</td>
<td>N/A</td>
<td>143 words</td>
<td>char *</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNint</td>
<td>1-30</td>
<td>14</td>
<td>32 bits</td>
<td>8 words</td>
<td>IQN</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>IQNfrac</td>
<td>1-30</td>
<td>17</td>
<td>32 bits</td>
<td>12 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQtonIQ</td>
<td>1-30</td>
<td>~ 4</td>
<td>N/A</td>
<td>N/A</td>
<td>GLOBAL_Q</td>
<td>IQN</td>
<td>C-MACRO</td>
</tr>
<tr>
<td>IQtonIQ2</td>
<td>1-30</td>
<td>~ 4</td>
<td>N/A</td>
<td>N/A</td>
<td>GLOBAL_Q</td>
<td>C-MACRO</td>
<td></td>
</tr>
<tr>
<td>IQtonQ</td>
<td>1-15</td>
<td>~ 4</td>
<td>N/A</td>
<td>N/A</td>
<td>GLOBAL_Q</td>
<td>QN</td>
<td>C-MACRO</td>
</tr>
<tr>
<td>QNtoIQ</td>
<td>1-15</td>
<td>~ 4</td>
<td>N/A</td>
<td>N/A</td>
<td>QN</td>
<td>GLOBAL_Q</td>
<td>C-MACRO</td>
</tr>
<tr>
<td><strong>Trigonometric Functions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mathematical Functions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Arithmetic Functions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Format Conversion Utilities</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Miscellaneous</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 6. Function Descriptions

6.1. Conversion Utilities

<table>
<thead>
<tr>
<th>IQN</th>
<th>Float to IQN data type</th>
</tr>
</thead>
</table>

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

- **Q format specific IQ Macro (IQ format = IQ1 to IQ29)**
  
  C  
  ```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

```c
// 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.

```c
#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) }
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]);
}
```
**atoIQ N**

**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 than 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:

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

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

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:

```c
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);
```
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;
**Description**
This function returns the fractional portion of IQ number.

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

```c
C    _iq _IQfrac(_iq A)
```

```cpp
iq IQfrac(const iq &A)
```

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

```c
C    _iqN _IQNfrac(_iqN A)
```

```cpp
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

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

```c
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.

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

```c
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

**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:

```c
#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**

<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>This Macro converts a 32-bit number in GLOBAL_Q format to 16-bit number in QN format.</th>
</tr>
</thead>
</table>
| **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:

\[
Y = X_0 \times C_0 + X_1 \times C_1 + X_2 \times C_2 \quad // \text{X_0, X_1, X_2 in Q15 format}
\]
\[
// C_0, C_1, C_2 in GLOBAL_Q format
\]

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);
```
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:

\[ Y = X_0 \cdot C_0 + X_1 \cdot C_1 + X_2 \cdot C_2 \] // \( X_0, X_1, X_2 \) in Q15 format

// \( C_0, C_1, C_2 \) 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:

```c
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);
```
### 6.2. Shift to Multiply or Divide by Powers of 2

<table>
<thead>
<tr>
<th>IQmpy2, 4, 8..64</th>
<th>Right Shift to Multiply by 2</th>
</tr>
</thead>
</table>

**Description**  
This #define macro in the IQmathLib.h can be used to perform a left shift in order to multiply 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);
```
This #define macro in the IQmathLib.h can be used to perform a right shift in order to multiply 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 \( Y = 2 \times X = X \ll 1 \)
\begin{verbatim}
  _iq Y;
  Y = _IQdiv2(X);
\end{verbatim}

Compute \( Y = 16 \times X = X \ll 4 \)
\begin{verbatim}
  _iq Y;
  Y = _IQdiv16(X);
\end{verbatim}
6.3. Arithmetic Operations

**IQNmpy**

**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
_C _iq _IQmpy(_iq A, _iq B)
```

```cpp
iq operator * (const iq &A, const iq &B)
```

```cpp
iq &iq :: operator *= (const iq &A)
```

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

```c
_C _iqN _IQNmpy(_iqN A, _iqN B)
```

```cpp
IQN operator * (const iqN &A, const iqN &B)
```

```cpp
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 \( Y = M \times X + B \) in GLOBAL_Q format with no rounding or saturation.

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

**Example 2:**
Compute \( Y = M \times X + B \) in IQ10 format with no rounding or saturation, assuming \( M, X, B \) are represented in IQ10 format.

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

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 \( Y = M \times X + B \) in GLOBAL_Q format with rounding but without saturation.

\[
_iq Y, M, X, B;
\]
\[
Y = _IQrmpy(M,X) + B;
\]

Example 2:
Compute \( Y = M \times X + B \) 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
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
<table>
<thead>
<tr>
<th>Description</th>
<th>This function multiplies an IQ number with a long integer and returns the integer part of the result.</th>
</tr>
</thead>
</table>
| 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;  

temp = IQmpyI32int(Output, 512);  
temp += 512;  
if( temp > 1023 ) temp = 1023;  
if( temp < 0 ) temp = 0;  
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.

---

©Texas Instruments Inc., 2002-2010
IQNmpyI32frac

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);
**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 _IQmpyIQX(_iqN1 A, int N1, _iqN2 B, int N2)  
C++  
 iqN IQmpyIQX(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 than IQ25.

**Case 1: GLOBAL_Q=IQ25**

```c
_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**

```c
_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);
```
IQDiv

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 _iq _IQdiv(_iq A, _iq B)

C++

iq operator / (const iq &A, const iq &B)
quotient &iq :: operator /= (const iq &A)

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

C

iq _iqN _IQNdiv(_iqN A, iq B)

C++

iqN operator / (const iqN &A, const iqN &B)
quotientN &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)

Accuracy

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

Usage
Example:
The following example obtains \( \sqrt{15} = 0.666 \) assuming that GLOBAL_Q is set to Q28 format in the IQmath header file.

```c
#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
IQDiv
Fixed Point vs. Floating Point Analysis

Fixed Point division vs. C Float division

Division function input: X=SIN(t) & Y=COS(t) in Q30 format

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

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

Error - IQ30div(X, Y) - DIV(X, Y)
Fixed Point vs. Floating Point Analysis

Fixed Point division vs. C Float division

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

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

3. IOHdiv(X, Y)-Floating Point Computation (Q16)

4. Error=IOHdiv(X, Y) - DIV(X, Y)
6.4. Trigonometric Functions

<table>
<thead>
<tr>
<th>IQNasin</th>
<th>Fixed point ASIN (radians)</th>
</tr>
</thead>
</table>

**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 \( \text{asin}(0.70710678) = \frac{0.25 \times \pi}{2} \). It assumes that GLOBAL_Q is set to Q29 format in the IQmath header file.

```c
#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);
}
```

©Texas Instruments Inc., 2002-2010
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.

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 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).

This function returns the sine of the input argument as fixed-point number in GLOBAL_Q format.

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

Accuracy

\[
20 \log_2 \left( \frac{\pi \times 2^{29}}{1} \right) = 30 \text{ bits}
\]

Example

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

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

void main(void ){
    // in1 = 0.25 \times \pi \times 2^{29} = 0\times1921FB54
    // out1 = \sin(0.25 \times \pi) \times 2^{29} = 0\times16A09E66
    // in2 = 0\times25 \times \pi \times 2^{29} = 0\times1921FB54
    // out2 = \sin(0.25 \times \pi) \times 2^{29} = 0\times16A09E66
    in1=_IQ(0.25*PI);
    out1=_IQsin(in1)
    in2=_IQ29(0.25*PI)
    out2=_IQ29sin(in2);
}
```
**Fixed Point vs. Floating Point Analysis**

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

![Graph: SIN(x) - Floating Point Output (Q29)](image)

![Graph: IQ29sin(x) - Fixed Point Output (Q29)](image)

![Graph: Error = IQ29sin(x) - SIN(x)](image)
IQNsinPU  

**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  _IQNsPU(_iqN A)
  C  iqN  IQNsPU(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**

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

**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.

```c
#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 = 0x80000000 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);
}
```

©Texas Instruments Inc., 2002-2010
IQNsinPU Function vs. C Float SIN: Input varies from 0 to $\pi$ 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
  
  ```c
  _iq  _IQacos(_iq  A)
  ```

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

- **Q format specific IQ function (IQ format = N = 1 to 29)**
  
  C
  
  ```c
  _iqN  _IQNacos(_iqN  A)
  ```

  C++
  
  ```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 \( \text{asin} \left( 0.70710678 \right) = \left( 0.25 \times \pi \right) \)
assuming that GLOBAL_Q is set to Q29 format in the IQmath header file.

```c
#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

```c
_iq IQcos(_iq A)
```

C++

```c
iq IQcos(const iq &A)
```

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

C

```c
_iqN IQNcos(_iqN A)
```

C++

```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**

\[ 20 \log_2 \left( \pi \times 2^{30} \right) - 20 \log_2 (2) = 30 \text{ bits} \]

**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.

```c
#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 \(\pi\)

\[\text{COS}(x)\] - Floating Point Output (Q29)

\[\text{IQ29cos}\] - Fixed Point Output (Q29)

Error = \[\text{IQ29cos}(x) - \text{COS}(x)\]
IQNcosPU

**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

```c
_iq _IQcosPU(_iq A)
```

C++

```cpp
_iq IQcosPU(const iq &A)
```

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

C

```c
_iqN _IQNcosPU(_iqN A)
```

C++

```cpp
_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**

\[
20 \log_2 \left(1 \times 2^{30}\right) - 20 \log_2 (2) = 29 \text{ bits}
\]

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

```c
#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 \(2\pi\) in per unit

COS(x)-Floating Point Output (Q30)

IQ30cosPU(x)- Fixed Point Output (Q30)

Error-IQ30cosPU(x) - COS(x)
## IQNatan2

**Description**  
This module computes 4-quadrant arctangent. Output of this module is in radians that varies from $-\pi$ to $\pi$.

**Declaration**  
Global IQ function (IQ format = GLOBAL_Q)  
\[ \text{C} \quad _{iq} \text{IQatan2}(_{iq} A, _{iq} B) \]  
\[ \text{C++} \quad _{iq} \text{IQatan2(const iq &A, const iq &B)} \]  
Q format specific IQ function (IQ format = IQ1 to IQ29)  
\[ \text{C} \quad _{iqN} \text{IQNatan2}(_{iqN} A, _{iqN} B) \]  
\[ \text{C++} \quad _{iqN} \text{IQNatan2(const iqN &A, const iqN &B)} \]

**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**  
\[ 20 \log_2 \left( \pi \times 2^{29} \right) - 20 \log_2 (32) = 26 \text{ bits} \]

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

```c
#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);
}
```

©Texas Instruments Inc., 2002-2010
Fixed Point ARCTAN Function vs. C Float ARCTAN

ATAN function input: Y=\sin(t) & X=\cos(t) in Q29 format

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

1Q29\text{atan2}(Y, X): Floating Point Computation(Q29)

Error=1Q29\text{atan2}(Y, X) - ATAN(Y, X)
IQNatan2PU

Fixed point 4-quadrant ATAN (in per unit)

Description
This module computes 4-quadrant arctangent. Output of this module is in per unit radians that varies from 0 (0 radians) to 1 ($\pi$ 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
  _iq _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 ($\pi$ 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 ($\pi$ radians).

Accuracy

\[
20 \log_2 \left( 1 \times 2^{29} \right) - 20 \log_2 \left( 6 \right) = 27 \text{ bits}
\]

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

```c
#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);
}
```

©Texas Instruments Inc., 2002-2010
Fixed Point ARCTAN Function vs. C Float ARCTAN

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

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

IQ29atan2PU(Y, X)-Floating Point Computation(Q29)

Error=IQ29atan2PU(Y, X) - ATAN(Y, X)
# Fixed point ATAN (in radians)

## Description
This module computes arctangent. Output of this module is in radians that vary from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.

## Declaration

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

C

```c
_iq _IQatan(_iq A)
```

C++

```c
iq IQatan(const iq &A)
```

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

C

```c
_iqN _IQNatan(_iqN A)
```

C++

```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 $[-\frac{\pi}{2}, +\frac{\pi}{2}]$.

**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 $[-\frac{\pi}{2}, +\frac{\pi}{2}]$.

## Accuracy

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

## Usage

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

```c
#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)
}
```
6.5. 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 \( e^{1.8} \approx 6.0496474 \), assuming that GLOBAL_Q is set to Q24 format in the IQmath header file.

```c
#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);
}
```
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
_C IQ _IQsqrt(_iq A)
```

C++

```c++
IQsqrt(const IQ &A)
```

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

```c
_C IQN _IQNsqrt(_iqN A)
```

C++

```c++
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**

\[ 20 \log_2 \left( \frac{1}{2} \right) - 20 \log_2 (6) = 29 \text{ bits} \]

**Usage**

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

```c
#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 vs. Floating Point Analysis

Fixed Point SQRT Function vs. C Float SQRT

SQRT(x)-Floating Point Computation (Q30)

IQ30sqrt(x)-Floating Point Computation (Q30)

Error=IQ30sqrt(x)-SQRT(x)
IQNisqrt

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

Usage
Calculate \( \sqrt{1.8} \approx 0.74535 \) assuming that GLOBAL_Q is set to Q30 format in the IQmath header file.

```c
#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 vs. Floating Point Analysis

**Fixed Point inverse SQRT Function vs. C Float inverse SQRT**

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

![Graph of 1/SQRT(x) for floating point computation (Q30)]

IQ30isqrt(x) - Fixed Point Computation (Q30)

![Graph of IQ30isqrt(x) for fixed point computation (Q30)]

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

![Graph showing the error between IQ30isqrt(x) and 1/SQRT(x)]
Fixed Point vs. Floating Point Analysis

Fixed Point inverse SQRT Function vs. C Float inverse SQRT

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

IQ16sqrt(x) Floating Point Computation (Q16)

IQ16sqrt(x) Floating Point Computation (Q16)

Error = IQ16sqrt(x) - 1/SQRT(x)
**Description**

This function calculates the magnitude of two orthogonal vectors as follows: \( \text{Mag} = \sqrt{A^2 + B^2} \). 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
_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
_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).

```c
#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);
}
```
6.6. Miscellaneous Utilities

<table>
<thead>
<tr>
<th>Description</th>
<th>This intrinsic calculates the absolute value of an IQ number.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td><strong>Global IQ function (IQ format = GLOBAL_Q)</strong>&lt;br&gt;C _iq _IQabs(_iq A)&lt;br&gt;C++ _iq _IQabs(const _iq &amp;A)&lt;br&gt;&lt;br&gt;<strong>Q format specific IQ function (IQ format = IQ1 to IQ30)</strong>&lt;br&gt;C _iqN _IQNabs(_iqN A)&lt;br&gt;C++ _iqN _IQNabs(const _iqN &amp;A)</td>
</tr>
<tr>
<td>Input</td>
<td><strong>Global IQ function (IQ format = GLOBAL_Q)</strong>&lt;br&gt;IQ number in GLOBAL_Q format&lt;br&gt;&lt;br&gt;<strong>Q format specific IQ function (IQ format = IQ1 to IQ30)</strong>&lt;br&gt;IQ number in IQN format</td>
</tr>
<tr>
<td>Output</td>
<td><strong>Global IQ function (IQ format = GLOBAL_Q)</strong>&lt;br&gt;Absolute value of input in GLOBAL_Q format&lt;br&gt;&lt;br&gt;<strong>Q format specific IQ function (IQ format = IQ1 to IQ30)</strong>&lt;br&gt;Absolute value of input in IQN format</td>
</tr>
<tr>
<td>Usage</td>
<td><strong>Example:</strong>&lt;br&gt;Calculate the absolute sum of three IQ numbers assuming GLOBAL_Q=IQ28 in the IQmath header file.</td>
</tr>
</tbody>
</table>

```c
#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

### 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
\[
\text{iq } \_\text{IQsat}(\text{iq } A, \text{long } P, \text{long } N)
\]

### 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 \( Y = M \times X + B \), 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:

```c
#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 7. Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.5c</td>
<td>June 6, 2010</td>
<td>Updates made to only the IQmathLib.h file. No changes were made to the library code itself.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added left shift and right shift #defines for multiplying and dividing by power of 2 in IQ_MATH:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>```c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define _IQmpy2(A) ((A)&lt;&lt;1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define _IQmpy4(A) ((A)&lt;&lt;2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define _IQmpy64(A) ((A)&lt;&lt;6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define _IQdiv2(A) ((A)&gt;&gt;1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define _IQmpy4(A) ((A)&gt;&gt;2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define _IQdiv64(A) ((A)&gt;&gt;6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Added corresponding Multiply/Divide in FLOAT_MATH:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>```c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define _IQmpy2(A) ((A)*2.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define _IQmpy4(A) ((A)*4.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#define _IQmpy64(A) ((A)*64.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FLOAT_MATH: Corrected the #defines for conversion from IQ to Q15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Removed the “L” from the constant so the compiler will not call the 64-bit floating point routine in the RTS library.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Removed the cast of the float to long before the multiply.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Examples: Fixed some issues with the projects being portable under CCS 4.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Fixed the 28335 project so it links in the fpu32 versions of the rts library and IQmath.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Fixed typos in this document.</td>
</tr>
<tr>
<td>V1.5b</td>
<td>January 12, 2010</td>
<td>Minor release to add examples and build information for Code Composer Studio V4. Changed the install directory to fit with ControlSuite.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Changes</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>V1.5a</td>
<td>June 1, 2009</td>
<td>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.</td>
</tr>
<tr>
<td>V1.5</td>
<td>July 8, 2008</td>
<td>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.</td>
</tr>
<tr>
<td>V1.4f</td>
<td>March 10, 2005</td>
<td>Fixed bug in IQexp.</td>
</tr>
<tr>
<td>V1.4e</td>
<td>June 17, 2004</td>
<td>Added IQexp, IQasin, IQacos.</td>
</tr>
<tr>
<td>V1.4d</td>
<td>March 30, 2003</td>
<td>Previous Web Release</td>
</tr>
</tbody>
</table>