NAME

qmath - fixed-point math library based on the "Q" number format

SYNOPSIS

#include <sys/qmath.h>

DESCRIPTION

The **qmath** data types and APIs support fixed-point math based on the "Q" number format. The APIs have been built around the following data types: *s8q_t*, *u8q_t*, *s16q_t*, *u16q_t*, *s32q_t*, *u32q_t*, *s64q_t*, and *u64q_t*, which are referred to generically in the earlier API definitions as *QTYPE*. The *ITYPE* refers to the stdint(7) integer types. *NTYPE* is used to refer to any numeric type and is therefore a superset of *QTYPE* and *ITYPE*.

This scheme can represent Q numbers with [2, 4, 6, 8, 16, 32, 48] bits of precision after the binary radix point, depending on the *rpshft* argument to **Q_INI**(). The number of bits available for the integral component is not explicitly specified, and implicitly consumes the remaining available bits of the chosen Q data type.

Operations on Q numbers maintain the precision of their arguments. The fractional component is truncated to fit into the destination, with no rounding. None of the operations is affected by the floating-point environment.

For more details, see the *IMPLEMENTATION DETAILS* below.

LIST OF FUNCTIONS

Functions which create/initialise a Q number

Name	Description
Q_INI(3)	initialise a Q number

Numeric functions which operate on two Q numbers

Name	Description
Q_QADDQ(3)	addition
Q_QDIVQ(3)	division
Q_QMULQ(3)	multiplication
Q_QSUBQ(3)	subtraction
Q_NORMPREC(3)	normalisation
Q_QMAXQ(3)	maximum function
Q_QMINQ(3)	minimum function
Q_QCLONEQ(3)	

identical copy

Q_QCPYVALQ(3)
representational copy

Numeric functions which apply integers to a Q number

Name	Description
Q_QADDI(3)	addition
Q_QDIVI(3)	division
Q_QMULI(3)	multiplication
Q_QSUBI(3)	subtraction
Q_QFRACI(3)	fraction
Q_QCPYVALI(3)	overwrite

Numeric functions which operate on a single Q number

Name	Description
Q_QABS(3)	absolute value
Q_Q2D(3)	double representation
Q_Q2F(3)	float representation

Comparison and logic functions

Name	Description
Q_SIGNED(3)	determine sign
Q_LTZ(3)	less than zero
Q_PRECEQ(3)	compare bits
Q_QLTQ(3)	less than
Q_QLEQ(3)	less or equal
Q_QGTQ(3)	greater than
Q_QGEQ(3)	greater or equal
Q_QEQ(3)	equal
Q_QNEQ(3)	not equal
Q_OFLOW(3)	would overflow
Q_RELPREC(3)	relative precision

Functions which manipulate the control/sign data bits

Name	Description
Q_SIGNSHFT(3)	sign bit position
Q_SSIGN(3)	sign bit

Q_CRAWMASK(3)	control bitmask
Q_SRAWMASK(3)	sign bitmask
Q_GCRAW(3)	raw control bits
Q_GCVAL(3)	value of control bits
Q_SCVAL(3)	set control bits

Functions which manipulate the combined integer/fractional data bits

Name	Description
Q_IFRAWMASK(3)	integer/fractional bitmask
Q_IFVALIMASK(3)	value of integer bits
Q_IFVALFMASK(3)	value of fractional bits
Q_GIFRAW(3)	raw integer/fractional bits
Q_GIFABSVAL(3)	absolute value of fractional bits
Q_GIFVAL(3)	real value of fractional bits
Q_SIFVAL(3)	set integer/fractional bits
Q_SIFVALS(3)	set separate integer/fractional values

Functions which manipulate the integer data bits

Name	Description
Q_IRAWMASK(3)	integer bitmask
Q_GIRAW(3)	raw integer bits
Q_GIABSVAL(3)	absolute value of integer bits
Q_GIVAL(3)	real value of integer bits
Q_SIVAL(3)	set integer bits

Functions which manipulate the fractional data bits

Name	Description
Q_FRAWMASK(3)	fractional bitmask
Q_GFRAW(3)	raw fractional bits
Q_GFABSVAL(3)	

	absolute value of fractional bits
Q_GFVAL(3)	real value of fractional bits
Q_SFVAL(3)	set fractional bits

Miscellaneous functions/variables

Name	Description
Q_NCBITS(3)	number of reserved control bits
Q_BT(3)	C data type
Q_TC(3)	casted data type
Q_NTBITS(3)	number of total bits
Q_NFCBITS(3)	number of control-encoded fractional bits
Q_MAXNFBITS(3)	number of maximum fractional bits
Q_NFBITS(3)	number of effective fractional bits
Q_NIBITS(3)	number of integer bits
Q_RPSHFT(3)	bit position of radix point
Q_ABS(3)	absolute value
Q_MAXSTRLEN(3)	number of characters to render string
Q_TOSTR(3)	render string
Q_SHL(3)	left-shifted value
Q_SHR(3)	right-shifted value
Q_DEBUG(3)	render debugging information
Q_DFV2BFV(3)	convert decimal fractional value

IMPLEMENTATION DETAILS

The **qmath** data types and APIs support fixed-point math based on the "Q" number format. This implementation uses the Q notation *Qm.n*, where *m* specifies the number of bits for integral data (excluding the sign bit for signed types), and *n* specifies the number of bits for fractional data.

The APIs have been built around the following q_t derived data types:

```
typedef int8_t          s8q_t;
typedef uint8_t         u8q_t;
typedef int16_t         s16q_t;
typedef uint16_t   u16q_t;
typedef int32_t         s32q_t;
typedef uint32_t   u32q_t;
```

```
typedef int64_t          s64q_t;
typedef uint64_t    u64q_t;
```

These types are referred to generically in the earlier API definitions as *QTYPE*, while *ITYPE* refers to the stdint(7) integer types the Q data types are derived from. *NTYPE* is used to refer to any numeric type and is therefore a superset of *QTYPE* and *ITYPE*.

The 3 least significant bits (LSBs) of all q_t data types are reserved for embedded control data:

- bits 1-2 specify the binary radix point shift index operand, with 00,01,10,11 == 1,2,3,4.

- bit 3 specifies the radix point shift index operand multiplier as 2 (0) or 16 (1).

This scheme can therefore represent Q numbers with [2,4,6,8,16,32,48,64] bits of precision after the binary radix point. The number of bits available for the integral component is not explicitly specified, and implicitly consumes the remaining available bits of the chosen Q data type.

Additionally, the most significant bit (MSB) of signed Q types stores the sign bit, with bit value 0 representing a positive number and bit value 1 representing a negative number. Negative numbers are stored as absolute values with the sign bit set, rather than the more typical two's complement representation. This avoids having to bit shift negative numbers, which can result in undefined behaviour from some compilers.

This binary representation used for Q numbers therefore comprises a set of distinct data bit types and associated bit counts. Data bit types/labels, listed in LSB to MSB order, are: control 'C', fractional 'F', integer 'I' and sign 'S'. The following example illustrates the binary representation of a Q20.8 number represented using a s32q_t variable:

```
M                                        L
S                                       S
B                                        B

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0

S I I I I I I I I I I I I I I I I I I I I F F F F F F F F C C C
```

Important bit counts are: total, control, control-encoded fractional, maximum fractional, effective fractional and integer bits.

The count of total bits is derived from the size of the q_t data type. For example, a s32q_t has 32 total bits.

The count of control-encoded fractional bits is derived from calculating the number of fractional bits per the control bit encoding scheme. For example, the control bits binary value of 101 encodes a fractional bit count of 2 x 16 = 32 fractional bits.

The count of maximum fractional bits is derived from the difference between the counts of total bits and control/sign bits. For example, a s32q_t has a maximum of 32 - 3 - 1 = 28 fractional bits.

The count of effective fractional bits is derived from the minimum of the control-encoded fractional bits and the maximum fractional bits. For example, a s32q_t with 32 control-encoded fractional bits is effectively limited to 28 fractional bits.

The count of integer bits is derived from the difference between the counts of total bits and all other non-integer data bits (the sum of control, fractional and sign bits.) For example, a s32q_t with 8 effective fractional bits has 32 - 3 - 8 - 1 = 20 integer bits. The count of integer bits can be zero if all available numeric data bits have been reserved for fractional data, e.g., when the number of control-encoded fractional bits is greater than or equal to the underlying Q data type's maximum fractional bits.

EXAMPLES

Calculating area of a circle with r=4.2 and rpshft=16

```
u64q_t a, pi, r;
char buf[32]

Q_INI(&a, 0, 0, 16);
Q_INI(&pi, 3, 14159, 16);
Q_INI(&r, 4, 2, 16);

Q_QCLONEQ(&a, r);
Q_QMULQ(&a, r);
Q_QMULQ(&a, pi);

Q_TOSTR(a, -1, 10, buf, sizeof(buf));
printf("%s\n", buf);
```

Debugging

Declare a Q20.8 s32q_t number *s32*, initialise it with the fixed-point value for 5/3, and render a debugging representation of the variable (including its full precision decimal C-string representation), to the console:

```
s32q_t s32;
Q_INI(&s32, 0, 0, 8);
Q_QFRACI(&s32, 5, 3);
char buf[Q_MAXSTRLEN(s32, 10)];
Q_TOSTR(s32, -1, 10, buf, sizeof(buf));
printf(Q_DEBUG(s32, "", "\n\ttostr=%s\n\n", 0), buf);
```

The above code outputs the following to the console:

```
"s32"@0x7fffffffe7d4
        type=s32q_t, Qm.n=Q20.8, rpshft=11, imin=0xfff00001, \
imax=0xfffff
        qraw=0x00000d53
        imask=0x7ffff800, fmask=0x000007f8, cmask=0x00000007, \
ifmask=0x7ffffff8
        iraw=0x00000800, iabsval=0x1, ival=0x1
        fraw=0x00000550, fabsval=0xaa, fval=0xaa
        tostr=1.664
```

Note: The "\" present in the rendered output above indicates a manual line break inserted to keep the man page within 80 columns and is not part of the actual output.

SEE ALSO

errno(2), math(3), Q_FRAWMASK(3), Q_IFRAWMASK(3), Q_INI(3), Q_IRAWMASK(3), Q_QABS(3), Q_QADDI(3), Q_QADDQ(3), Q_SIGNED(3), Q_SIGNSHFT(3), stdint(7)

HISTORY

The **qmath** functions first appeared in FreeBSD 13.0.

AUTHORS

The **qmath** functions and this manual page were written by Lawrence Stewart <*lstewart@FreeBSD.org*> and sponsored by Netflix, Inc.