# MSP430® IQmathLib Users Guide version 01.10.00.05

# USER'S GUIDE

# Copyright



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this document.

Texas Instruments
Post Office Box 655303
Dallas, TX 75265
http://www.ti.com/msp430


# Revision Information

This is version 01.10.00.05 of this document, last updated on January 19, 2015.

# Table of Contents

# 1 Introduction

The Texas Instruments® IQmath and Qmath Libraries are a collection of highly optimized and high-precision mathematical functions for C programmers to seamlessly port a floating-point algorithm into fixed-point code on MSP430 and MSP432 devices. These routines are typically used in computationally intensive real-time applications where optimal execution speed, high accuracy and ultra low energy are critical. By using the IQmath and Qmath libraries, it is possible to achieve execution speeds considerably faster and energy consumption considerably lower than equivalent code written using floating-point math.

The Qmath library provides functions for use with 16-bit fixed point data types. These functions have been optimized for all devices and can efficiently be used with or without a hardware multiplier. The functions provide up to 16 bits of accuracy to satisfy the majority of applications on MSP430 devices.

The IQmath library provides the same functions as the Qmath library with 32-bit data types and higher accuracy. These functions are provided for when an application requires accuracy comparable or greater than the equivalent floating point math functions.

The following tool chains are supported:

- Texas Instruments Code Composer Studio
- IAR Embedded Workbench for MSP430 and MSP432

# 2 Using The Qmath and IQmath Libraries

## 2.1 Qmath Data Types

The Qmath library uses a 16-bit fixed-point signed number (an “int16_t” in C99) as its basic data type. The Q format of this fixed-point number can range from Q1 to Q15, where the Q format number indicates the number of fractional bits. The Q format value is stored as an integer with an implied scale based on the Q format and the number of fractional bits. Equation 2.1 shows how a Q format decimal number $x_q$ is stored using an integer value $x_i$ with an implied scale, where $n$ represents the number of fractional bits.

$$Qn(x_q) = x_i * 2^{-n} \tag{2.1}$$

For example the Q12 value of 3.625 is stored as an integer value of 14848, shown in equation 2.2 below.

$$14848 * 2^{-12} = Q12(3.625) \tag{2.2}$$

C typedefs are provided for the various Q formats, and these Qmath data types should be used in preference to the underlying “int16_t” data type to make it clear which variables are in Q format.

The following table provides the characteristics of the various Q formats (the C data type, the number of integer bits, the number of fractional bits, the smallest negative value that can be represented, the largest positive value that can be represented, and the smallest difference that can be represented):

| Type | Bits | | Range | | Resolution |
|---|---|---|---|---|---|
| | Integer | Fractional | Min | Max | |
| _q15 | 1 | 15 | -1 | 0.999 970 | 0.000 030 |
| _q14 | 2 | 14 | -2 | 1.999 940 | 0.000 061 |
| _q13 | 3 | 13 | -4 | 3.999 830 | 0.000 122 |
| _q12 | 4 | 12 | -8 | 7.999 760 | 0.000 244 |
| _q11 | 5 | 11 | -16 | 15.999 510 | 0.000 488 |
| _q10 | 6 | 10 | -32 | 31.999 020 | 0.000 976 |
| _q9 | 7 | 9 | -64 | 63.998 050 | 0.001 953 |
| _q8 | 8 | 8 | -128 | 127.996 090 | 0.003 906 |
| _q7 | 9 | 7 | -256 | 255.992 190 | 0.007 812 |
| _q6 | 10 | 6 | -512 | 511.984 380 | 0.015 625 |
| _q5 | 11 | 5 | -1,024 | 1,023.968 750 | 0.031 250 |
| _q4 | 12 | 4 | -2,048 | 2047.937 500 | 0.062 500 |
| _q3 | 13 | 3 | -4,096 | 4,095.875 000 | 0.125 000 |
| _q2 | 14 | 2 | -8,192 | 8,191.750 000 | 0.250 000 |
| _q1 | 15 | 1 | -16,384 | 16,383.500 000 | 0.500 000 |

Table 2.1: Qmath Data Types

In addition to these specific Q format types, there is an additional type that corresponds to the GLOBAL_Q format. This is _q, and it matches one of the above Q formats (based on the setting of

GLOBAL_Q). The GLOBAL_Q format has no impact when using the specific _qN types and function such as _q12.

## 2.2 IQmath Data Types

The IQmath library uses a 32-bit fixed-point signed number (an “int32_t” in C99) as its basic data type. The IQ format of this fixed-point number can range from IQ1 to IQ30, where the IQ format number indicates the number of fractional bits. The IQ format value is stored as an integer with an implied scale based on the IQ format and the number of fractional bits. Equation 2.3 shows how a IQ format decimal number $x_{iq}$ is stored using an integer value $x_i$ with an implied scale, where $n$ represents the number of fractional bits.

$$IQn(x_{iq}) = x_i * 2^{-n} \tag{2.3}$$

For example the IQ24 value of 3.625 is stored as an integer value of 60817408, shown in equation 2.4 below.

$$60817408 * 2^{-24} = IQ24(3.625) \tag{2.4}$$

C typedefs are provided for the various IQ formats, and these IQmath data types should be used in preference to the underlying “int32_t” data type to make it clear which variables are in IQ format.

The following table provides the characteristics of the various IQ formats (the C data type, the number of integer bits, the number of fractional bits, the smallest negative value that can be represented, the largest positive value that can be represented, and the smallest difference that can be represented):

| Type | Bits | | Range | | Resolution |
|---|---|---|---|---|---|
| | Integer | Fractional | Min | Max | |
| _iq30 | 2 | 30 | -2 | 1.999 999 999 | 0.000 000 001 |
| _iq29 | 3 | 29 | -4 | 3.999 999 998 | 0.000 000 002 |
| _iq28 | 4 | 28 | -8 | 7.999 999 996 | 0.000 000 004 |
| _iq27 | 5 | 27 | -16 | 15.999 999 993 | 0.000 000 007 |
| _iq26 | 6 | 26 | -32 | 31.999 999 985 | 0.000 000 015 |
| _iq25 | 7 | 25 | -64 | 63.999 999 970 | 0.000 000 030 |
| _iq24 | 8 | 24 | -128 | 127.999 999 940 | 0.000 000 060 |
| _iq23 | 9 | 23 | -256 | 255.999 999 881 | 0.000 000 119 |
| _iq22 | 10 | 22 | -512 | 511.999 999 762 | 0.000 000 238 |
| _iq21 | 11 | 21 | -1,024 | 1,023.999 999 523 | 0.000 000 477 |
| _iq20 | 12 | 20 | -2,048 | 2,047.999 999 046 | 0.000 000 954 |
| _iq19 | 13 | 19 | -4,096 | 4,095.999 998 093 | 0.000 001 907 |
| _iq18 | 14 | 18 | -8,192 | 8,191.999 996 185 | 0.000 003 815 |
| _iq17 | 15 | 17 | -16,384 | 16,383.999 992 371 | 0.000 007 629 |
| _iq16 | 16 | 16 | -32,768 | 32,767.999 984 741 | 0.000 015 259 |
| _iq15 | 17 | 15 | -65,536 | 65,535.999 969 483 | 0.000 030 518 |
| _iq14 | 18 | 14 | -131,072 | 131,071.999 938 965 | 0.000 061 035 |
| _iq13 | 19 | 13 | -262,144 | 262,143.999 877 930 | 0.000 122 070 |
| _iq12 | 20 | 12 | -524,288 | 524,287.999 755 859 | 0.000 244 141 |
| _iq11 | 21 | 11 | -1,048,576 | 1,048,575.999 511 720 | 0.000 488 281 |
| _iq10 | 22 | 10 | -2,097,152 | 2,097,151.999 023 440 | 0.000 976 563 |
| _iq9 | 23 | 9 | -4,194,304 | 4,194,303.998 046 880 | 0.001 953 125 |
| _iq8 | 24 | 8 | -8,388,608 | 8,388,607.996 093 750 | 0.003 906 250 |
| _iq7 | 25 | 7 | -16,777,216 | 16,777,215.992 187 500 | 0.007 812 500 |
| _iq6 | 26 | 6 | -33,554,432 | 33,554,431.984 375 000 | 0.015 625 000 |
| _iq5 | 27 | 5 | -67,108,864 | 67,108,863.968 750 000 | 0.031 250 000 |
| _iq4 | 28 | 4 | -134,217,728 | 134,217,727.937 500 000 | 0.062 500 000 |
| _iq3 | 29 | 3 | -268,435,456 | 268,435,455.875 000 000 | 0.125 000 000 |
| _iq2 | 30 | 2 | -536,870,912 | 536,870,911.750 000 000 | 0.250 000 000 |
| _iq1 | 31 | 1 | -1,073,741,824 | 1,073,741,823.500 000 000 | 0.500 000 000 |

Table 2.2: IQmath Data Types

In addition to these specific IQ format types, there is an additional type that corresponds to the GLOBAL_IQ format. This is _iq, and it matches one of the above IQ formats (based on the setting of GLOBAL_IQ).The GLOBAL_IQ format has no impact when using the specific _iqN types and function such as _iq24.

## 2.3 Using the Libraries

The Qmath and IQmath libraries are available for a wide range of MSP430 and MSP432 devices from value line to the F5xx series to the latest FRAM and ARM based devices. The libraries are available in the top level libraries directory and are divided by IDE and multiplier hardware support.

Each library name is constructed with the IDE and multiplier hardware used such that it matches the directory path. The name is then followed by a specifier for the CPU version and code and data models if applicable.

### 2.3.1 Code Composer Studio

The Code Composer Studio (CCS) libraries are provided in easy to use archive files, QmathLib.a and IQmathLib.a. The archive files should be used with projects in place of any .lib files. When linking, the archive file will select the correct library based on CPU, memory and data model compiler settings.

To add a library to an existing CCS project, simply navigate to the device directory under IQmathLib/libraries/CCS and drag and drop the QmathLib.a or IQmathLib.a file to the CCS project. When prompted select "Link to files" and you are done!

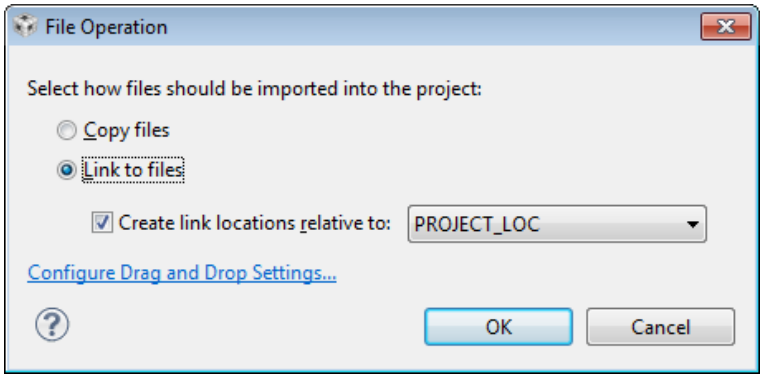


Figure 2.1: CCS file add prompt

The full list of CCS libraries are provided in the tables below. These are automatically selected when using QmathLib.a or IQmathLib.a but can also be added directly to a project.

| Library Name | CPU | Multiply Hardware | Code Model | Data Model |
|---|---|---|---|---|
| *_MPYsoftware_CPU.lib | CPU | Software | - | - |
| *_MPYsoftware_CPUX_small_code_small_data.lib | CPUX | Software | small | small |
| *_MPYsoftware_CPUX_large_code_small_data.lib | CPUX | Software | large | small |
| *_MPYsoftware_CPUX_large_code_restricted_data.lib | CPUX | Software | large | restricted |
| *_MPYsoftware_CPUX_large_code_large_data.lib | CPUX | Software | large | large |

Table 2.3: CCS software multiply libraries for all MSP430 devices.

| Library Name | CPU | Multiply Hardware | Code Model | Data Model |
|---|---|---|---|---|
| *_MPY32_4xx_CPU.lib | CPU | MPY32 | - | - |
| *_MPY32_4xx_CPUX_small_code_small_data.lib | CPUX | MPY32 | small | small |
| *_MPY32_4xx_CPUX_large_code_small_data.lib | CPUX | MPY32 | large | small |
| *_MPY32_4xx_CPUX_large_code_restricted_data.lib | CPUX | MPY32 | large | restricted |
| *_MPY32_4xx_CPUX_large_code_large_data.lib | CPUX | MPY32 | large | large |

Table 2.4: CCS MPY32 libraries for F4xx devices.

| Library Name | CPU | Multiply Hardware | Code Model | Data Model |
|---|---|---|---|---|
| *_MPY32_5xx_6xx_CPUX_small_code_small_data.lib | CPUX | MPY32 | small | small |
| *_MPY32_5xx_6xx_CPUX_large_code_small_data.lib | CPUX | MPY32 | large | small |
| *_MPY32_5xx_6xx_CPUX_large_code_restricted_data.lib | CPUX | MPY32 | large | restricted |
| *_MPY32_5xx_6xx_CPUX_large_code_large_data.lib | CPUX | MPY32 | large | large |

Table 2.5: CCS MPY32 libraries for F5xx, F6xx, FR5xx and FR6xx devices.

| Library Name | CPU | Multiply Hardware | Code Model | Data Model |
|---|---|---|---|---|
| *_MSP432.lib | ARM M4F | Yes | n/a | n/a |

Table 2.6: CCS libraries for MSP432 devices.

### 2.3.2 IAR Embedded Workbench

The IAR Embedded Workbench libraries are provided under the IQmathLib/libraries/IAR folder and organized by multiplier hardware support and device family. When selecting a library the CPU version, code and data model must match the options used in the project settings shown below.

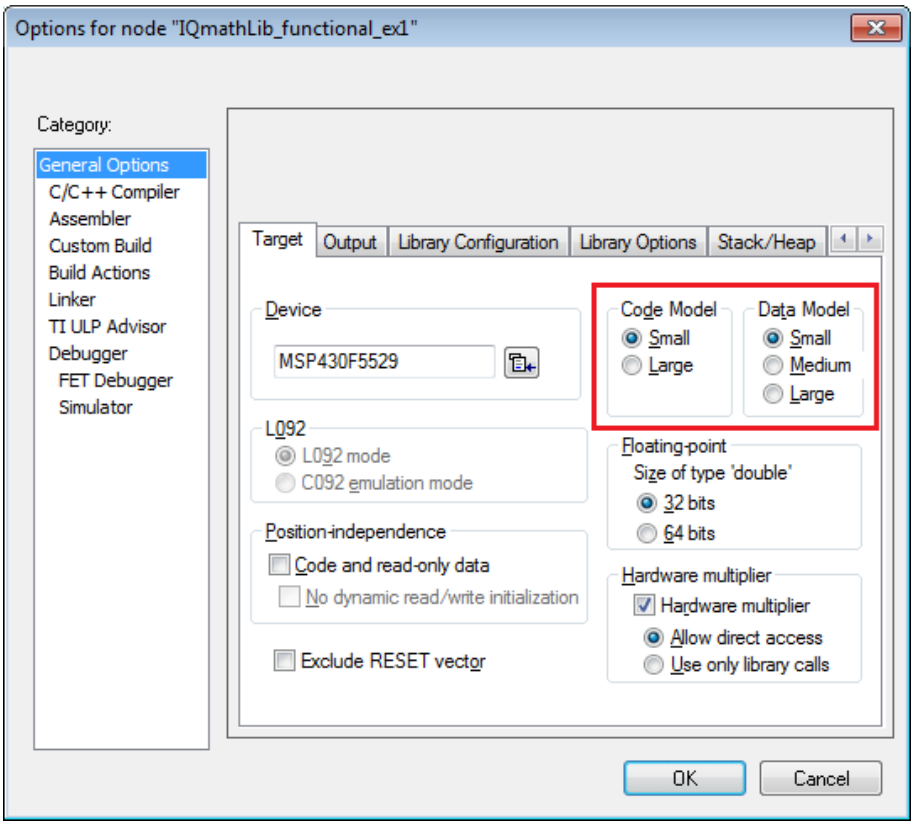


Figure 2.2: IAR code and data model options

The IAR library files can be added to a project either by dragging and dropping the library to the project or right clicking the project and selecting Add -> Add Files as shown below.

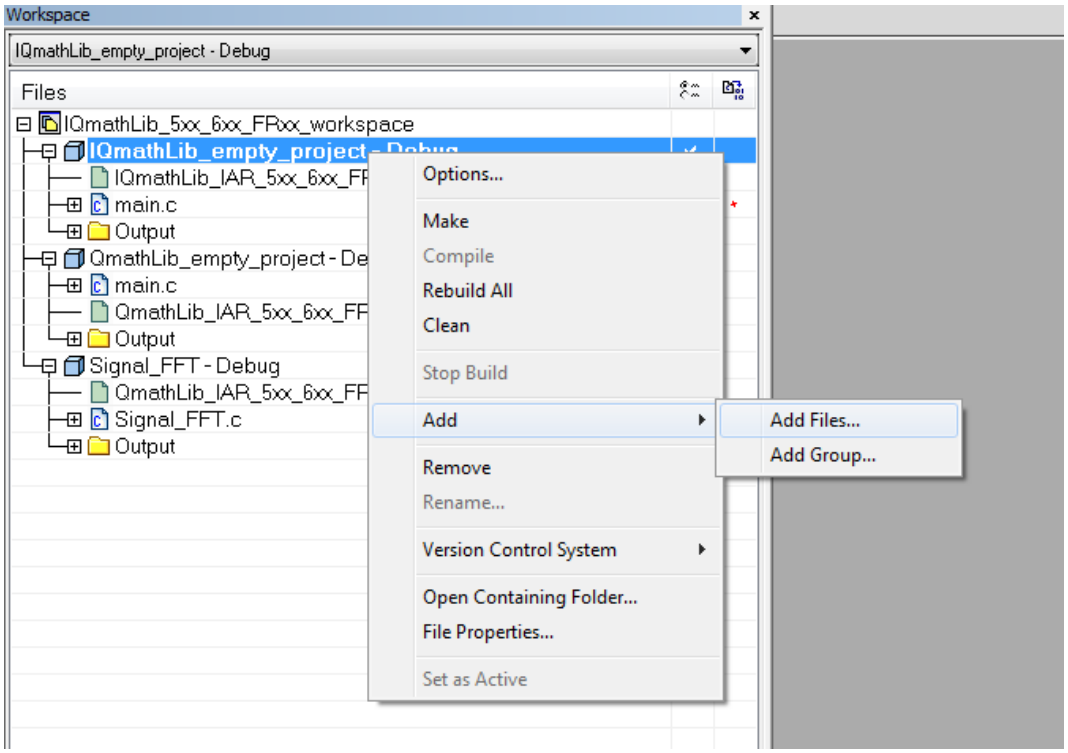


Figure 2.3: IAR add files option

When selecting a library file for versions of IAR previous to 6.10, the code model is always set to large for CPUX based devices. The library selected must be large code model and then the data model that matches the project settings shown above.

The full list of IAR libraries are provided in the tables below.

| Library Name | CPU | Multiply Hardware | Code Model | Data Model |
|---|---|---|---|---|
| *_MPYsoftware_CPU.lib | CPU | Software | - | - |
| *_MPYsoftware_CPUX_small_code_small_data.lib | CPUX | Software | small | small |
| *_MPYsoftware_CPUX_small_code_medium_data.lib | CPUX | Software | small | medium |
| *_MPYsoftware_CPUX_small_code_large_data.lib | CPUX | Software | small | large |
| *_MPYsoftware_CPUX_large_code_small_data.lib | CPUX | Software | large | small |
| *_MPYsoftware_CPUX_large_code_medium_data.lib | CPUX | Software | large | medium |
| *_MPYsoftware_CPUX_large_code_large_data.lib | CPUX | Software | large | large |

Table 2.7: IAR software multiply libraries for all MSP430 devices.

| Library Name | CPU | Multiply Hardware | Code Model | Data Model |
|---|---|---|---|---|
| *_MPY32_4xx_CPU.lib | CPU | MPY32 | - | - |
| *_MPY32_4xx_CPUX_small_code_small_data.lib | CPUX | MPY32 | small | small |
| *_MPY32_4xx_CPUX_small_code_medium_data.lib | CPUX | MPY32 | small | medium |
| *_MPY32_4xx_CPUX_small_code_large_data.lib | CPUX | MPY32 | small | large |
| *_MPY32_4xx_CPUX_large_code_small_data.lib | CPUX | MPY32 | large | small |
| *_MPY32_4xx_CPUX_large_code_restricted_data.lib | CPUX | MPY32 | large | medium |
| *_MPY32_4xx_CPUX_large_code_large_data.lib | CPUX | MPY32 | large | large |

Table 2.8: IAR MPY32 libraries for F4xx devices.

| Library Name | CPU | Multiply Hardware | Code Model | Data Model |
|---|---|---|---|---|
| *_MPY32_5xx_6xx_CPUX_small_code_small_data.lib | CPUX | MPY32 | small | small |
| *_MPY32_5xx_6xx_CPUX_small_code_medium_data.lib | CPUX | MPY32 | small | medium |
| *_MPY32_5xx_6xx_CPUX_small_code_large_data.lib | CPUX | MPY32 | small | large |
| *_MPY32_5xx_6xx_CPUX_large_code_small_data.lib | CPUX | MPY32 | large | small |
| *_MPY32_5xx_6xx_CPUX_large_code_restricted_data.lib | CPUX | MPY32 | large | restricted |
| *_MPY32_5xx_6xx_CPUX_large_code_large_data.lib | CPUX | MPY32 | large | large |

Table 2.9: IAR MPY32 libraries for F5xx, F6xx, FR5xx and FR6xx devices.

| Library Name | CPU | Multiply Hardware | Code Model | Data Model |
|---|---|---|---|---|
| *_MSP432.lib | ARM M4F | Yes | n/a | n/a |

Table 2.10: IAR libraries for MSP432 devices.

## 2.4 MSP430-GCC Beta Support

The CCS libraries are built for and intended for use with the TI compiler tool chain. These libraries can be linked with and used by MSP430-GCC projects, however this feature is untested and is the responsibility of the user to verify end application functionality.

When linking the libraries with MSP430-GCC the *.a archive files cannot be used and the library file must be manually selected. Additionally, the `--gc-sections` linker option must be applied to discard unused sections and avoid including additional code and data sections than is required by the application. This can be applied in the CCS GUI on the linker settings page shown below.

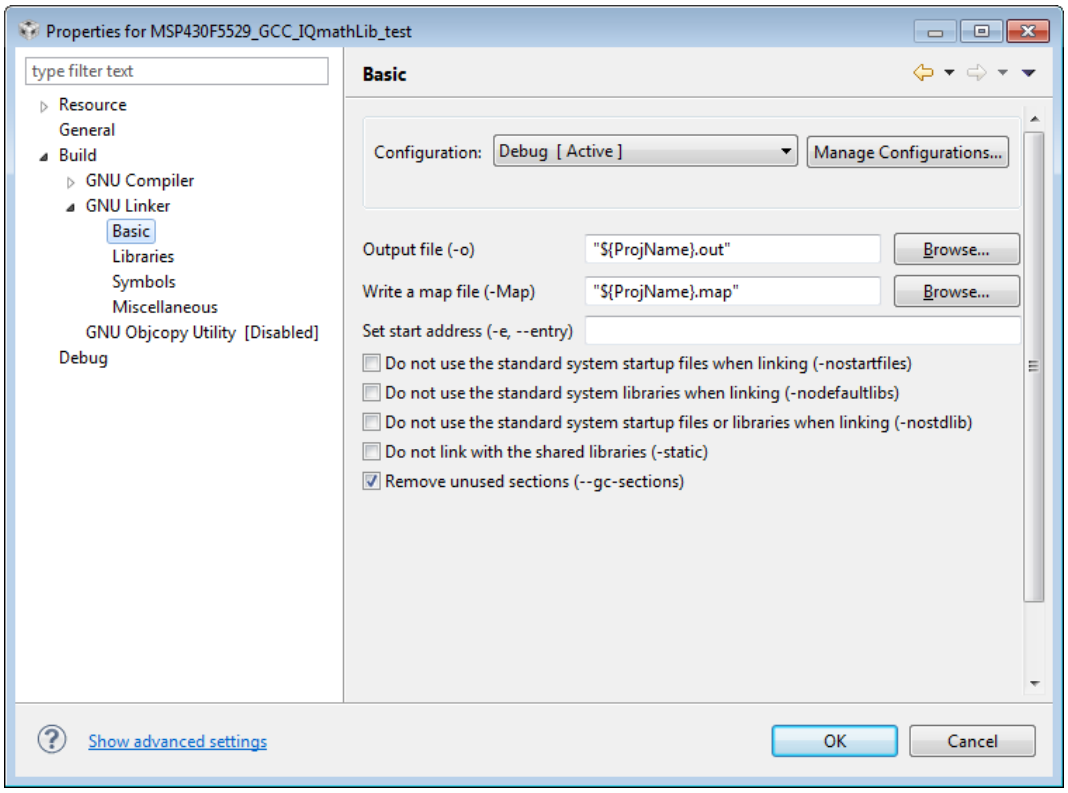


Figure 2.4: MSP430-GCC linker options

## 2.5 Calling Functions From C

In order to call a Qmath or IQmath function from C, the C header file must be included. Then, the `_q`, `_qN`, `_iq` and `_iqN` data types, along with the Qmath and IQmath functions can be used by the application.

As an example, the following code performs some simple arithmetic in Q12 format:

```
#include "QmathLib.h"

int
main(void)
{
    _q12 X, Y, Z;

    X = _Q12(1.0);
    Y = _Q12(7.0);

    Z = _Q12div(X, Y);
}
```

## 2.6 Selecting The Global Q and IQ Formats

Numerical precision and dynamic range requirements vary considerably from one application to another. The libraries provides a GLOBAL_Q and GLOBAL_IQ format (using the _q and _iq data types respectively) that an application can use to perform its computations in a generic format which can be changed at compile time. An application written using the GLOBAL_Q and GLOBAL_IQ formats can be changed from one format to another by simply changing the GLOBAL_Q and GLOBAL_IQ values and recompiling, allowing the precision and performance effects of different formats to be easily measured and evaluated.

The setting of GLOBAL_Q and GLOBAL_IQ does not have any influence in the _qN and _iqN format and corresponding functions. These types will always have the same fixed accuracy regardless of the GLOBAL_Q or GLOBAL_IQ formats.

The default GLOBAL_Q format is Q10 and the default GLOBAL_IQ format is IQ24. This can be easily overridden in one of two ways:

- In the source file, the format can be selected prior to including the header file. The following example selects a GLOBAL_Q format of Q8:

```
//
// Set GLOBAL_Q to 8 prior to including QmathLib.h.
//
#define GLOBAL_Q    8
#include "QmathLib.h"
```

- In the project file, add a predefined value for GLOBAL_Q or GLOBAL_IQ. The method to add a predefined value varies from tool chain to tool chain.

The first method allows different modules in the application to have different global format values, while the second method changes the global format value for the entire application. The method that is most appropriate varies from application to application.

**Note:** Some functions are not available when GLOBAL_Q and GLOBAL_IQ are set to 15 and 30 respectively. Please see the Qmath and IQmath function chapters for a list of functions and the available Q and IQ formats.

# 2.7 Example Projects

The IQmathLib provides four example projects for use with CCS or IAR:

- Empty QmathLib project
- Empty IQmathLib project
- QmathLib basic functional example
- QmathLib signal generator and FFT example

The empty QmathLib and IQmathLib projects provide a starting point for building a fixed point application. These projects will already have the libraries added and the include path set to include the header files.

The third example (QmathLib_functional_ex3) demonstrates how to use several of the QmathLib functions and data types to perform basic math calculations.

The fourth example (QmathLib_signal_FFT_ex4) is a code example that demonstrates how the QmathLib can be used to write application code. The example can be separated into two parts:

- Generate an input signal from multiple cosine waves.
- Perform a complex DFT (FFT) on the input signal.

The result of the complex DFT can be used to approximate the original signals amplitude and phase angle at each of the frequency bins.

## 2.7.1 Importing CCS Example Projects

The CCS example projects are provided as .projectspec files for each device family. These files can be imported to the workspace as a new project using the "Import" option and selecting the "Existing CCS Eclipse Projects" category shown below.

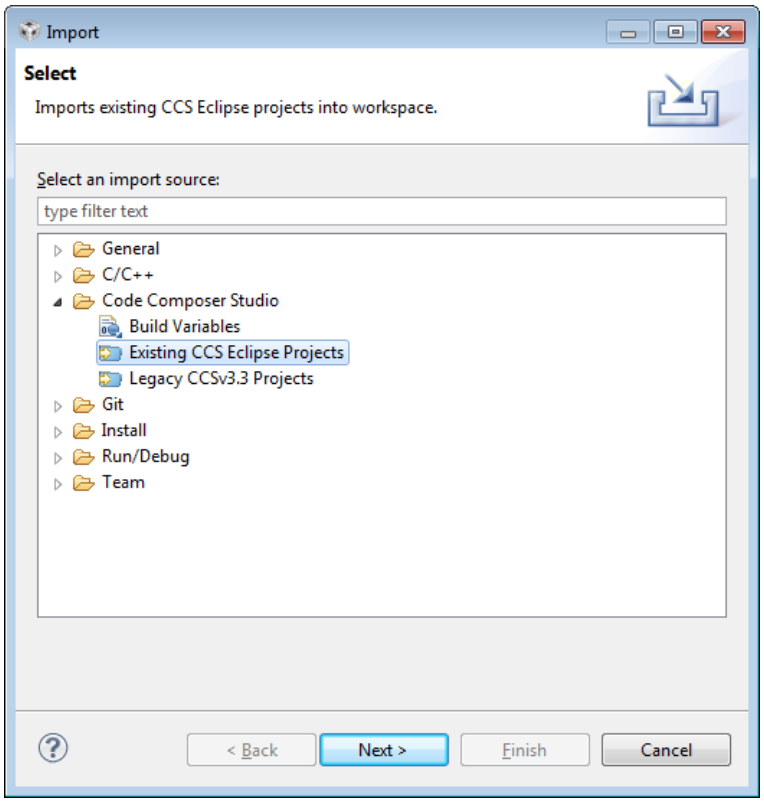


Figure 2.5: CCS Import Options

Select next and browse to the IQmathLib installation directory. The example projects for all devices will be listed and can be imported.

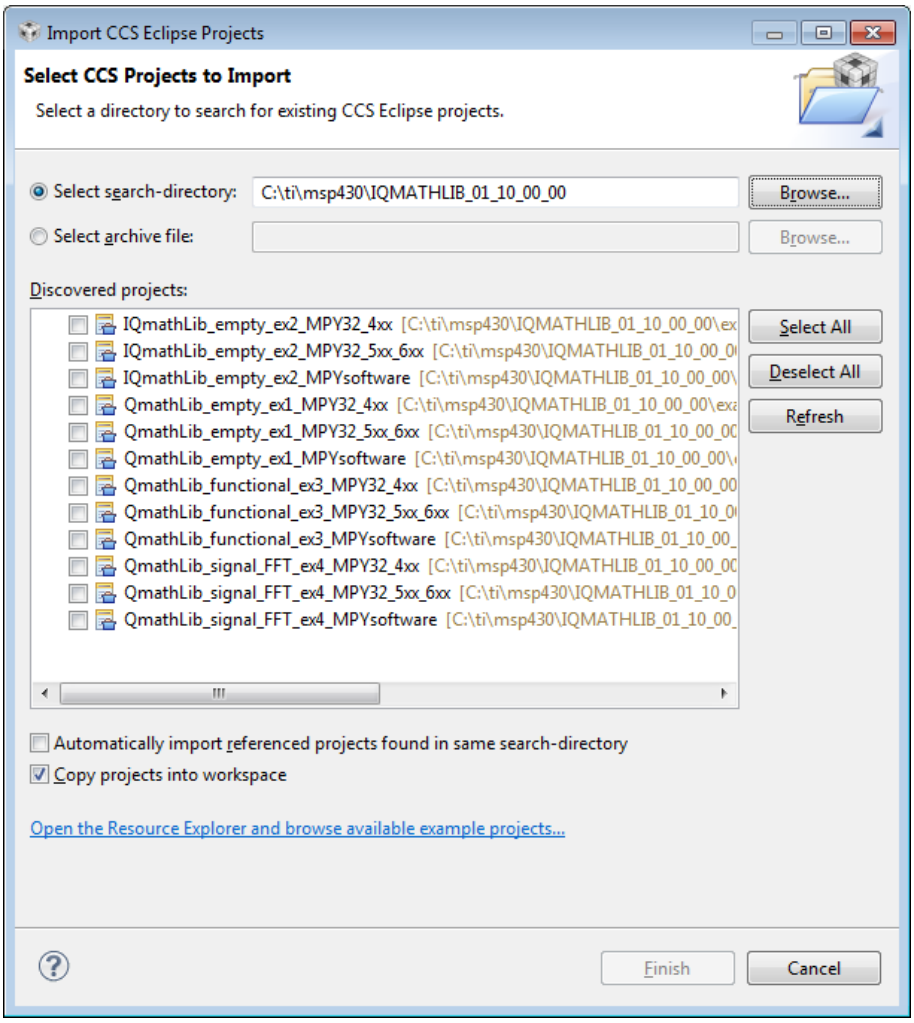


Figure 2.6: CCS Import Projects Window

## 2.8 Function Groups

The Qmath and IQmath routines are organized into five groups:

- Format conversion functions - methods to convert numbers to and from the various formats.
- Arithmetic functions - methods to perform basic arithmetic (addition, subtraction, multiplication, division).
- Trigonometric functions - methods to perform trigonometric functions (sin, cos, atan, and so on).
- Mathematical functions - methods to perform advanced arithmetic (square root, $e^x$, and so on).
- Miscellaneous - miscellaneous methods (saturation and absolute value).

In the chapters that follow, the methods in each of these groups are covered in detail.

# 3 Qmath Functions



## 3.1 Qmath Introduction

The Qmath library provides 16-bit fixed point math functions that have been optimized for the 16-bit MSP430 architecture. The library has been optimized to make efficient use of resources for all MSP430 devices. Execution times, code size and constant data tables are kept to a minimum for each function.

The Qmath library takes advantage of the MPY32 multiplier peripheral when it is available. If the device does not have the MPY32 peripheral then the CPU is used to perform a software multiply. For this reason some functions will utilize larger constant data tables to reduce the number of multiplies required to compute the result. Many of these tables are shared between functions and will only need to be included into the applications constant memory once.

The majority of applications will only require 16-bit accuracy. If greater accuracy is required for calculation see the IQmath chapter for a list of equivalent 32-bit functions.

# 3.2 Qmath Format Conversion Functions

The format conversion functions provide a way to convert numbers to and from various Q formats. There are functions to convert Q numbers to and from single-precision floating-point numbers, to and from integers, to and from strings, to and from various Q formats, and to extract the integer and fractional portion of a Q number. The following table summarizes the format conversion functions:

| Function Name | Q Format | Input Format | Output Format |
|---|---|---|---|
| _atoQN | 1-15 | char * | QN |
| _QN | 1-15 | float | QN |
| _QNfrac | 1-15 | QN | QN |
| _QNint | 1-15 | QN | short |
| _QNtoa | 1-15 | QN | char * |
| _QNtoF | 1-15 | QN | float |
| _QNtoQ | 1-15 | QN | GLOBAL_Q |
| _QtoQN | 1-15 | GLOBAL_Q | QN |

## 3.2.1 _atoQN

Converts a string to a Q number.

**Prototype:**

```
_qN
_atoQN(const char *A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_atoQ(const char *A)
```

for the global Q format

**Parameters:**

***A*** is the string to be converted.

**Description:**

This function converts a string into a Q number. The input string may contain (in order) an optional sign and a string of digits optionally containing a decimal point. A unrecognized character ends the string and returns zero. If the input string converts to a number greater than the minimum or maximum values for the given Q format, the return value is limited to the minimum or maximum value.

**Returns:**

Returns the Q number corresponding to the input string.

## 3.2.2 _QN

Converts a floating-point constant or variable into a Q number.

**Prototype:**

```
_qN
_QN(float A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Q(float A)
```

for the global Q format

**Parameters:**

***A*** is the floating-point variable or constant to be converted.

**Description:**

This function converts a floating-point constant or variable into the equivalent Q number. If the input value is greater than the minimum or maximum values for the given Q format, the return value wraps around and produces inaccurate results.

**Returns:**

Returns the Q number corresponding to the floating-point variable or constant.

## 3.2.3 _QNfrac

Returns the fractional portion of a Q number.

**Prototype:**

```
_qN
_QNfrac(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qfrac(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input number in Q format.

**Description:**

This function returns the fractional portion of a Q number as a Q number.

**Returns:**

Returns the fractional portion of the input Q number.

## 3.2.4 _QNint

Returns the integer portion of a Q number.

**Prototype:**

```
long
_QNint(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
long
_Qint(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input number in Q format.

**Description:**

This function returns the integer portion of a Q number.

**Returns:**

Returns the integer portion of the input Q number.

## 3.2.5 _QNtoa

Converts a Q number to a string.

**Prototype:**

```
int
_QNtoa(char *A,
        const char *B,
        _qN C)
```

for a specific Q format (1 <= N <= 15)

- or -

```
int
_Qtoa(char *A,
       const char *B,
       _q C)
```

for the global Q format

**Parameters:**

***A*** is a pointer to the buffer to store the converted Q number.

***B*** is the format string specifying how to convert the Q number. Must be of the form “%xx.yyf” with xx and yy at most 2 characters in length.

***C*** is the Q number to convert.

**Description:**

This function converts the Q number to a string, using the specified format.

**Example:**

```
_Qtoa(buffer, "%2.4f", qInput)
```

**Returns:**
Returns 0 if there is no error, 1 if the width is too small to hold the integer characters, and 2 if an illegal format was specified.

## 3.2.6 _QNtoF

Converts a Q number to a single-precision floating-point number.

**Prototype:**

```
float
_QNtoF(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
float
_QtoF(_q A)
```

for the global Q format

**Parameters:**
***A*** is the Q number to be converted.

**Description:**
This function converts a Q number into a single-precision floating-point number.

**Returns:**
Returns the single-precision floating-point number corresponding to the input Q number.

## 3.2.7 _QNtoQ

Converts a Q number in QN format to the global Q format.

**Prototype:**

```
_q
_QNtoQ(_qN A)
```

for a specific Q format (1 <= N <= 15)

**Parameters:**
***A*** is Q number to be converted.

**Description:**
This function converts a Q number in the specified Q format to a Q number in the global Q format.

**Returns:**
Returns the Q number converted into the global Q format.

## 3.2.8 _QtoQN

Converts a Q number in the global Q format to the QN format.

**Prototype:**

```
_qN
_QtoQN(_q A)
```

for a specific Q format (1 <= N <= 15)

**Parameters:**

***A*** is the Q number to be converted.

**Description:**

This function converts a Q number in the global Q format to a Q number in the specified Q format. be limited to the minimum or maximum value.

**Returns:**

Returns the Q number converted to the specified Q format.

# 3.3 Qmath Arithmetic Functions

The arithmetic functions provide basic arithmetic (addition, subtraction, multiplication, division) of Q numbers. No special functions are required for addition or subtraction; Q numbers can simply be added and subtracted using the underlying C addition and subtraction operators. Multiplication and division require special treatment in order to maintain the Q number of the result. The following table summarizes the arithmetic functions:

| Function Name | Q Format | Input Format | Output Format |
|---|---|---|---|
| _Qdiv2 | 1-15 | QN | QN |
| _Qdiv4 | 1-15 | QN | QN |
| _Qdiv8 | 1-15 | QN | QN |
| _Qdiv16 | 1-15 | QN | QN |
| _Qdiv32 | 1-15 | QN | QN |
| _Qdiv64 | 1-15 | QN | QN |
| _Qmpy2 | 1-15 | QN | QN |
| _Qmpy4 | 1-15 | QN | QN |
| _Qmpy8 | 1-15 | QN | QN |
| _Qmpy16 | 1-15 | QN | QN |
| _Qmpy32 | 1-15 | QN | QN |
| _Qmpy64 | 1-15 | QN | QN |
| _QNdiv | 1-15 | QN/QN | QN |
| _QNmpy | 1-15 | QN*QN | QN |
| _QNmpyI16 | 1-15 | QN*short | QN |
| _QNmpyI16frac | 1-15 | QN*short | QN |
| _QNmpyI16int | 1-15 | QN*short | short |
| _QNmpyQX | 1-15 | QN*QN | QN |
| _QNrmpy | 1-15 | QN*QN | QN |
| _QNrsmpy | 1-15 | QN*QN | QN |

## 3.3.1 _Qdiv2

Divides a Q number by two.

**Prototype:**

```
_qN
_Qdiv2(_qN A)
```

**Parameters:**

***A*** is the number to be divided, in Q format.

**Description:**

This function divides a Q number by two. This will work for any Q format.

**Returns:**

Returns the number divided by two.

## 3.3.2 _Qdiv4

Divides a Q number by four.

**Prototype:**

```
_qN
_Qdiv4(_qN A)
```

**Parameters:**

***A*** is the number to be divided, in Q format.

**Description:**

This function divides a Q number by four. This will work for any Q format.

**Returns:**

Returns the number divided by four.

### 3.3.3 _Qdiv8

Divides a Q number by eight.

**Prototype:**

```
_qN
_Qdiv8(_qN A)
```

**Parameters:**

***A*** is the number to be divided, in Q format.

**Description:**

This function divides a Q number by eight. This will work for any Q format.

**Returns:**

Returns the number divided by eight.

### 3.3.4 _Qdiv16

Divides a Q number by sixteen.

**Prototype:**

```
_qN
_Qdiv16(_qN A)
```

**Parameters:**

***A*** is the number to be divided, in Q format.

**Description:**

This function divides a Q number by sixteen. This will work for any Q format.

**Returns:**

Returns the number divided by sixteen.

### 3.3.5 _Qdiv32

Divides a Q number by thirty two.

**Prototype:**

```
_qN
_Qdiv32(_qN A)
```

**Parameters:**

***A*** is the number to be divided, in Q format.

**Description:**

This function divides a Q number by thirty two. This will work for any Q format.

**Returns:**

Returns the number divided by thirty two.

### 3.3.6 _Qdiv64

Divides a Q number by sixty four.

**Prototype:**

```
_qN
_Qdiv64(_qN A)
```

**Parameters:**

***A*** is the number to be divided, in Q format.

**Description:**

This function divides a Q number by sixty four. This will work for any Q format.

**Returns:**

Returns the number divided by sixty four.

### 3.3.7 _Qmpy2

Multiplies a Q number by two.

**Prototype:**

```
_qN
_Qmpy2(_qN A)
```

**Parameters:**

***A*** is the number to be multiplied, in Q format.

**Description:**

This function multiplies a Q number by two. This will work for any Q format.

**Returns:**

Returns the number multiplied by two.

### 3.3.8 _Qmpy4

Multiplies a Q number by four.

**Prototype:**

```
_qN
_Qmpy4(_qN A)
```

**Parameters:**

***A*** is the number to be multiplied, in Q format.

**Description:**

This function multiplies a Q number by four. This will work for any Q format.

**Returns:**

Returns the number multiplied by four.

### 3.3.9 _Qmpy8

Multiplies a Q number by eight.

**Prototype:**

```
_qN
_Qmpy8(_qN A)
```

**Parameters:**

***A*** is the number to be multiplied, in Q format.

**Description:**

This function multiplies a Q number by eight. This will work for any Q format.

**Returns:**

Returns the number multiplied by eight.

### 3.3.10 _Qmpy16

Multiplies a Q number by sixteen.

**Prototype:**

```
_qN
_Qmpy16(_qN A)
```

**Parameters:**

***A*** is the number to be multiplied, in Q format.

**Description:**

This function multiplies a Q number by sixteen. This will work for any Q format.

**Returns:**

Returns the number multiplied by sixteen.

### 3.3.11 _Qmpy32

Multiplies a Q number by thirty two.

**Prototype:**

```
_qN
_Qmpy32(_qN A)
```

**Parameters:**

***A*** is the number to be multiplied, in Q format.

**Description:**

This function multiplies a Q number by thirty two. This will work for any Q format.

**Returns:**

Returns the number multiplied by thirty two.

## 3.3.12 _Qmpy64

Multiplies a Q number by sixty four.

**Prototype:**

```
_qN
_Qmpy64(_qN A)
```

**Parameters:**

***A*** is the number to be multiplied, in Q format.

**Description:**

This function multiplies a Q number by sixty four. This will work for any Q format.

**Returns:**

Returns the number multiplied by sixty four.

## 3.3.13 _QNdiv

Divides two Q numbers.

**Prototype:**

```
_qN
_QNdiv(_qN A,
       _qN B)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qdiv(_q A,
      _q B)
```

for the global Q format

**Parameters:**

***A*** is the numerator, in Q format.

***B*** is the denominator, in Q format.

**Description:**
This function divides two Q numbers, returning the quotient in Q format. The result is saturated if it exceeds the capacity of the Q format, and division by zero always results in positive saturation (regardless of the sign of A).

**Returns:**
Returns the quotient in Q format.

## 3.3.14 _QNmpy

Multiplies two Q numbers.

**Prototype:**

```
_qN
_QNmpy(_qN A,
       _qN B)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qmpy(_q A,
      _q B)
```

for the global Q format

**Parameters:**
***A*** is the first number, in Q format.
***B*** is the second number, in Q format.

**Description:**
This function multiplies two Q numbers, returning the product in Q format. The result is neither rounded nor saturated, so if the product is greater than the minimum or maximum values for the given Q format, the return value wraps around and produces inaccurate results.

**Returns:**
Returns the product in Q format.

## 3.3.15 _QNmpyI16

Multiplies a Q number by an integer.

**Prototype:**

```
_qN
_QNmpyI16(_qN A,
          long B)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_QmpyI16(_q A,
         long B)
```

for the global Q format

**Parameters:**

***A*** is the first number, in Q format.

***B*** is the second number, in integer format.

**Description:**

This function multiplies a Q number by an integer, returning the product in Q format. The result is not saturated, so if the product is greater than the minimum or maximum values for the given Q format, the return value wraps around and produces inaccurate results.

**Returns:**

Returns the product in Q format.

## 3.3.16 _QNmpyI16frac

Multiplies a Q number by an integer, returning the fractional portion of the product.

**Prototype:**

```
_qN
_QNmpyI16frac(_qN A,
              long B)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_QmpyI16frac(_q A,
             long B)
```

for the global Q format

**Parameters:**

***A*** is the first number, in Q format.

***B*** is the second number, in integer format.

**Description:**

This function multiplies a Q number by an integer, returning the fractional portion of the product in Q format.

**Returns:**

Returns the fractional portion of the product in Q format.

## 3.3.17 _QNmpyI16int

Multiplies a Q number by an integer, returning the integer portion of the result.

**Prototype:**

```
long
_QNmpyI16int(_qN A,
             long B)
```

for a specific Q format (1 <= N <= 15)

- or -

```
long
_QmpyI16int(_q A,
            long B)
```

for the global Q format

**Parameters:**

***A*** is the first number, in Q format.

***B*** is the second number, in integer format.

**Description:**

This function multiplies a Q number by an integer, returning the integer portion of the product. The result is saturated, so if the integer portion of the product is greater than the minimum or maximum values for an integer, the result will be saturated to the minimum or maximum value.

**Returns:**

Returns the product in Q format.

## 3.3.18 _QNmpyQX

Multiplies two Q numbers.

**Prototype:**

```
_qN
_QNmpyQX(_qN A,
         long QA,
         _qN B,
         long QB)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_QmpyQX(_q A,
        long QA,
        _q B,
        long QB,)
```

for the global Q format

**Parameters:**

***A*** is the first number, in Q format.

***QA*** is the Q format for the first number.

***B*** is the second number, in Q format.

***QB*** is the Q format for the second number.

**Description:**

This function multiplies two Q numbers in different Q formats, returning the product in a third Q format. The result is neither rounded nor saturated, so if the product is greater than the minimum or maximum values for the given output Q format, the return value will wrap around and produce inaccurate results.

**Returns:**
Returns the product in Q format.

## 3.3.19 _QNrmpy

Multiplies two Q numbers, with rounding.

**Prototype:**

```
_qN
_QNrmpy(_qN A,
        _qN B)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qrmpy(_q A,
       _q B)
```

for the global Q format

**Parameters:**
***A*** is the first number, in Q format.
***B*** is the second number, in Q format.

**Description:**
This function multiplies two Q numbers, returning the product in Q format. The result is rounded but not saturated, so if the product is greater than the minimum or maximum values for the given Q format, the return value wraps around and produces inaccurate results.

**Returns:**
Returns the product in Q format.

## 3.3.20 _QNrsmpy

Multiplies two Q numbers, with rounding and saturation.

**Prototype:**

```
_qN
_QNrsmpy(_qN A,
         _qN B)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qrsmpy(_q A,
        _q B)
```

for the global Q format

**Parameters:**
***A*** is the first number, in Q format.

***B*** is the second number, in Q format.

**Description:**

This function multiplies two Q numbers, returning the product in Q format. The result is rounded and saturated, so if the product is greater than the minimum or maximum values for the given Q format, the return value is saturated to the minimum or maximum value for the given Q format (as appropriate).

**Returns:**

Returns the product in Q format.

# 3.4 Qmath Trigonometric Functions

The trigonometric functions compute a variety of the trigonometric functions for Q numbers. Functions are provided that take the traditional radians inputs (or produce the traditional radians output for the inverse functions), as well as a cycles per unit format where the range [0, 1) is mapped onto the circle (in other words, 0.0 is 0 radians, 0.25 is $\pi/2$ radians, 0.5 is $\pi$ radians, 0.75 is $3\pi/2$ radians, and 1.0 is $2\pi$ radians). The following table summarizes the trigonometric functions.

| Function Name | Q Format | Input Format | Output Format |
|---|---|---|---|
| _QNacos | 1-14 | QN | QN |
| _QNasin | 1-14 | QN | QN |
| _QNatan | 1-15 | QN | QN |
| _QNatan2 | 1-15 | QN,QN | QN |
| _QNatan2PU | 1-15 | QN,QN | QN |
| _QNcos | 1-15 | QN | QN |
| _QNcosPU | 1-15 | QN | QN |
| _QNsin | 1-15 | QN | QN |
| _QNsinPU | 1-15 | QN | QN |

## 3.4.1 _QNacos

Computes the inverse cosine of the input value.

**Prototype:**

```
_qN
_QNacos(_qN A)
```

for a specific Q format (1 <= N <= 14)

- or -

```
_q
_Qacos(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input value in Q format.

**Description:**

This function computes the inverse cosine of the input value.

**Note:**

This function is not available for Q15 format.

**Returns:**

The inverse cosine of the input value, in radians.

## 3.4.2 _QNasin

Computes the inverse sine of the input value.

**Prototype:**

```
_qN
_QNasin(_qN A)
```

for a specific Q format (1 <= N <= 14)

- or -

```
_q
_Qasin(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input value in Q format.

**Description:**

This function computes the inverse sine of the input value.

**Note:**

This function is not available for Q15 format.

**Returns:**

The inverse sine of the input value, in radians.

### 3.4.3 _QNatan

Computes the inverse tangent of the input value.

**Prototype:**

```
_qN
_QNatan(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qatan(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input value in Q format.

**Description:**

This function computes the inverse tangent of the input value.

**Returns:**

The inverse tangent of the input value, in radians.

### 3.4.4 _QNatan2

Computes the inverse four-quadrant tangent of the input point.

**Prototype:**

```
_qN
_QNatan2(_qN A,
         _qN B)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qatan2(_q A,
        _q B)
```

for the global Q format

**Parameters:**

***A*** is the Y coordinate input value in Q format.

***B*** is the X coordinate input value in Q format.

**Description:**

This function computes the inverse four-quadrant tangent of the input point.

**Returns:**

The inverse four-quadrant tangent of the input point, in radians.

## 3.4.5 _QNatan2PU

Computes the inverse four-quadrant tangent of the input point, returning the result in cycles per unit.

**Prototype:**

```
_qN
_QNatan2PU(_qN A,
           _qN B)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qatan2PU(_q A,
          _q B)
```

for the global Q format

**Parameters:**

***A*** is the X coordinate input value in Q format.

***B*** is the Y coordinate input value in Q format.

**Description:**

This function computes the inverse four-quadrant tangent of the input point, returning the result in cycles per unit.

**Returns:**

The inverse four-quadrant tangent of the input point, in cycles per unit.

## 3.4.6 _QNcos

Computes the cosine of the input value.

**Prototype:**

```
_qN
_QNcos(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qcos(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input value in radians, in Q format.

**Description:**

This function computes the cosine of the input value.

**Returns:**

The cosine of the input value.

## 3.4.7 _QNcosPU

Computes the cosine of the input value in cycles per unit.

**Prototype:**

```
_qN
_QNcosPU(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_QcosPU(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input value in cycles per unit, in Q format.

**Description:**

This function computes the cosine of the input value.

**Returns:**

The cosine of the input value.

## 3.4.8 _QNsin

Computes the sine of the input value.

**Prototype:**

```
_qN
_QNsin(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qsin(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input value in radians, in Q format.

**Description:**

This function computes the sine of the input value.

**Returns:**

The sine of the input value.

## 3.4.9 _QNsinPU

Computes the sine of the input value in cycles per unit.

**Prototype:**

```
_qN
_QNsinPU(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_QsinPU(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input value in cycles per unit, in Q format.

**Description:**

This function computes the sine of the input value.

**Returns:**

The sine of the input value.

# 3.5 Qmath Mathematical Functions

The mathematical functions compute a variety of advanced mathematical functions for Q numbers. The following table summarizes the mathematical functions:

| Function Name | Q Format | Input Format | Output Format |
|---|---|---|---|
| _QNexp | 1-15 | QN | QN |
| _QNlog | 1-15 | QN | QN |
| _QNsqrt | 1-15 | QN | QN |
| _QNisqrt | 1-15 | QN | QN |
| _QNmag | 1-15 | QN,QN | QN |
| _QNimag | 1-15 | QN,QN | QN |

## 3.5.1 _QNexp

Computes the base-e exponential value of a Q number.

**Prototype:**

```
_qN
_QNexp(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qexp(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input value, in Q format.

**Description:**

This function computes the base-e exponential value of the input, and saturates the result if it exceeds the range of the Q format in use.

**Returns:**

Returns the base-e exponential of the input.

## 3.5.2 _QNlog

Computes the base-e logarithm of a Q number.

**Prototype:**

```
_qN
_QNlog(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qlog(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input value, in Q format.

**Description:**

This function computes the base-e logarithm of the input, and saturates the result if it exceeds the range of the Q format in use.

**Returns:**

Returns the base-e logarithm of the input.

### 3.5.3 _QNsqrt

Computes the square root of a Q number.

**Prototype:**

```
_qN
_QNsqrt(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qsqrt(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input value, in Q format.

**Description:**

This function computes the square root of the input. Negative inputs result in an output of 0.

**Returns:**

Returns the square root of the input.

### 3.5.4 _QNisqrt

Computes the inverse square root of a Q number.

**Prototype:**

```
_qN
_QNisqrt(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qisqrt(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input value, in Q format.

**Description:**

This function computes the inverse square root (1 / sqrt) of the input, and saturates the result if it exceeds the range of the Q format in use. Negative inputs result in an output of 0.

**Returns:**

Returns the inverse square root of the input.

## 3.5.5 _QNmag

Computes the magnitude of a two dimensional vector.

**Prototype:**

```
_qN
_QNmag(_qN A,
       _qN B)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qmag(_q A,
      _q B)
```

for the global Q format

**Parameters:**

***A*** is the first input value, in Q format.

***B*** is the second input value, in Q format.

**Description:**

This function computes the magnitude of a two-dimensional vector provided in Q format. The result is always positive and saturated if it exceeds the range of the Q format in use.

This is functionally equivalent to _QNsqrt(_QNrmpy(A, A) + _QNrmpy(B, B)), but provides better accuracy, speed, and intermediate overflow handling than building this computation from _QNsqrt() and _QNrmpy().

**Returns:**

Returns the magnitude of a two dimensional vector.

## 3.5.6 _QNimag

Computes the inverse magnitude of a two dimensional vector.

**Prototype:**

```
_qN
_QNimag(_qN A,
        _qN B)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qimag(_q A,
       _q B)
```

for the global Q format

**Parameters:**

***A*** is the first input value, in Q format.

***B*** is the second input value, in Q format.

**Description:**

This function computes the inverse magnitude (1 / QNmag) of a two-dimensional vector provided in Q format. The result is always positive and saturated if it exceeds the range of the Q format in use.

**Returns:**

Returns the inverse of the magnitude of a two dimensional vector.

# 3.6 Qmath Miscellaneous Functions

The miscellaneous functions are useful functions that do not otherwise fit elsewhere. The following table summarizes the miscellaneous functions:

| Function Name | Q Format | Input Format | Output Format |
|---|---|---|---|
| _QNabs | 1-15 | QN | QN |
| _QNsat | 1-15 | QN | QN |

## 3.6.1 _QNabs

Finds the absolute value of a Q number.

**Prototype:**

```
_qN
_QNabs(_qN A)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qabs(_q A)
```

for the global Q format

**Parameters:**

***A*** is the input value in Q format.

**Description:**

This function computes the absolute value of the input Q number.

**Returns:**

Returns the absolute value of the input.

## 3.6.2 _QNsat

Satures a Q number.

**Prototype:**

```
_qN
_QNsat(_qN A,
       _qN Pos,
       _qN Neg)
```

for a specific Q format (1 <= N <= 15)

- or -

```
_q
_Qsat(_q A,
      _q Pos,
      _q Neg)
```

for the global Q format

**Parameters:**

***A*** is the input value in Q format.

***Pos*** is the positive limit in Q format.

***Neg*** is the negative limit in Q format.

**Description:**

This function limits the input Q number between the range specified by the positive and negative limits.

**Returns:**

Returns the saturated input value.

# 4 IQmath Functions



## 4.1 IQmath Introduction

The IQmath library provides the same function set as the Qmath library with 32-bit data types and higher accuracy. These functions are provided for when an application requires accuracy comparable or greater than the equivalent floating point math functions. As a result the code size and constant data tables are going to be larger than the Qmath library counterparts.

Execution time is increased however it remains manageable for devices with the MPY32 peripheral. For devices without the MPY32 peripheral the execution time will be an order of magnitude higher than the Qmath counterparts and it is recommended to only use the IQmath functions when greater than 16-bit accuracy is necessary.

When mixing Qmath and IQmath, the IQmath library provides functions for converting between Q and IQ data types to make combining Qmath and IQmath easy and seamless.

# 4.2 IQmath Format Conversion Functions

The format conversion functions provide a way to convert numbers to and from various IQ formats. There are functions to convert IQ numbers to and from single-precision floating-point numbers, to and from integers, to and from strings, to and from 16-bit QN format numbers, to and from various IQ formats, and to extract the integer and fractional portion of an IQ number. The following table summarizes the format conversion functions:

| Function Name | Q Format | Input Format | Output Format |
|---|---|---|---|
| _atoIQN | 1-30 | char * | IQN |
| _IQN | 1-30 | float | IQN |
| _IQNfrac | 1-30 | IQN | IQN |
| _IQNint | 1-30 | IQN | long |
| _IQNtoa | 1-30 | IQN | char * |
| _IQNtoF | 1-30 | IQN | float |
| _IQNtoIQ | 1-30 | IQN | GLOBAL_IQ |
| _IQtoIQN | 1-30 | GLOBAL_IQ | IQN |
| _IQtoQN | 1-15 | GLOBAL_IQ | QN |
| _QNtoIQ | 1-15 | QN | GLOBAL_IQ |

## 4.2.1 _atoIQN

Converts a string to an IQ number.

**Prototype:**

```
_iqN
_atoIQN(const char *A)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_atoIQ(const char *A)
```

for the global IQ format

**Parameters:**

***A*** is the string to be converted.

**Description:**

This function converts a string into an IQ number. The input string may contain (in order) an optional sign and a string of digits optionally containing a decimal point. A unrecognized character ends the string and returns zero. If the input string converts to a number greater than the minimum or maximum values for the given IQ format, the return value is limited to the minimum or maximum value.

**Returns:**

Returns the IQ number corresponding to the input string.

## 4.2.2 _IQN

Converts a floating-point constant or variable into an IQ number.

**Prototype:**

```
_iqN
_IQN(float A)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQ(float A)
```

for the global IQ format

**Parameters:**

***A*** is the floating-point variable or constant to be converted.

**Description:**

This function converts a floating-point constant or variable into the equivalent IQ number. If the input value is greater than the minimum or maximum values for the given IQ format, the return value wraps around and produces inaccurate results.

**Returns:**

Returns the IQ number corresponding to the floating-point variable or constant.

## 4.2.3 _IQNfrac

Returns the fractional portion of an IQ number.

**Prototype:**

```
_iqN
_IQNfrac(_iqN A)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQfrac(_iq A)
```

for the global IQ format

**Parameters:**

***A*** is the input number in IQ format.

**Description:**

This function returns the fractional portion of an IQ number as an IQ number.

**Returns:**

Returns the fractional portion of the input IQ number.

### 4.2.4 _IQNint

Returns the integer portion of an IQ number.

**Prototype:**

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

for a specific IQ format (1 <= N <= 30)

- or -

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

for the global IQ format

**Parameters:**

***A*** is the input number in IQ format.

**Description:**

This function returns the integer portion of an IQ number.

**Returns:**

Returns the integer portion of the input IQ number.

### 4.2.5 _IQNtoa

Converts an IQ number to a string.

**Prototype:**

```
int
_IQNtoa(char *A,
        const char *B,
        _iqN C)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
int
_IQtoa(char *A,
       const char *B,
       _iq C)
```

for the global IQ format

**Parameters:**

***A*** is a pointer to the buffer to store the converted IQ number.

***B*** is the format string specifying how to convert the IQ number. Must be of the form “%xx.yyf” with xx and yy at most 2 characters in length.

***C*** is the IQ number to convert.

**Description:**

This function converts the IQ number to a string, using the specified format.

**Example:**

```
_IQtoa(buffer, "%4.8f", iqInput)
```

**Returns:**

Returns 0 if there is no error, 1 if the width is too small to hold the integer characters, and 2 if an illegal format was specified.

## 4.2.6 _IQNtoF

Converts an IQ number to a single-precision floating-point number.

**Prototype:**

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

for a specific IQ format (1 <= N <= 30)

- or -

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

for the global IQ format

**Parameters:**

***A*** is the IQ number to be converted.

**Description:**

This function converts an IQ number into a single-precision floating-point number. Since single-precision floating-point values have only 24 bits of mantissa, 8 bits of accuracy will be lost via this conversion.

**Returns:**

Returns the single-precision floating-point number corresponding to the input IQ number.

## 4.2.7 _IQNtoIQ

Converts an IQ number in IQN format to the global IQ format.

**Prototype:**

```
_iq
_IQNtoIQ(_iqN A)
```

for a specific IQ format (1 <= N <= 30)

**Parameters:**

***A*** is IQ number to be converted.

**Description:**

This function converts an IQ number in the specified IQ format to an IQ number in the global IQ format.

**Returns:**

Returns the IQ number converted into the global IQ format.

### 4.2.8 _IQtoIQN

Converts an IQ number in the global IQ format to the IQN format.

**Prototype:**

```
_iqN
_IQtoIQN(_iq A)
```

for a specific IQ format (1 <= N <= 30)

**Parameters:**

***A*** is the IQ number to be converted.

**Description:**

This function converts an IQ number in the global IQ format to an IQ number in the specified IQ format. be limited to the minimum or maximum value.

**Returns:**

Returns the IQ number converted to the specified IQ format.

### 4.2.9 _IQtoQN

Converts an IQ number to a 16-bit number in QN format.

**Prototype:**

```
short
_IQtoQN(_iq A)
```

for a specific Q format (1 <= N <= 15)

**Parameters:**

***A*** is the IQ number to be converted.

**Description:**

This function converts an IQ number in the global IQ format to a 16-bit number in QN format.

**Returns:**

Returns the QN number corresponding to the input IQ number.

### 4.2.10 _QNtoIQ

Converts a 16-bit QN number to an IQ number.

**Prototype:**

```
_iq
_QNtoIQ(short A)
```

for a specific Q format (1 <= N <= 15)

**Parameters:**

***A*** is the QN number to be converted.

**Description:**

This function converts a 16-bit QN number to an IQ number in the global IQ format.

**Returns:**

Returns the IQ number corresponding to the input QN number.

# 4.3 IQmath Arithmetic Functions

The arithmetic functions provide basic arithmetic (addition, subtraction, multiplication, division) of IQ numbers. No special functions are required for addition or subtraction; IQ numbers can simply be added and subtracted using the underlying C addition and subtraction operators. Multiplication and division require special treatment in order to maintain the IQ number of the result. The following table summarizes the arithmetic functions:

| Function Name | Q Format | Input Format | Output Format |
|---|---|---|---|
| _IQdiv2 | 1-30 | IQN | IQN |
| _IQdiv4 | 1-30 | IQN | IQN |
| _IQdiv8 | 1-30 | IQN | IQN |
| _IQdiv16 | 1-30 | IQN | IQN |
| _IQdiv32 | 1-30 | IQN | IQN |
| _IQdiv64 | 1-30 | IQN | IQN |
| _IQmpy2 | 1-30 | IQN | IQN |
| _IQmpy4 | 1-30 | IQN | IQN |
| _IQmpy8 | 1-30 | IQN | IQN |
| _IQmpy16 | 1-30 | IQN | IQN |
| _IQmpy32 | 1-30 | IQN | IQN |
| _IQmpy64 | 1-30 | IQN | IQN |
| _IQNdiv | 1-30 | IQN/IQN | IQN |
| _IQNmpy | 1-30 | IQN*IQN | IQN |
| _IQNmpyI32 | 1-30 | IQN*long | IQN |
| _IQNmpyI32frac | 1-30 | IQN*long | IQN |
| _IQNmpyI32int | 1-30 | IQN*long | long |
| _IQNmpyIQX | 1-30 | IQN*IQN | IQN |
| _IQNrmpy | 1-30 | IQN*IQN | IQN |
| _IQNrsmpy | 1-30 | IQN*IQN | IQN |

## 4.3.1 _IQdiv2

Divides an IQ number by two.

**Prototype:**

```
_iqN
_IQdiv2(_iqN A)
```

**Parameters:**

***A*** is the number to be divided, in IQ format.

**Description:**

This function divides an IQ number by two. This will work for any IQ format.

**Returns:**

Returns the number divided by two.

## 4.3.2 _IQdiv4

Divides an IQ number by four.

**Prototype:**

```
_iqN
_IQdiv4(_iqN A)
```

**Parameters:**

***A*** is the number to be divided, in IQ format.

**Description:**

This function divides an IQ number by four. This will work for any IQ format.

**Returns:**

Returns the number divided by four.

### 4.3.3 _IQdiv8

Divides an IQ number by eight.

**Prototype:**

```
_iqN
_IQdiv8(_iqN A)
```

**Parameters:**

***A*** is the number to be divided, in IQ format.

**Description:**

This function divides an IQ number by eight. This will work for any IQ format.

**Returns:**

Returns the number divided by eight.

### 4.3.4 _IQdiv16

Divides an IQ number by sixteen.

**Prototype:**

```
_iqN
_IQdiv16(_iqN A)
```

**Parameters:**

***A*** is the number to be divided, in IQ format.

**Description:**

This function divides an IQ number by sixteen. This will work for any IQ format.

**Returns:**

Returns the number divided by sixteen.

### 4.3.5 _IQdiv32

Divides an IQ number by thirty two.

**Prototype:**

```
_iqN
_IQdiv32(_iqN A)
```

**Parameters:**

***A*** is the number to be divided, in IQ format.

**Description:**

This function divides an IQ number by thirty two. This will work for any IQ format.

**Returns:**

Returns the number divided by thirty two.

### 4.3.6 _IQdiv64

Divides an IQ number by sixty four.

**Prototype:**

```
_iqN
_IQdiv64(_iqN A)
```

**Parameters:**

***A*** is the number to be divided, in IQ format.

**Description:**

This function divides an IQ number by sixty four. This will work for any IQ format.

**Returns:**

Returns the number divided by sixty four.

### 4.3.7 _IQmpy2

Multiplies an IQ number by two.

**Prototype:**

```
_iqN
_IQmpy2(_iqN A)
```

**Parameters:**

***A*** is the number to be multiplied, in IQ format.

**Description:**

This function multiplies an IQ number by two. This will work for any IQ format.

**Returns:**

Returns the number multiplied by two.

### 4.3.8 _IQmpy4

Multiplies an IQ number by four.

**Prototype:**

```
_iqN
_IQmpy4(_iqN A)
```

**Parameters:**

***A*** is the number to be multiplied, in IQ format.

**Description:**

This function multiplies an IQ number by four. This will work for any IQ format.

**Returns:**

Returns the number multiplied by four.

### 4.3.9 _IQmpy8

Multiplies an IQ number by eight.

**Prototype:**

```
_iqN
_IQmpy8(_iqN A)
```

**Parameters:**

***A*** is the number to be multiplied, in IQ format.

**Description:**

This function multiplies an IQ number by eight. This will work for any IQ format.

**Returns:**

Returns the number multiplied by eight.

### 4.3.10 _IQmpy16

Multiplies an IQ number by sixteen.

**Prototype:**

```
_iqN
_IQmpy16(_iqN A)
```

**Parameters:**

***A*** is the number to be multiplied, in IQ format.

**Description:**

This function multiplies an IQ number by sixteen. This will work for any IQ format.

**Returns:**

Returns the number multiplied by sixteen.

### 4.3.11 _IQmpy32

Multiplies an IQ number by thirty two.

**Prototype:**

```
_iqN
_IQmpy32(_iqN A)
```

**Parameters:**

***A*** is the number to be multiplied, in IQ format.

**Description:**

This function multiplies an IQ number by thirty two. This will work for any IQ format.

**Returns:**

Returns the number multiplied by thirty two.

## 4.3.12 _IQmpy64

Multiplies an IQ number by sixty four.

**Prototype:**

```
_iqN
_IQmpy64(_iqN A)
```

**Parameters:**

***A*** is the number to be multiplied, in IQ format.

**Description:**

This function multiplies an IQ number by sixty four. This will work for any IQ format.

**Returns:**

Returns the number multiplied by sixty four.

## 4.3.13 _IQNdiv

Divides two IQ numbers.

**Prototype:**

```
_iqN
_IQNdiv(_iqN A,
        _iqN B)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQdiv(_iq A,
       _iq B)
```

for the global IQ format

**Parameters:**

***A*** is the numerator, in IQ format.

***B*** is the denominator, in IQ format.

**Description:**
This function divides two IQ numbers, returning the quotient in IQ format. The result is saturated if it exceeds the capacity of the IQ format, and division by zero always results in positive saturation (regardless of the sign of A).

**Returns:**
Returns the quotient in IQ format.

## 4.3.14 _IQNmpy

Multiplies two IQ numbers.

**Prototype:**

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

for a specific IQ format (1 <= N <= 30)

- or -

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

for the global IQ format

**Parameters:**
***A*** is the first number, in IQ format.
***B*** is the second number, in IQ format.

**Description:**
This function multiplies two IQ numbers, returning the product in IQ format. The result is neither rounded nor saturated, so if the product is greater than the minimum or maximum values for the given IQ format, the return value wraps around and produces inaccurate results.

**Returns:**
Returns the product in IQ format.

## 4.3.15 _IQNmpyI32

Multiplies an IQ number by an integer.

**Prototype:**

```
_iqN
_IQNmpyI32(_iqN A,
           long B)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQmpyI32(_iq A,
          long B)
```

for the global IQ format

**Parameters:**

***A*** is the first number, in IQ format.

***B*** is the second number, in integer format.

**Description:**

This function multiplies an IQ number by an integer, returning the product in IQ format. The result is not saturated, so if the product is greater than the minimum or maximum values for the given IQ format, the return value wraps around and produces inaccurate results.

**Returns:**

Returns the product in IQ format.

## 4.3.16 _IQNmpyI32frac

Multiplies an IQ number by an integer, returning the fractional portion of the product.

**Prototype:**

```
_iqN
_IQNmpyI32frac(_iqN A,
               long B)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQmpyI32frac(_iq A,
              long B)
```

for the global IQ format

**Parameters:**

***A*** is the first number, in IQ format.

***B*** is the second number, in integer format.

**Description:**

This function multiplies an IQ number by an integer, returning the fractional portion of the product in IQ format.

**Returns:**

Returns the fractional portion of the product in IQ format.

## 4.3.17 _IQNmpyI32int

Multiplies an IQ number by an integer, returning the integer portion of the result.

**Prototype:**

```
long
_IQNmpyI32int(_iqN A,
              long B)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
long
_IQmpyI32int(_iq A,
             long B)
```

for the global IQ format

**Parameters:**

***A*** is the first number, in IQ format.

***B*** is the second number, in integer format.

**Description:**

This function multiplies an IQ number by an integer, returning the integer portion of the product. The result is saturated, so if the integer portion of the product is greater than the minimum or maximum values for an integer, the result will be saturated to the minimum or maximum value.

**Returns:**

Returns the product in IQ format.

## 4.3.18 _IQNmpyIQX

Multiplies two IQ numbers.

**Prototype:**

```
_iqN
_IQNmpyIQX(_iqN A,
           long IQA,
           _iqN B,
           long IQB)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQmpyIQX(_iq A,
          long IQA,
          _iq B,
          long IQB,)
```

for the global IQ format

**Parameters:**

***A*** is the first number, in IQ format.

***IQA*** is the IQ format for the first number.

***B*** is the second number, in IQ format.

***IQB*** is the IQ format for the second number.

**Description:**

This function multiplies two IQ numbers in different IQ formats, returning the product in a third IQ format. The result is neither rounded nor saturated, so if the product is greater than the minimum or maximum values for the given output IQ format, the return value will wrap around and produce inaccurate results.

**Returns:**
Returns the product in IQ format.

## 4.3.19 _IQNrmpy

Multiplies two IQ numbers, with rounding.

**Prototype:**

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

for a specific IQ format (1 <= N <= 30)

- or -

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

for the global IQ format

**Parameters:**
***A*** is the first number, in IQ format.
***B*** is the second number, in IQ format.

**Description:**
This function multiplies two IQ numbers, returning the product in IQ format. The result is rounded but not saturated, so if the product is greater than the minimum or maximum values for the given IQ format, the return value wraps around and produces inaccurate results.

**Returns:**
Returns the product in IQ format.

## 4.3.20 _IQNrsmpy

Multiplies two IQ numbers, with rounding and saturation.

**Prototype:**

```
_iqN
_IQNrsmpy(_iqN A,
          _iqN B)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQrsmpy(_iq A,
         _iq B)
```

for the global IQ format

**Parameters:**
***A*** is the first number, in IQ format.

***B*** is the second number, in IQ format.

**Description:**

This function multiplies two IQ numbers, returning the product in IQ format. The result is rounded and saturated, so if the product is greater than the minimum or maximum values for the given IQ format, the return value is saturated to the minimum or maximum value for the given IQ format (as appropriate).

**Returns:**

Returns the product in IQ format.

# 4.4 IQmath Trigonometric Functions

The trigonometric functions compute a variety of the trigonometric functions for IQ numbers. Functions are provided that take the traditional radians inputs (or produce the traditional radians output for the inverse functions), as well as a cycles per unit format where the range [0, 1) is mapped onto the circle (in other words, 0.0 is 0 radians, 0.25 is $\pi/2$ radians, 0.5 is $\pi$ radians, 0.75 is $3\pi/2$ radians, and 1.0 is $2\pi$ radians). The following table summarizes the trigonometric functions.

| Function Name | Q Format | Input Format | Output Format |
|---|---|---|---|
| _IQNacos | 1-29 | IQN | IQN |
| _IQNasin | 1-29 | IQN | IQN |
| _IQNatan | 1-29 | IQN | IQN |
| _IQNatan2 | 1-29 | IQN,IQN | IQN |
| _IQNatan2PU | 1-30 | IQN,IQN | IQN |
| _IQNcos | 1-29 | IQN | IQN |
| _IQNcosPU | 1-30 | IQN | IQN |
| _IQNsin | 1-29 | IQN | IQN |
| _IQNsinPU | 1-30 | IQN | IQN |

## 4.4.1 _IQNacos

Computes the inverse cosine of the input value.

**Prototype:**

```
_iqN
_IQNacos(_iqN A)
```

for a specific IQ format (1 <= N <= 29)

- or -

```
_iq
_IQacos(_iq A)
```

for the global IQ format

**Parameters:**

***A*** is the input value in IQ format.

**Description:**

This function computes the inverse cosine of the input value.

**Note:**

This function is not available for IQ30 format because the full output range ($-\pi$ through $\pi$) cannot be represented in IQ30 format (which ranges from -2 through 2).

**Returns:**

The inverse cosine of the input value, in radians.

## 4.4.2 _IQNasin

Computes the inverse sine of the input value.

**Prototype:**

```
_iqN
_IQNasin(_iqN A)
```

for a specific IQ format (1 <= N <= 29)

- or -

```
_iq
_IQasin(_iq A)
```

for the global IQ format

**Parameters:**

***A*** is the input value in IQ format.

**Description:**

This function computes the inverse sine of the input value.

**Note:**

This function is not available for IQ30 format because the full output range (-$\pi$ through $\pi$) cannot be represented in IQ30 format (which ranges from -2 through 2).

**Returns:**

The inverse sine of the input value, in radians.

## 4.4.3 _IQNatan

Computes the inverse tangent of the input value.

**Prototype:**

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

for a specific IQ format (1 <= N <= 29)

- or -

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

for the global IQ format

**Parameters:**

***A*** is the input value in IQ format.

**Description:**

This function computes the inverse tangent of the input value.

**Note:**

This function is not available for IQ30 format because the full output range (-$\pi$ through $\pi$) cannot be represented in IQ30 format (which ranges from -2 through 2).

**Returns:**
The inverse tangent of the input value, in radians.

## 4.4.4 _IQNatan2

Computes the inverse four-quadrant tangent of the input point.

**Prototype:**

```
_iqN
_IQNatan2(_iqN A,
          _iqN B)
```

for a specific IQ format (1 <= N <= 29)

- or -

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

for the global IQ format

**Parameters:**
***A*** is the Y coordinate input value in IQ format.
***B*** is the X coordinate input value in IQ format.

**Description:**
This function computes the inverse four-quadrant tangent of the input point.

**Note:**
This function is not available for IQ30 format because the full output range (-$\pi$ through $\pi$) cannot be represented in IQ30 format (which ranges from -2 through 2).

**Returns:**
The inverse four-quadrant tangent of the input point, in radians.

## 4.4.5 _IQNatan2PU

Computes the inverse four-quadrant tangent of the input point, returning the result in cycles per unit.

**Prototype:**

```
_iqN
_IQNatan2PU(_iqN A,
            _iqN B)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQatan2PU(_iq A,
           _iq B)
```

for the global IQ format

**Parameters:**

***A*** is the X coordinate input value in IQ format.

***B*** is the Y coordinate input value in IQ format.

**Description:**

This function computes the inverse four-quadrant tangent of the input point, returning the result in cycles per unit.

**Returns:**

The inverse four-quadrant tangent of the input point, in cycles per unit.

## 4.4.6 _IQNcos

Computes the cosine of the input value.

**Prototype:**

```
_iqN
_IQNcos(_iqN A)
```

for a specific IQ format (1 <= N <= 29)

- or -

```
_iq
_IQcos(_iq A)
```

for the global IQ format

**Parameters:**

***A*** is the input value in radians, in IQ format.

**Description:**

This function computes the cosine of the input value.

**Note:**

This function is not available for IQ30 format because the full input range (-$\pi$ through $\pi$) cannot be represented in IQ30 format (which ranges from -2 through 2).

**Returns:**

The cosine of the input value.

## 4.4.7 _IQNcosPU

Computes the cosine of the input value in cycles per unit.

**Prototype:**

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

for a specific IQ format (1 <= N <= 30)

- or -

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

for the global IQ format

**Parameters:**

***A*** is the input value in cycles per unit, in IQ format.

**Description:**

This function computes the cosine of the input value.

**Returns:**

The cosine of the input value.

## 4.4.8 _IQNsin

Computes the sine of the input value.

**Prototype:**

```
_iqN
_IQNsin(_iqN A)
```

for a specific IQ format (1 <= N <= 29)

- or -

```
_iq
_IQsin(_iq A)
```

for the global IQ format

**Parameters:**

***A*** is the input value in radians, in IQ format.

**Description:**

This function computes the sine of the input value.

**Note:**

This function is not available for IQ30 format because the full input range (-$\pi$ through $\pi$) cannot be represented in IQ30 format (which ranges from -2 through 2).

**Returns:**

The sine of the input value.

## 4.4.9 _IQNsinPU

Computes the sine of the input value in cycles per unit.

**Prototype:**

```
_iqN
_IQNsinPU(_iqN A)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQsinPU(_iq A)
```

for the global IQ format

**Parameters:**

***A*** is the input value in cycles per unit, in IQ format.

**Description:**

This function computes the sine of the input value.

**Returns:**

The sine of the input value.

# 4.5 IQmath Mathematical Functions

The mathematical functions compute a variety of advanced mathematical functions for IQ numbers. The following table summarizes the mathematical functions:

| Function Name | Q Format | Input Format | Output Format |
|---|---|---|---|
| _IQNexp | 1-30 | IQN | IQN |
| _IQNlog | 1-30 | IQN | IQN |
| _IQNsqrt | 1-30 | IQN | IQN |
| _IQNisqrt | 1-30 | IQN | IQN |
| _IQNmag | 1-30 | IQN,IQN | IQN |
| _IQNimag | 1-30 | IQN,IQN | IQN |

## 4.5.1 _IQNexp

Computes the base-e exponential value of an IQ number.

**Prototype:**

```
_iqN
_IQNexp(_iqN A)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQexp(_iq A)
```

for the global IQ format

**Parameters:**

***A*** is the input value, in IQ format.

**Description:**

This function computes the base-e exponential value of the input, and saturates the result if it exceeds the range of the IQ format in use.

**Returns:**

Returns the base-e exponential of the input.

## 4.5.2 _IQNlog

Computes the base-e logarithm of an IQ number.

**Prototype:**

```
_iqN
_IQNlog(_iqN A)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQlog(_iq A)
```

for the global IQ format

**Parameters:**

***A*** is the input value, in IQ format.

**Description:**

This function computes the base-e logarithm of the input, and saturates the result if it exceeds the range of the IQ format in use.

**Returns:**

Returns the base-e logarithm of the input.

### 4.5.3 _IQNsqrt

Computes the square root of an IQ number.

**Prototype:**

```
_iqN
_IQNsqrt(_iqN A)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQsqrt(_iq A)
```

for the global IQ format

**Parameters:**

***A*** is the input value, in IQ format.

**Description:**

This function computes the square root of the input. Negative inputs result in an output of 0.

**Returns:**

Returns the square root of the input.

### 4.5.4 _IQNisqrt

Computes the inverse square root of an IQ number.

**Prototype:**

```
_iqN
_IQNisqrt(_iqN A)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQisqrt(_iq A)
```

for the global IQ format

**Parameters:**

***A*** is the input value, in IQ format.

**Description:**

This function computes the inverse square root (1 / sqrt) of the input, and saturates the result if it exceeds the range of the IQ format in use. Negative inputs result in an output of 0.

**Returns:**

Returns the inverse square root of the input.

## 4.5.5 _IQNmag

Computes the magnitude of a two dimensional vector.

**Prototype:**

```
_iqN
_IQNmag(_iqN A,
        _iqN B)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQmag(_iq A,
       _iq B)
```

for the global IQ format

**Parameters:**

***A*** is the first input value, in IQ format.

***B*** is the second input value, in IQ format.

**Description:**

This function computes the magnitude of a two-dimensional vector provided in IQ format. The result is always positive and saturated if it exceeds the range of the IQ format in use.

This is functionally equivalent to _IQNsqrt(_IQNrmpy(A, A) + _IQNrmpy(B, B)), but provides better accuracy, speed, and intermediate overflow handling than building this computation from _IQNsqrt() and _IQNrmpy(). For example, _IQ16mag(_IQ16(30000), _IQ16(1000)) correctly returns _IQ16(30016.6...), even though the intermediate value of _IQ16rmpy(_IQ16(30000), _IQ16(1000)) overflows an _iq16.

**Returns:**

Returns the magnitude of a two dimensional vector.

## 4.5.6 _IQNimag

Computes the inverse magnitude of a two dimensional vector.

**Prototype:**

```
_iqN
_IQNimag(_iqN A,
         _iqN B)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQimag(_iq A,
        _iq B)
```

for the global IQ format

**Parameters:**

***A*** is the first input value, in IQ format.

***B*** is the second input value, in IQ format.

**Description:**

This function computes the inverse magnitude (1 / IQNmag) of a two-dimensional vector provided in IQ format. The result is always positive and saturated if it exceeds the range of the IQ format in use.

**Returns:**

Returns the inverse of the magnitude of a two dimensional vector.

# 4.6 IQmath Miscellaneous Functions

The miscellaneous functions are useful functions that do not otherwise fit elsewhere. The following table summarizes the miscellaneous functions:

| Function Name | Q Format | Input Format | Output Format |
|---|---|---|---|
| _IQNabs | 1-30 | IQN | IQN |
| _IQNsat | 1-30 | IQN | IQN |

## 4.6.1 _IQNabs

Finds the absolute value of an IQ number.

**Prototype:**

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

for a specific IQ format (1 <= N <= 30)

- or -

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

for the global IQ format

**Parameters:**

***A*** is the input value in IQ format.

**Description:**

This function computes the absolute value of the input IQ number.

**Returns:**

Returns the absolute value of the input.

## 4.6.2 _IQNsat

Satures an IQ number.

**Prototype:**

```
_iqN
_IQNsat(_iqN A,
        _iqN Pos,
        _iqN Neg)
```

for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQsat(_iq A,
       _iq Pos,
       _iq Neg)
```

for the global IQ format

**Parameters:**

***A*** is the input value in IQ format.

***Pos*** is the positive limit in IQ format.

***Neg*** is the negative limit in IQ format.

**Description:**

This function limits the input IQ number between the range specified by the positive and negative limits.

**Returns:**

Returns the saturated input value.

# 5 Optimization Guide For Advanced Users

## 5.1 Introduction

This chapter will cover optimizations for advanced users of fixed point math. Often times there are several ways to implement the same fixed point algorithm, all with varying differences in complexity, code size, execution time and energy consumption. It is up to the application programmer to implement the algorithms in the most efficient manner that best suits the application goals.

## 5.2 Advanced Multiplication

It is very rare that an application will use the same fixed point format for all calculations. Usually it is necessary to convert arguments to the same type however there are properties of fixed point multiplication that can be used to avoid these conversions. The fixed point multiplication function can be written as follows using the Q and IQ formats represented in equations 2.1 and 2.3 respectively.

$$(x_i * 2^{-n_1}) * (y_i * 2^{-n_2}) = x_i * y_i * 2^{-(n_1+n_2)} \tag{5.1}$$

The result of the integer multiply will have an integer component with double the precision of the original type ("int32_t" for Q and "int64_t" for IQ) and the implied scale exponents will be a combination of the two. Thus with no further steps the result of Q and IQ multiplication will be in $Q_{(n_1+n_2)}$ or $IQ_{(n_1+n_2)}$ format.

The result must be converted to the desired Q or IQ type by adding a constant $s$ to equation 5.1 to manipulate both the integer result and implied scale.

$$(x_i * 2^{-n_1}) * (y_i * 2^{-n_2}) = x_i * y_i * 2^{-s} * 2^{(s-(n_1+n_2))} \tag{5.2}$$

The real integer component will be solved for by implementing equation 5.3. The resulting implied scale does not need to be implemented since it is only implied, equation 5.4 gives the implied scale of the result and thus the Q or IQ format.

$$x_i * y_i * 2^{-s} \tag{5.3}$$

$$2^{(s-(n_1+n_2))} \tag{5.4}$$

For multiplication of two identical Q or IQ types the scale exponents $n_1$ and $n_2$ will both be equal to $n$, giving a resulting exponent of $2n$. In order to obtain a result in the same Q or IQ format as the arguments the constant $s$ must also be equal to $n$.

$$2^{(s-(n_1+n_2))} = 2^{(n-(n+n))} = 2^{-n} \tag{5.5}$$

For example, the IQ24 multiplication function is implemented below with a scale constant of $24$. It is important to remember that one of the scales is implied and will not actually be solved.

$$(x_i * 2^{-24}) * (y_i * 2^{-24}) = x_i * y_i * 2^{-24} * 2^{(24-(24+24))} = x_i * y_i * 2^{-24} * 2^{-24} \tag{5.6}$$

Thus we can see each Q and IQ multiplication function will implement a constant scale $s$ equal to the Q or IQ type. This can be used to our advantage when mixing Q or IQ types into equation 5.2.

For example an application requires the multiplication of two arguments in IQ20 and IQ27 format and would like the result in IQ24 format. First we must solved equation 5.4 for the desired constant scale $s$ that gives us the result in the correct format.

$$2^{(s-(20+27))} = 2^{-24} \tag{5.7}$$

The result of solving for $s$ is $23$. Instead of implementing a custom multiplication function for this set of arguments we can use the IQ23 multiply functions since it will also implement a scale of $23$. Substituting our arguments into the full equation 5.2 will give the full result.

$$(x_i * 2^{-20}) * (y_i * 2^{-27}) = x_i * y_i * 2^{-23} * 2^{(23-(20+27))} = x_i * y_i * 2^{-23} * 2^{-24} \quad (5.8)$$

To solve the integer component the IQ23 multiply function is used and the implied scale will be $2^{-24}$, or IQ24 format.

The C code for this multiply operation can be written in many ways, three of which are shown below.

```
#define GLOBAL_IQ   24
#include "IQmathLib.h"

int16_t main1(void)
{
    _iq20 X = _IQ20(10);
    _iq27 Y = _IQ27(0.1);
    _iq Z;

    // Z = X * Y
    Z = _IQmpyIQX(X, Q20, Y, Q27);
}

int16_t main2(void)
{
    _iq20 X = _IQ20(10);
    _iq27 Y = _IQ27(0.1);
    _iq Z, Xt, Yt;

    // Scale X and Y to the global format.
    Xt = _IQ20toIQ(X);
    Yt = _IQ27toIQ(Y);

    // Z = X * Y
    Z = _IQmpy(Xt, Yt);
}

int16_t main3(void)
{
    _iq20 X = _IQ20(10);
    _iq27 Y = _IQ27(0.1);
    _iq Z;

    // Z = X * Y
    Z = _IQ23mpy(X, Y);
}
```

The `_IQmpyIQX` function used in `main1` will consume the most cycles and energy of the three implementation. This function correctly calculates the result as `1.0`.

The code in `main2` is more efficient however it requires conversion between IQ and the `GLOBAL_IQ` formats. This method is prone to overflows and loss of precision as the IQ formats must be scaled to match the `GLOBAL_IQ` format before they can be multiplied. In this example argument Y loses four bits of accuracy when it is scale to the global IQ format and the result is calculated as `0.9999996424`. In addition to the loss of accuracy the code produced by the compiler will be larger than necessary and require the use of temporary registers, decreasing overall performance.

The code in `main3` demonstrates the best way to perform this multiplication. Using the method outlined in equation 5.2 that has been solved in equation 5.8, only a single line of code is required. This method will yield the fastest execution time, lowest energy consumption, lowest code size and experience no possibility of intermediate saturation or loss of precision due to scaling to intermediate values. The result is correctly calculated as `1.0` and no precision is lost. Although this is the most efficient method to perform the multiplication, extra care must be taken to make sure the correct multiplication function is used.

## 5.3 Advanced Division

Division operations can be simplified in many of the same ways as multiplication. Similar to equation 5.2, equation 5.9 below gives a the fixed point divide function with a scale constant $s$.

$$\frac{x_i * 2^{-n_1}}{y_i * 2^{-n_2}} = \frac{x_i}{y_i} * 2^s * 2^{(n_2 - n_1 - s)} \tag{5.9}$$

It is important to note that for division the scale is added with a positive exponent for the integer component and a negative exponent for the implied scale. In the same way as multiplication, each Q and IQ multiplication function will implement a constant scale $s$ equal to the Q or IQ type.

For example, an application requires a division with an IQ29 numerator and IQ30 denominator with the result in IQ24 format. For this operation a scale constant of 25 is used by using the IQ25 divide function. The corresponding C code is given below.

$$\frac{x_i * 2^{-29}}{y_i * 2^{-30}} = \frac{x_i}{y_i} * 2^{25} * 2^{(30-29-25)} = \frac{x_i}{y_i} * 2^{25} * 2^{-24} \tag{5.10}$$

```
#include "IQmathLib.h"

extern _iq29 X;
extern _iq30 Y;

int16_t main(void)
{
    // Z = X / Y
    _iq24 Z = _IQ25div(X, Y);
}
```

For a second example, two integers are divided with the result in Q15 format by using the Q15 divide function. This operation is very useful for taking any two arguments of identical format and calculating the ratio in Q15 format.

$$\frac{x_i * 2^{-0}}{y_i * 2^{-0}} = \frac{x_i}{y_i} * 2^{15} * 2^{(0-0-15)} = \frac{x_i}{y_i} * 2^{15} * 2^{-15} \tag{5.11}$$

```
#include "QmathLib.h"

extern int16_t X;
extern int16_t Y;

int16_t main(void)
{
    // Z = X / Y
    _q15 Z = _Q15div(X, Y);
}
```

# 5.4 Inlined Multiplication with the MPY32 Peripheral

Accessing the MPY32 multiplier peripheral directly in-line with the application code can significantly speed up processing time by removing the overhead of function calls, returns and context saving. Each multiply function implemented in the Qmath and IQmath libraries saves context of the multiplier peripheral and disables interrupts to ensure safe operation in either main or interrupts. When adding direct access to the multiplier it is not always necessary to save context of the multiplier or disable interrupts. It is the responsibility of the application programmer to determine if saving multiplier context is necessary based on the usage of the multiplier within interrupts.

The following code snippets show how the multiplier can be used to perform Q15 and IQ31 multiplications with direct access to the peripheral.

```
static inline _q _Q15mpy_inline(_q q15Arg1, _q q15Arg2)
{
    uint16_t ui16Result;
    uint16_t ui16IntState;
    uint16_t ui16MPYState;

    /* Disable interrupts and save multiplier mode. [optional] */
    ui16IntState = __get_interrupt_state();
    __disable_interrupt();
    ui16MPYState = MPY32CTL0;

    /* Set the multiplier to fractional mode. */
    MPY32CTL0 = MPYFRAC;

    /* Perform multiplication and save result. */
    MPYS = q15Arg1;
    OP2 = q15Arg2;
    __delay_cycles(3);  //Delay for the result to be ready
    ui16Result = RESHI;

    /* Restore multiplier mode and interrupts. [optional] */
    MPY32CTL0 = ui16MPYState;
    __set_interrupt_state(ui16IntState);

    return (_q)ui16Result;
}

static inline _iq _IQ31mpy_inline(_iq iq31Arg1, _iq iq31Arg2)
{
    uint32_t ui32Result;
    uint16_t ui16IntState;
    uint16_t ui16MPYState;

    /* Disable interrupts and save multiplier mode. [optional] */
    ui16IntState = __get_interrupt_state();
    __disable_interrupt();
    ui16MPYState = MPY32CTL0;

    /* Set the multiplier to fractional mode. */
    MPY32CTL0 = MPYFRAC;

    /* Perform multiplication and save result. */
    MPYS32L = iq31Arg1;
    MPYS32H = iq31Arg1 >> 16;
    OP2L = iq31Arg2;
    OP2H = iq31Arg2 >> 16;
    __delay_cycles(5);  //Delay for the result to be ready
    ui32Result = RES2;
    ui32Result |= (uint32_t)RES3 << 16;
```

```
    /* Restore multiplier mode and interrupts. [optional] */
    MPY32CTL0 = ui16MPYState;
    __set_interrupt_state(ui16IntState);

    return (_iq)ui32Result;
}
```

For more details about using the MPY32 peripheral and the required delay timings please see the MPY32 chapter in the device Family User Guide.

# 6 Benchmarks



This chapter gives benchmarks of the available Qmath and IQmath functions for each device family. The benchmarks are given with the following considerations:

- The number of execution cycles and program memory usage provided assumes the Q14 or Q15 formats for Qmath functions and the IQ29 or IQ30 format for IQmath functions. Execution cycles may vary for inputs that are not within a normal input range and other Q and IQ formats.
- Program memory usage may vary by a few bytes for other Q and IQ formats.
- Some functions that are implemented as C preprocessor macros do not have benchmarks for execution cycles or code size. These entries will be left empty.
- The number of execution cycles provided in the table includes the call and return and assumes that the library is running from internal flash or FRAM.
- There are cross functional dependencies that may result in additional functions being included into the application. The code size can vary based on application and functions used.
- Some of the constant data tables are shared across functions. As a result the code size may be less than the benchmarks indicate if multiple functions use the same constant data table.
- Accuracy should always be tested and verified within the end application.

# 6.1 MSP430 Software Multiply

These benchmarks have been run using MSP430G2553 with the following libraries:

- libraries/IAR/MPYsoftware/QmathLib_IAR_MPYsoftware_CPU.lib
- libraries/IAR/MPYsoftware/IQmathLib_IAR_MPYsoftware_CPU.lib

## 6.1.1 MSP430 Software Multiply Qmath Benchmarks

| Function | Accuracy (Bits) | Execution Cycles | Code Size | Const Data |
|---|---|---|---|---|
| _atoQN | 16 | - | 194 | 0 |
| _QN | 16 | - | - | 0 |
| _QNfrac | 16 | 14 | 10 | 0 |
| _QNint | 16 | - | - | 0 |
| _QNtoa | 16 | - | 324 | 0 |
| _QNtoF | 16 | 39 | 24 | 0 |
| _Qdiv2 | 16 | 1 | - | 0 |
| _Qdiv4 | 16 | 2 | - | 0 |
| _Qdiv8 | 16 | 3 | - | 0 |
| _Qdiv16 | 16 | 4 | - | 0 |
| _Qdiv32 | 16 | 5 | - | 0 |
| _Qdiv64 | 16 | 6 | - | 0 |
| _Qmpy2 | 16 | 1 | - | 0 |
| _Qmpy4 | 16 | 2 | - | 0 |
| _Qmpy8 | 16 | 3 | - | 0 |
| _Qmpy16 | 16 | 4 | - | 0 |
| _Qmpy32 | 16 | 5 | - | 0 |
| _Qmpy64 | 16 | 6 | - | 0 |
| _QNdiv | 14 | 366 | 162 | 256 |
| _QNmpy | 16 | 178 | 92 | 0 |
| _QNmpyI16 | 16 | - | - | 0 |
| _QNmpyI16frac | 16 | - | - | 0 |
| _QNmpyI16int | 16 | - | - | 0 |
| _QNmpyQX | 16 | - | - | 0 |
| _QNrmpy | 16 | 177 | 114 | 0 |
| _QNrsmpy | 16 | 196 | 166 | 0 |
| _QNacos | 13 | 201 | 106 | 68 |
| _QNasin | 13 | 201 | 106 | 68 |
| _QNatan | 12 | 549 | 120 | 132 |
| _QNatan2 | 12 | 549 | 120 | 132 |
| _QNatan2PU | 14 | 543 | 96 | 132 |
| _QNcos | 14 | 423 | 156 | 140 |
| _QNcosPU | 14 | 587 | 196 | 140 |
| _QNsin | 14 | 435 | 160 | 140 |
| _QNsinPU | 14 | 588 | 198 | 140 |
| _QNexp | 13 | 497 | 100 | 80 |
| _QNlog | 13 | 357 | 128 | 64 |
| _QNsqrt | 14 | 220 | 190 | 192 |
| _QNisqrt | 14 | 515 | 200 | 96 |
| _QNmag | 14 | 706 | 256 | 192 |
| _QNimag | 13 | 954 | 268 | 96 |
| _QNabs | 16 | - | - | 0 |
| _QNsat | 16 | - | - | 0 |

## 6.1.2 MSP430 Software Multiply IQmath Benchmarks

| Function | Accuracy (Bits) | Execution Cycles | Code Size | Const Data |
|---|---|---|---|---|
| _atoIQN | 32 | - | 266 | 0 |
| _IQN | 23 | - | - | 0 |
| _IQNfrac | 32 | 15 | 10 | 0 |
| _IQNint | 32 | - | - | 0 |
| _IQNtoa | 32 | - | 412 | 0 |
| _IQNtoF | 23 | 50 | 116 | 0 |
| _IQNtoIQ | 32 | - | - | 0 |
| _IQtoIQN | 32 | - | - | 0 |
| _IQtoQN | 32 | - | - | 0 |
| _QNtoIQ | 32 | - | - | 0 |
| _IQdiv2 | 32 | 2 | - | 0 |
| _IQdiv4 | 32 | 4 | - | 0 |
| _IQdiv8 | 32 | 6 | - | 0 |
| _IQdiv16 | 32 | 8 | - | 0 |
| _IQdiv32 | 32 | 10 | - | 0 |
| _IQdiv64 | 32 | 12 | - | 0 |
| _IQmpy2 | 32 | 2 | - | 0 |
| _IQmpy4 | 32 | 4 | - | 0 |
| _IQmpy8 | 32 | 6 | - | 0 |
| _IQmpy16 | 32 | 8 | - | 0 |
| _IQmpy32 | 32 | 10 | - | 0 |
| _IQmpy64 | 32 | 12 | - | 0 |
| _IQNdiv | 30 | 2573 | 154 | 65 |
| _IQNmpy | 32 | 374 | 126 | 0 |
| _IQNmpyI32 | 32 | - | - | 0 |
| _IQNmpyI32frac | 32 | - | - | 0 |
| _IQNmpyI32int | 32 | - | - | 0 |
| _IQNmpyIQX | 32 | - | - | 0 |
| _IQNrmpy | 32 | 374 | 156 | 0 |
| _IQNrsmpy | 32 | 413 | 238 | 0 |
| _IQNacos | 26 | 1658 | 216 | 340 |
| _IQNasin | 26 | 1658 | 216 | 340 |
| _IQNatan | 27 | 4113 | 188 | 528 |
| _IQNatan2 | 27 | 4113 | 188 | 528 |
| _IQNatan2PU | 30 | 3706 | 184 | 528 |
| _IQNcos | 27 | 1697 | 326 | 208 |
| _IQNcosPU | 27 | 2078 | 342 | 208 |
| _IQNsin | 27 | 1690 | 330 | 208 |
| _IQNsinPU | 27 | 2081 | 346 | 208 |
| _IQNexp | 30 | 4401 | 268 | 132 |
| _IQNlog | 28 | 6229 | 260 | 60 |
| _IQNsqrt | 31 | 3551 | 368 | 192 |
| _IQNisqrt | 31 | 3217 | 362 | 192 |
| _IQNmag | 30 | 5469 | 484 | 192 |
| _IQNimag | 30 | 5154 | 480 | 192 |
| _IQNabs | 32 | - | - | 0 |
| _IQNsat | 32 | - | - | 0 |

# 6.2 MSP430F4xx Family

These benchmarks have been run using MSP430F4794 with the following libraries:

- libraries/IAR/MPY32/4xx/QmathLib_IAR_MPY32_4xx_CPU.lib
- libraries/IAR/MPY32/4xx/IQmathLib_IAR_MPY32_4xx_CPU.lib

## 6.2.1 MSP430F4xx Family Qmath Benchmarks

| Function | Accuracy (Bits) | Execution Cycles | Code Size | Const Data |
|---|---|---|---|---|
| _atoQN | 16 | - | 216 | 0 |
| _QN | 16 | - | - | 0 |
| _QNfrac | 16 | 14 | 10 | 0 |
| _QNint | 16 | - | - | 0 |
| _QNtoa | 16 | - | 558 | 0 |
| _QNtoF | 16 | 39 | 24 | 0 |
| _Qdiv2 | 16 | 1 | - | 0 |
| _Qdiv4 | 16 | 2 | - | 0 |
| _Qdiv8 | 16 | 3 | - | 0 |
| _Qdiv16 | 16 | 4 | - | 0 |
| _Qdiv32 | 16 | 5 | - | 0 |
| _Qdiv64 | 16 | 6 | - | 0 |
| _Qmpy2 | 16 | 1 | - | 0 |
| _Qmpy4 | 16 | 2 | - | 0 |
| _Qmpy8 | 16 | 3 | - | 0 |
| _Qmpy16 | 16 | 4 | - | 0 |
| _Qmpy32 | 16 | 5 | - | 0 |
| _Qmpy64 | 16 | 6 | - | 0 |
| _QNdiv | 14 | 120 | 196 | 256 |
| _QNmpy | 16 | 52 | 48 | 0 |
| _QNmpyI16 | 16 | - | - | 0 |
| _QNmpyI16frac | 16 | - | - | 0 |
| _QNmpyI16int | 16 | - | - | 0 |
| _QNmpyQX | 16 | - | - | 0 |
| _QNrmpy | 16 | 55 | 54 | 0 |
| _QNrsmpy | 16 | 63 | 90 | 0 |
| _QNacos | 13 | 88 | 138 | 68 |
| _QNasin | 13 | 88 | 138 | 68 |
| _QNatan | 12 | 187 | 150 | 132 |
| _QNatan2 | 12 | 187 | 150 | 132 |
| _QNatan2PU | 14 | 182 | 126 | 132 |
| _QNcos | 14 | 102 | 194 | 140 |
| _QNcosPU | 14 | 137 | 234 | 140 |
| _QNsin | 14 | 114 | 202 | 140 |
| _QNsinPU | 14 | 140 | 238 | 140 |
| _QNexp | 13 | 118 | 142 | 80 |
| _QNlog | 13 | 111 | 168 | 64 |
| _QNsqrt | 14 | 102 | 214 | 192 |
| _QNisqrt | 14 | 134 | 248 | 96 |
| _QNmag | 14 | 163 | 300 | 192 |
| _QNimag | 13 | 184 | 328 | 96 |
| _QNabs | 16 | - | - | 0 |
| _QNsat | 16 | - | - | 0 |

## 6.2.2 MSP430F4xx Family IQmath Benchmarks

| Function | Accuracy (Bits) | Execution Cycles | Code Size | Const Data |
|---|---|---|---|---|
| _atoIQN | 32 | - | 336 | 0 |
| _IQN | 23 | - | - | 0 |
| _IQNfrac | 32 | 15 | 10 | 0 |
| _IQNint | 32 | - | - | 0 |
| _IQNtoa | 32 | - | 706 | 0 |
| _IQNtoF | 23 | 50 | 116 | 0 |
| _IQNtoIQ | 32 | - | - | 0 |
| _IQtoIQN | 32 | - | - | 0 |
| _IQtoQN | 32 | - | - | 0 |
| _QNtoIQ | 32 | - | - | 0 |
| _IQdiv2 | 32 | 2 | - | 0 |
| _IQdiv4 | 32 | 4 | - | 0 |
| _IQdiv8 | 32 | 6 | - | 0 |
| _IQdiv16 | 32 | 8 | - | 0 |
| _IQdiv32 | 32 | 10 | - | 0 |
| _IQdiv64 | 32 | 12 | - | 0 |
| _IQmpy2 | 32 | 2 | - | 0 |
| _IQmpy4 | 32 | 4 | - | 0 |
| _IQmpy8 | 32 | 6 | - | 0 |
| _IQmpy16 | 32 | 8 | - | 0 |
| _IQmpy32 | 32 | 10 | - | 0 |
| _IQmpy64 | 32 | 12 | - | 0 |
| _IQNdiv | 30 | 319 | 150 | 65 |
| _IQNmpy | 32 | 69 | 68 | 0 |
| _IQNmpyI32 | 32 | - | - | 0 |
| _IQNmpyI32frac | 32 | - | - | 0 |
| _IQNmpyI32int | 32 | - | - | 0 |
| _IQNmpyIQX | 32 | - | - | 0 |
| _IQNrmpy | 32 | 73 | 76 | 0 |
| _IQNrsmpy | 32 | 81 | 116 | 0 |
| _IQNacos | 26 | 225 | 300 | 340 |
| _IQNasin | 26 | 225 | 300 | 340 |
| _IQNatan | 27 | 515 | 268 | 528 |
| _IQNatan2 | 27 | 515 | 268 | 528 |
| _IQNatan2PU | 30 | 480 | 248 | 528 |
| _IQNcos | 27 | 260 | 456 | 208 |
| _IQNcosPU | 27 | 286 | 478 | 208 |
| _IQNsin | 27 | 272 | 468 | 208 |
| _IQNsinPU | 27 | 296 | 482 | 208 |
| _IQNexp | 30 | 494 | 320 | 132 |
| _IQNlog | 28 | 629 | 318 | 60 |
| _IQNsqrt | 31 | 382 | 544 | 192 |
| _IQNisqrt | 31 | 364 | 520 | 192 |
| _IQNmag | 30 | 548 | 734 | 192 |
| _IQNimag | 30 | 530 | 710 | 192 |
| _IQNabs | 32 | - | - | 0 |
| _IQNsat | 32 | - | - | 0 |

# 6.3 MSP430F5xx, MSP430F6xx and MSP430FRxx Family

These benchmarks have been run using MSP430F5529 with the following libraries:

- libraries/IAR/MPY32/5xx_6xx/QmathLib_IAR_MPY32_5xx_6xx_CPUX_small_code_small_data.lib
- libraries/IAR/MPY32/5xx_6xx/IQmathLib_IAR_MPY32_5xx_6xx_CPUX_small_code_small_data.lib

## 6.3.1 MSP430F5xx, MSP430F6xx and MSP430FRxx Family Qmath Benchmarks

| Function | Accuracy (Bits) | Execution Cycles | Code Size | Const Data |
|---|---|---|---|---|
| _atoQN | 16 | - | 206 | 0 |
| _QN | 16 | - | - | 0 |
| _QNfrac | 16 | 14 | 10 | 0 |
| _QNint | 16 | - | - | 0 |
| _QNtoa | 16 | - | 532 | 0 |
| _QNtoF | 16 | 39 | 26 | 0 |
| _Qdiv2 | 16 | 1 | - | 0 |
| _Qdiv4 | 16 | 2 | - | 0 |
| _Qdiv8 | 16 | 3 | - | 0 |
| _Qdiv16 | 16 | 4 | - | 0 |
| _Qdiv32 | 16 | 5 | - | 0 |
| _Qdiv64 | 16 | 6 | - | 0 |
| _Qmpy2 | 16 | 1 | - | 0 |
| _Qmpy4 | 16 | 2 | - | 0 |
| _Qmpy8 | 16 | 3 | - | 0 |
| _Qmpy16 | 16 | 4 | - | 0 |
| _Qmpy32 | 16 | 5 | - | 0 |
| _Qmpy64 | 16 | 6 | - | 0 |
| _QNdiv | 14 | 103 | 184 | 256 |
| _QNmpy | 16 | 47 | 48 | 0 |
| _QNmpyI16 | 16 | - | - | 0 |
| _QNmpyI16frac | 16 | - | - | 0 |
| _QNmpyI16int | 16 | - | - | 0 |
| _QNmpyQX | 16 | - | - | 0 |
| _QNrmpy | 16 | 50 | 54 | 0 |
| _QNrsmpy | 16 | 58 | 90 | 0 |
| _QNacos | 13 | 80 | 132 | 68 |
| _QNasin | 13 | 80 | 132 | 68 |
| _QNatan | 12 | 175 | 142 | 132 |
| _QNatan2 | 12 | 175 | 142 | 132 |
| _QNatan2PU | 14 | 169 | 122 | 132 |
| _QNcos | 14 | 91 | 174 | 140 |
| _QNcosPU | 14 | 118 | 210 | 140 |
| _QNsin | 14 | 98 | 180 | 140 |
| _QNsinPU | 14 | 121 | 214 | 140 |
| _QNexp | 13 | 105 | 134 | 80 |
| _QNlog | 13 | 99 | 162 | 64 |
| _QNsqrt | 14 | 95 | 190 | 192 |
| _QNisqrt | 14 | 118 | 220 | 96 |
| _QNmag | 14 | 147 | 280 | 192 |
| _QNimag | 13 | 162 | 302 | 96 |
| _QNabs | 16 | - | - | 0 |
| _QNsat | 16 | - | - | 0 |

### 6.3.2 MSP430F5xx, MSP430F6xx and MSP430FRxx Family IQmath Benchmarks

| Function | Accuracy (Bits) | Execution Cycles | Code Size | Const Data |
|---|---|---|---|---|
| _atoIQN | 32 | - | 312 | 0 |
| _IQN | 23 | - | - | 0 |
| _IQNfrac | 32 | 15 | 10 | 0 |
| _IQNint | 32 | - | - | 0 |
| _IQNtoa | 32 | - | 676 | 0 |
| _IQNtoF | 23 | 50 | 116 | 0 |
| _IQNtoIQ | 32 | - | - | 0 |
| _IQtoIQN | 32 | - | - | 0 |
| _IQtoQN | 32 | - | - | 0 |
| _QNtoIQ | 32 | - | - | 0 |
| _IQdiv2 | 32 | 2 | - | 0 |
| _IQdiv4 | 32 | 4 | - | 0 |
| _IQdiv8 | 32 | 6 | - | 0 |
| _IQdiv16 | 32 | 8 | - | 0 |
| _IQdiv32 | 32 | 10 | - | 0 |
| _IQdiv64 | 32 | 12 | - | 0 |
| _IQmpy2 | 32 | 2 | - | 0 |
| _IQmpy4 | 32 | 4 | - | 0 |
| _IQmpy8 | 32 | 6 | - | 0 |
| _IQmpy16 | 32 | 8 | - | 0 |
| _IQmpy32 | 32 | 10 | - | 0 |
| _IQmpy64 | 32 | 12 | - | 0 |
| _IQNdiv | 30 | 278 | 136 | 65 |
| _IQNmpy | 32 | 62 | 68 | 0 |
| _IQNmpyI32 | 32 | - | - | 0 |
| _IQNmpyI32frac | 32 | - | - | 0 |
| _IQNmpyI32int | 32 | - | - | 0 |
| _IQNmpyIQX | 32 | - | - | 0 |
| _IQNrmpy | 32 | 66 | 76 | 0 |
| _IQNrsmpy | 32 | 74 | 116 | 0 |
| _IQNacos | 26 | 194 | 282 | 340 |
| _IQNasin | 26 | 194 | 282 | 340 |
| _IQNatan | 27 | 445 | 256 | 528 |
| _IQNatan2 | 27 | 445 | 256 | 528 |
| _IQNatan2PU | 30 | 417 | 238 | 528 |
| _IQNcos | 27 | 219 | 438 | 208 |
| _IQNcosPU | 27 | 241 | 460 | 208 |
| _IQNsin | 27 | 230 | 450 | 208 |
| _IQNsinPU | 27 | 250 | 464 | 208 |
| _IQNexp | 30 | 431 | 286 | 132 |
| _IQNlog | 28 | 556 | 294 | 60 |
| _IQNsqrt | 31 | 333 | 522 | 192 |
| _IQNisqrt | 31 | 318 | 498 | 192 |
| _IQNmag | 30 | 480 | 710 | 192 |
| _IQNimag | 30 | 464 | 686 | 192 |
| _IQNabs | 32 | - | - | 0 |
| _IQNsat | 32 | - | - | 0 |

# 6.4 MSP432 Devices

These benchmarks have been run using MSP432P401R with the following libraries at 1.5 MHz and 0 wait-states:

- libraries/IAR/MSP432/QmathLib_IAR_MSP432.lib
- libraries/IAR/MSP432/IQmathLib_IAR_MSP432.lib

## 6.4.1 MSP432 Qmath Benchmarks

| Function | Accuracy (Bits) | Execution Cycles | Code Size | Const Data |
|---|---|---|---|---|
| _atoQN | 16 | - | 200 | 0 |
| _QN | 16 | - | - | 0 |
| _QNfrac | 16 | 8 | 12 | 0 |
| _QNint | 16 | - | - | 0 |
| _QNtoa | 16 | - | 492 | 0 |
| _QNtoF | 16 | 32 | 92 | 0 |
| _Qdiv2 | 16 | 1 | - | 0 |
| _Qdiv4 | 16 | 1 | - | 0 |
| _Qdiv8 | 16 | 1 | - | 0 |
| _Qdiv16 | 16 | 1 | - | 0 |
| _Qdiv32 | 16 | 1 | - | 0 |
| _Qdiv64 | 16 | 1 | - | 0 |
| _Qmpy2 | 16 | 1 | - | 0 |
| _Qmpy4 | 16 | 1 | - | 0 |
| _Qmpy8 | 16 | 1 | - | 0 |
| _Qmpy16 | 16 | 1 | - | 0 |
| _Qmpy32 | 16 | 1 | - | 0 |
| _Qmpy64 | 16 | 1 | - | 0 |
| _QNdiv | 14 | 61 | 154 | 256 |
| _QNmpy | 16 | 10 | 10 | 0 |
| _QNmpyI16 | 16 | - | - | 0 |
| _QNmpyI16frac | 16 | - | - | 0 |
| _QNmpyI16int | 16 | - | - | 0 |
| _QNmpyQX | 16 | - | - | 0 |
| _QNrmpy | 16 | 10 | 14 | 0 |
| _QNrsmpy | 16 | 10 | 16 | 0 |
| _QNacos | 13 | 48 | 96 | 68 |
| _QNasin | 13 | 48 | 96 | 68 |
| _QNatan | 12 | 98 | 128 | 132 |
| _QNatan2 | 12 | 98 | 128 | 132 |
| _QNatan2PU | 14 | 94 | 106 | 132 |
| _QNcos | 14 | 40 | 106 | 140 |
| _QNcosPU | 14 | 60 | 144 | 140 |
| _QNsin | 14 | 46 | 114 | 140 |
| _QNsinPU | 14 | 60 | 144 | 140 |
| _QNexp | 13 | 70 | 98 | 80 |
| _QNlog | 13 | 50 | 128 | 64 |
| _QNsqrt | 14 | 46 | 174 | 192 |
| _QNisqrt | 14 | 56 | 180 | 96 |
| _QNmag | 14 | 58 | 190 | 192 |
| _QNimag | 13 | 50 | 194 | 96 |
| _QNabs | 16 | - | - | 0 |
| _QNsat | 16 | - | - | 0 |

## 6.4.2 MSP432 IQmath Benchmarks

| Function | Accuracy (Bits) | Execution Cycles | Code Size | Const Data |
|---|---|---|---|---|
| _atoIQN | 32 | - | 240 | 0 |
| _IQN | 23 | - | - | 0 |
| _IQNfrac | 32 | 6 | 8 | 0 |
| _IQNint | 32 | - | - | 0 |
| _IQNtoa | 32 | - | 546 | 0 |
| _IQNtoF | 23 | 36 | 104 | 0 |
| _IQNtoIQ | 32 | - | - | 0 |
| _IQtoIQN | 32 | - | - | 0 |
| _IQtoQN | 32 | - | - | 0 |
| _QNtoIQ | 32 | - | - | 0 |
| _IQdiv2 | 32 | 1 | - | 0 |
| _IQdiv4 | 32 | 1 | - | 0 |
| _IQdiv8 | 32 | 1 | - | 0 |
| _IQdiv16 | 32 | 1 | - | 0 |
| _IQdiv32 | 32 | 1 | - | 0 |
| _IQdiv64 | 32 | 1 | - | 0 |
| _IQmpy2 | 32 | 1 | - | 0 |
| _IQmpy4 | 32 | 1 | - | 0 |
| _IQmpy8 | 32 | 1 | - | 0 |
| _IQmpy16 | 32 | 1 | - | 0 |
| _IQmpy32 | 32 | 1 | - | 0 |
| _IQmpy64 | 32 | 1 | - | 0 |
| _IQNdiv | 30 | 117 | 260 | 65 |
| _IQNmpy | 32 | 8 | 12 | 0 |
| _IQNmpyI32 | 32 | - | - | 0 |
| _IQNmpyI32frac | 32 | - | - | 0 |
| _IQNmpyI32int | 32 | - | - | 0 |
| _IQNmpyIQX | 32 | - | - | 0 |
| _IQNrmpy | 32 | 10 | 20 | 0 |
| _IQNrsmpy | 32 | 24 | 60 | 0 |
| _IQNacos | 26 | 92 | 194 | 340 |
| _IQNasin | 26 | 92 | 194 | 340 |
| _IQNatan | 27 | 166 | 184 | 528 |
| _IQNatan2 | 27 | 166 | 248 | 528 |
| _IQNatan2PU | 30 | 156 | 170 | 528 |
| _IQNcos | 27 | 82 | 250 | 208 |
| _IQNcosPU | 27 | 86 | 248 | 208 |
| _IQNsin | 27 | 84 | 256 | 208 |
| _IQNsinPU | 27 | 90 | 258 | 208 |
| _IQNexp | 30 | 184 | 210 | 132 |
| _IQNlog | 28 | 220 | 194 | 60 |
| _IQNsqrt | 31 | 94 | 302 | 192 |
| _IQNisqrt | 31 | 98 | 304 | 192 |
| _IQNmag | 30 | 136 | 406 | 192 |
| _IQNimag | 30 | 132 | 402 | 192 |
| _IQNabs | 32 | - | - | 0 |
| _IQNsat | 32 | - | - | 0 |

# IMPORTANT NOTICE

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

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

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

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

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

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

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

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

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

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

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

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