IQmath Library
A Virtual Floating Point Engine

Module user’s Guide
C28x Foundation Software
## Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1.4.1</td>
<td>June 24, 2002</td>
<td>Original Draft Release</td>
</tr>
</tbody>
</table>
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 trademark mentioned herein are property of their respective companies

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

1. Introduction...........................................................................................................1
2. Installing IQmath library.......................................................................................2
3. Using IQmath Library...........................................................................................3
   3.1. IQmath Arguments and Data types.................................................................3
   3.2. IQmath Data Type: Range & Resolution..........................................................4
   3.3. Calling IQmath Function from C Program......................................................5
   3.4. IQmath Function Naming Convention ............................................................6
   3.5. Selecting GLOBAL_Q format ........................................................................6
   3.6. Using IQmath GEL file for De-bugging..........................................................7
4. IQmath Function Summary....................................................................................9
5. IQmath Function Description..............................................................................13
   5.1. Format Conversion utilities..........................................................................14
      5.1.1. IQN: Float to IQN data type .................................................................14
      5.1.2. IQNtoF: Float to IQN data type ..............................................................15
      5.1.3. atoIQN: String to IQN data type ..........................................................16
      5.1.4. IQNint: Integer part of IQN number .....................................................17
      5.1.5. IQNfrac: Fractional Part of IQN number .............................................18
      5.1.6. IQtoIQN: GLOBAL_Q number to IQN number ..................................19
      5.1.7. IQNtoIQ: IQN number to GLOBAL_Q number ....................................20
      5.1.8. IQtoQN: GLOBAL_Q number to 16-bit QN number ............................21
      5.1.9. QNtoIQ: 16-bit QN number to GLOBAL_Q number .............................22
   5.2. Arithmatic Functions......................................................................................23
      5.2.1. IQNmpy: IQ multiplication (IQN * IQN) .............................................23
      5.2.2. IQNrmpy: IQ multiplication with rounding (IQN*IQN) .......................24
      5.2.3. IQNrsmpy: IQ multiplication with rounding & saturation (IQN*IQN)....25
5.2.4. IQNmpyl32: Multiplication (IQN*LONG) .................................................................26
5.2.5. IQNmpyl32int: Integer portion of IQN*LONG .......................................................27
5.2.6. IQNmpyl32frac: Fractional portion of IQN*LONG .................................................28
5.2.7. IQNmpylQX: Multiplication (IQ=IQN1*IQN2) .........................................................29
5.2.8. IQNdvi: Fractional portion of IQ*LONG .................................................................30

5.3. Trignomatric Functions. ............................................................................................34
5.3.1. IQNsin: SIN(radians) .........................................................................................34
5.3.2. IQNsinPU: SIN(per unit) .....................................................................................37
5.3.3. IQNcos: COS(radians) ....................................................................................40
5.3.4. IQNcosPU: COS(per unit) .................................................................................43
5.3.6. IQNatan2: 4-quadrant ATAN (radians) ...............................................................46
5.3.7. IQNatan2: 4-quadrant ATAN (per unit) ...............................................................49
5.3.8. IQNatan: ATAN(radians) ..................................................................................52

5.4. Mathematical Functions. .........................................................................................54
5.3.6. IQNsqrt: Square-root ......................................................................................54
5.3.7. IQNisqrt: Inverse Square root ........................................................................57
5.4.3. IQNmag: Magnitude Square .............................................................................61

5.5. Miscellaneous. ........................................................................................................63
5.3.6. IQSat: Saturate IQ number ..............................................................................63
5.3.7. IQNabs: Absolute value of IQN number .........................................................64

Appendix A: IQmath C-Calling convention .................................................................65
### 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</th>
<th>Input format</th>
<th>Output format</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQNsinv</td>
<td>N=1 to 29</td>
<td>46</td>
<td>30 bits</td>
<td>49 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNsinvPU</td>
<td>N=1 to 30</td>
<td>40</td>
<td>30 bits</td>
<td>41 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNcos</td>
<td>N=1 to 29</td>
<td>44</td>
<td>30 bits</td>
<td>47 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNcosPU</td>
<td>N=1 to 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>N=1 to 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>N=1 to 29</td>
<td>117</td>
<td>27 bits</td>
<td>136 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQQatan</td>
<td>N=1 to 29</td>
<td>109</td>
<td>25 bits</td>
<td>123 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNsqrt</td>
<td>N=1 to 30</td>
<td>63</td>
<td>29 bits</td>
<td>66 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNisqrt</td>
<td>N=1 to 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>N=1 to 30</td>
<td>86</td>
<td>29 bits</td>
<td>96 words</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNmpy</td>
<td>N=1 to 30</td>
<td>~6 cycles</td>
<td>32 bits</td>
<td>NA</td>
<td>IQN*IQN</td>
<td>IQN</td>
<td>INTRINSIC</td>
</tr>
<tr>
<td>IQNrmmpy</td>
<td>N=1 to 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>IQNrsmpy</td>
<td>N=1 to 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>IQNmPypyl32</td>
<td>N=1 to 30</td>
<td>~4 cycles</td>
<td>32 bits</td>
<td>NA</td>
<td>IQN*long</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNmPypyl32int</td>
<td>N=1 to 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>IQNmPypyl32frac</td>
<td>N=1 to 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>IQNmPypylQX</td>
<td>N=1 to 30</td>
<td>~7 cycles</td>
<td>32 bits</td>
<td>NA</td>
<td>IQN1*IQN2</td>
<td>IQN</td>
<td>INTRINSIC</td>
</tr>
<tr>
<td>IQNdiv</td>
<td>N=1 to 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>N=1 to 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>N=1 to 30</td>
<td>22</td>
<td>N/A</td>
<td>20 words</td>
<td>IQN</td>
<td>Float</td>
<td></td>
</tr>
<tr>
<td>atolIQN</td>
<td>N=1 to 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>N=1 to 30</td>
<td>14</td>
<td>32 bits</td>
<td>8</td>
<td>IQN</td>
<td>long</td>
<td></td>
</tr>
<tr>
<td>IQNfrac</td>
<td>N=1 to 30</td>
<td>17</td>
<td>32 bits</td>
<td>12</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQN3t0QN</td>
<td>N=1 to 30</td>
<td>~4 cycles</td>
<td>N/A</td>
<td>N/A</td>
<td>GLOBAL_Q</td>
<td>IQN</td>
<td>C-MACRO</td>
</tr>
<tr>
<td>IQN3toIQ</td>
<td>N=1 to 30</td>
<td>~4 cycles</td>
<td>N/A</td>
<td>N/A</td>
<td>IQN</td>
<td>GLOBAL_Q</td>
<td>C-MACRO</td>
</tr>
<tr>
<td>IQN3toQN</td>
<td>N=1 to 15</td>
<td>~4 cycles</td>
<td>N/A</td>
<td>N/A</td>
<td>GLOBAL_Q</td>
<td>QN</td>
<td>C-MACRO</td>
</tr>
<tr>
<td>QN3toIQ</td>
<td>N=1 to 15</td>
<td>~4 cycles</td>
<td>N/A</td>
<td>N/A</td>
<td>QN</td>
<td>GLOBAL_Q</td>
<td>C-MACRO</td>
</tr>
</tbody>
</table>

### Miscellaneous

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Execution Cycles</th>
<th>Accuracy (in bits)</th>
<th>Program Memory</th>
<th>Input format</th>
<th>Output format</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQSsat</td>
<td>N=1 to 30</td>
<td>~7 cycles</td>
<td>N/A</td>
<td>N/A</td>
<td>IQN</td>
<td>IQN</td>
</tr>
<tr>
<td>IQNabs</td>
<td>N=1 to 30</td>
<td>~2 cycles</td>
<td>N/A</td>
<td>N/A</td>
<td>IQN</td>
<td>IQN</td>
</tr>
</tbody>
</table>

### Notes:

- Execution cycles & Program memory usage mentioned in the Table assumes IQ24 format.
- Execution cycles may vary by few cycles for some other IQ format.
- Program memory may vary by few words for some other IQ format.
- Execution Cycles mentioned in the table includes the CALL and RETURN (LCR + LRETR) and it assumes that the IQmath table is loaded in internal memory.
Chapter 1: Introduction

1.1. Introduction
Texas Instruments TMS320C28x IQmath Library is collection of highly optimized and high precision mathematical Function Library for C/C++ programmers to seamlessly port the floating-point algorithm into fixed point code on TMS320C28x devices. These routines are typically used in computationally intensive real-time applications where optimal execution speed & high accuracy is critical. By using these routines you can achieve execution speeds considerable faster than equivalent code written in standard ANSI C language. In addition, by providing ready-to-use high precision functions, TI IQmath library can shorten significantly your DSP application development time.
Chapter 2: Installing IQmath Library

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

1) IQmath header file: 
   *IQmath*.h
2) IQmath object library containing all function & look-up tables 
   *IQmath.lib*
3) Linker Command File 
   *IQmath.cmd*
4) IQmath GEL file for debugging 
   *IQmath.gel*
5) Example programs

2.2 How to Install IQmath Library
IQmath library is distributed in the form of an self-extracting ZIP file. The zip file automatically restores the IQmath library individual components in the directory structure shown below. Read README.TXT File for Specific Details of Release
Chapter 3: Using IQmath Library

3.1. IQmath Arguments and Data Types

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

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

```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 function like IQNsin, IQNcos, IQNatan2, IQNatan2PU, IQatan does not support Q30 format, due to the fact that these functions input or output need to vary between $-\pi$ to $\pi$ radians.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Range</th>
<th>Resolution/Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>iq30</td>
<td>-2</td>
<td>1.999 999 999</td>
</tr>
<tr>
<td>iq29</td>
<td>-4</td>
<td>3.999 999 999</td>
</tr>
<tr>
<td>iq28</td>
<td>-8</td>
<td>7.999 999 999</td>
</tr>
<tr>
<td>iq27</td>
<td>-16</td>
<td>15.999 999 993</td>
</tr>
<tr>
<td>iq26</td>
<td>-32</td>
<td>31.999 999 985</td>
</tr>
<tr>
<td>iq25</td>
<td>-64</td>
<td>63.999 999 970</td>
</tr>
<tr>
<td>iq24</td>
<td>-128</td>
<td>127.999 999 940</td>
</tr>
<tr>
<td>iq23</td>
<td>-256</td>
<td>255.999 999 981</td>
</tr>
<tr>
<td>iq22</td>
<td>-512</td>
<td>511.999 999 972</td>
</tr>
<tr>
<td>iq21</td>
<td>-1024</td>
<td>1023.999 999 523</td>
</tr>
<tr>
<td>iq20</td>
<td>-2048</td>
<td>2047.999 999 046</td>
</tr>
<tr>
<td>iq19</td>
<td>-4096</td>
<td>4095.999 999 093</td>
</tr>
<tr>
<td>iq18</td>
<td>-8192</td>
<td>8191.999 999 185</td>
</tr>
<tr>
<td>iq17</td>
<td>-16384</td>
<td>16383.999 999 371</td>
</tr>
<tr>
<td>iq16</td>
<td>-32768</td>
<td>32767.999 984 741</td>
</tr>
<tr>
<td>iq15</td>
<td>-65536</td>
<td>65535.999 969 482</td>
</tr>
<tr>
<td>iq14</td>
<td>-131072</td>
<td>131071.999 938 965</td>
</tr>
<tr>
<td>iq13</td>
<td>-262144</td>
<td>262143.999 877 930</td>
</tr>
<tr>
<td>iq12</td>
<td>-524288</td>
<td>524287.999 755 859</td>
</tr>
<tr>
<td>iq11</td>
<td>-1048576</td>
<td>1048575.999 511 719</td>
</tr>
<tr>
<td>iq10</td>
<td>-2097152</td>
<td>2097151.999 023 437</td>
</tr>
<tr>
<td>iq9</td>
<td>-4194304</td>
<td>4194303.999 046 875</td>
</tr>
<tr>
<td>iq8</td>
<td>-8388608</td>
<td>8388607.999 093 750</td>
</tr>
<tr>
<td>iq7</td>
<td>-16777216</td>
<td>16777215.992 187 500</td>
</tr>
<tr>
<td>iq6</td>
<td>-33554432</td>
<td>3355431.984 375 000</td>
</tr>
<tr>
<td>iq5</td>
<td>-67108864</td>
<td>67108863.968 750 000</td>
</tr>
<tr>
<td>iq4</td>
<td>-134217728</td>
<td>134217727.937 500 000</td>
</tr>
<tr>
<td>iq3</td>
<td>-268435456</td>
<td>268435455.875 000 000</td>
</tr>
<tr>
<td>iq2</td>
<td>-536870912</td>
<td>536870911.750 000 000</td>
</tr>
<tr>
<td>iq1</td>
<td>-1073741824</td>
<td>1 073741823.500 000 000</td>
</tr>
</tbody>
</table>
### 3.3. Calling a IQmath Function from C

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

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

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

<table>
<thead>
<tr>
<th>IQmath Linker Command File (F28x device)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORY</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>PAGE 0:</td>
</tr>
<tr>
<td>BOOTROM (RW) : origin = 0x3ff000, length = 0x000fc0</td>
</tr>
<tr>
<td>RAMH0 (RW) : origin = 0x3f8000, length = 0x002000</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>SECTIONS</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>IQmathTables : load = BOOTROM, type = NOLOAD, PAGE = 0</td>
</tr>
<tr>
<td>IQmath : load = RAMH0, PAGE = 0</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

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

```c
#include<IQmathLib.h> /* Header file for IQmath routine */
#define PI 3.14159

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

©Texas Instruments Inc., June 2002 5
3.4. IQmath Function Naming Convention
Each IQmath function provides, two types of function handles, viz.,

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

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

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

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

<table>
<thead>
<tr>
<th>IQmath Function Naming Convention</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL_Q Function</td>
</tr>
<tr>
<td>_IQXXX(func) , Where “XXX” is the Function Name</td>
</tr>
<tr>
<td>Q Specific Function</td>
</tr>
<tr>
<td>_IQNXXX(func) , Where “XXX” is the Function Name &amp; “N” is the Q format of input/output</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>IQmathLib.h : Selecting GLOBAL_Q format</th>
</tr>
</thead>
<tbody>
<tr>
<td>#ifndef GLOBAL_Q</td>
</tr>
<tr>
<td>#define GLOBAL_Q 24 /* Q1 to Q29 */</td>
</tr>
<tr>
<td>#endif</td>
</tr>
</tbody>
</table>
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>
```

3.6. Using IQmath GEL file for De-bugging

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

**Step 1: Define “GlobalQ” variable**
In one of the user source file, the following global variable must be defined:

```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`

![Watch Window](image)

©Texas Instruments Inc., June 2002
Step 4: Modifying IQmath variable

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

**IQ C Support**

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

**IQ C++ Support**

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

Invoking one of the above GEL operations will bring up a dialogue box window, which the user can enter the variable name and the floating-point value to set. The function will convert the float value to the appropriate IQ value.
Chapter 4: Function Summary

The routines included within the IQmath library are organized as follows

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

4.1 Arguments and Conventions Used

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

<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>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.</td>
</tr>
<tr>
<td>_iqN</td>
<td>Data type definition equating a long, a 32-bit value representing a IQN number, where N=1:30</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: “+1.32”, “0.232”, “-2.343” 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>
4.2. IQmath Functions

**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></td>
</tr>
<tr>
<td>float _IQtoF(_iq A)</td>
<td>IQ to Floating point</td>
<td>Q=GLOBAL_Q, Q=1:30</td>
</tr>
<tr>
<td>float _IQNtoF(_iqN A)</td>
<td></td>
<td></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></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></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></td>
</tr>
<tr>
<td>_iq _IQtoIQN(_iq A)</td>
<td>Convert IQ number to IQN number (32-bit)</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iq _IQNtoIQ(_iqN A)</td>
<td>Convert IQN (32-bit) number to IQ number</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>_iq _QNtoIQ(int A)</td>
<td>Convert QN (16-bit) number to IQ number</td>
<td>Q=GLOBAL_Q</td>
</tr>
</tbody>
</table>

**Arithmetic Operations:**

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>IQ format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_iq _IQmpy(_iq A, _iq B)</td>
<td>IQ Multiplication</td>
<td>Q=GLOBAL_Q, Q=1:30</td>
</tr>
<tr>
<td>_iqN _IQNmppy(_iqN A, _iqN B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_iq _IQrmpy(_iq A, _iq B)</td>
<td>IQ Multiplication with rounding</td>
<td>Q=GLOBAL_Q, Q=1:30</td>
</tr>
<tr>
<td>_iqN _IQNrmppy(_iqN A, _iqN B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_iq _IQrsmpy(_iq A, _iq B)</td>
<td>IQ multiplication with rounding &amp; saturation</td>
<td>Q=GLOBAL_Q, Q=1:30</td>
</tr>
<tr>
<td>_iq _IQNrsmpy(_iqN A, _iqN B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_iq _IQmpyI32(_iq A, long B)</td>
<td>Multiply IQ with &quot;long&quot; integer</td>
<td>Q=GLOBAL_Q, Q=1:30</td>
</tr>
<tr>
<td>_iqN _IQNmppyI32(_iqN A, long B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long _IQmpyI32int(_iq A, long B)</td>
<td>Multiply IQ with &quot;long&quot;, return integer part</td>
<td>Q=GLOBAL_Q, Q=1:30</td>
</tr>
<tr>
<td>long _IQNmppyI32int(_iqN A, long B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long _IQmpyI32frac(_iq A, long B)</td>
<td>Multiply IQ with &quot;long&quot;, return fraction part</td>
<td>Q=GLOBAL_Q, Q=1:30</td>
</tr>
<tr>
<td>long _IQNmppyI32frac(_iqN A, long B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_iq _IQmpyIQX(_iqN1 A, _iqN1, _iqN2 B, N2)</td>
<td>Multiply two 2-different IQ number</td>
<td>Q=GLOBAL_Q, Q=1:30</td>
</tr>
<tr>
<td>_iqN _IQNmppyIQX(_iqN1 A, _iqN1, _iqN2 B, N2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_iq _IQdiv(_iq A, _iq B)</td>
<td>Fixed point division</td>
<td>Q=GLOBAL_Q, Q=1:30</td>
</tr>
<tr>
<td>_iqN _IQNdiv(_iqN A, _iqN B)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Trignometric Functions:

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

Mathematical Functions:

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>IQ format</th>
</tr>
</thead>
<tbody>
<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></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></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>
<tr>
<td>_iqN  _IQNmag(_iqN A, _iqN B)</td>
<td></td>
<td>Q=1:30</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>
Chapter 5: Function Description
Description
This C-Macro converts a floating-point constant or variable to the equivalent IQ value.

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

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

Input
Floating point variable or constant

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

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

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

Example 1: Implementing equation in IQmath way
Floating point equation: \( Y = M \times 1.26 + 2.345 \)
IQmath equation (Type 1): \( Y = \_IQ\text{mpy}(M, \_IQ(1.26)) + \_IQ(2.345) \)
IQmath equation (Type 2): \( Y = \_IQ23\text{mpy}(M, \_IQ23(1.26)) + \_IQ23(2.345) \)

Example 2: Converting Floating point variable to IQ data type
float x=3.343;
_iq y1;
_iq23 y2
IQmath (Type 1): \( y1 = \_IQ(x) \)
IQmath (Type 2): \( y2 = \_IQ23(x) \)

Example 3: Initialing Global variables or Tables
IQmath (Type 1):
_iq Array[4] = {\_IQ(1.0), \_IQ(2.5) \_IQ(-0.2345), \_IQ(0.0) }

IQmath (Type 2):
_iq23 Array[4] = {\_IQ23(1.0), \_IQ23(2.5) \_IQ23(-0.2345), \_IQ23(0.0) }
Description
This function converts a IQ number to equivalent floating point value in IEEE 754 format.

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

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

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

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

Output
Floating point equivalent of fixed-point input.

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

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

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

for(i = 0; i < N, i++)
    DataF[i] = _IQtoF(DataIQ[i]);
Description
This function converts a string to IQ number.

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

Q format specific IQ function (IQ format = IQ1 to IQ30)
float _atolQN( 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];
_ilq X;

printf("Enter value X = ");
gets(buffer);
X = _atolIQ(buffer);        // IQ value (GLOBAL_Q)
```
**IQNint**

**Description**
This function returns the integer portion of IQ number.

**Declaration**

Global IQ function (IQ format = GLOBAL_Q)

```c
long _IQint( _iq A)
```

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

```c
long _IQNint( _iqN A)
```

**Input**

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

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

**Output**

Integer part of the IQ number

**Usage**

**Example 1:** Extracting Integer & fractional part of IQ number

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

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

Y0int = _IQint(Y0); // Y0int = 2
Y1int = _IQint(Y1); // Y1int = -2
Y0frac = _IQfrac(Y0); // Y0frac = 0.3456
Y1frac = _IQfrac(Y1); // Y1frac = -0.3456
```

**Example 2:** Building IQ number from integer & Fractional part

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

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

Y = _IQmpyI32(_IQ(1.0), Yint) + Yfrac;
```
This function returns the fractional portion of IQ number.

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

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

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.

Fractional part of the IQ number

Example 1: Extracting Integer & fractional part of IQ number

Following example extracts integer & fractional part of two IQ numbers

_iq Y0 = 2.3456;
_iq Y1 = -2.3456
long Y0int, Y1int;
_iq Y0frac, Y1frac;

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

Example 2: Building IQ number from integer & Fractional part

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

_iq Y;
long Yint;
_iq Yfrac;

Y = _IQmpyl32(_IQ(1.0), Yint) + Yfrac;
This Macro converts an IQ number in GLOBAL_Q format to the specified IQ format.

Declaration

_iqN _IQtoIQN( _iq A)

Input
IQ number in GLOBAL_Q format

Output
Equivalent value of input in IQN format

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

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

\[ Z = \sqrt{X^2 + Y^2} \]

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

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

\begin{verbatim}
_iq Z, Y, X; // GLOBAL_Q = 26
_iq23 temp;

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

Y = _IQ23toIQ(temp);
\end{verbatim}
Description  
This Macro converts an IQ number in IQN format to the GLOBAL_Q format.

Declaration  
_iq _IQNtoIQ( _iqN a)

Input  
IQ number in IQN format

Output  
Equivalent value of input in GLOBAL_Q format

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

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

\[ Z = \sqrt{X^2 + Y^2} \]

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

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

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

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

Y = _IQ23toIQ(temp);
This Macro converts a 32-bit number in GLOBAL_Q format to 16-bit number in QN format.

**Declaration**

```c
int _IQtoQN(_iq A)
```

**Input**

IQ number in GLOBAL_Q format

**Output**

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

**Usage**

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

**Example:**

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

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

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

```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);
```
This Macro converts a 16-bit number in QN format to 32-bit number in GLOBAL_Q format.

Declaration

_iq _QNtoIQ( int A)

Input

16-bit fixed point number in QN format

Output

Equivalent value of input in GLOBAL_Q format

Usage

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

Example:

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

\[
Y = X0 \cdot C0 + X1 \cdot C1 + X2 \cdot C2 \quad // X0, X1, X2 in Q15 format
\]

\[
// C0, C1, C2 in GLOBAL_Q format
\]

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

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

sum = _IQmpy(_Q15toIQ(X0), C0);
sum += _IQmpy(_Q15toIQ(X1), C1);
sum += _IQmpy(_Q15toIQ(X2), C2);
Y = _IQtoQ15(sum);
```
Description
This “C” compiler intrinsic multiplies two IQ number. It does not perform saturation and rounding. In most cases, the multiplication of two IQ variables will not exceed the range of the IQ variable. This operation takes the least amount of cycles and code size and should be used most often.

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

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

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

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

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

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

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

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

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

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

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

## Declaration

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

```
_iq _IQrmpy(_iq A, _iq B)
```

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

```
_iqN _IQNrmpy(_iqN A, _iqN B)
```

## Input Format

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

Input “A” & “B” are IQ number in GLOBAL_Q format

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

Input “A” & “B” are IQ number in IQN format

## Output Format

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

Result of multiplication in GLOBAL_Q format

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

Result of multiplication in IQN format.

## Usage

### Example 1:

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

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

### Example 2:

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

```
_iq10 Y, M, X, B;
Y = _IQ10rmpy(M,X) + B;
```
Description
This function multiplies two IQ number with rounding and saturation. In cases where the calculation may possibly exceed the range of the IQ variable, then this operation will round and then saturate the result to the maximum IQ value range before storing.

Declaration
Global IQ function (IQ format = GLOBAL_Q)

 iq _IQrsmpy(_iq A, _iq B)

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

 iqN _IQNrsmpy(_iqN A, _iqN B)

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

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

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

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

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

Example 1:
Following code computes “Y = M*X” in GLOBAL_Q format with rounding and saturation (Assuming GLOBAL_Q=IQ26):

 iq Y, M, X;

 M=_IQ(10.9); // M=10.9
 X=_IQ(4.5); // X=4.5
 Y = _IQrmpy(M,X); // Y= ~32.0, output is Saturated to MAX

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

 iq26 Y, M, X;

 M=_IQ26(-10.9); // M=-10.9
 X=_IQ26(4.5); // X=4.5
 Y = _IQ26rmpy(M,X); // Y= -32.0, output is Saturated to MIN
Description
This macro multiplies an IQ number with a long integer.

Declaration
Global IQ Macro (IQ format = GLOBAL_Q)

qi _IQmpyl32(_iq A, long B)

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

qiN _IQNmpyl32( _iqN A, long B)

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

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

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

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

Usage
Example 1:
Following code computes "Y = 5*X" in GLOBAL_Q format (assuming GLOBAL_Q = IQ26)

_iq Y, X;
X=IQ(5.1); // X=5.1 in GLOBAL_Q format
Y = IQmpyl32(X,5); // Y = 25.5 in GLOBAL_Q format

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

_iq26 Y, X;
long M;
M=5; // M=5
X=IQ26(5.1); // X=5.1 in IQ29 format
Y = _IQ26mpyl32(X,M); // Y=25.5 in IQ29 format
Description
This function multiplies an IQ number with a long integer and returns the integer part of the result.

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

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

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

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

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

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

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

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

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

Note: The integer operation performs the multiply and calculates the integer portion from the resulting 64-bit calculation. Hence it avoids any overflow conditions.
**Description**  
This function multiplies an IQ number with a long integer and returns the fractional part of the result.

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

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

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

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

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

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

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

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

Y1frac = IQmpyI32frac(X1, M1);       // Y1frac = 0.5 in GLOBAL_Q  
Y2frac = IQ26mpyI32frac(X2, M2);     // Y2frac = -0.9 in GLOBAL_Q  
```
Description
This "C" compiler intrinsic multiplies two IQ number that are represented in different IQ format.

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

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

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

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

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

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

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

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

Case 1: GLOBAL_Q=IQ25

\[
\begin{align*}
_iq30 & \ X0, X1, X2; \quad \text{// All values IQ30} \\
_iq28 & \ C0, C1, C2; \quad \text{// All values IQ28} \\
_iq & \ Y; \quad \text{// Result GLOBAL_Q = IQ25} \\
\end{align*}
\]

\[
\begin{align*}
Y & = \_IQmpyIQX(X0, 30, C0, 28) \\
Y & += \_IQmpyIQX(X1, 30, C1, 28) \\
Y & += \_IQmpyIQX(X2, 30, C2, 28) \\
\end{align*}
\]

Case 2: IQ Specific computation

\[
\begin{align*}
_iq30 & \ X0, X1, X2; \quad \text{// All values IQ30} \\
_iq28 & \ C0, C1, C2; \quad \text{// All values IQ28} \\
_iq25 & \ Y; \quad \text{// Result GLOBAL_Q = IQ25} \\
\end{align*}
\]

\[
\begin{align*}
Y & = \_IQ25mpyIQX(X0, 30, C0, 28) \\
Y & += \_IQ25mpyIQX(X1, 30, C1, 28) \\
Y & += \_IQ25mpyIQX(X2, 30, C2, 28) \\
\end{align*}
\]
Description
This module divides two IQN number and provide 32-bit quotient (IQN format) using Newton-Raphson technique

Availability
C-Callable Assembly (CcA)

Module Properties
Type: Target Independent, Application Independent
Target Devices: x28xx
C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>71 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>Multiple instances</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQdiv(_iq A, _iq B)

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

_iqN _IQNdiv(_iqN A, iq B)

Input Format

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

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

Output Format

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

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

Example

The following example obtains $\frac{15}{25} = 0.666$ assuming that GLOBAL_Q is set to Q28 format in IQmath header file.

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

_iq in1 out1;
_iq28 in2 out2;

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

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

Error=Q30div(X, Y) - DIV(X, Y)
IQNsin  

**Fixed point SIN (radians)**

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

**Availability**
C/C++ Callable Assembly

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

**Target Devices:** x28xx

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

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>49 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>Multiple instances</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

**Accuracy**

\[
\text{Accuracy} = 20\log_2 \left( \pi \times 2^{39} \right) - 20\log_2 (1) = 30 \text{ bits}
\]
C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQsin(_iq A)

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

_iqN _IQNsin(_iqN A)

Input Format

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

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

Output Format

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

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

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

```c
#include<IQmathLib.h> /* Header file for IQmath routine */
#define PI 3.14156
_iq in1, out1;
_iq28 in2, out2;

void main(void )
{
    in1=_IQ(0.25*PI); /* in1=0.25 \times \pi \times 2^{29} = 1921FB54h */
    out1=_IQsin(in1)
        /* out1=\sin(0.25 \times \pi) \times 2^{29} = 16A09E66h */
    in2=_IQ29(0.25*PI)
        /* in2=0.25 \times \pi \times 2^{29} = 1921FB54h */
    out2=_IQ29sin(in2);
        /* out2=\sin(0.25 \times \pi) \times 2^{29} = 16A09E66h */
}
```

©Texas Instruments Inc., June 2002
Fixed Point vs Floating Point Analysis

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

---

**SIN(x): Floating Point Output (Q29)**

---

**IQ29sin(x): Fixed Point Output (Q29)**

---

**Error = IQ29sin(x) - SIN(x)**
Description

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

Availability

C/C++ Callable Assembly

Module Properties

Type: Target Independent, Application Independent

Target Devices: x28xx

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

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>41 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>Multiple instances</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

Accuracy

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

**Declaration**

Global IQ function (IQ format = GLOBAL_Q)

\[ \text{iq } \_IQ\text{sinPU(_iq } A) \]

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

\[ \text{iqN } \_IQ\text{NsinPU(_iqN } A) \]

**Input Format**

Global IQ function (IQ format = GLOBAL_Q)

Input argument is in per-unit radians and represented as fixed-point number in GLOBAL_Q format.

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

Input argument is in per-unit radians and represented as fixed-point number in IQN format (N=1:30).

**Output Format**

Global IQ function (IQ format = GLOBAL_Q)

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

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

This function returns the sine of the input argument as fixed-point number in IQN format (N=1:30).

**Example**

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

```c
#include<IQmathLib.h> /* Header file for IQmath routine */
#define PI 3.14156
_iq in1, out1;
_iq30 in2, out2;

void main(void )
{
  in1=_IQ(0.25*PI/PI); /* in1 = \frac{0.25 \times \pi}{\pi} \times 2^{30} = 08000000h */
  out1=_IQsinPU(in1) /* out1= \sin\left( \frac{0.25 \times \pi}{\pi} \right) \times 2^{30} = 2D413CCCh */
  in2=_IQ30(0.25*PI/PI); /* in2 = \frac{0.25 \times \pi}{\pi} \times 2^{30} = 08000000h */
  out2=_IQ30sinPU(in2); /* out2= \sin\left( \frac{0.25 \times \pi}{\pi} \right) \times 2^{30} = 2D413CCCh */
}
```
IQNsinPU Function vs C Float SIN: Input varies from 0 to \( \frac{2\pi}{2} \) in per unit representation.
**Description**
This module computes the cosine value of the input (in radians) using table look-up and Taylor series expansion between the look up table entries.

**Availability**
C/C++ Callable Assembly

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

**Target Devices:** x28xx

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

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>47 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>Multiple instances</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

**Accuracy**

\[
= 20 \log_2 \left( \pi \times 2^{29} \right) - 20 \log_2 (2) \\
= 30 \text{ bits}
\]
C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)

\_iq _IQcos(_iq A)

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

\_iqN _IQNcos(_iqN A)

Input Format

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

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

Output Format

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

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

Example

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

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

_iq in1, out1;
_iq29 in2 out2;

void main(void )
{
    in1=_IQ(0.25*PI); /* in=0.25 \times \pi \times 2^{29} = 1921FB54h */
    out1=_IQcos(in1); /* out1=\cos(0.25 \times \pi) \times 2^{29} = 16A09E66h */
    in2=_IQ29(0.25*PI); /* in2=0.25 \times \pi \times 2^{29} = 1921FB54h */
    out2=_IQ29cos(in2); /* out2=\cos(0.25 \times \pi) \times 2^{29} = 16A09E66h */
}
```
Fixed Point COS Function vs C Float COS: Input varies from $-\pi$ to $\pi$
This module computes the cosine value of the input (in per-unit radians) using table look-up and Taylor series expansion between the look up table entries.

**Availability**

C/C++ Callable Assembly

**Module Properties**

**Type:** Target Independent, Application Independent

**Target Devices:** x28xx

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

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>39 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>Multiple instances</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

**Accuracy**

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

 iq _IQcosPU(_iq A)

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

 iqN _IQNcosPU(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

Input argument is in per-unit radians and represented as fixed-point number in GLOBAL_Q format.

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

Input argument is in per-unit radians and represented as fixed-point number in IQN format (N=1:30).

Output Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

This function returns the sine of the input argument as fixed-point number in IQN format (N=1:30).

Example

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

Sample Code

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

define PI 3.14156

 iq in1, out1;
 iq30 in2, out2

void main(void)
{
    in1=_IQ(0.25*PI/PI); /* in1 = \( 0.25 \times \pi \times 2^{30} = 08000000h \) */
    out1=_IQcosPU(in1) /* out1 = \( \cos(0.25 \times \pi) \times 2^{30} = 2D413CCCh \) */

    in2=_IQ30(0.25*PI/PI); /* in2 = \( 0.25 \times \pi \times 2^{30} = 08000000h \) */
    out2=_IQ30cosPU(in2) /* out2 = \( \cos(0.25 \times \pi) \times 2^{30} = 2D413CCCh \) */
}
```
Fixed Point COS Function vs C Float COS: Input varies from 0 to $\frac{2\pi}{2}$ in per unit representation
**IQNatan2**

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

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

**Availability**
C/C++ Callable Assembly

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

**Target Devices:** x28xx

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

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>123 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>Multiple instances</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

**Accuracy**

\[
\begin{align*}
\text{Accuracy} &= 20\log_2\left(\pi \times 2^{-y}\right) - 20\log_2(32) \\
&= 26 \text{ bits}
\end{align*}
\]
C/C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)
 iq IQatan2(iq A, iq B)

Q format specific IQ function (IQ format = IQ1 to IQ29)
 iqN IQNatan2(iqN A, iqN B), where the Q format “N” can vary from 1 to 29

Input Format

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

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

Output Format

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

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

Example

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

```c
#include<QmathLib.h> /* Header file for IQ math routine */
#define PI 3.14156

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

void main(void )
{
    xin1=_IQ(0.809) /* xin1=cos(\frac{\pi}{5})\times2^{29} = 19E3779Bh */
    yin1=_IQ(0.5877) /* yin1=sin(\frac{\pi}{5})\times2^{29} = 12CF 2304h */
    out1=_IQatan2(yin1,xin1); /* out1=\frac{\pi}{5}\times2^{29} = 141B2F 76h */

    xin2=_IQ29(0.809) /* xin1=cos(\frac{\pi}{5})\times2^{29} = 19E3779Bh */
    yin2=_IQ29(0.5877) /* yin1=sin(\frac{\pi}{5})\times2^{29} = 12CF 2304h */
    out2=_IQ29atan2(yin2,xin2); /* out2=\frac{\pi}{5}\times2^{29} = 141B2F 76h */
}
```
Fixed Point ARCTAN Function vs C Float ARCTAN

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

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

ATAN(Y, X) - Floating Point Computation (Q20)

Error=1029atan2(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 ($\frac{\pi}{2}$ radians).

**Availability**
C/C++ Callable Assembly

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

Target Devices: x28xx

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

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>136 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>Multiple instances</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

**Accuracy**

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

$$= 27 \text{ bits}$$
C/C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)

 iq _IQatan2PU(_iq A, _iq B)

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

 iqN _IQNatan2PU(_iqN A, _iqN B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

Input 'A' & 'B' are fixed-point number represented in GLOBAL_Q format.

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

Input 'A' & 'B' are fixed-point number in IQN format (N=1:29)

Output Format

Global IQ function (IQ format = GLOBAL_Q)

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

Example

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

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

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

void main(void )
{
    xin1=_IQ(0.809) /* xin1=cos(\frac{\pi}{4}) \times 2^{29} = 19E3779Bh */
    yin1=_IQ(0.5877) /* yin1=sin(\frac{\pi}{4}) \times 2^{29} = 12CF2304h */
    out1=_IQatan2PU(yin1,xin1); /* ou1 = \frac{\pi}{2\pi} \times 2^{29} = 03333333h */

    xin2=_IQ29(0.809) /* xin2=cos(\frac{\pi}{4}) \times 2^{29} = 19E3779Bh */
    yin2=_IQ29(0.5877) /* yin2=sin(\frac{\pi}{4}) \times 2^{29} = 12CF2304h */
    out2=_IQ29atan2PU(yin2,xin2) /* ou2 = \frac{\pi}{2\pi} \times 2^{29} = 03333333h */
}
```
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)
**IQNatan**

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

**Availability**
C/C++ Callable Assembly

**Module Properties**
- **Type:** Target Independent, Application Independent
- **Target Devices:** x28xx
- **C/CPP Interface Files:** IQmathLib.h, IQmathCPP.h & IQmath.lib

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>123 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>Multiple instances</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

**Accuracy**

\[
= 20\log_{2}\left(\frac{\pi}{2} \times 2^{29}\right) - 20\log_{2}(2)
\]

\[
= 25 \text{ bits}
\]
C/C-Callable ASM Interface

Declaration

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

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

Input Format

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

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

Output Format

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

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

Example

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> /* Header file for IQ math routine */

_iq in1, out1;
_iq29 in2, out2;

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

    in2=_IQ29(1.0);
    out2=_IQ29atan(in2)
}
```
Description: This module computes the square root of the input using table lookup and Newton-Raphson approximation.

Availability: C-Callable Assembly (CcA)

Module Properties: Type: Target Independent, Application Independent
Target Devices: x28xx
C/CPP Interface Files: IQmathLib.h, IQmathCPP.h & IQmath.lib

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>66 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>Multiple instances</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

Accuracy: 
\[
= 20 \log_2 \left(2^{31}\right) - 20 \log_2 \left(6\right)
\]
\[
= 29 \text{ bits}
\]
C/C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)

 iq _IQsqrt(_iq A)

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

 iqN _IQNsqrt(_iqN A)

Input Format

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

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

Output Format

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

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

Example

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

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

_iq in1, out1;
_iq30 in2, out2;

void main(void )
{
  in1=_IQ(1.8); /* in1= 1.8 × 2^{30} = 733333332h */
  out1=_IQsqrt(x); /* out1= √(1.8 × 2^{30}) = 55DD7151h */
  in2=_IQ30(1.8); /* in2= 1.8 × 2^{30} = 733333333h */
  out2=_IQ30sqrt(x); /* out2= √(1.8 × 2^{30}) = 55DD7151h */
}
```
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.

**Availability**  
C-Callable Assembly (CcA)

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

*Target Devices:* x28xx

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

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>69 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>Multiple instances</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

**Accuracy**  
\[
= 20\log_2\left(2^{31}\right) - 20\log_2(5) \\
= 29 \text{ bits}
\]
C/C-Callable ASM Interface

Declaration
Global IQ function (IQ format = GLOBAL_Q)

_qiQ _IQisqrt(_qi A)

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

_qiN _IQNisqrt(_qiN A)

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

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

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

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

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

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

_qi in1, out1;
_qi30 in2, out2;

void main(void )
{
    in1=_IQ(1.8); /* in1= 1.8 x 2^30 = 733333332h */
    out1=_IQisqrt(in1); /* out1= $\sqrt{1.8} x 2^{30} = 2FB3E99Eh$ */

    in2=_IQ30(1.8); /* in2= 1.8 x 2^{30} = 733333333h */
    out2=_IQ30isqrt(in2); /* out2= $\sqrt{1.8} x 2^{30} = 2FB3E99Eh$ */
}
```
Fixed Point vs Floating Point Analysis

Fixed Point inverse SQRT Function vs C Float inverse SQRT

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

IQ30isqrt(x) - Fixed Point Computation (Q30)

Error = IQ30isqrt(x) - 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)-Fixed Point Computation (Q16)

IQ16sqrt(x)-Fixed Point Computation (Q16)

Error=IQ16sqrt(x)*1SQRT(x)

©Texas Instruments Inc., June 2002
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.

**Availability**
C-Callable Assembly (CcA)

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

**Target Devices:** x28xx

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

---

<table>
<thead>
<tr>
<th>Item</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>96 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>Multiple instances</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>2 words</td>
<td>Stack grows by 2 words</td>
</tr>
</tbody>
</table>

**Accuracy**
29-bits (Same as SQRT function)
C/C-Callable ASM Interface

Declaration

Global IQ function (IQ format = GLOBAL_Q)

 iq IQmag(iq A, iq B)

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

 iqN IQNmag(iqN A, iqN B)

Input Format

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

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

Output Format

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

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

Example

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

Sample Code

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

 iq real1, imag1, mag1; // Complex number = real1 + j*imag1
 iq28 real2, imag2, mag2; // Complex number = real2 + j*imag2

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

    real2=_IQ28(7.0);
    imag2=_IQ28(7.0);
    mag2=_IQ28mag(real2, imag2); // mag2=8.0, saturated to MAX value (IQ28) !!!
}
### Description
This intrinsic calculates the absolute value of an IQ number:

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

  ```
  _iq _IQabs(_iq A)
  ```

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

  ```
  _iqN _IQNabs(_iqN A)
  ```

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

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

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

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

### Usage
**Example:**
Calculate the absolute sum of three IQ numbers (GLOBAL_Q=IQ28)

```
_iq xin1, xin2, xin3, xsum;
_iq20 yin1, yin2, yin3, ysum;

xsum = _IQabs(X0) + _IQabs(X1) + _IQabs(X2);
xsum = _IQ28abs(X0) + _IQ28abs(X1) + _IQ28abs(X2);
```
Description
This intrinsic saturates an IQ value to the given Positive and Negative limits. This operation is useful in areas where there is potential for overflow in a calculation.

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

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

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

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

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

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

```c
_iq Y, M, X, B; // GLOBAL_Q = 26 (+/- 32 range)
_iq20 temp; // IQ = 20 (+/- 2048 range)
temp = _IQ20mpy(_IQtoIQ20(M), _IQtoIQ20(X)) + _IQtoIQ20(B);
temp = _IQsat(temp, _IQtoIQ20(MAX_IQ_POS), _IQtoIQ20(MAX_IQ_NEG));
Y = _IQ20toIQ(temp);
```
Appendix A: IQmath C-Calling Convention

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