# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>Installing the IQmath Library</td>
<td>7</td>
</tr>
<tr>
<td>1.1</td>
<td>IQmath Content</td>
<td>8</td>
</tr>
<tr>
<td>1.2</td>
<td>How to Install the IQmath Library</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Using the IQmath Library</td>
<td>9</td>
</tr>
<tr>
<td>2.1</td>
<td>IQmath Arguments and Data Types</td>
<td>10</td>
</tr>
<tr>
<td>2.2</td>
<td>IQmath Data Type: Range and Resolution</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>Calling IQmath Functions from C</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>IQmath Function Naming Convention</td>
<td>12</td>
</tr>
<tr>
<td>2.5</td>
<td>Selecting GLOBAL_Q format</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Function Summary</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Arguments and Conventions Used</td>
<td>16</td>
</tr>
<tr>
<td>3.2</td>
<td>IQmath Functions</td>
<td>16</td>
</tr>
<tr>
<td>3.3</td>
<td>C64x+ IQmath Library Benchmarks</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>Libraries Available</td>
<td>21</td>
</tr>
<tr>
<td>4.1</td>
<td>Target Build (C64x+ Little Endian)</td>
<td>22</td>
</tr>
<tr>
<td>4.1.1</td>
<td>ROM Build (C643x Little Endian)</td>
<td>22</td>
</tr>
<tr>
<td>4.1.2</td>
<td>RAM Build (C64x+ Little Endian)</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>Target Build (C64x+ Big Endian)</td>
<td>22</td>
</tr>
<tr>
<td>4.3</td>
<td>Host Build</td>
<td>22</td>
</tr>
<tr>
<td>4.4</td>
<td>Usage</td>
<td>22</td>
</tr>
<tr>
<td>5</td>
<td>Function Description</td>
<td>23</td>
</tr>
</tbody>
</table>
List of Figures

1-1 IQmath Directory Structure ........................................................................................................ 8

List of Tables

2-1 Q Format Range and Resolution .................................................................................................. 11
3-1 Argument Descriptions .................................................................................................................. 16
3-2 Format Conversion Utilities ........................................................................................................ 16
3-3 Arithmetic Operations .................................................................................................................. 17
3-4 Trigonometric Functions .............................................................................................................. 17
3-5 Mathematical Functions ................................................................................................................ 18
3-6 Miscellaneous Functions ............................................................................................................... 18
3-7 IQmath Library Benchmarks ........................................................................................................ 18
5-1 List of Functions and Macros .......................................................................................................... 23
About This Manual

The Texas Instruments TMS320C64x+™ IQmath Library is a collection of highly-optimized and high-precision mathematical functions. The library is intended for C programmers to seamlessly port their floating-point algorithms into fixed-point code for execution on TMS320C64x+ devices. These routines are typically used in computationally intensive real-time applications, where optimal execution speed and high accuracy are 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, the TI IQmath library can significantly shorten your digital signal processor (DSP) application development time.

Acronyms

IQmath — High-accuracy mathematical functions (32-bit implementation)
QMATH — Fixed-point mathematical computation

Trademarks

TMS320C64x+ is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.
Installing the IQmath Library

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 IQmath Content</td>
<td>8</td>
</tr>
<tr>
<td>1.2 How to Install the IQmath Library</td>
<td>8</td>
</tr>
</tbody>
</table>
1.1 IQmath Content
The TI IQmath library consists of the following components:

- IQmath header file:
  IQmath.h: Includes definitions needed to interface with the IQMath library.

- IQmath inline header file:
  IQmath_inline.h: Includes source code for certain IQMath APIs to enable inlining. For more details on inlining, see the TMS320C6000 Optimizing Compiler User's Guide (SPRU187).

- IQmath object library containing all functions for little-endian devices:
  IQmath_c64x+.lib

- IQmath object library containing all functions and data tables for big-endian devices:
  IQmath_c64x+.e.lib

- Object library containing IQMath tables (little endian) to be linked in RAM:
  IQmath_RAM_c64x+.lib

- Object library containing IQMath tables (little endian) to be linked from device ROM (DM643x only):
  IQmath_ROM_c64x3x.lib

- Additionally, an x86-based host library is also provided for host (PC) testing of the IQmath code. This is functionally equivalent to the target library:
  IQmath_pc.lib

The code generation tool version used to create the IQmath libraries is v6.0.16.

1.2 How to Install the IQmath Library
The IQmath installation provides the directory structure shown in Figure 1-1.

![Figure 1-1. IQmath Directory Structure](image)

IQmath_v[X.Y.Z]  IQmath version X.Y.Z installation
   docs  IQmath Library User's Guide
   examples  Examples to demonstrate IQmath usage
      dotprod  Dot product example
      fft  FFT example
      fir  FIR example
   include  IQmath interface and inlining header files
   lib  Precompiled libraries
   README.txt
# Using the IQmath Library

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 IQmath Arguments and Data Types</td>
<td>10</td>
</tr>
<tr>
<td>2.2 IQmath Data Type: Range and Resolution</td>
<td>10</td>
</tr>
<tr>
<td>2.3 Calling IQmath Functions from C</td>
<td>11</td>
</tr>
<tr>
<td>2.4 IQmath Function Naming Convention</td>
<td>12</td>
</tr>
<tr>
<td>2.5 Selecting GLOBAL_Q format</td>
<td>13</td>
</tr>
</tbody>
</table>
2.1 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 Q0 to Q31.

Typedefs have been used to create aliases for IQ data types. This facilitates the user's ability to define the IQmath data type variable in the application program.

```c
typedef int _iq; /* Fixed point data type: GLOBAL_Q format */
typedef int _iq31; /* Fixed point data type: Q31 format */
typedef int _iq30; /* Fixed point data type: Q30 format */
typedef int _iq29; /* Fixed point data type: Q29 format */
typedef int _iq28; /* Fixed point data type: Q28 format */
typedef int _iq27; /* Fixed point data type: Q27 format */
typedef int _iq26; /* Fixed point data type: Q26 format */
typedef int _iq25; /* Fixed point data type: Q25 format */
typedef int _iq24; /* Fixed point data type: Q24 format */
typedef int _iq23; /* Fixed point data type: Q23 format */
typedef int _iq22; /* Fixed point data type: Q22 format */
typedef int _iq21; /* Fixed point data type: Q21 format */
typedef int _iq20; /* Fixed point data type: Q20 format */
typedef int _iq19; /* Fixed point data type: Q19 format */
typedef int _iq18; /* Fixed point data type: Q18 format */
typedef int _iq17; /* Fixed point data type: Q17 format */
typedef int _iq16; /* Fixed point data type: Q16 format */
typedef int _iq15; /* Fixed point data type: Q15 format */
typedef int _iq14; /* Fixed point data type: Q14 format */
typedef int _iq13; /* Fixed point data type: Q13 format */
typedef int _iq12; /* Fixed point data type: Q12 format */
typedef int _iq11; /* Fixed point data type: Q11 format */
typedef int _iq10; /* Fixed point data type: Q10 format */
typedef int _iq9; /* Fixed point data type: Q9 format */
typedef int _iq8; /* Fixed point data type: Q8 format */
typedef int _iq7; /* Fixed point data type: Q7 format */
typedef int _iq6; /* Fixed point data type: Q6 format */
typedef int _iq5; /* Fixed point data type: Q5 format */
typedef int _iq4; /* Fixed point data type: Q4 format */
typedef int _iq3; /* Fixed point data type: Q3 format */
typedef int _iq2; /* Fixed point data type: Q2 format */
typedef int _iq1; /* Fixed point data type: Q1 format */
typedef int _iq0; /* Fixed point data type: Q0 format */
```

2.2 IQmath Data Type: Range and Resolution

Table 2-1 summarizes the range and resolution of a 32-bit fixed-point number for different Q format representations. All IQmath functions support Q1 to Q29 format. Moreover, most of the functions also support Q0, Q30-Q31. For further details, see Section 3.3.

Trigonometric functions do not support Q formats above 29 because their input or output needs to vary between -π to π radians and this range cannot be represented in Q30 format. A few other functions do not support Q0 or Q30-Q31 because of performance overheads arising out of supporting 64-bit intermediate computations.
Table 2-1. Q Format Range and Resolution

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Min</th>
<th>Max</th>
<th>Resolution/Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>_iq31</td>
<td>-1</td>
<td>0.999 999 999</td>
<td>0.000 000 0005</td>
</tr>
<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>-67108664</td>
<td>67108663.968 750 000</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>
<tr>
<td>_iq0</td>
<td>-2147483648</td>
<td>2147483648.000 000 000</td>
<td>1.000 000 000</td>
</tr>
</tbody>
</table>

2.3 Calling IQmath Functions from C

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

- Include the IQmath.h include file.
- Link your code with the IQmath object code library. If executing the code on a C64x+ big-endian device, include IQmath_c64x+.h. If executing the code on a C64x+ little-endian device, include the library IQmath_c64x+.lib. If the little-endian device of choice is DM643x, include the library IQmath_ROM_c643x.lib to refer the IQmath tables from the device ROM. If the little-endian device is not DM643x, if executing the code on a device simulator, or if it is required to link the tables from RAM, include the library IQmath_RAM_c64x+.lib. Thus, for big-endian devices, only a single library needs to be included, whereas, for little-endian devices, two libraries need to be included.
- Use the linker command file to place the IQmath section in program memory. This step is optional and only required if a finer control is desired on the memory location where the IQmath code and tables are linked.
**Note:** IQmath functions are assembled in the `.text:IQmath` section and the look-up tables used to perform high-precision computation are placed in the `.data:IQmathTables` section.

---

**Example 2-1. IQmath Linker Command File (64x+ Device)**

```c
MEMORY
{
 ERAM_01 : o = 0x81000000, l = 0x00100000
}
SECTIONS
{
 .data:IQmathTables : > ERAM_01
 .text:IQmath : > ERAM_01
}
```

For example, the following code contains a call to the IQ29sin routines in the IQmath Library:

```c
#include<IQmath.h>    /* Header file for IQmath routine */
#define PI 3.14159F
_iq input, sin_out;    /* Definition of variables using IQmath datatype */
void main(void )
{
    input =_IQ29( 0.25*PI );   /* radians represented in Q29 format */
    sin_out =_IQ29sin (input );
}
```

---

### 2.4 IQmath Function Naming Convention

Each IQmath function supports two APIs from which the function can be called:

- Functions can be called using a global Q-point definition. The GLOBAL_Q macro is defined in the IQmath header file. When the IQmath functions are called using this API, they use the GLOBAL_Q macro definition as the input Q value for the arguments. The GLOBAL_Q macro is defined to a particular value and all the IQmath functions operate with that Q format. The valid values for global Q are from 0 to 31 (inclusive). The default value for GLOBAL_Q is 24. Natively, all the IQmath functions are written to accept the Q value information as part of the function arguments. When using the GLOBAL_Q method, the IQmath header file translates (using macros) the function calls to native format that includes the Q information as part of the argument.

**Examples:**

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

- Functions can be called using a specific Q format. In this method, the user explicitly provides the Q format information as part of the function arguments. Thus, if more than one Q value is used to represent the fixed point variables, this method can be used to specify the Q format information.

**Examples:**

- `_IQ29sin(A)` /* High Precision SIN: input/output are in Q29 */
- `_IQ25cos(A)` /* High Precision COS: input/output are in Q25 */
- `_IQ20rmpy(A, B)` /* fixed point multiply: input/output are in Q20 */

---

**Example 2-2. IQmath Function Naming Convention**

<table>
<thead>
<tr>
<th>GLOBAL_Q Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_IQxxx( )</code> , Where &quot;xxx&quot; is the Function Name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q Specific Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_IQNxxx( )</code> , Where &quot;xxx&quot; is the Function Name &amp; &quot;N&quot; is the Q format of input/output</td>
</tr>
</tbody>
</table>
2.5 Selecting GLOBAL_Q format

Numerical precision and dynamic range requirement vary considerably from one application to other. The 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 precisions and, finally, fix the numerical resolution. As explained in Section 2.2, higher 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 the IQmath.h header file to modify this value as required; user can choose from Q0 to Q31 as GLOBAL_Q format. Note that by modifying this value all the GLOBAL_Q functions use the modified Q format for input/output, unless this symbolic definition is overridden in the source code.

*Example 2-3. IQmath.h: Selecting GLOBAL_Q Format*

```c
#ifndef GLOBAL_Q
#define GLOBAL_Q 24 /* Q0 to Q31 */
#endif
```

**CASE II:**

A complete system consists of various modules. Some modules may require different precision than the rest of the system. In such situations, we need to override the GLOBAL_Q defined in the IQmath.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.

*Example 2-4. MODULE6.C: Selecting Local Q Format*

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

**CASE III:**

In certain cases, more than one Q format is required in the same source. In such an event, the GLOBAL_Q can be defined as the most frequently used Q format. The other Q formats can be handled by using explicit functions.

*Example 2-5. mixedQ.C: Use Multiple Q Formats*

```c
#define GLOBAL_Q 20 /* Set the Global Q value */
#include <IQmath.h>
int main()
{
   _iq a, b, c;
   _iq25 d, e;
   a = _IQ(2.5);
   b = _IQ(3.5)
   c = _IQmpy(a, b);
   d = _IQ25(1);
   e = _IQ25asin(d);
}
```
The routines included within the IQmath library are organized as follows:

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

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Arguments and Conventions Used</td>
<td>16</td>
</tr>
<tr>
<td>3.2 IQmath Functions</td>
<td>16</td>
</tr>
<tr>
<td>3.3 C64x+ IQmath Library Benchmarks</td>
<td>18</td>
</tr>
</tbody>
</table>
3.1 Arguments and Conventions Used

Table 3-1 shows conventions that have been followed when describing the arguments for each individual function:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQN</td>
<td>32-bit fixed point Q number, where N=0:31</td>
</tr>
<tr>
<td>Int, INT, I32_IQ</td>
<td>32-bit signed number</td>
</tr>
<tr>
<td>_iq</td>
<td>Data type definition equating to int, a 32-bit value, representing a GLOBAL_Q number. Usage of _iq instead of int is recommended to increase future portability across devices.</td>
</tr>
<tr>
<td>_iqN</td>
<td>Data type definition equating to int, a 32-bit value, representing a IQN number, where N=0:31</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>

3.2 IQmath Functions

Table 3-2 through Table 3-6 describe the IQmath functions.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>IQ Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>_iq_FtoIQ(float F)</td>
<td>Converts float to IQ value</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iqN_FtoIQN(float F)</td>
<td>Q=1:31</td>
<td></td>
</tr>
<tr>
<td>_iq_IQ(int A)</td>
<td>Convert an integer to IQ format</td>
<td>Q=0:31</td>
</tr>
<tr>
<td>float_IQtoF(_iq A)</td>
<td>IQ to floating point</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>float_IQNtoF(_iqN A)</td>
<td>Q=0:31</td>
<td></td>
</tr>
<tr>
<td>_iq_atoIQ(char *S)</td>
<td>Float ASCII string to IQ</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iqN_atoIQN(char *S)</td>
<td>Q=0:31</td>
<td></td>
</tr>
<tr>
<td>int_IQint(_iq A)</td>
<td>Extract integer portion of IQ</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>int_IQNint(_iqN A)</td>
<td>Q=0:29</td>
<td></td>
</tr>
<tr>
<td>_iq_IQfrac(_iq A)</td>
<td>Extract fractional portion of IQ</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iqN_IQfrac(_iqN A)</td>
<td>Q=1:31</td>
<td></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>Convert IQN (32-bit) number to IQ number</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iqY_IQXtoIQY(_iqX A, int x, int y)</td>
<td>Convert IQX number to IQY number</td>
<td>Q=0:31</td>
</tr>
</tbody>
</table>
### Table 3-3. Arithmetic Operations

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>IQ Format</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>iq</em> IQmpy(_iq A, _iq B)</td>
<td>IQ multiplication</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td><em>iqN</em> IQNmpy(_iqN A, _iqN B)</td>
<td>IQ multiplication with rounding</td>
<td>Q=1:31</td>
</tr>
<tr>
<td><em>iq</em> IQmpyR(_iq A, _iq B)</td>
<td>IQ multiplication with rounding and saturation</td>
<td>Q=1:31</td>
</tr>
<tr>
<td><em>iqN</em> IQNmpyR(_iqN A, _iqN B)</td>
<td>Multiply IQ with &quot;int&quot; integer</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td><em>iq</em> IQmpy32(_iq A, int B)</td>
<td>Multiply IQ with &quot;int&quot;, return integer part</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td><em>iqN</em> IQNmpy32(_iqN A, int B)</td>
<td>Multiply IQ with &quot;int&quot;, return fraction part</td>
<td>Q=0:30</td>
</tr>
<tr>
<td>int / IQmpy32int(_iq A, int B)</td>
<td>Multiply two 2-different IQ numbers</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>int / IQNmpy32int(_iqN A, int B)</td>
<td>Fixed-point division</td>
<td>Q=GLOBAL_Q</td>
</tr>
</tbody>
</table>

### Table 3-4. Trigonometric Functions

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>IQ Format</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>iq</em> IQsin(_iq A)</td>
<td>High-precision SIN (input in radians)</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td><em>iqN</em> IQsin(_iqN A)</td>
<td>High-precision SIN (input in units)(^{(1)})</td>
<td>Q=1:29</td>
</tr>
<tr>
<td><em>iq</em> IQsinPU(_iq A)</td>
<td>High-precision inverse SIN (output in radians)</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td><em>iqN</em> IQsinPU(_iqN A)</td>
<td>High-precision inverse SIN (output in radians)</td>
<td>Q=1:29</td>
</tr>
<tr>
<td><em>iq</em> IQcos(_iq A)</td>
<td>High-precision COS (input in radians)</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td><em>iqN</em> IQcos(_iqN A)</td>
<td>High-precision COS (input in units)</td>
<td>Q=1:29</td>
</tr>
<tr>
<td><em>iq</em> IQcosPU(_iq A)</td>
<td>High-precision inverse COS (output in radians)</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td><em>iqN</em> IQcosPU(_iqN A)</td>
<td>High-precision inverse COS (output in radians)</td>
<td>Q=1:29</td>
</tr>
<tr>
<td><em>iq</em> IQtan(_iq A)</td>
<td>4-quadrant ATAN (output in radians); produces atan(A / B) result</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td><em>iqN</em> IQNtan(_iqN A)</td>
<td>4-quadrant ATAN (output in radians); produces atan(A / B) result</td>
<td>Q=1:29</td>
</tr>
<tr>
<td><em>iqN</em> IQNtanPU(_iqN A)</td>
<td>4-quadrant ATAN (output in units); produces atan(A / B) result</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td><em>iqN</em> IQNatan(_iqN A)</td>
<td>4-quadrant ATAN (output in units)</td>
<td>Q=1:29</td>
</tr>
</tbody>
</table>

\(^{(1)}\) One unit is \((2\pi / 512)\) radians
Table 3-5. Mathematical Functions

<table>
<thead>
<tr>
<th>Function</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>High-precision inverse square root</td>
<td>Q=GLOBAL_Q</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>Magnitude square: sqrt(A^2 + B^2)</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iq _IQexp(_iq A)</td>
<td>Exponential</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iqN _IQNexp(_iqN A)</td>
<td>Exponential</td>
<td>Q=1:30</td>
</tr>
<tr>
<td>_iq _IQlog(_iq A)</td>
<td>Natural logarithm</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iqN _IQNlog(_iqN A)</td>
<td>Natural logarithm</td>
<td>Q=1:30</td>
</tr>
<tr>
<td>_iq _IQpow(_iq A)</td>
<td>Power: pow = A^B</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iqN _IQNpow(_iqN A)</td>
<td>Power: pow = A^B</td>
<td>Q=1:30</td>
</tr>
</tbody>
</table>

Table 3-6. Miscellaneous Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Q Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>I32 _IQsat( I32 A)</td>
<td>Saturates an integer to the range of governing Q format</td>
<td>Int32</td>
</tr>
<tr>
<td>_iq _IQabs(_iq A)</td>
<td>Absolute value of IQ number</td>
<td>Q=GLOBAL_Q</td>
</tr>
<tr>
<td>_iqN IQNabs(_iqN A)</td>
<td>Absolute value of IQ number</td>
<td>Q=0 :31</td>
</tr>
</tbody>
</table>

3.3 C64x+ IQmath Library Benchmarks

Table 3-7. IQmath Library Benchmarks

<table>
<thead>
<tr>
<th>Function Name</th>
<th>IQ Format</th>
<th>Execution Cycles (Stand alone)</th>
<th>Execution Cycles (Pipeline)</th>
<th>Program Memory (Words)(1)</th>
<th>Input Format</th>
<th>Output Format</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N=1 to 29</td>
<td>56</td>
<td>4</td>
<td>52</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=1 to 29</td>
<td>45</td>
<td>3.5</td>
<td>36</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=1 to 29</td>
<td>122</td>
<td>n.a.</td>
<td>118</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=1 to 29</td>
<td>54</td>
<td>4</td>
<td>46</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=1 to 29</td>
<td>48</td>
<td>3.5</td>
<td>39</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=1 to 29</td>
<td>122</td>
<td>n.a.</td>
<td>118</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=1 to 29</td>
<td>118</td>
<td>32.1</td>
<td>115</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=1 to 29</td>
<td>123</td>
<td>31.1</td>
<td>122</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=1 to 29</td>
<td>134</td>
<td>32.1</td>
<td>115</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=0 to 30</td>
<td>79</td>
<td>8.6</td>
<td>77</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=0 to 30</td>
<td>83</td>
<td>11.1</td>
<td>92</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=0 to 30</td>
<td>89</td>
<td>12.6</td>
<td>75</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=0 to 30</td>
<td>550</td>
<td>n.a.</td>
<td>14</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=0 to 30</td>
<td>20</td>
<td>1</td>
<td>7</td>
<td>IQN*IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=0 to 30</td>
<td>23</td>
<td>2</td>
<td>13</td>
<td>IQN*IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=0 to 30</td>
<td>27</td>
<td>3</td>
<td>20</td>
<td>IQN*IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>N=0 to 30</td>
<td>27</td>
<td>3.5</td>
<td>23</td>
<td>IQN*int</td>
<td>int</td>
<td></td>
</tr>
</tbody>
</table>

(1) 1 word is 4 bytes.
Table 3-7. IQmath Library Benchmarks (continued)

<table>
<thead>
<tr>
<th>Function Name</th>
<th>IQ Format</th>
<th>Execution Cycles (Stand alone)</th>
<th>Execution Cycles (Pipelined)</th>
<th>Program Memory (Words)</th>
<th>Input Format</th>
<th>Output Format</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQNmpyI32frac</td>
<td>N=0 to 30</td>
<td>21</td>
<td>3</td>
<td>16</td>
<td>IQN*int</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNmpyIQX</td>
<td>N=0 to 31</td>
<td>27</td>
<td>2</td>
<td>41</td>
<td>IQN1*IQN2</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNdiv</td>
<td>N=0 to 31</td>
<td>73</td>
<td>11.1</td>
<td>70</td>
<td>IQN/IQN</td>
<td>IQN</td>
<td></td>
</tr>
</tbody>
</table>

Format Conversion Utilities

<table>
<thead>
<tr>
<th>Function Name</th>
<th>IQ Format</th>
<th>Execution Cycles</th>
<th>Program Memory</th>
<th>Input Format</th>
<th>Output Format</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>FtoIQN</td>
<td>N=1 to 31</td>
<td>26</td>
<td>25</td>
<td>Float</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNtoF</td>
<td>N=0 to 31</td>
<td>23</td>
<td>17</td>
<td>IQN</td>
<td>Float</td>
<td></td>
</tr>
<tr>
<td>atofIQN</td>
<td>N=1 to 31</td>
<td>874</td>
<td>n.a.</td>
<td>char *</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNint</td>
<td>N=0 to 29</td>
<td>19</td>
<td>7</td>
<td>IQN</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>IQNfrac</td>
<td>N=1 to 31</td>
<td>19</td>
<td>7</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQtoIQN</td>
<td>N=0 to 31</td>
<td>≈4</td>
<td>n.a.</td>
<td>GLOBAL_Q</td>
<td>IQN</td>
<td>C-MACRO</td>
</tr>
<tr>
<td>IQNtoIQ</td>
<td>N=0 to 31</td>
<td>≈4</td>
<td>n.a.</td>
<td>IQN</td>
<td>GLOBAL_Q</td>
<td>C-MACRO</td>
</tr>
<tr>
<td>IQXtoIQY</td>
<td>N=0 to 31</td>
<td>≈4</td>
<td>n.a.</td>
<td>IQX</td>
<td>IQY</td>
<td>C-MACRO</td>
</tr>
</tbody>
</table>

Miscellaneous

<table>
<thead>
<tr>
<th>Function Name</th>
<th>IQ Format</th>
<th>Execution Cycles</th>
<th>Program Memory</th>
<th>Input Format</th>
<th>Output Format</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IQSat</td>
<td>N=0 to 31</td>
<td>24</td>
<td>10</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
<tr>
<td>IQNabs</td>
<td>N=0 to 31</td>
<td>18</td>
<td>2</td>
<td>IQN</td>
<td>IQN</td>
<td></td>
</tr>
</tbody>
</table>

Notes:
- Execution cycles are measured on a cycle accurate simulator and do not incorporate memory latencies.
- The pipelined loop cycles mentioned are measured with only a single function being called in a loop for 1024 iterations. This figure also includes cycles for loading the input and storing output data. A combination of functions may not yield scalable performance. If a large number of functions are used in a loop, the loop may not schedule.
- Program memory mentioned above is for the standalone version of code and may change with different compiler versions.
The chapter lists the various builds available for the IQmath source.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Target Build (C64x+ LittleEndian)</td>
<td>22</td>
</tr>
<tr>
<td>4.2 Target Build (C64x+ BigEndian)</td>
<td>22</td>
</tr>
<tr>
<td>4.3 Host Build</td>
<td>22</td>
</tr>
<tr>
<td>4.4 Usage</td>
<td>22</td>
</tr>
</tbody>
</table>
4.1 Target Build (C64x+ Little Endian)

This is the base version of the IQmath library and runs on all C64x+ cores. It consists of object code for all the IQmath functions. However, for operation this requires the RAM or ROM data table library, depending on the device under consideration. The ROM library can be used only with the C643x series devices, whereas, the RAM library can be used with all C64x+ core devices, including the C643x series. The RAM library needs to be used when using the 643x device simulator.

Library name: IQmath_c64x+.lib.

4.1.1 ROM Build (C643x Little Endian)

This version runs only on the C643x family of processors. This library includes the references to the IQmath data tables present on the ROM of C643x devices, thus reducing the memory footprint. To be able to use this version, the revision of the ROM should be later than 0x27B2A120. To determine your ROM version, see section 9 of the Using the TMS320DM643x Bootloader application report (SPRAAG0). The RAM library needs to be used when using the C643x device simulator.

Library name: IQmath_ROM_c643x.lib.

4.1.2 RAM Build (C64x+ Little Endian)

This is the generic add-on for the IQmath library. It consists of data tables required by the IQmath library and can run on all C64x+ little-endian devices.

Library name: IQmath_RAM_c64x+.lib.

4.2 Target Build (C64x+ Big Endian)

This is the base version of the IQmath library and runs on all C64x+ big-endian cores. It consists of object code and data tables for all the IQmath functions.

Library name: IQmath_c64x+.e.lib.

4.3 Host Build

The host (PC/VC++) build is useful in the case where PC testing of the algorithm is required. This feature is greatly helpful in the development phase of the algorithms, as the algorithm performance can be tuned on the PC using the IQmath PC library. Once the algorithm performance is verified, the algorithm can be easily ported to the C64x+ DSP simply by replacing the IQmath library. Thus, the host build helps bridge the gap between the PC development environment and the C64x+ device.

Library name: IQmath_pc.lib.

4.4 Usage

The usage of all the above described IQmath libraries is identical. The libraries provide the same APIs and the only care the user needs to take is to include the library that is appropriate to the project.
This chapter provides detailed descriptions of the C64x+ IQmath library functions and macros.

### Table 5-1. List of Functions and Macros

<table>
<thead>
<tr>
<th>Function/Macro</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>_FtoIQN</td>
<td>24</td>
</tr>
<tr>
<td>_IQN</td>
<td>25</td>
</tr>
<tr>
<td>IQNtoF</td>
<td>26</td>
</tr>
<tr>
<td>atoIQN</td>
<td>27</td>
</tr>
<tr>
<td>IQNint</td>
<td>28</td>
</tr>
<tr>
<td>IQNfrac</td>
<td>29</td>
</tr>
<tr>
<td>IQNtoIQ</td>
<td>30</td>
</tr>
<tr>
<td>IQNmpy</td>
<td>31</td>
</tr>
<tr>
<td>IQNmpympy</td>
<td>32</td>
</tr>
<tr>
<td>IQNmpympympy</td>
<td>33</td>
</tr>
<tr>
<td>IQNmpympy32</td>
<td>34</td>
</tr>
<tr>
<td>IQNmpympy32int</td>
<td>35</td>
</tr>
<tr>
<td>IQNmpympy32frac</td>
<td>36</td>
</tr>
<tr>
<td>IQNmpympyOX</td>
<td>37</td>
</tr>
<tr>
<td>IQNdiv</td>
<td>38</td>
</tr>
<tr>
<td>IQNsin</td>
<td>39</td>
</tr>
<tr>
<td>IQNsinPU</td>
<td>40</td>
</tr>
<tr>
<td>IQNcos</td>
<td>41</td>
</tr>
<tr>
<td>IQNcosPU</td>
<td>42</td>
</tr>
<tr>
<td>IQNatan2</td>
<td>43</td>
</tr>
<tr>
<td>IQNatan2PU</td>
<td>44</td>
</tr>
<tr>
<td>IQNatan</td>
<td>45</td>
</tr>
<tr>
<td>IQNsqrt</td>
<td>46</td>
</tr>
<tr>
<td>IQNisqrt</td>
<td>47</td>
</tr>
<tr>
<td>IQNmag</td>
<td>48</td>
</tr>
<tr>
<td>IQNexp</td>
<td>49</td>
</tr>
<tr>
<td>IQNlog</td>
<td>50</td>
</tr>
<tr>
<td>IQNpow</td>
<td>51</td>
</tr>
<tr>
<td>IQNabs</td>
<td>52</td>
</tr>
<tr>
<td>IQSat</td>
<td>53</td>
</tr>
</tbody>
</table>

The list includes various functions and macros for fixed-point and integer operations, as well as math functions like addition, multiplication, division, and square root. Each function is categorized under the C64x+ IQmath library and is detailed in the chapter for users to understand their implementation and usage.
_FtoIQN — Float-to-IQN Data Type

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

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

```c
_iq _FtoIQ(float F)
```

Q format specific IQ function (IQ format = IQ1 to IQ31)

```c
_iQN _FtoIQN(float F)
```

**Input Format**
Floating-point variable or constant

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

Fixed-point equivalent of floating-point input in GLOBAL_Q format

Q-format-specific IQ function (IQ format = IQ1 to IQ31)

Fixed-point equivalent of floating-point input in IQN format

**Saturation**
If the input is out of limits for a given Q value, the function returns 0 for positive input and 0x80000000 for negative input.

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

**Examples**

**Example 1:** Implementing equation in IQmath:
Floating point equation: \( Y = M \times 1.26 + 2.345 \)
IQmath equation (Type 1): \( Y = _Iqmpy(M, _FtoIQ(1.26)) + _FtoIQ(2.345) \)
IQmath equation (Type 2): \( Y = _IQ23mpy(M, _FtoIQ23(1.26)) + _FtoIQ23(2.345) \)

**Example 2:** Converting Floating point variable to IQ data type:
```c
float x=3.343;
_iq y1;
_iq23 y2
```
IQmath (Type 1): \( y1 = _FtoIQ(x) \)
IQmath (Type 2): \( y2 = _FtoIQ23(x) \)
**IQN**

**Integer-to-IQN Data Type**

<table>
<thead>
<tr>
<th>Description</th>
<th>This macro converts an integer (short, char) to an equivalent IQ value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>Global IQ Macro (IQ format = GLOBAL_Q)</td>
</tr>
<tr>
<td></td>
<td>_iq _IQ(int A);</td>
</tr>
<tr>
<td></td>
<td>Q format specific IQ macro (IQ format = IQ0 to IQ31)</td>
</tr>
<tr>
<td></td>
<td>_iqN _IQN (int A);</td>
</tr>
<tr>
<td>Input Format</td>
<td>Global IQ function (IQ format = GLOBAL_Q) or Q-format-specific IQ function (IQ format = IQ0 to IQ31)</td>
</tr>
<tr>
<td></td>
<td>Integer</td>
</tr>
<tr>
<td>Output Format</td>
<td>IQ equivalent of the input.</td>
</tr>
<tr>
<td>Usage</td>
<td>This operation is typically used to convert an integer constant or variable to its equivalent IQ value.</td>
</tr>
<tr>
<td>Note</td>
<td>This macro can also be used to convert floating-point numbers to IQ format. However, it is strongly advised not to do so because of the large performance overhead involved. This includes calling two floating-point RTS functions. The _FtoIQ() function should be used for this purpose.</td>
</tr>
<tr>
<td>Example</td>
<td>Converting array of IQ numbers to the equivalent floating-point values:</td>
</tr>
<tr>
<td></td>
<td>int A = 3; short B = 5;</td>
</tr>
<tr>
<td></td>
<td>_iq a, b, c;</td>
</tr>
<tr>
<td></td>
<td>a = _IQ(A); // Uses GLOBAL_Q</td>
</tr>
<tr>
<td></td>
<td>b = _IQ10(B);</td>
</tr>
</tbody>
</table>
## IQNtoF — *Float-to-IQN Data Type*

<table>
<thead>
<tr>
<th>Description</th>
<th>This function converts a IQ number to equivalent floating-point value in IEEE 754 format.</th>
</tr>
</thead>
</table>
| Declaration | **Global IQ function (IQ format = GLOBAL_Q)**

```c
float _IQtoF(_iq A)
```

**Q-format-specific IQ function (IQ format = IQ0 to IQ31)**

```c
float _IQNtoF(_iqN A)
```

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

Fixed-point IQ number in GLOBAL_Q format.

**Q-format-specific IQ function (IQ format = IQ0 to IQ31)**

Fixed-point IQ number in IQN format. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Output Format</td>
<td>Floating-point equivalent of fixed-point input.</td>
</tr>
<tr>
<td>Usage</td>
<td>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.</td>
</tr>
<tr>
<td>Example</td>
<td>Converting array of IQ numbers to the equivalent floating-point values:</td>
</tr>
</tbody>
</table>
```c
_iq DataIQ[N];
_iq24 DataIQ24[N];
float DataF1[N], DataF2[N];

for(i = 0; i < N, i++) {
  DataF1[i] = _IQtoF(DataIQ[i]);
  DataF2[i] = _IQ24toF(DataIQ24[i]);
}
```
<table>
<thead>
<tr>
<th><strong>atoIQN</strong></th>
<th><strong>String to IQN</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This function converts a string to IQ number.</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td><strong>Global IQ function (IQ format = GLOBAL_Q)</strong>&lt;br&gt;float _atoIQ(char *S)&lt;br&gt;<strong>Q-format-specific IQ function (IQ format = IQ1 to IQ31)</strong>&lt;br&gt;float _atoIQN(char *S)</td>
</tr>
<tr>
<td><strong>Input Format</strong></td>
<td>This function recognizes (in order) an optional sign, a string of digits optionally containing a radix character.&lt;br&gt;Valid Input strings: &quot;12.23456&quot;, &quot;-12.23456&quot;, &quot;0.2345&quot;, &quot;0&quot;, &quot;0&quot;, &quot;127&quot;, &quot;-89&quot;</td>
</tr>
<tr>
<td><strong>Output Format</strong></td>
<td>The first unrecognized character ends the string and returns zero. If the input string converts to a number greater then the max/min values for the given Q value, then the returned value is limited to the min/max values for the given Q format.</td>
</tr>
<tr>
<td><strong>Global IQ function (IQ format = GLOBAL_Q)</strong></td>
<td>Fixed-point equivalent of input string in GLOBAL_Q format.</td>
</tr>
<tr>
<td><strong>Q-format-specific IQ function (IQ format = IQ1 to IQ31)</strong></td>
<td>Fixed-point equivalent of input string in IQN format.</td>
</tr>
<tr>
<td><strong>Saturation</strong></td>
<td>If the input string converts to a number out of range for the given Q value, then the returned value is limited to the min/max values.</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>This is useful for programs that need to process user input or ASCII strings.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>The following code prompts the user to enter the value X:&lt;br&gt;char buffer[N];&lt;br&gt;_iq X;&lt;br&gt;_iq20 Y;&lt;br&gt;printf(&quot;Enter value X = &quot;);&lt;br&gt;gets(buffer);&lt;br&gt;X = _atoIQ(buffer);            // IQ value (GLOBAL_Q)&lt;br&gt;Y = _atoIQ20(buffer);</td>
</tr>
</tbody>
</table>
**IQNint — Integer Part of IQN Number**

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

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

```c
int _IQint(_iq A)
```

**Q-format-specific IQ function (IQ format = IQ0 to IQ29)**

```c
int _IQNint(_iqN A)
```

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

**Q-format-specific IQ function (IQ format = IQ0 to IQ29)**
Fixed-point IQ number in IQN format.

**Output Format**  
Integer part of the IQ number.

**Example**  
Extracting integer and fractional part of IQ number.

The following example extracts the integer and fractional part of two IQ number:

```c
int Y0int, Y1int;
_iq Y0frac, Y1frac;

_iq Y0 = _FtoIQ(2.3456);
_iq Y1 = _FtoIQ(-2.3456);

Y0int = _IQint(Y0);   // Y0int = 2
Y1int = _IQint(Y1);   // Y1int = -2
Y0frac = _IQfrac(Y0); // Y0frac = 0.3456
Y1frac = _IQfrac(Y1); // Y1frac = -0.3456
```
## IQNfrac — Fractional Part of IQN Number

<table>
<thead>
<tr>
<th>Description</th>
<th>This function returns the fractional portion of IQ number.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td><strong>Global IQ function (IQ format = GLOBAL_Q)</strong></td>
</tr>
<tr>
<td></td>
<td><code>_iq _IQfrac(_iq A)</code></td>
</tr>
<tr>
<td></td>
<td><strong>Q-format-specific IQ function (IQ format = IQ1 to IQ31)</strong></td>
</tr>
<tr>
<td></td>
<td><code>iqN _IQNfrac(_iqN A)</code></td>
</tr>
<tr>
<td>Input Format</td>
<td><strong>Global IQ function (IQ format = GLOBAL_Q)</strong></td>
</tr>
<tr>
<td></td>
<td>Fixed-point IQ number in GLOBAL_Q format.</td>
</tr>
<tr>
<td></td>
<td><strong>Q-format-specific IQ function (IQ format = IQ1 to IQ31)</strong></td>
</tr>
<tr>
<td></td>
<td>Fixed-point IQ number in IQN format.</td>
</tr>
<tr>
<td>Output Format</td>
<td>Fractional part of the IQ number.</td>
</tr>
<tr>
<td>Example</td>
<td>Extracting integer and fractional part of IQ number.</td>
</tr>
<tr>
<td></td>
<td>The following example extracts integer and fractional part of two IQ numbers:</td>
</tr>
</tbody>
</table>
|           | ```
|           | int Y0int, Y1int;
|           | _iq Y0frac, Y1frac;
|           | _iq Y0 = _FtoIQ(2.3456);
|           | _iq Y1 = _FtoIQ(-2.3456);
|           | Y0int = _IQint(Y0);   // Y0int = 2
|           | Y1int = _IQint(Y1);   // Y1int = -2
|           | Y0frac = _IQfrac(Y0); // Y0frac = 0.3456
|           | Y1frac = _IQfrac(Y1); // Y1frac = -0.3456
|           | ```
### IQtoIQN — GLOBAL_Q Number to IQN

<table>
<thead>
<tr>
<th>Description</th>
<th>This macro converts an IQ number in GLOBAL_Q format to the specified IQ format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>_iqN _IQtoIQN(_iq A)</td>
</tr>
<tr>
<td>Input Format</td>
<td>IQ number in GLOBAL_Q format.</td>
</tr>
<tr>
<td>Output Format</td>
<td>Equivalent value of input in IQN format.</td>
</tr>
<tr>
<td>Note</td>
<td>This functionality is implemented by a simple shift. Rounding is not performed.</td>
</tr>
<tr>
<td>Usage</td>
<td>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.</td>
</tr>
<tr>
<td>Example</td>
<td>The following example calculates the magnitude of complex number (X+jY) in Q26 format:</td>
</tr>
<tr>
<td></td>
<td>Z = sqrt(X^2 + Y^2)</td>
</tr>
<tr>
<td></td>
<td>The values Z, X, and Y are given as GLOBAL_Q = 26, but the equation itself may generate an overflow. To guard against this, the intermediate calculations are performed in a lower Q format (say Q = 23), in which we know that overflow will not occur. The result is converted back to GLOBAL_Q at the end, as shown below:</td>
</tr>
<tr>
<td></td>
<td>_iq Z, Y, X; // GLOBAL_Q = 26</td>
</tr>
<tr>
<td></td>
<td>_iq23 temp;</td>
</tr>
<tr>
<td></td>
<td>temp = _IQ23sqrt(_IQ23mpy(_IQtoIQ23(X), _IQtoIQ23(X)) + _IQ23mpy(_IQtoIQ23(Y), _IQtoIQ23(Y)));</td>
</tr>
<tr>
<td></td>
<td>Y = _IQ23toIQ(temp);</td>
</tr>
</tbody>
</table>
IQNtoIQ — IQN Number to GLOBAL_Q

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

Declaration
_iq _IQNtoIQ(_iqN A)

Input Format
IQ number in IQN format.

Output Format
Equivalent value of input in GLOBAL_Q format.

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

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

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

The values \(Z\), \(X\), and \(Y\) are given as GLOBAL_Q = 26, but the equation itself may generate an overflow. To guard against this, the intermediate calculations are performed using \(Q = 23\) and the value converted back at the end as shown below:

\[
_iq Z, Y, X; \quad // \text{GLOBAL}_Q = 26
\]

\[
_iq23 temp;
\]

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

\[
Y = _IQ23toIQ(temp);
\]
**Description**

This function multiplies two IQ number. It does not perform saturation and rounding. In most cases, the multiplication of two IQ variables do not exceed the range of the IQ variable. Amongst all IQ multiply flavors available, this operation takes the least amount of cycles and code size.

**Declaration**

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

```c
_iq _IQmpy(_iq A, _iq B)
```

**Q-format-specific IQ (IQ format = IQ1 to IQ31)**

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

**Input Format**

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

The input arguments A and B are IQ numbers in GLOBAL_Q format.

**Q-format-specific IQ (IQ format = IQ1 to IQ31)**

The input arguments A and B are IQ numbers in IQN format.

**Output Format**

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

Result of multiplication in GLOBAL_Q format.

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

Result of multiplication in IQN format.

**Examples**

**Example 1:** The following code computes \( Y = MX + B \) in GLOBAL_Q format with no rounding or saturation:

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

**Example 2:** The following code computes \( Y = MX + B \) in IQ10 format with no rounding or saturation, assuming \( M, X, \) and \( B \) are represented in IQ10 format:

```c
_iq10 Y, M, X, B;
Y = _IQ10mpy(M, X) + B;
```
IQNrmyp — 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 LSB 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 IQ31)

_iqN _IQNrmyp(_iqN A, _iqN B)

Input Format
Global IQ function (IQ format = GLOBAL_Q)
The input arguments A and B are IQ numbers in GLOBAL_Q format.

Q-format-specific IQ function (IQ format = IQ1 to IQ31)
The input arguments A and B are IQ numbers 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 IQ31)
Result of multiplication in IQN format.

Examples
Example 1: The following code computes $Y = M \times X + B$ in GLOBAL_Q format with rounding, but no saturation:

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

Example 2: The following code computes $Y = M \times X + B$ in IQ10 format with rounding, but no saturation:

_iq10 Y, M, X, B;
Y = _IQ10rmpy(M, X) + B;
**IQNrsmpy**  
*IQ Multiplication With Rounding and 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, this operation rounds and then saturates the result to the maximum IQ value range before storing.

**Declaration**  
Global IQ function (IQ format = GLOBAL_Q)  
\[
\text{iq} \_\text{IQrsmpy}(_\text{iq} \ A, _\text{iq} \ B)
\]

Q-format-specific IQ function (IQ format = IQ1 to IQ31)  
\[
\text{iqN} \_\text{IQNrsmpy}(_\text{iqN} \ A, _\text{iqN} \ B)
\]

**Input Format**  
Global IQ function (IQ format = GLOBAL_Q)  
The input arguments A and B are IQ numbers in GLOBAL_Q format.

Q-format-specific IQ function (IQ format = IQ1 to IQ31)  
The input arguments A and B are IQ numbers 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 IQ31)  
Result of multiplication in IQN format.

**Saturation**  
The function saturates between maximum and minimum limits of Q format.

**Usage**  
Let us assume that we use IQ26 as GLOBAL_Q format. This means that the range of the numbers is approximately [-32.0, 32.0] (see Section 2.2). If two IQ variables are multiplied together, then the maximum range of the result is [-1024, 1024]. The IQNrsmpy operation ensures that the result is saturated to ±32 in cases where the result exceeds this range.

**Examples**

**Example 1:** The following code computes \( Y = M \times X \) in GLOBAL_Q format with rounding and saturation (assuming GLOBAL_Q=26):  
\[
\text{iq} \ Y, \ M, \ X;
\]
\[
\text{M} = \text{FtoIQ}(10.9); \quad // \text{M}=10.9
\]
\[
\text{X} = \text{FtoIQ}(4.5); \quad // \text{X}=4.5
\]
\[
\text{Y} = \text{IQrsmpy}(\text{M}, \text{X}); \quad // \text{Y}=-32.0, \text{output is saturated to MAX}
\]

**Example 2:** The following code computes \( Y = M \times X \) in IQ26 format with rounding and saturation:  
\[
\text{iq26} \ Y, \ M, \ X;
\]
\[
\text{M} = \text{IQ26}(-10.9); \quad // \text{M}=-10.9
\]
\[
\text{X} = \text{IQ26}(4.5); \quad // \text{X}=4.5
\]
\[
\text{Y} = \text{IQ26rsmpy}(\text{M}, \text{X}); \quad // \text{Y}=-32.0, \text{output is saturated to MIN}
\]
IQNmpyI32 — Multiplication (IQN\*INT)

Description
This macro multiplies an IQ number with an integer.

Declaration
Global IQ macro (IQ format = GLOBAL_Q)

```c
_iq _IQmpyI32(_iq A, int B)
```

Q-format-specific IQ macro (IQ format = IQ0 to IQ31)

```c
_iqN _ IQNmpyI32(_iqN A, int B)
```

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

Q-format-specific IQ macro (IQ format = IQ1 to IQ31)
Operand A is an IQ number in IQN format and B is the integer.

Output Format
A 64-bit result is returned (while maintaining the same Q format). The user is free to use either the 64-bit result or truncate the result to 32 bits.

Examples
Example 1: The following code computes \( Y = 5 \times X \) in GLOBAL_Q format (assuming GLOBAL_Q = IQ26):

```c
_iq Y, X;
X=_FtoIQ(5.1);      // X = 5.1 in GLOBAL_Q format
Y = IQmpyI32(X, 5); // Y = 25.5 in GLOBAL_Q format
```

Example 2: The following code computes \( Y = 5 \times X \) in IQ26 format:

```c
I64_IQ Z;
int M;
M = 5;          // M = 5
X = _FtoIQN (5.1, 26); // X = 5.1 in IQ26 format
Y = _IQ26mpyI32(X, M); // Y = 25.5 in IQ26 format (32 bit result)
Z = _IQ26mpyI32(X, 0x80000000u); // 64 bit result in Z
```
**IQNmpyI32int — Integer Portion of (IQN*INT)**

**Description**
This function multiplies an IQ number with an integer and returns the integer part of the result.

**Declaration**
- **Global IQ function (IQ format = GLOBAL_Q)**
  ```c
  int _IQmpyI32int(_iq A, int B)
  ```
- **Q-format-specific IQ function (IQ format = IQ0 to IQ30)**
  ```c
  int _IQNmpyI32int(_iqN A, int B)
  ```

**Input Format**
- **Global IQ function (IQ format = GLOBAL_Q)**
  Operand A is an IQ number in GLOBAL_Q format and B is the integer.
- **Q-format-specific IQ function (IQ format = IQ0 to IQ30)**
  Operand A is an IQ number in IQN format and B is the integer.

**Output Format**
- **Global IQ function (IQ format = GLOBAL_Q)**
  Integer part of the result (32-bit).
- **Q-format-specific IQ function (IQ format = IQ0 to IQ30)**
  Integer part of the result (32-bit).

**Saturation**
The function saturates between maximum and minimum limits of Q format.

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

```c
int temp;
short OutputDAC;

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

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

if( temp < 0 )
  temp = 0;

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

**Note:** An integer operation performs the multiply and calculates the integer portion from the resulting 64-bit result.
**IQNmpyl32frac — Fractional Part of (IQN*INT)**

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

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

```c
_iq _IQmpyI32frac(_iq A, int B)
```

**Q-format-specific IQ function (IQ format = IQ0 to IQ30)**

```c
_iqN _IQNmpyI32frac(_iqN A, int B)
```

### Input Format
**Global IQ function (IQ format = GLOBAL_Q)**
Operand A is an IQ number in GLOBAL_Q format and B is the 32-bit integer.

**Q-format-specific IQ function (IQ format = IQ0 to IQ30)**
Operand A is an IQ number in IQN format and B is the 32-bit integer.

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

**Q-format-specific IQ function (IQ format = IQ0 to IQ30)**
Fractional part of the result (32-bit).

### Example
The following example extracts the fractional part of result after multiplication (assuming GLOBAL_Q=IQ26):

```c
int M1=5, M2=9;
_iq Y1frac, Y2frac;
_iq X1=_FtoIQ(2.5);
_iq X2=_FtoIQ26(-1.1);

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

This function multiplies two IQ number that are represented in different IQ formats.

**Declaration**

Global IQ function (IQ format = GLOBAL_Q)

```c
_iq _iqmpyIQX(_iqN1 A, int N1, _iqN2 B, int N2)
```

Q-format-specific IQ function (IQ format = IQ0 to IQ31)

```c
_iqN _IQmpyIQX(_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 = IQ0 to IQ31)

Result of the multiplication in IQN format.

**Saturation**

The function saturates between maximum and minimum limits of the Q format.

**Usage**

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

**Example**

We want to calculate the following equation:

\[ Y = X_0 \cdot C_0 + X_1 \cdot C_1 + X_2 \cdot C_2 \]

Where,

- \( X_0, X_1, \) and \( X_2 \) values are in IQ30 format (range -2 to +2)
- \( C_0, C_1, \) and \( C_2 \) values are in IQ28 format (range –8 to +8)

Maximum range of \( Y \) is -48 to +48. Hence, we should store the result in 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
_iq28 C0, C1, C2;       // All values IQ28
_iq25 Y;                // Result GLOBAL_Q = IQ25

Y = _IQ25mpyIQX(X0, 30, C0, 28);
Y += _IQ25mpyIQX(X1, 30, C1, 28);
Y += _IQ25mpyIQX(X2, 30, C2, 28);
```
**IQNdiv — Fixed-Point Division**

**Description**
This module divides two IQN number and provides 32-bit results (IQN format) using the Newton-Raphson technique.

**Declaration**
- **Global IQ function (IQ format = GLOBAL_Q)**
  
  \_iq \_IQdiv(_iq A, \_iq B)

- **Q-format-specific IQ function (IQ format = IQ0 to IQ31)**
  
  \_iqN \_IQNdiv(_iqN A, iqN B)

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

- **Q-format-specific IQ function (IQ format = IQ0 to IQ31)**
  The input arguments A and B are fixed-point numbers in IQN format (N=0:31).

**Output Format**
- **Global IQ function (IQ format = GLOBAL_Q)**
  Output in GLOBAL_Q format.

- **Q-format-specific IQ function (IQ format = IQ0 to IQ31)**
  Output in IQN format (N=0:31).

**Saturation**
This function saturates between maximum and minimum limits of Q format. No special handling of the divide by 0 has been implemented. Thus, the results are undefined if divide by 0 is attempted.

**Example**
The following example obtains 1/1.5=0.666, assuming that GLOBAL_Q is set to Q28 format in the IQmath header file:

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

  _iq in1, out1;
  _iq28 in2, out2;

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

    in2 = _IQ28 (1.5);  
    out2 = _IQ28div (_FtoIQ28 (1.0), in2);
  }
```
#include<IQmath.h> /* Header file for IQmath routine */
define PI 3.14156

_iq in1, out1;
_iq28 in2, out2;

void main (void)
{
    in1 = _FtoIQ (0.25*PI); /* in1=0.25 x π x 2^29 = 1921FB54h */
    out1 = _IQsin (in1); /* out1=sin(0.25 x π x 2^29) = 16A09E66h */
    in2 = _FtoIQ29 (0.25*PI); /* in2=0.25 x π x 2^29 = 1921FB54h */
    out2 = _IQ29sin (in2); /* out2=sin(0.25 x π x 2^29) = 16A09E66h */
}

---

### IQNsin — Fixed-Point SIN (in radians)

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

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

\[
_iq \_IQsin(_iq A)
\]

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

\[
_iqN \_IQNsin(_iqN A)
\]

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

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

**Saturation**
This function saturates between maximum and minimum limits of Q format.

**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<IQmath.h> /* Header file for IQmath routine */
define PI 3.14156

_iq in1, out1;
_iq28 in2, out2;

void main (void)
{
    in1 = _FtoIQ (0.25*PI); /* in1=0.25 x π x 2^29 = 1921FB54h */
    out1 = _IQsin (in1); /* out1=sin(0.25 x π x 2^29) = 16A09E66h */
    in2 = _FtoIQ29 (0.25*PI); /* in2=0.25 x π x 2^29 = 1921FB54h */
    out2 = _IQ29sin (in2); /* out2=sin(0.25 x π x 2^29) = 16A09E66h */
}
```
**IQNsinPU**  
*Fixed-Point SIN (in per-unit radians)*

**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)  
\[
\_iq\ _IQsinPU(_iq\ A)
\]

Q-format-specific IQ function (IQ format = IQ1 to IQ29)  
\[
\_iqN\ _IQNsinPU(_iqN\ A)
\]

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

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

**Saturation**  
The function saturates between maximum and minimum limits of the Q format.

**Example**  
The following example obtains the \( \sin(0.25 \times \pi) = 0.707 \), assuming that GLOBAL_Q is set to Q30 format in the IQmath header file:

```c
#include<IQmath.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 x \pi / 2^30 = 08000000h */
  out1=_IQsinPU (in1); /* out1=sin(0.25 x \pi) x 2^30 = 2D413CCCh */

  in2=IQ30 (0.25*PI/PI); /* in2 =0.25 x \pi / 2^30 = 08000000h */
  out2=_IQ30sinPU (in2); /* out2=sin(0.25 x \pi) x 2^30 = 2D413CCCh */
}
```
#include <IQmath.h> /* Header file for IQmath routine */
#define PI 3.14156
_iq in1, out1;
_iq29 in2 out2;

void main(void)
{
    in1 = _IQ (0.25*PI); /* in1=0.25 x x 2^29 = 1921FB54h */
    out1 = _IQcos (in1); /* out1=cos(0.25 x x) x 2^29 = 16A09E66h */
    in2 = _IQ29 (0.25*PI); /* in2=0.25 x x 2^29 = 1921FB54h */
    out2 = _IQ29cos (in2); /* out2=cos(0.25 x x) x 2^29 = 16A09E66h */
}
---

### IQNcosPU  
**Fixed-Point COS (in per-unit radians)**

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

**Declaration**

- **Global IQ function (IQ format = GLOBAL_Q)**
  
  ```c
  _iq _IQcosPU(_iq A)
  ```

- **Q-format-specific IQ function (IQ format = IQ1 to IQ29)**
  
  ```c
  _iqN _IQNcosPU(_iqN A)
  ```

**Input Format**

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

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

**Saturation**
The function saturates between maximum and minimum limits of the Q format.

**Example**
The following sample code obtains the cos(0.25 × π) = 0.707, assuming that GLOBAL_Q is set to Q30 format in the IQmath header file:

```c
#include<IQmath.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 × π/2 × 2^30 = 08000000h */
    out1=_IQcosPU (in1);  /* out1=cos(0.25 × π) × 2^30 = 2D413CCCh */

    in2=_IQ30 (0.25*PI/PI);
    out2=_IQ30cosPU (in2);  /* out2=cos(0.25 × π) × 2^30 = 2D413CCCh */
}
```
#include <IQmath.h>

/* Header file for IQ math routine */
#define PI 3.14156

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

void main(void)
{
    xin1 = FtoIQ(0.809); /* xin1 = \cos(\pi/5) \times 2^{29} = 19E3779Bh */
    yin1 = FtoIQ(0.5877); /* yin1 = \sin(\pi/5) \times 2^{29} = 12CF2304h */
    out1 = IQatan2(yin1, xin1); /* out1 = \pi/5 \times 2^{29} = 141B2F76h */

    xin2 = FtoIQ29(0.809); /* xin2 = \cos(\pi/5) \times 2^{29} = 19E3779Bh */
    yin2 = FtoIQ29(0.5877); /* yin2 = \sin(\pi/5) \times 2^{29} = 12CF2304h */
    out2 = IQ29atan2(yin2, xin2); /* out2 = \pi/5 \times 2^{29} = 141B2F76h */
}

---

**IQNatan2**

### Fixed-Point 4-Quadrant ATAN (in radians)

**Description**

This module computes the 4-quadrant arctangent. Module output in radians and varies from -\pi to \pi.

**Declaration**

Global IQ function (IQ format = GLOBAL_Q)

```c
_iq _IQatan2(_iq A, _iq B)
```

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

```c
_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)

The input arguments A and B are fixed-point numbers represented in GLOBAL_Q format.

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

The input arguments A and B are fixed-point numbers 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 a 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 a 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(\pi/5), \cos(\pi/5)] = \pi/5, assuming that GLOBAL_Q is set to Q29 format in the IQmath header file:

```c
#include <IQmath.h>
#define PI 3.14156

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

void main(void)
{
    xin1 = FtoIQ(0.809); /* xin1 = \cos(\pi/5) \times 2^{29} = 19E3779Bh */
    yin1 = FtoIQ(0.5877); /* yin1 = \sin(\pi/5) \times 2^{29} = 12CF2304h */
    out1 = IQatan2(yin1, xin1); /* out1 = \pi/5 \times 2^{29} = 141B2F76h */

    xin2 = FtoIQ29(0.809); /* xin2 = \cos(\pi/5) \times 2^{29} = 19E3779Bh */
    yin2 = FtoIQ29(0.5877); /* yin2 = \sin(\pi/5) \times 2^{29} = 12CF2304h */
    out2 = IQ29atan2(yin2, xin2); /* out2 = \pi/5 \times 2^{29} = 141B2F76h */
}
```
## IQNatan2PU

### Fixed-Point 4-Quadrant ATAN (in per-unit radians)

#### Description

This module computes the 4-quadrant arctangent. Module output is in per-unit radians and varies from 0 (0 radians) to 1 (2\(\pi\) radians).

#### 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)

The input arguments A and B are fixed-point numbers represented in GLOBAL_Q format.

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

The input arguments A and B are fixed-point numbers 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 a fixed-point number in GLOBAL_Q format. The output contains the angle in per-unit radians and varies from 0 (0 radians) to 1 (2\(\pi\) radians).

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

This function returns the inverse tangent of the input argument as a fixed-point number in IQN format (\(N=1:29\)). The output contains the angle in per-unit radians and varies from 0 (0 radians) to 1 (2\(\pi\) radians).

#### Example

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<IQmath.h> /* Header file for IQ math routine */

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

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

    xin2=_IQ29(0.809) /* xin2=cos(\pi/5) \times 2^{29} = 19E3779Bh */
    yin2=_IQ29(0.5877) /* yin2=sin(\pi/5) \times 2^{29} = 12CF2304h */
    out2=_IQ29atan2PU(yin2,xin2) /* ou2 = (\pi/5)/2\pi \times 2^{29} = 03333333h */
}
```
**Description**
This module computes the arctangent. Module output is in radians and varies from \(-\pi/2\) to \(\pi/2\).

**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)**
The input argument is a fixed-point number in GLOBAL_Q format.

**Q-format-specific IQ function (IQ format = IQ1 to IQ29)**
The input argument is a 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 a fixed-point number in GLOBAL_Q format. The output contains the angle in radians between \([-\pi/2, +\pi/2]\).

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

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

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

_iq in1, out1;
_iq29 in2, out2;

void main(void )
{
    in1=_IQ(1.0);
    out1=_IQatan(in1);
    in2=_IQ29(1.0);
    out2=_IQ29atan(in2)
}
```
# IQNsqrt — Fixed-Point Square Root

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

## Declaration
- **Global IQ function (IQ format = GLOBAL_Q)**
  
  ```c
  _iq _IQsqrt(_iq A)
  ```

- **Q-format-specific IQ function (IQ format = IQ0 to IQ30)**
  
  ```c
  _iqN _IQNsqrt(_iqN A)
  ```

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

- **Q-format-specific IQ function (IQ format = IQ0 to IQ30)**
  The input argument is a fixed-point number in IQN format (N=0:30).

## Output Format
- **Global IQ function (IQ format = GLOBAL_Q)**
  Square root of input in GLOBAL_Q format.

- **Q-format-specific IQ function (IQ format = IQ0 to IQ30)**
  Square root of input in IQN format (N=0:30).

## Saturation
The function saturates between maximum and minimum limits of the Q format.

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

_iq in1, out1;
_iq30 in2, out2;

void main(void )
{
    in1=_FtoIQ(1.8); /* in1= 1.8x2^30 = 73333333h */
    out1=_IQsqrt(x); /* out1= \sqrt{1.8}x2^30 = 55DD7151h */

    in2=_FtoIQ30(1.8); /* in2= 1.8x2^30 = 73333333h */
    out2=_IQ30sqrt(x); /* out2= \sqrt{1.8}x2^30 = 55DD7151h */
}
```
**IQNisqrt — Fixed-Point Inverse Square Root**

**Description**
This module computes the inverse square root of the input using table look-up and Newton-Raphson approximation.

**Declaration**
- **Global IQ function (IQ format = GLOBAL_Q)**
  
  ```c
  _iq _IQisqrt(_iq A)
  ```

- **Q-format-specific IQ function (IQ format = IQ0 to IQ30)**
  
  ```c
  _iqN _IQNisqrt(_iqN A)
  ```

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

- **Q-format-specific IQ function (IQ format = IQ0 to IQ30)**
The input argument is a fixed-point number in IQN format (N=0:30).

**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 = IQ0 to IQ30)**
Inverse square root of input in IQN format (N=0:30).

**Saturation**
The function saturates between maximum and minimum limits of the Q format.

**Example**
The following example obtains \(1/\sqrt{1.8} = 0.74535\), assuming that GLOBAL_Q is set to Q30 format in IQmath header file:

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

_iq in1, out1;
_iq30 in2, out2;

void main(void )
{

    in1=_FtoIQ(1.8); /* in1 = 1.8x30 = 73333333h */
    out1=_IQisqrt(in1); /* out1 = 1/\sqrt{1.8} x30 = 2FB3E99Ehh */

    in2=_FtoIQ30(1.8); /* in2 = 1.8x30 = 73333333h */
    out2=_IQ30isqrt(in2); /* out2 = 1/\sqrt{1.8} x30 = 2FB3E99Ehh */
}
```
**IQNmag — Magnitude of IQ Complex Number**

**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. This is because the internal computations \((A^2 + B^2)\) are maintained at 64-bit accuracy.

**Declaration**
**Global IQ function (IQ format = GLOBAL\_Q)**

\_iq \_IQmag\(_iq A, \_iq B\)

**Q-format-specific IQ function (IQ format = IQ0 to IQ30)**

\_iqN \_IQNmag\(_iqN A, \_iqN B\)

**Input Format**
**Global IQ function (IQ format = GLOBAL\_Q)**
The input arguments \( A \) and \( B \) are IQ numbers represented in GLOBAL\_Q format.

**Q-format-specific IQ function (IQ format = IQ0 to IQ30)**
The input arguments \( A \) and \( B \) are IQ numbers 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 = IQ0 to IQ30)**
Magnitude of the input vector in IQN format.

**Saturation**
This function saturates between maximum and minimum limits of the Q format.

**Example**
The following sample code obtains the magnitude of the complex number (assuming GLOBAL\_Q=IQ28):

```c
#include <IQmath.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=_FtoIQ(4.0);
    imag1=_FtoIQ(4.0);
    mag1=_IQmag(real1, imag1);    /* mag1=5.6568 in IQ28 format */

    real2=_FtoIQ28(7.0);
    imag2=_FtoIQ28(7.0);
    mag2=_IQ28mag(real2, imag2);  /* mag2=8.0, saturated to MAX value (IQ28) */
}
```

---

**SPRUGG9—December 2008**

Submit Documentation Feedback
**IQNexp**  
---

**Magnitude of IQ Complex Number**

**Description**
This function calculates the fixed-point exponential of a value A.

**Declaration**

- **Global IQ function (IQ format = GLOBAL_Q)**
  
  `_iq _IQexp(_iq A)`

- **Q-format-specific IQ function (IQ format = IQ1 to IQ30)**
  
  `_iqN _IQNexp(_iqN A)`

**Input Format**

- **Global IQ function (IQ format = GLOBAL_Q)**
  The input argument A is an IQ number represented in GLOBAL_Q format.

- **Q-format-specific IQ function (IQ format = IQ1 to IQ30)**
  The input argument A is an IQ number represented in IQN format.

**Output Format**

- **Global IQ function (IQ format = GLOBAL_Q)**
  Exponential of the input vector in GLOBAL_Q format.

- **Q-format-specific IQ function (IQ format = IQ1 to IQ30)**
  Exponential of the input vector in IQN format.

**Example**

The following sample code obtains the exponential of a number (assuming GLOBAL_Q=IQ28):

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

_iq real1,exp1;
_iq28 real2,exp2;
void main(void )
{
    real1=_IQ(1.0);
    exp1=_IQexp(real1);  /* 2.71828 in GLOBAL_Q */

    real2=_IQ28(1.0);
    exp2=_IQ28exp(real2);  /* 2.71828 in Q28 Format */
}
```
<table>
<thead>
<tr>
<th>IQNlog</th>
<th><strong>Magnitude of IQ Complex Number</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>This function calculates the fixed-point natural logarithm.</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td><strong>Global IQ function (IQ format = GLOBAL_Q)</strong>&lt;br&gt;_iq _IQlog(_iq A)&lt;br&gt;<strong>Q-format-specific IQ function (IQ format = IQ1 to IQ30)</strong>&lt;br&gt;_iq_IQNlog(_iq A)</td>
</tr>
<tr>
<td><strong>Input Format</strong></td>
<td><strong>Global IQ function (IQ format = GLOBAL_Q)</strong>&lt;br&gt;The input argument A is an IQ number represented in GLOBAL_Q format.&lt;br&gt;<strong>Q-format-specific IQ function (IQ format = IQ1 to IQ30)</strong>&lt;br&gt;The input argument A is an IQ number represented in IQN format.</td>
</tr>
<tr>
<td><strong>Output Format</strong></td>
<td><strong>Global IQ function (IQ format = GLOBAL_Q)</strong>&lt;br&gt;Logarithm of the input vector in GLOBAL_Q format.&lt;br&gt;<strong>Q-format-specific IQ function (IQ format = IQ1 to IQ30)</strong>&lt;br&gt;Logarithm of the input vector in IQN format.</td>
</tr>
</tbody>
</table>
| **Example** | The following sample code obtains the logarithm of the complex number (assuming GLOBAL_Q=IQ28):<br>#include< IQmath.h >      /* Header file for IQ math routine  */<br>\_iq real1, log1;<br>\_iq\_IQ28 real2, log2;<br>void main(void )<br>{<br>    real1=_IQ(4.0);<br>    mag1=_IQlog(real1);    /* 1.38629 in GLOBAL_Q format */<br>    real2=_IQ28(4.0);<br>    mag2=_IQ28log(real2);    /* 1.38629 in Q28 */
} |
**IQNpow — Magnitude of IQ Complex Number**

**Description**
This function calculates the fixed-point power of two orthogonal vectors as follows:
\[ \text{pow} = A^B. \]

**Declaration**
- **Global IQ function (IQ format = GLOBAL_Q)**
  \[ \_iq \_IQpow(_iq A, \_iq B) \]
- **Q-format-specific IQ function (IQ format = IQ1 to IQ30)**
  \[ \_iqN \_IQNpow(_iqN A, \_iqN B) \]

**Input Format**
- **Global IQ function (IQ format = GLOBAL_Q)**
The input arguments A and B are IQ numbers represented in GLOBAL_Q format.
- **Q-format-specific IQ function (IQ format = IQ1 to IQ30)**
The input arguments A and B are IQ numbers represented in IQN format.

**Output Format**
- **Global IQ function (IQ format = GLOBAL_Q)**
  Power of the input vector in GLOBAL_Q format.
- **Q-format-specific IQ function (IQ format = IQ1 to IQ30)**
  Power of the input vector in IQN format.

**Example**
The following sample code demonstrates the usage of IQNpow function:

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

\_iq real1, imag1, mag1;
\_iq28 real2, imag2, mag2;
void main(void)
{    
    real1=_FtoIQ(4.0);
    imag1=_FtoIQ(1.0);
    mag1=_IQpow(real1, imag1);       /* mag1=4 in GLOBAL_Q */
    real2=_FtoIQ28(4.0);
    imag2=_FtoIQ28(1.0);
    mag2=_IQ28pow(real2, imag2);    /* mag2=4.0 in IQ28 format */
}
```

---

Submit Documentation Feedback
## IQNabs — Absolute Value of IQ Number

<table>
<thead>
<tr>
<th>Description</th>
<th>This function calculates the absolute value of an IQ number.</th>
</tr>
</thead>
</table>
| Declaration | **Global IQ function** (IQ format = GLOBAL_Q)  
_\_iq_ _IQabs(_iq A)  
**Q-format-specific IQ function** (IQ format = IQ0 to IQ31)  
_\_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 = IQ0 to IQ31)  
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 = IQ0 to IQ31)  
Absolute value of input in IQN format. |
| Saturation | This function does not saturate. |
| Example | Calculate the absolute sum of three IQ numbers (GLOBAL_Q=IQ28):  
_\_iq xin1, xin2, xin3, xsum;  
_\_iq20 yin1, yin2, yin3, ysum;  
\[xsum = _\_IQabs(X0) + _\_IQabs(X1) + _\_IQabs(X2);\]  
\[xsum = _\_IQ28abs(X0) + _\_IQ28abs(X1) + _\_IQ28abs(X2);\] |
**IQsat — Saturate the IQ Number**

**Description**
This function 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**

```c
_iq _IQsat(_iq A, int P, int 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)**
Saturated output in GLOBAL_Q format.

**Saturation**
The function saturates between maximum and minimum limits of the Q format.

**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
_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);
```
IMPORTANT NOTICE

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

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

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

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

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

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

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

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

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

<table>
<thead>
<tr>
<th>Products</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amplifiers</td>
<td>Audio</td>
</tr>
<tr>
<td>Data Converters</td>
<td>Automotive</td>
</tr>
<tr>
<td>DSP</td>
<td>Broadband</td>
</tr>
<tr>
<td>Clocks and Timers</td>
<td>Digital Control</td>
</tr>
<tr>
<td>Interface</td>
<td>Medical</td>
</tr>
<tr>
<td>Logic</td>
<td>Military</td>
</tr>
<tr>
<td>Power Mgmt</td>
<td>Optical Networking</td>
</tr>
<tr>
<td>Microcontrollers</td>
<td>Security</td>
</tr>
<tr>
<td>RFID</td>
<td>Telephony</td>
</tr>
<tr>
<td>RF/I and ZigBee® Solutions</td>
<td>Video &amp; Imaging</td>
</tr>
<tr>
<td></td>
<td>Wireless</td>
</tr>
</tbody>
</table>

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2008, Texas Instruments Incorporated