TMS320C64x+ IQmath Library

User's Guide

Literature Number: SPRUGG9
December 2008

Contents

List of Figures

List of Tables

Read This First

About This Manual

The Texas Instruments TMS320C64x+™ IQmath Library is collection of highly-optimized and high-precision mathematical functions. The library is intended for C programmers to seamlessly port their floating-point algorithms into fixed-point code for execution on TMS320C64x+ devices. These routines are typically used in computationally intensive real-time applications, where optimal execution speed and high accuracy are critical. By using these routines you can achieve execution speeds considerably faster than equivalent code written in standard ANSI C language. In addition, by providing ready-to-use high-precision functions, the TI IQmath library can significantly shorten your digital signal processor (DSP) application development time.

Acronyms

IQmath — High-accuracy mathematical functions (32-bit implementation)

QMATH — Fixed-point mathematical computation

Trademarks

TMS320C64x+ is a trademark of Texas Instruments.

All other trademarks are the property of their respective owners.

Chapter 1
SPRUGG9–December 2008

Installing the IQmath Library

1.1 IQmath Content

The TI IQmath library consists of the following components:

- IQmath header file:
  IQmath.h: Includes definitions needed to interface with the IQMath library.
- IQmath inline header file:
  IQmath_inline.h: Includes source code for certain IQMath APIs to enable inlining. For more details on inlining, see the *TMS320C6000 Optimizing Compiler User's Guide* (SPRU187)
- IQmath object library containing all functions for little-endian devices:
  IQmath_c64x+.lib
- IQmath object library containing all functions and data tables for big-endian devices:
  IQmath_c64x+e.lib
- Object library containing IQMath tables (little endian) to be linked in RAM:
  IQmath_RAM_c64x+.lib
- Object library containing IQMath tables (little endian) to be linked from device ROM (DM643x only):
  IQmath_ROM_c643x.lib
- Additionally, an x86-based host library is also provided for host (PC) testing of the IQmath code. This is functionally equivalent to the target library:
  IQmath_pc.lib

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

1.2 How to Install the IQmath Library

The IQmath installation provides the directory structure shown in Figure 1-1.

Figure 1-1. IQmath Directory Structure

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

Using the IQmath Library

Topic **Page**

2.1 IQmath Arguments and Data Types

Input/output of the IQmath functions are typically 32-bit fixed-point numbers and the Q format of the fixed-point number can vary from Q0 to Q31.

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

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

2.2 IQmath Data Type: Range and Resolution

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

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

Table 2-1. Q Format Range and Resolution

Data Type	Range		Resolution/Precision
	Min	Max	
_iq31	-1	0.999 999 999	0.000 000 0005
_iq30	-2	1.999 999 999	0.000 000 001
_iq29	-4	3.999 999 998	0.000 000 002
_iq28	-8	7.999 999 996	0.000 000 004
_iq27	-16	15.999 999 993	0.000 000 007
_iq26	-32	31.999 999 985	0.000 000 015
_iq25	-64	63.999 999 970	0.000 000 030
_iq24	-128	127.999 999 940	0.000 000 060
_iq23	-256	255.999 999 981	0.000 000 119
_iq22	-512	511.999 999 762	0.000 000 238
_iq21	-1024	1023.999 999 523	0.000 000 477
_iq20	-2048	2047.999 999 046	0.000 000 954
_iq19	-4096	4095.999 998 093	0.000 001 907
_iq18	-8192	8191.999 996 185	0.000 003 815
_iq17	-16384	16383.999 992 371	0.000 007 629
_iq16	-32768	32767.999 984 741	0.000 015 259
_iq15	-65536	65535.999 969 482	0.000 030 518
_iq14	-131072	131071.999 938 965	0.000 061 035
_iq13	-262144	262143.999 877 930	0.000 122 070
_iq12	-524288	524287.999 755 859	0.000 244 141
_iq11	-1048576	1048575.999 511 719	0.000 488 281
_iq10	-2097152	2097151.999 023 437	0.000 976 563
_iq9	-4194304	4194303.998 046 875	0.001 953 125
_iq8	-8388608	8388607.996 093 750	0.003 906 250
_iq7	-16777216	16777215.992 187 500	0.007 812 500
_iq6	-33554432	33554431.984 375 000	0.015 625 000
_iq5	-67108864	67108863.968 750 000	0.031 250 000
_iq4	-134217728	134217727.937 500 000	0.062 500 000
_iq3	-268435456	268435455.875 000 000	0.125 000 000
_iq2	-536870912	536870911.750 000 000	0.250 000 000
_iq1	-1073741824	1073741823.500 000 000	0.500 000 000
_iq0	-2147483648	2147483648.000 000 000	1.000 000 000

2.3 Calling IQmath Functions from C

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

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

Note: IQmath functions are assembled in the ***.text:IQmath*** section and the look-up tables used to perform high-precision computation are placed in the ***.data:IQmathTables*** section.

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

```
MEMORY
{
  ERAM_01        :   o = 0x81000000, l = 0x00100000
}

SECTIONS
{
  .data:IQmathTables     :  >   ERAM_01
  .text:IQmath           :  >   ERAM_01
}
```

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

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

_iq input, sin_out;               /* Definition of variables using IQmath datatype      */
void main(void )
{
      input =_IQ29( 0.25*PI );    /*   radians represented in Q29 format                 */
      sin_out =_IQ29sin (input );
}
```

2.4 IQmath Function Naming Convention

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

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

  Examples:

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

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

  Examples:

  ```
  _IQ29sin(A)            /* High Precision SIN: input/output are in Q29      */
  _IQ25cos(A)            /* High Precision COS: input/output are in Q25      */
  _IQ20mpy(A, B)         /* fixed point multiply: input/output are in Q20    */
  ```

Example 2-2. IQmath Function Naming Convention

```
GLOBAL_Q Function
_IQxxx( ), Where "xxx" is the Function Name

Q Specific Function
_IQNxxx( ), Where "xxx" is the Function Name  &
                   "N" is the Q format of input/output
```

2.5 Selecting GLOBAL_Q format

Numerical precision and dynamic range requirement vary considerably from one application to other. The IQmath library facilitates the application programming in fixed-point arithmetic without fixing the numerical precision up front. This allows the system engineer to check the application performance with different numerical precisions and, finally, fix the numerical resolution. As explained in Section 2.2, higher precision results in lower dynamic range. Hence, the system designer must trade off between the range and resolution before choosing the GLOBAL_Q format.

CASE I:

Default GLOBAL_Q format is set to Q24. Edit the *IQmath.h* header file to modify this value as required; user can choose from Q0 to Q31 as GLOBAL_Q format. Note that by modifying this value all the GLOBAL_Q functions use the modified Q format for input/output, unless this symbolic definition is overridden in the source code.

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

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

CASE II:

A complete system consists of various modules. Some modules may require different precision than the rest of the system. In such situations, we need to override the GLOBAL_Q defined in the *IQmath.h* file and use the local Q format.

This can be easily done by defining the GLOBAL_Q constant in the source file of the module before the include statement.

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

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

CASE III:

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

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

```
#define  GLOBAL_Q      20      /*      Set the Global Q value      */
#include <IQmath.h>

int main()
{
        _iq a, b, c;
        _iq25 d, e;

        a = _IQ(2.5);
        b = _IQ(3.5)
        c = _IQmpy(a, b);

        d = _IQ25(1);
        e = _IQ25asin(d);
}
```

Function Summary

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

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

Topic **Page**

3.1 Arguments and Conventions Used

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

Table 3-1. Argument Descriptions

Argument	Description
IQN	32-bit fixed point Q number, where N=0:31
Int, INT, I32_IQ	32-bit signed number
_iq	Data type definition equating to int, a 32-bit value, representing a GLOBAL_Q number. Usage of _iq instead of int is recommended to increase future portability across devices.
_iqN	Data type definition equating to int, a 32-bit value, representing a IQN number, where N=0:31
A, B	Input operand to IQmath function or macro
F	Floating point input: Ex: -1.232, +22.433, 0.4343, -0.32
S	Floating point string: "+1.32", "0.232", "-2.343", etc.
P	Positive Saturation value
N	Negative Saturation value

3.2 IQmath Functions

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

Table 3-2. Format Conversion Utilities

Functions	Description	IQ Format
_iq _FtoIQ(float F)	Converts float to IQ value	Q=GLOBAL_Q
_iqN _ FtoIQN(float F)		Q=1:31
_iq _IQ(int A)	Convert an integer to IQ format	Q=0:31
float _IQtoF(_iq A)	IQ to floating point	Q=GLOBAL_Q
float _IQNtoF(_iqN A)		Q=0:31
_iq _atoIQ(char *S)	Float ASCII string to IQ	Q=GLOBAL_Q
_iqN _atoIQN(char *S)		Q=1:31
int _IQint(_iq A)	Extract integer portion of IQ	Q=GLOBAL_Q
int _IQNint(_iqN A)		Q=0:29
_iq _IQfrac(_iq A)	Extract fractional portion of IQ	Q=GLOBAL_Q
_iqN _IQNfrac(_iqN A)		Q=1:31
_iqN _IQtoIQN(_iq A)	Convert IQ number to IQN number (32-bit)	Q=GLOBAL_Q
_iq _IQNtoIQ(_iqN A)	Convert IQN (32-bit) number to IQ number	Q=GLOBAL_Q
_iqY _IQXtoIQY(_iqX A, int x, int y)	Convert IQX number to IQY number	Q=0:31

Table 3-3. Arithmetic Operations

Functions	Description	IQ Format
_iq _IQmpy(_iq A, _iq B) _iqN _IQNmpy(_iqN A, _iqN B)	IQ multiplication	Q=GLOBAL_Q Q=1:31
_iq _IQrmpy(_iq A, _iq B) _iqN _IQNrmpy(_iqN A, _iqN B)	IQ multiplication with rounding	Q=GLOBAL_Q Q=1:31
_iq _IQrsmpy(_iq A, _iq B) _iqN _IQNrsmpy(_iqN A, _iqN B)	IQ multiplication with rounding and saturation	Q=GLOBAL_Q Q=1:31
_iq _IQmpyI32(_iq A, int B) _iqN _IQNmpyI32(_iqN A, int B)	Multiply IQ with "int" integer	Q=GLOBAL_Q Q=1:30
int _IQmpyI32int(_iq A, int B) int _IQNmpyI32int(_iqN A, int B)	Multiply IQ with "int", return integer part	Q=GLOBAL_Q Q=0:30
int _IQmpyI32frac(_iq A, int B) int _IQNmpyI32frac(_iqN A, int B)	Multiply IQ with "int", return fraction part	Q=GLOBAL_Q Q=0:30
_iq _IQmpyIQX(_iqN1 A, N1, _iqN2 B, N2) _iqN _IQNmpyIQX(_iqN1 A, N1, _iqN2 B, N2)	Multiply two 2-different IQ numbers	Q=GLOBAL_Q Q=0:31
_iq _IQdiv(_iq A, _iq B) _iqN _IQNdiv(_iqN A, _iqN B)	Fixed-point division	Q=GLOBAL_Q Q=0:31

Table 3-4. Trigonometric Functions

Functions	Description	IQ Format
_iq _IQsin(_iq A) _iqN _IQNsin(_iqN A)	High-precision SIN (input in radians)	Q=GLOBAL_Q Q=1:29
_iq _IQsinPU(_iq A) _iqN _IQNsinPU(_iqN A)	High-precision SIN (input in units)[1]	Q=GLOBAL_Q Q=1:30
_iq _IQasin(_iq A) _iqN _IQNasin(_iqN A)	High-precision inverse SIN (output in radians)	Q=GLOBAL_Q Q=1:29
_iq _IQcos(_iq A) _iqN _IQNcos(_iqN A)	High-precision COS (input in radians)	Q=GLOBAL_Q Q=1:29
_iq _IQcosPU(_iq A) _iqN _IQNcosPU(_iqN A)	High-precision COS (input in units)	Q=GLOBAL_Q Q=1:30
_iq _IQacos(_iq A) _iqN _IQNacos(_iq A)	High-precision inverse COS (output in radians)	Q=GLOBAL_Q Q=1:29
_iq _IQatan2(_iq A, _iq B) _iqN _IQNatan2(_iqN A, _iqN B)	4-quadrant ATAN (output in radians); produces atan(A / B) result	Q=GLOBAL_Q Q=1:29
_iq _IQatan2PU(_iq A, _iq B) _iqN _IQNatanPU(_iqN A, _iqN B)	4-quadrant ATAN (output in units); produces atan(A / B) result	Q=GLOBAL_Q Q=1:30
_iq _IQatan(_iq A) _iqN _IQNatan(_iqN A)	4-quadrant ATAN (output in radians)	Q=GLOBAL_Q Q=1:29

(1) One unit is (2π / 512) radians

Table 3-5. Mathematical Functions

Functions	Description	IQ Format
_iq _IQsqrt(_iq A)	High-precision square root	Q=GLOBAL_Q
_iqN _IQNsqrt(_iqN A)		Q=0:30
_iq _IQisqrt(_iq A)	High-precision inverse square root	Q=GLOBAL_Q
_iqN _IQNisqrt(_iqN A)		Q=0:30
_iq _IQmag(_iq A, _iq B)	Magnitude square: sqrt($A^2 + B^2$)	Q=GLOBAL_Q
_iqN _IQNmag(_iqN A, _iqN B)		Q=0:30
_iq _IQexp(_iq A)	Exponential	Q=GLOBAL_Q
_iqN _IQNexp(_iqN A)		Q=1:30
_iq _IQlog(_iq A)	Natural logarithm	Q=GLOBAL_Q
_iqN _IQNlog(_iqN A)		Q=1:30
_iq _IQpow(_iq A)	Power: pow = A^B	Q=GLOBAL_Q
_iqN _IQNpow(_iqN A)		Q=1:30

Table 3-6. Miscellaneous Functions

Functions	Description	Q Format
I32 _IQsat(I32 A) I32 _IQNsat(I32 A)	Saturates an integer to the range of the governing Q format	Int32
_iq _IQabs(_iq A)	Absolute value of IQ number	Q=GLOBAL_Q
_iqN IQNabs(_iqN A)		Q=0 :31

3.3 C64x+ IQmath Library Benchmarks

Table 3-7. IQmath Library Benchmarks

Function Name	IQ Format	Execution Cycles (Stand alone)	Execution Cycles (Pipelined)	Program Memory (Words)[(1)]	Input Format	Output Format	Remarks
Trigonometric Functions							
IQNsin	N=1 to 29	56	4	52	IQN	IQN	
IQNsinPU	N=1 to 29	45	3.5.	36	IQN	IQN	
IQNasin	N=1 to 29	122	n.a.	118	IQN	IQN	
IQNcos	N=1 to 29	54	4	46	IQN	IQN	
IQNcosPU	N=1 to 29	48	3.5	39	IQN	IQN	
IQNacos	N=1 to 29	122	n.a.	118	IQN	IQN	
IQNatan2	N=1 to 29	118	32.1	115	IQN	IQN	
IQNatan2PU	N=1 to 29	123	31.1	122	IQN	IQN	
IQatan	N=1 to 29	134	32.1	115	IQN	IQN	
Mathematical Functions							
IQNsqrt	N=0 to 30	79	8.6	77	IQN	IQN	
IQNisqrt	N=0 to 30	83	11.1	92	IQN	IQN	
IQNmag	N=0 to 30	89	12.6	75	IQN	IQN	
IQpow	N=1 to 30	550	n.a.	14	IQN	IQN	
Arithmetic Functions							
IQNmpy	N=1 to 31	20	1	7	IQN*IQN	IQN	
IQNrmpy	N=1 to 31	23	2	13	IQN*IQN	IQN	
IQNrsmpy	N=1 to 31	27	3	20	IQN*IQN	IQN	
IQNmpyI32int	N=0 to 30	27	3.5	23	IQN*int	int	

(1) 1 word is 4 bytes.

Table 3-7. IQmath Library Benchmarks (continued)

Function Name	IQ Format	Execution Cycles (Stand alone)	Execution Cycles (Pipelined)	Program Memory (Words)[1]	Input Format	Output Format	Remarks
IQNmpyI32frac	N=0 to 30	21	3	16	IQN*int	IQN	
IQNmpyIQX	N= 0 to 31	27	2	41	IQN1*IQN2	IQN	
IQNdiv	N=0 to 31	73	11.1	70	IQN/IQN	IQN	
Format Conversion Utilities							
FtoIQN	N=1 to 31	26	3.5	25	Float	IQN	
IQNtoF	N=0 to 31	23	3	17	IQN	Float	
atoIQN	N=1 to 31	874	n.a.	207	char *	IQN	
IQNint	N=0 to 29	19	1.5	7	IQN	int	
IQNfrac	N=1 to 31	19	1.5	7	IQN	IQN	
IQtoIQN	N=0 to 31	≈4		n.a.	GLOBAL_Q	IQN	C-MACRO
IQNtoIQ	N=0 to 31	≈4		n.a.	IQN	GLOBAL_Q	C-MACRO
IQXtoIQY	N=0 to 31	≈4		n.a.	IQX	IQY	C-MACRO
Miscellaneous							
IQsat	N=0 to 31	24	1	10	IQN	IQN	
IQNabs	N=0 to 31	18	1	2	IQN	IQN	

Notes:

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

Libraries Available

The chapter lists the various builds available for the IQmath source.

4.1 Target Build (C64x+ Little Endian)

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

Library name: *IQmath_c64x+.lib*.

4.1.1 ROM Build (C643x Little Endian)

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

Library name: *IQmath_ROM_c643x.lib*.

4.1.2 RAM Build (C64x+ Little Endian)

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

Library name: *IQmath_RAM_c64x+.lib*.

4.2 Target Build (C64x+ Big Endian)

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

Library name: *IQmath_c64x+e.lib*.

4.3 Host Build

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

Library name: *IQmath_pc.lib*.

4.4 Usage

The usage of all the above described IQmath libraries is identical. The libraries provide the same APIs and the only care the user needs to take is to include the library that is appropriate to the project.

Function Description

This chapter provides detailed descriptions of the C64x+ IQmath library functions and macros.

Table 5-1. List of Functions and Macros

Function/Macro **Page**

_FtoIQN *Float-to-IQN Data Type*

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _FtoIQ(float F)

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

_IQN _FtoIQN(float F)

Input Format Floating-point variable or constant

Output Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

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

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

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

Examples

Example 1: Implementing equation in IQmath:

```
Floating point equation:  Y = M*1.26 + 2.345
IQmath equation (Type 1): Y =_IQmpy(M, _FtoIQ(1.26)) + _FtoIQ(2.345)
IQmath equation (Type 2): Y =_IQ23mpy(M, _FtoIQ23(1.26)) +_FtoIQ23(2.345)
```

Example 2: Converting Floating point variable to IQ data type:

```
float x=3.343;
_iq y1;
_iq23 y2

IQmath (Type 1):      y1=_FtoIQ(x)
IQmath (Type 2):      y2=_FtoIQ23(x)
```

_IQN — *Integer-to-IQN Data Type*

Description This macro converts an integer (short, char) to an equivalent IQ value

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

_iq _IQ(int A);

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

_iqN _IQN(int A);

Input Format **Global IQ function (IQ format = GLOBAL_Q) or Q-format-specific IQ function (IQ format = IQ0 to IQ31)**

Integer

Output Format IQ equivalent of the input.

Usage This operation is typically used to convert an integer constant or variable to its equivalent IQ value.

Note This macro can also be used to convert floating-point numbers to IQ format. However, it is strongly advised not to do so because of the large performance overhead involved. This includes calling two floating-point RTS functions. The _FtoIQ() function should be used for this purpose.

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

```
int   A = 3;
short B = 5;

_iq a, b, c;

a = _IQ(A);             // Uses GLOBAL_Q
b = _IQ10(B);
```

IQNtoF — *Float-to-IQN Data Type*

Description

This function converts a IQ number to equivalent floating-point value in IEEE 754 format.

Declaration

Global IQ function (IQ format = GLOBAL_Q)

float _IQtoF(_iq A)

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

float _IQNtoF(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

Fixed-point IQ number in GLOBAL_Q format.

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

Fixed-point IQ number in IQN format.

Output Format

Floating-point equivalent of fixed-point input.

Usage

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

Example

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

```
_iq DataIQ[N];
_iq24 DataIQ24[N];
float DataF1[N], DataF2[N];

for(i = 0; i < N, i++) {
    DataF1[i] = _IQtoF(DataIQ[i]);
    DataF2[i] = _IQ24toF(DataIQ24[i]);
}
```

atoIQN *String to IQN*

Description This function converts a string to IQ number.

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

float _atoIQ(char *S)

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

float _atoIQN(char *S)

Input Format This function recognizes (in order) an optional sign, a string of digits optionally containing a radix character.

Valid Input strings: "12.23456", "-12.23456", "0.2345", "0.0", "0", "127", "-89"

Output Format The first unrecognized character ends the string and returns zero. If the input string converts to a number greater then the max/min values for the given Q value, then the returned value is limited to the min/max values for the given Q format.

Global IQ function (IQ format = GLOBAL_Q)

Fixed-point equivalent of input string in GLOBAL_Q format.

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

Fixed-point equivalent of input string in IQN format.

Saturation If the input string converts to a number out of range for the given Q value, then the returned value is limited to the min/max values.

Usage This is useful for programs that need to process user input or ASCII strings.

Example The following code prompts the user to enter the value X:

```
char buffer[N];
_iq X;
_iq20 Y;

printf("Enter value X = ");
gets(buffer);
X = _atoIQ(buffer);            // IQ value (GLOBAL_Q)
Y = _atoIQ20(buffer);
```

IQNint *Integer Part of IQN Number*

Description This function returns the integer portion of IQ number.

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

int _IQint(_iq A)

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

int _IQNint(_iqN A)

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

Fixed-point IQ number in GLOBAL_Q format.

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

Fixed-point IQ number in IQN format.

Output Format Integer part of the IQ number.

Example Extracting integer and fractional part of IQ number.

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

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

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

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

IQNfrac — *Fractional Part of IQN Number*

Description

This function returns the fractional portion of IQ number.

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQfrac(_iq A)

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

iqN _IQNfrac (_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

Fixed-point IQ number in GLOBAL_Q format.

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

Fixed-point IQ number in IQN format.

Output Format

Fractional part of the IQ number.

Example

Extracting integer and fractional part of IQ number.

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

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

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

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

IQtoIQN *GLOBAL_Q Number to IQN*

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

Declaration _iqN _IQtoIQN(_iq A)

Input Format IQ number in GLOBAL_Q format.

Output Format Equivalent value of input in IQN format.

Note This functionality is implemented by a simple shift. Rounding is not performed.

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

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

```
Z = sqrt(X^2 + Y^2)
```

The values Z, X, and Y are given as GLOBAL_Q = 26, but the equation itself may generate an overflow. To guard against this, the intermediate calculations are performed in a lower Q format (say Q = 23), in which we know that overflow will not occur. The result is converted back to GLOBAL_Q at the end, as shown below:

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

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

Y = _IQ23toIQ(temp);
```

IQNtoIQ — *IQN Number to GLOBAL_Q*

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

Declaration _iq _IQNtoIQ(_iqN A)

Input Format IQ number in IQN format.

Output Format Equivalent value of input in GLOBAL_Q format.

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

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

```
Z = sqrt(X^2 + Y^2)
```

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

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

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

Y = _IQ23toIQ(temp);
```

IQNmpy *IQ Multiplication(IQN*IQN)*

Description

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

Declaration

Global IQ (IQ format = GLOBAL_Q)

_iq _IQmpy(_iq A, _iq B)

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

_iqN _IQNmpy(_iqN A, _iqN B)

Input Format

Global IQ (IQ format = GLOBAL_Q)

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

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

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

Output Format

Global IQ (IQ format = GLOBAL_Q)

Result of multiplication in GLOBAL_Q format.

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

Result of multiplication in IQN format.

Examples

Example 1: The following code computes Y = M*X + B in GLOBAL_Q format with no rounding or saturation:

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

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

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

IQNrmpy — *IQ Multiplication With Rounding (IQN*IQN)*

Description

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQrmpy(_iq A, _iq B)

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

_iqN _IQNrmpy(_iqN A, _iqN B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

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

Output Format

Global IQ function (IQ format = GLOBAL_Q)

Result of multiplication in GLOBAL_Q format.

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

Result of multiplication in IQN format.

Examples

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

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

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

```
_iq10 Y, M, X, B;
Y = _IQ10rmpy(M,X) + B;
```

IQNrsmpy

*IQ Multiplication With Rounding and Saturation (IQN*IQN)*

Description

This function multiplies two IQ number with rounding and saturation. In cases where the calculation may possibly exceed the range of the IQ variable, this operation rounds and then saturates the result to the maximum IQ value range before storing.

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQrsmpy(_iq A, _iq B)

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

_iqN _IQNrsmpy(_iqN A, _iqN B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

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

Output Format

Global IQ function (IQ format = GLOBAL_Q)

Result of multiplication in GLOBAL_Q format.

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

Result of multiplication in IQN format.

Saturation

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

Usage

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

Examples

Example 1: The following code computes Y = M*X in GLOBAL_Q format with rounding and saturation (assuming GLOBAL_Q=26):

```
_iq Y, M, X;

M = _FtoIQ(10.9);           // M=10.9
X = _FtoIQ(4.5);            // X=4.5
Y = _IQrsmpy(M,X);          // Y= ~32.0, output is Saturated to MAX
```

Example 2: The following code computes Y = M*X in IQ26 format with rounding and saturation:

```
_iq26 Y, M, X;

M=_IQ26(-10.9);            // M=-10.9
X=_IQ26(4.5);              // X=4.5
Y = _IQ26rsmpy(M,X);       // Y= -32.0, output is Saturated to MIN
```

IQNmpyI32 *Multiplication (IQN*INT)*

Description This macro multiplies an IQ number with an integer.

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

_iq _IQmpyI32(_iq A, int B)

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

_iqN _IQNmpyI32(_iqN A, int B)

Input Format **Global IQ macro (IQ format = GLOBAL_Q)**

Operand A is an IQ number in GLOBAL_Q format and B is the integer.

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

Operand A is an IQ number in IQN format and B is the integer.

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

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

```
_iq Y,  X;

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

Example 2: The following code computes Y = 5*X in IQ26 format:

```
I64_IQ Z;
int M;

M = 5;                        // M=5
X =_FtoIQN (5.1, 26);         // X=5.1 in IQ26 format
Y = _IQ26mpyI32(X, M);        // Y=25.5 in IQ26 format (32 bit result)
Z = _IQ26mpyI32(X, 0x80000000u);  // 64 bit result in Z
```

IQNmpyI32int — *Integer Portion of (IQN*INT)*

Description

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

int _IQmpyI32int(_iq A, int B)

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

int _IQNmpyI32int(_iqN A, int B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

Operand A is an IQ number in GLOBAL_Q format and B is the integer.

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

Operand A is an IQ number in IQN format and B is the integer.

Output Format

Global IQ function (IQ format = GLOBAL_Q)

Integer part of the result (32-bit).

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

Integer part of the result (32-bit).

Saturation

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

Example

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

```
int temp;
short OutputDAC;

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

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

if( temp < 0 )
    temp = 0;

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

Note: An integer operation performs the multiply and calculates the integer portion from the resulting 64-bit result.

IQNmpyI32frac — *Fractional Part of (IQN*INT)*

Description

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQmpyI32frac(_iq A, int B)

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

_iqN _IQNmpyI32frac(_iqN A, int B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

Operand A is an IQ number in GLOBAL_Q format and B is the 32-bit integer.

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

Operand A is an IQ number in IQN format and B is the 32-bit integer.

Output Format

Global IQ function (IQ format = GLOBAL_Q)

Fractional part of the result (32-bit).

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

Fractional part of the result (32-bit).

Example

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

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

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

IQNmpyIQX *Multiplication (GLOBAL_Q=IQN1*IQN2)*

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

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

_iq _IQmpyIQX(_iqN1 A, int N1, _iqN2 B, int N2)

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

_iqN _IQNmpyIQX(_iqN1 A, int N1, _iqN2 B, int N2)

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

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

Result of the multiplication in GLOBAL_Q format.

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

Result of the multiplication in IQN format.

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

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

Example We want to calculate the following equation:

Y = X0*C0 + X1*C1 + X2*C2

Where,

X0, X1, and X2 values are in IQ30 format (range -2 to +2)

C0, C1, and C2 values are in IQ28 format (range –8 to +8)

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

Case 1: GLOBAL_Q=IQ25

```
_iq30 X0, X1, X2;                        // All values IQ30
_iq28 C0, C1, C2;                        // All values IQ28
_iq Y;                                   // Result GLOBAL_Q = IQ25

Y = _IQmpyIQX(X0, 30, C0, 28);
Y += _IQmpyIQX(X1, 30, C1, 28);
Y += _IQmpyIQX(X2, 30, C2, 28);
```

Case 2: IQ-Specific Computation

```
_iq28 C0, C1, C2;                        // All values IQ28
_iq25 Y;                                 // Result GLOBAL_Q = IQ25

Y = _IQ25mpyIQX(X0, 30, C0, 28);
Y += _IQ25mpyIQX(X1, 30, C1, 28);
Y += _IQ25mpyIQX(X2, 30, C2, 28);
```

IQNdiv — *Fixed-Point Division*

Description

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQdiv(_iq A, _iq B)

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

_iqN _IQNdiv(_iqN A, iqN B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

The input arguments A and B are fixed-point numbers in IQN format (N=0:31).

Output Format

Global IQ function (IQ format = GLOBAL_Q)

Output in GLOBAL_Q format.

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

Output in IQN format (N=0:31).

Saturation

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

Example

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

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

        _iq in1 out1;
        _iq28 in2, out2;

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

                in2 = _IQ28 (1.5);
                out2 = _IQ28div (_FtoIQ28 (1.0), in2);
        }
```

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

Description

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQsin(_iq A)

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

_iqN _IQNsin(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

The input argument is in radians and represented as a fixed-point number in GLOBAL_Q format.

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

The input argument is in radians and represented as a fixed-point number in IQN format (N=1:29).

Output Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

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

Saturation

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

Example

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

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

_iq in1, out1;
_iq28 in2, out2;

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

   in2  = _FtoIQ29 (0.25*PI)   /* in2=0.25 x π x 2^29 = 1921FB54h           */
   out2 = _IQ29sin (in2);      /* out2=sin(0.25 x π) x 2^29 = 16A09E66h     */
}
```

IQNsinPU *Fixed-Point SIN (in per-unit radians)*

Description

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQsinPU(_iq A)

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

_iqN _IQNsinPU(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

The input argument is in per-unit radians and represented as a fixed-point number in IQN format (N=1:29).

Output Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

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

Saturation

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

Example

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

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

_iq in1, out1;
_iq30 in2, out2;

void main(void )
{
   in1=_IQ ( 0.25*PI/PI );    /* in1 =0.25 x π/2π x 2^30 = 08000000h     */
   out1=_IQsinPU (in1);       /* out1=sin(0.25 x π) x 2^30 = 2D413CCCh   */

   in2=_IQ30 (0.25*PI/PI);    /* in2 =0.25 x π/2π x 2^30 = 08000000h     */
   out2= _IQ30sinPU (in2);    /* out2=sin(0.25 x π) x 2^30 = 2D413CCCh   */
}
```

IQNcos *Fixed-Point COS (in radians)*

Description

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQcos(_iq A)

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

_iqN _IQNcos(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

The input argument is in radians and represented as a fixed-point number in GLOBAL_Q format.

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

The input argument is in radians and represented as a fixed-point number in IQN format (N=1:29).

Output Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

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

Saturation

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

Example

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

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

_iq in1, out1;
_iq29 in2 out2;

void main(void )
{
   in1  = _IQ (0.25*PI);        /* in1=0.25 x π x 2^29 = 1921FB54h             */
   out1 = _IQcos (in1);         /* out1=cos(0.25 x π) x 2^29 = 16A09E66h       */

   in2  = _IQ29 (0.25*PI);      /* in2=0.25 x π x 2^29 = 1921FB54h             */
   out2 = _IQ29cos (in2);       /* out2=cos(0.25 x π) x 2^29 = 16A09E66h       */
}
```

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

Description

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQcosPU(_iq A)

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

_iqN _IQNcosPU(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

The input argument is in per-unit radians and represented as a fixed-point number in IQN format (N=1:29).

Output Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

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

Saturation

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

Example

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

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

_iq in1, out1;
_iq30 in2, out2;

void main(void )
{
   in1=_IQ (0.25*PI/PI );      /* in1 =0.25 x π/2π x 2^30 = 08000000h     */
   out1=_IQcosPU (in1);        /* out1=cos(0.25 x π) x 2^30 = 2D413CCCh   */

   in2=_IQ30 (0.25*PI/PI);     /* in2 =0.25 x π/2π x 2^30 = 08000000h     */
   out2= _IQ30cosPU (in2);     /* out2=cos(0.25 x π) x 2^30 = 2D413CCCh   */
}
```

IQNatan2 *Fixed-Point 4-Quadrant ATAN (in radians)*

Description

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQatan2(_iq A, _iq B)

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

_iqN _IQNatan2(_iqN A, _iqN B), where the Q format N can vary from 1 to 29

Input Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

The input arguments A and B are fixed-point numbers in IQN format (N=1:29).

Output Format

Global IQ function (IQ format = GLOBAL_Q)

This function returns the inverse tangent of the input argument as a fixed-point number in GLOBAL_Q format. The output contains the angle in radians between [-π, +π].

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

This function returns the inverse tangent of the input argument as a fixed-point number in IQN format (N=1:29). The output contains the angle in radians between [-π, +π].

Example

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

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

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

void main(void )
{
  xin1=_FtoIQ(0.809);           /* xin1=cos(π/5) x 2^29 = 19E3779Bh          */
  yin1=_FtoIQ(0.5877);          /* yin1=sin(π/5) x 2^29 = 12CF2304h          */
  out1=_IQatan2(yin1,xin1);     /* out1=π/5 x 2^29 = 141B2F76h               */

  xin2=_FtoIQ29(0.809);         /* xin1=cos(π/5) x 2^29 = 19E3779Bh          */
  yin2=_FtoIQ29(0.5877);        /* yin1=sin(π/5) x 2^29 = 12CF2304h          */
  out2=_IQ29atan2(yin2,xin2);   /* out2=π/5 x 2^29 = 141B2F76h               */
}
```

IQNatan2PU — *Fixed-Point 4-Quadrant ATAN (in per-unit radians)*

Description

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQatan2PU(_iq A, _iq B)

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

_iqN _IQNatan2PU(_iqN A, _iqN B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

The input arguments A and B are fixed-point numbers in IQN format (N=1:29).

Output Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

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

Example

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

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

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

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

  xin2=_IQ29(0.809)                /* xin2=cos(π/5) x 2^29 = 19E3779Bh    */
  yin2=_IQ29(0.5877)               /* yin2=sin(π/5) x 2^29 = 12CF2304h    */
  out2=_IQ29atan2PU(yin2,xin2)     /* ou2 =(π/5)/2π x 2^29 = 03333333h    */
}
```

IQNatan *Fixed-Point ATAN (in radians)*

Description

This module computes the arctangent. Module output is in radians and varies from -π/2 to π/2.

Declaration

Global IQ Macro (IQ format = GLOBAL_Q)

#define _IQatan(A) _IQatan2(A , _IQ(1.0))

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

#define _IQNatan(A) _IQNatan2(A , _IQN(1.0))

Input Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

The input argument is a fixed-point number in IQN format (N=1:29).

Output Format

Global IQ function (IQ format = GLOBAL_Q)

This function returns the inverse tangent of the input argument as a fixed-point number in GLOBAL_Q format. The output contains the angle in radians between [-π/2, +π/2].

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

This function returns the inverse tangent of the input argument as a fixed-point number in IQN format (N=1:29). The output contains the angle in radians between [-π/2, +π/2].

Example

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

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

_iq in1, out1;
_iq29 in2, out2;

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

  in2=_IQ29(1.0);
  out2=_IQ29atan(in2)
}
```

IQNsqrt — *Fixed-Point Square Root*

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQsqrt(_iq A)

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

_iqN _IQNsqrt(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

The input argument is a fixed-point number in IQN format (N=0:30).

Output Format

Global IQ function (IQ format = GLOBAL_Q)

Square root of input in GLOBAL_Q format.

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

Square root of input in IQN format (N=0:30).

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

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

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

_iq in1, out1;
_iq30 in2, out2;

void main(void )

{
    in1=_FtoIQ(1.8);            /* in1= 1.8x2^30  = 73333333h            */
    out1=_IQsqrt(x);            /* out1= √1.8x2^30  = 55DD7151h          */

    in2=_FtoIQ30(1.8);          /* in2= 1.8x2^30  = 73333333h            */
    out2=_IQ30sqrt(x);          /* out2= √1.8x2^30  = 55DD7151h          */
}
```

IQNisqrt *Fixed-Point Inverse Square Root*

Description

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

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQisqrt(_iq A)

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

_iqN _IQNisqrt(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

The input argument is a fixed-point number in IQN format (N=0:30).

Output Format

Global IQ function (IQ format = GLOBAL_Q)

Inverse square-root of input in GLOBAL_Q format.

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

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

Saturation

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

Example

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

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

_iq in1, out1;
_iq30 in2, out2;

void main(void )

{
   in1=_FtoIQ(1.8);           /* in1= 1.8x2^30  = 73333333h            */
   out1=_IQisqrt(in1);        /* out1=1/√1.8 x2^30  = 2FB3E99Ehh       */

   in2=_FtoIQ30(1.8);         /* in2= 1.8x2^30  = 73333333h            */
   out2=_IQ30isqrt(in2);      /* out2=1/√1.8 x2^30  = 2FB3E99Ehh       */
}
```

IQNmag

Magnitude of IQ Complex Number

Description

This function calculates the magnitude of two orthogonal vectors as follows: Mag = sqrt(A^2 + B^2). This operation achieves better accuracy and avoids overflow problems that may be encountered by using the _IQsqrt function. This is because the internal computations (A^2 + B^2) are maintained at 64-bit accuracy.

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQmag(_iq A, _iq B)

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

_iqN _IQNmag(_iqN A, _iqN B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

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

Output Format

Global IQ function (IQ format = GLOBAL_Q)

Magnitude of the input vector in GLOBAL_Q format.

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

Magnitude of the input vector in IQN format.

Saturation

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

Example

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

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

_iq real1, imag1, mag1;           /* Complex number = real1 + j*imag1         */
_iq28 real2, imag2, mag2;         /* Complex number = real2 + j*imag2         */

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

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

IQNexp *Magnitude of IQ Complex Number*

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

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

_iq _IQexp(_iq A)

Q-format-specific IQ function (IQ format = IQ1to IQ30)

_iqN _IQNexp(_iqN A)

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

The input argument A is an IQ number represented in GLOBAL_Q format.

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

The input argument A is an IQ number represented in IQN format.

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

Exponential of the input vector in GLOBAL_Q format.

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

Exponential of the input vector in IQN format.

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

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

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

  real2=_IQ28(1.0);
  exp2=_IQ28exp(real2);         /* 2.71828 in Q28 Format               */

}
```

IQNlog — *Magnitude of IQ Complex Number*

Description

This function calculates the fixed-point natural logarithm.

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQlog(_iq A)

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

_iqN _IQNlog(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

The input argument A is an IQ number represented in GLOBAL_Q format.

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

The input argument A is an IQ number represented in IQN format.

Output Format

Global IQ function (IQ format = GLOBAL_Q)

Logarithm of the input vector in GLOBAL_Q format.

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

Logarithm of the input vector in IQN format.

Example

The following sample code obtains the logarithm of the complex number (assuming GLOBAL_Q=IQ28):

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

_iq real1, log1;
_iq28 real2, log2;
void main(void )
{
  real1=_IQ(4.0);
  mag1=_IQlog(real1);                /* 1.38629 in GLOBAL_Q format          */

 real2=_IQ28(4.0);
 mag2=_IQ28log(real2);               /* 1.38629 in Q28                      */
}
```

IQNpow — *Magnitude of IQ Complex Number*

Description

This function calculates the fixed-point power of two orthogonal vectors as follows: pow = A^B.

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQpow(_iq A, _iq B)

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

_iqN _IQNpow(_iqN A, _iqN B)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

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

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

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

Output Format

Global IQ function (IQ format = GLOBAL_Q)

Power of the input vector in GLOBAL_Q format.

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

Power of the input vector in IQN format.

Example

The following sample code demonstrates the usage of IQNpow function:

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

_iq real1, imag1, mag1;
_iq28 real2, imag2, mag2;
void main(void )
{
  real1=_FtoIQ(4.0);
  imag1=_FtoIQ(1.0);
  mag1=_IQpow(real1, imag1);        /* mag1=4 in GLOBAL_Q                  */

  real2=_FtoIQ28(4.0);
  imag2=_FtoIQ28(1.0);
  mag2=_IQ28pow(real2, imag2);      /* mag2=4.0 in IQ28 format             */
}
```

IQNabs — *Absolute Value of IQ Number*

Description

This function calculates the absolute value of an IQ number.

Declaration

Global IQ function (IQ format = GLOBAL_Q)

_iq _IQabs(_iq A)

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

_iqN _IQNabs(_iqN A)

Input Format

Global IQ function (IQ format = GLOBAL_Q)

IQ number in GLOBAL_Q format.

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

IQ number in IQN format.

Output Format

Global IQ function (IQ format = GLOBAL_Q)

Absolute value of input in GLOBAL_Q format.

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

Absolute value of input in IQN format.

Saturation

This function does not saturate.

Example

Calculate the absolute sum of three IQ numbers (GLOBAL_Q=IQ28):

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

xsum = _IQabs(X0) + _IQabs(X1) + _IQabs(X2);
xsum = _IQ28abs(X0) + _IQ28abs(X1) + _IQ28abs(X2);
```

IQsat *Saturate the IQ Number*

Description This function saturates an IQ value to the given positive and negative limits. This operation is useful in areas where there is potential for overflow in a calculation.

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

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

IQ number in GLOBAL_Q format.

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

Saturated output in GLOBAL_Q format.

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

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

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

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

```
iq Y, M, X, B;          // GLOBAL_Q = 26 (+/- 32 range)
_iq20 temp;             // IQ = 20 (+/- 2048 range)

temp = _IQ20mpy(_IQtoIQ20(M), _IQtoIQ20(X)) + _IQtoIQ20(B);
temp = _IQsat(temp, _IQtoIQ20(MAX_IQ_POS), _IQtoIQ20(MAX_IQ_NEG));

Y = _IQ20toIQ(temp);
```