Texas Instruments

TivaWare™ IQmath Library

USER'S GUIDE

SW-TM4C-IQMATH-UG-2.1.0.12573

Copyright

Texas Instruments
108 Wild Basin, Suite 350
Austin, TX 78746
www.ti.com/tiva-c

Revision Information

This is version 2.1.0.12573 of this document, last updated on February 07, 2014.

Table of Contents

1 Introduction

The Texas Instruments® Tiva™ IQmath Library is a collection of highly optimized and high-precision mathematical functions for C/C++ programmers to seamlessly port a floating-point algorithm into fixed-point code on Tiva devices. These routines are typically used in computationally intensive real-time applications where optimal execution speed and high accuracy is critical. By using the IQmath library, it is possible to achieve execution speeds considerably faster than equivalent code written using floating-point math.

The following tool chains are supported:

- Keil™ RealView® Microcontroller Development Kit
- Mentor Graphics® Sourcery™ CodeBench
- IAR Embedded Workbench®
- Texas Instruments Code Composer Studio™

2 Using The IQmath Library

2.1 IQmath Data Type

The IQmath library uses a 32-bit fixed-point signed number (a "long" in C) 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. C typedefs are provided for the various IQ formats, and these IQmath data types should be used in preference to the underlying "long" 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 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

Type	Bits		Range		Resolution
	Integer	Fractional	Min	Max	
_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

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

2.2 Calling IQmath Functions From C

In order to call an IQmath function from C (or from C++ using the normal C bindings), the IQmath C header file (`IQmath/IQmathLib.h`) must be included. Then, the `_iq` and `_iqN` data types, along with the IQmath functions can be used by the application.

As an example, the following code performs some simple arithmetic in IQ24 format:

```
#include "IQmath/IQmathLib.h"

int
main(void)
{
    _iq24 X, Y, Z;

    X = _IQ24(1.0);
    Y = _IQ24(7.0);

    Z = _IQ24div(X, Y);
}
```

2.3 Calling IQmath Functions From C++

In C++, the `_iq` type becomes the `iq` class, allowing for operator overloading of operations such as multiply and divide. To access the library from C++, the IQmath C++ header file (`IQmath/IQmathCPP.h`) must be included after the IQmath C header file has been included. Then, call the functions using the `iq` and `iqN` classes along with the C++ functions, which have the leading underscore removed and the math operations overloaded. For example:

C Code	C++ Code	Note
`_iq`, `_iqN`	`iq`, `iqN`	IQ data types
`_IQ(A)`, `_IQN(A)`	`IQ(A)`, `IQN(A)`	Convert float to IQ
`_IQdiv(A, B)`	`A / B`	Division
`_IQsqrt(A)`	`IQsqrt(A)`	Square root

As an example, the following code is equivalent to the C example provided above, but using the

C++ functions:

```
#include "IQmath/IQmathLib.h"
#include "IQmath/IQmathCPP.h"

int
main(void)
{
    iq24 X, Y, Z;

    X = IQ24(1.0);
    Y = IQ24(7.0);

    Z = X / Y;
}
```

2.4 Selecting The GLOBAL_Q Format

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

The default `GLOBAL_Q` format is IQ24. This can be easily overridden in one of two ways:

- In the source file, the `GLOBAL_Q` format can be selected prior to including `IQmath/IQmathLib.h`. The following example selects a `GLOBAL_Q` format of IQ8:

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

- In the project file, add a predefined value for `GLOBAL_Q` for the entire project. 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_Q` values, while the second method changes the `GLOBAL_Q` value for the entire application. The method that is most appropriate varies from application to application.

2.5 Converting An IQmath Application To Floating-Point

An IQmath application can be easily converted to use floating-point math instead of IQmath. `MATH_TYPE` selects the type of math to use; it can have one of two values:

- `IQ_MATH` - the default value, which performs all IQmath functions using fixed-point arithmetic in the IQmath library.
- `FLOAT_MATH` - which provides stubs for all the IQmath functions causing the arithmetic to be done in floating-point using the tool chain's C and math library.

By changing the definition of `MATH_TYPE` to `FLOAT_MATH`, all the IQmath calls are replaced by their floating-point equivalents. This change can be easily made in one of two ways:

- In the source file, the `MATH_TYPE` can be selected prior to including `IQmath/IQmathLib.h`. The following example selects floating-point math:

```
//
// Select floating-point math.
//
#define MATH_TYPE               FLOAT_MATH
#include "IQmath/IQmathLib.h"
```

- In the project file, add a predefined value for `MATH_TYPE` for the entire project. The method to add a predefined value varies from tool chain to tool chain.

The first method allows different modules in the application to use different math types, while the second method changes the math type for the entire application. The method that is most appropriate varies from application to application.

2.6 IQmath Function Groups

The IQmath routines are organized into five groups:

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

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

3 Format Conversion Functions

3.1 Introduction

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- and double-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	IQ Format	Execution Cycles	Accuracy (bits)	Program Memory (bytes)	Input Format	Output Format
_atoIQN	1-30	n/a	n/a	304	char *	IQN
_IQN	1-30	n/a	n/a	n/a	float	IQN
_IQNfrac	1-30	8	32 bits	12	IQN	IQN
_IQNint	1-30	1	32 bits	2	IQN	long
_IQNtoa	1-30	n/a	n/a	286	IQN	char *
_IQNtoD	1-30	20	n/a	52	IQN	double
_IQNtoF	1-30	17	n/a	36	IQN	float
_IQNtoIQ	1-30	1	n/a	2	IQN	GLOBAL_Q
_IQtoIQN	1-30	1	n/a	2	GLOBAL_Q	IQN
_IQtoQN	1-15	1	n/a	2	GLOBAL_Q	QN
_QNtoIQ	1-15	1	n/a	2	QN	GLOBAL_Q

- The number of execution cycles and program memory usage provided above assumes IQ24 format. Execution cycles may vary by a few cycles for other IQ formats, and program memory usage may vary by a few bytes for other IQ formats.
- The number of execution cycles provided in the table includes the call and return and assumes that the IQmath library is running from internal flash.
- Accuracy should always be tested and verified within the end application.

3.2 API Functions

3.2.1 _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 radix character. 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.

3.2.2 _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.

3.2.3 _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.

3.2.4 _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.

3.2.5 _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.

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. If **MATH_TYPE** is set to **FLOAT_MATH**, the return is the number of characters written into the output buffer.

3.2.6 _IQNtoD

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

Prototype:

```
double
_IQNtoD(_iqN A)
```

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

- or -

```
double
_IQtoD(_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 double-precision floating-point number.

Returns:

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

3.2.7 _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.

3.2.8 _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.

3.2.9 _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.

3.2.10 _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.

3.2.11 _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 Arithmetic Functions

4.1 Introduction

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	IQ Format	Execution Cycles	Accuracy (bits)	Program Memory (bytes)	Input Format	Output Format
_IQdiv2	1-30	1	32 bits	2	IQN	IQN
_IQdiv4	1-30	1	32 bits	2	IQN	IQN
_IQdiv8	1-30	1	32 bits	2	IQN	IQN
_IQdiv16	1-30	1	32 bits	2	IQN	IQN
_IQdiv32	1-30	1	32 bits	2	IQN	IQN
_IQdiv64	1-30	1	32 bits	2	IQN	IQN
_IQmpy2	1-30	1	32 bits	2	IQN	IQN
_IQmpy4	1-30	1	32 bits	2	IQN	IQN
_IQmpy8	1-30	1	32 bits	2	IQN	IQN
_IQmpy16	1-30	1	32 bits	2	IQN	IQN
_IQmpy32	1-30	1	32 bits	2	IQN	IQN
_IQmpy64	1-30	1	32 bits	2	IQN	IQN
_IQNdiv	1-30	59	32 bits	144	IQN/IQN	IQN
_IQNmpy	1-30	13	32 bits	16	IQN*IQN	IQN
_IQNmpyI32	1-30	1	32 bits	2	IQN*long	IQN
_IQNmpyI32frac	1-30	14	32 bits	20	IQN*long	IQN
_IQNmpyI32int	1-30	17	32 bits	20	IQN*long	long
_IQNmpyIQX	1-30	20	32 bits	52	IQN*IQN	IQN
_IQNrmpy	1-30	13	32 bits	12	IQN*IQN	IQN
_IQNrsmpy	1-30	17	32 bits	36	IQN*IQN	IQN

- The number of execution cycles and program memory usage provided above assumes IQ24 format. Execution cycles may vary by a few cycles for other IQ formats, and program memory usage may vary by a few bytes for other IQ formats.
- The number of execution cycles provided in the table includes the call and return and assumes that the IQmath library is running from internal flash.
- Accuracy should always be tested and verified within the end application.

4.2 API Functions

4.2.1 _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.

4.2.2 _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.

4.2.3 _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.

4.2.4 _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.

4.2.5 _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.

4.2.6 _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.

4.2.7 _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.

4.2.8 _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.

4.2.9 _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.

4.2.10 _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.

4.2.11 _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.

4.2.12 _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.

4.2.13 _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.

4.2.14 _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.

4.2.15 _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.

4.2.16 _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.

4.2.17 _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.

4.2.18 _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.

4.2.19 _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.

4.2.20 _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.

5 Trigonometric Functions

5.1 Introduction

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 π radians, 0.75 is $3\pi/2$ radians, and 1.0 is 2π radians). The following table summarizes the trigonometric functions.

Function Name	IQ Format	Execution Cycles	Accuracy (bits)	Program Memory (bytes)	Input Format	Output Format
_IQNacos	1-29	58	28 bits	148	IQN	IQN
_IQNasin	1-29	54	28 bits	140	IQN	IQN
_IQNatan	1-29	109	30 bits	218	IQN	IQN
_IQNatan2	1-29	107	30 bits	216	IQN,IQN	IQN
_IQNatan2PU	1-30	99	31 bits	208	IQN,IQN	IQN
_IQNcos	1-29	56	30 bits	184	IQN	IQN
_IQNcosPU	1-30	50	30 bits	128	IQN	IQN
_IQNsin	1-29	56	30 bits	180	IQN	IQN
_IQNsinPU	1-30	50	30 bits	124	IQN	IQN

- The number of execution cycles and program memory usage provided above assumes IQ24 format. Execution cycles may vary by a few cycles for other IQ formats, and program memory usage may vary by a few bytes for other IQ formats.
- The number of execution cycles provided in the table includes the call and return and assumes that the IQmath library is running from internal flash.
- Accuracy should always be tested and verified within the end application.

5.2 API Functions

5.2.1 _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 (-π through π) cannot be represented in IQ30 format (which ranges from -2 through 2).

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

5.2.2 _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 (-π through π) cannot be represented in IQ30 format (which ranges from -2 through 2).

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

5.2.3 _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 (-π through π) cannot be represented in IQ30 format (which ranges from -2 through 2).

Returns:

The inverse tangent of the input value, in radians.

5.2.4 _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 (-π through π) cannot be represented in IQ30 format (which ranges from -2 through 2).

Returns:

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

5.2.5 _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.

5.2.6 _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 π) cannot be represented in IQ30 format (which ranges from -2 through 2).

Returns:
The cosine of the input value.

5.2.7 _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.

5.2.8 _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 (-π through π) cannot be represented in IQ30 format (which ranges from -2 through 2).

Returns:
The sine of the input value.

5.2.9 _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.

6 Mathematical Functions

6.1 Introduction

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

Function Name	IQ Format	Execution Cycles	Accuracy (bits)	Program Memory (bytes)	Input Format	Output Format
_IQNexp	1-30	96	30 bits	184	IQN	IQN
_IQNexp2	1-30	72	30 bits	128	IQN	IQN
_IQNisqrt	1-30	62	30 bits	116	IQN	IQN
_IQNmag	1-30	83	30 bits	136	IQN,IQN	IQN
_IQNsqrt	1-30	63	31 bits	108	IQN	IQN

- The number of execution cycles and program memory usage provided above assumes IQ24 format. Execution cycles may vary by a few cycles for other IQ formats, and program memory usage may vary by a few bytes for other IQ formats.

- The number of execution cycles provided in the table includes the call and return and assumes that the IQmath library is running from internal flash.

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

6.2 API Functions

6.2.1 _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.

6.2.2 _IQNexp2

Computes the base-2 exponential value of an IQ number.

Prototype:
```
_iqN
_IQNexp2(_iqN A)
```
for a specific IQ format (1 <= N <= 30)

- or -

```
_iq
_IQexp2(_iq A)
```
for the global IQ format

Parameters:
A is the input value, in IQ format.

Description:
This function computes the base-2 exponential value of the input, and saturates the result if it exceeds the range of the IQ format in use.

Returns:
Returns the base-2 exponential of the input.

6.2.3 _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.

6.2.4 _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(30000)) overflows an _iq16.

Returns:

Returns the inverse square root of the input.

6.2.5 _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.

7 Miscellaneous Functions

7.1 Introduction

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

Function Name	IQ Format	Execution Cycles	Accuracy (bits)	Program Memory (bytes)	Input Format	Output Format
_IQNabs	1-30	3	32 bits	6	IQN	IQN
_IQNsat	1-30	6	32 bits	14	IQN	IQN

- The number of execution cycles and program memory usage provided above assumes IQ24 format. Execution cycles may vary by a few cycles for other IQ formats, and program memory usage may vary by a few bytes for other IQ formats.
- The number of execution cycles provided in the table includes the call and return and assumes that the IQmath library is running from internal flash.
- Accuracy should always be tested and verified within the end application.

7.2 API Functions

7.2.1 _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.

7.2.2 _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.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as "components") are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

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

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license 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 significant portions 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. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI's goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or "enhanced plastic" are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

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