NAME

avr_math

SYNOPSIS

Macros

#define **M_E** 2.7182818284590452354
#define **M_LOG2E** 1.4426950408889634074 /* log_2 e */
#define **M_LOG10E** 0.43429448190325182765 /* log_10 e */
#define **M_LN2** 0.69314718055994530942 /* log_e 2 */
#define **M_LN10** 2.30258509299404568402 /* log_e 10 */
#define **M_PI** 3.14159265358979323846 /* pi */
#define **M_PI_2** 1.57079632679489661923 /* pi/2 */
#define **M_PI_4** 0.78539816339744830962 /* pi/4 */
#define **M_1_PI** 0.31830988618379067154 /* 1/pi */
#define **M_2_PI** 0.63661977236758134308 /* 2/pi */
#define **M_2_SQRTPI** 1.12837916709551257390 /* 2/**sqrt**(pi) */
#define **M_SQRT2** 1.41421356237309504880 /* **sqrt**(2) */
#define **M_SQRT1_2** 0.70710678118654752440 /* 1/**sqrt**(2) */
#define **NAN** __builtin_nan(’’)
#define **INFINITY** __builtin_inf()
#define **cosf** **cos**
#define **sinf** **sin**
#define **tanf** **tan**
#define **fabsf** **fabs**
#define **fmodf** **fmod**
#define **cbrtf** **cbrt**
#define **hypotf** **hypot**
#define **squaref** **square**
#define **floorf** **floor**
#define **ceilf** **ceil**
#define **frexpf** **frexp**
#define **ldexpf** **ldexp**
#define **expf** **exp**
#define **coshf** **cosh**
#define **sinhf** **sinh**
#define **tanhf** **tanh**
#define **acosf** **acos**
#define **asinf** **asin**
#define **atanf** **atan**
#define **atan2f** **atan2**
#define **logf** **log**
#define **log10f** **log10**
#define **powf** **pow**
#define **isnanf** **isnan**
#define **isinff** **isinf**
#define **isfinitef** **isfinite**
#define **copysignf** **copysign**
#define **signbitf** **signbit**
#define **fdimf** **fdim**
#define **fmaf** **fma**
#define **fmaxf** **fmax**
#define **fminf** **fmin**
#define **truncf** **trunc**
#define **roundf** **round**
#define **lroundf** **lround**
#define **lrintf** **lrint**

Functions

double **cos** (double __x)
double **sin** (double __x)

double **tan** (double __x)
double **fabs** (double __x)
double **fmod** (double __x, double __y)
double **modf** (double __x, double *__iptr)
float **modff** (float __x, float *__iptr)
double **sqrt** (double __x)
float **sqrtf** (float)
double **cbrt** (double __x)
double **hypot** (double __x, double __y)
double **square** (double __x)
double **floor** (double __x)
double **ceil** (double __x)
double **frexp** (double __x, int *__pexp)
double **ldexp** (double __x, int __exp)
double **exp** (double __x)
double **cosh** (double __x)
double **sinh** (double __x)
double **tanh** (double __x)
double **acos** (double __x)
double **asin** (double __x)
double **atan** (double __x)
double **atan2** (double __y, double __x)
double **log** (double __x)
double **log10** (double __x)
double **pow** (double __x, double __y)
int **isnan** (double __x)
int **isinf** (double __x)
static int **isfinite** (double __x)
static double **copysign** (double __x, double __y)
int **signbit** (double __x)
double **fdim** (double __x, double __y)
double **fma** (double __x, double __y, double __z)
double **fmax** (double __x, double __y)
double **fmin** (double __x, double __y)
double **trunc** (double __x)
double **round** (double __x)
long **lround** (double __x)
long **lrint** (double __x)

Detailed Description

#include <math.h>

This header file declares basic mathematics constants and functions.

Notes:

- In order to access the functions declared herein, it is usually also required to additionally link against the library `libm.a`. See also the related **FAQ entry**.
- Math functions do not raise exceptions and do not change the `errno` variable. Therefore the majority of them are declared with const attribute, for better optimization by GCC.

Macro Definition Documentation

#define acosf acos

The alias for **acos()**.

#define asinf asin

The alias for **asin()**.

#define atan2f atan2
The alias for **atan2()**.

#define atanf atan
The alias for **atan()**.

#define cbrtf cbrt
The alias for **cbrt()**.

#define ceilf ceil
The alias for **ceil()**.

#define copysignf copysign
The alias for **copysign()**.

#define cosf cos
The alias for **cos()**.

#define coshf cosh
The alias for **cosh()**.

#define expf exp
The alias for **exp()**.

#define fabsf fabs
The alias for **fabs()**.

#define fdimf fdim
The alias for **fdim()**.

#define floorf floor
The alias for **floor()**.

#define fmaf fma
The alias for **fma()**.

#define fmaxf fmax
The alias for **fmax()**.

#define fminf fmin
The alias for **fmin()**.

#define fmodf fmod
The alias for **fmod()**.

#define frexpf frexp
The alias for **frexp()**.

#define hypotf hypot
The alias for **hypot()**.

#define INFINITY __builtin_inf()
INFINITY constant.

#define isfinitef isfinite
The alias for **isfinite()**.

#define isinff isinf
The alias for **isinf()**.

#define isnanf isnan
The alias for **isnan()**.

#define ldexpf ldexp
The alias for **ldexp()**.

#define log10f log10
The alias for **log10()**.

#define logf log
The alias for **log()**.

#define lrintf lrint
The alias for **lrint()**.

#define lroundf lround
The alias for **lround()**.

#define M_1_PI 0.31830988618379067154 /* 1/pi */
The constant *1/pi*.

#define M_2_PI 0.63661977236758134308 /* 2/pi */
The constant *2/pi*.

#define M_2_SQRTPI 1.12837916709551257390 /* 2/sqrt(pi) */
The constant *2/sqrt*(pi).

#define M_E 2.7182818284590452354
The constant *e*.

#define M_LN10 2.30258509299404568402 /* log_e 10 */
The natural logarithm of the 10.

#define M_LN2 0.69314718055994530942 /* log_e 2 */
The natural logarithm of the 2.

#define M_LOG10E 0.43429448190325182765 /* log_10 e */
The logarithm of the *e* to base 10.

#define M_LOG2E 1.4426950408889634074 /* log_2 e */
The logarithm of the *e* to base 2.

#define M_PI 3.14159265358979323846 /* pi */
The constant *pi*.

#define M_PI_2 1.57079632679489661923 /* pi/2 */
The constant *pi/2*.

#define M_PI_4 0.78539816339744830962 /* pi/4 */
The constant *pi/4*.

#define M_SQRT1_2 0.70710678118654752440 /* 1/sqrt(2) */
The constant *1/sqrt*(2).

#define M_SQRT2 1.41421356237309504880 /* sqrt(2) */
The square root of 2.

#define NAN __builtin_nan('')
NAN constant.

#define powf pow
The alias for **pow()**.

#define roundf round
The alias for **round()**.

#define signbitf signbit
The alias for **signbit()**.

#define sinf sin
The alias for **sin()**.

#define sinhf sinh
The alias for **sinh()**.

#define squaref square
The alias for **square()**.

#define tanf tan
The alias for **tan()**.

#define tanhf tanh
The alias for **tanh()**.

#define truncf trunc
The alias for **trunc()**.

Function Documentation

double acos (double __x)
The **acos()** function computes the principal value of the arc cosine of *__x*. The returned value is in the range [0, pi] radians. A domain error occurs for arguments not in the range [-1, +1].

double asin (double __x)
The **asin()** function computes the principal value of the arc sine of *__x*. The returned value is in the range [-pi/2, pi/2] radians. A domain error occurs for arguments not in the range [-1, +1].

double atan (double __x)
The **atan()** function computes the principal value of the arc tangent of *__x*. The returned value is in the range [-pi/2, pi/2] radians.

double atan2 (double __y, double __x)
The **atan2()** function computes the principal value of the arc tangent of *__y / __x*, using the signs of both arguments to determine the quadrant of the return value. The returned value is in the range [-pi, +pi] radians.

double cbrt (double __x)
The **cbrt()** function returns the cube root of *__x*.

double ceil (double __x)
The **ceil()** function returns the smallest integral value greater than or equal to *__x*, expressed as a floating-point number.

static double copysign (double __x, double __y) `[static]`
The **copysign()** function returns *__x* but with the sign of *__y*. They work even if *__x* or *__y* are NaN or zero.

double cos (double __x)
The **cos()** function returns the cosine of *__x*, measured in radians.

double cosh (double __x)
The **cosh()** function returns the hyperbolic cosine of *__x*.

double exp (double __x)
The **exp()** function returns the exponential value of *__x*.

double fabs (double __x)
The **fabs()** function computes the absolute value of a floating-point number *__x*.

double fdim (double __x, double __y)
The **fdim()** function returns *max(__x - __y, 0)*. If *__x* or *__y* or both are NaN, NaN is returned.

double floor (double __x)
The **floor()** function returns the largest integral value less than or equal to *__x*, expressed as a floating-point number.

double fma (double __x, double __y, double __z)
The **fma()** function performs floating-point multiply-add. This is the operation *(__x * __y) + __z*, but the intermediate result is not rounded to the destination type. This can sometimes improve the precision of a calculation.

double fmax (double __x, double __y)
The **fmax()** function returns the greater of the two values *__x* and *__y*. If an argument is NaN, the other argument is returned. If both arguments are NaN, NaN is returned.

double fmin (double __x, double __y)
The **fmin()** function returns the lesser of the two values *__x* and *__y*. If an argument is NaN, the other argument is returned. If both arguments are NaN, NaN is returned.

double fmod (double __x, double __y)
The function **fmod()** returns the floating-point remainder of *__x / __y*.

double frexp (double __x, int * __pexp)

The **frexp()** function breaks a floating-point number into a normalized fraction and an integral power of 2. It stores the integer in the `int` object pointed to by *__pexp*.

If *__x* is a normal float point number, the **frexp()** function returns the value `v`, such that `v` has a magnitude in the interval [1/2, 1) or zero, and *__x* equals `v` times 2 raised to the power *__pexp*. If *__x* is zero, both parts of the result are zero. If *__x* is not a finite number, the **frexp()** returns *__x* as is and stores 0 by *__pexp*.

Note:

This implementation permits a zero pointer as a directive to skip a storing the exponent.

double hypot (double __x, double __y)

The **hypot()** function returns *sqrt(__x*__x + __y*__y)*. This is the length of the hypotenuse of a right triangle with sides of length *__x* and *__y*, or the distance of the point (*__x*, *__y*) from the origin. Using this function instead of the direct formula is wise, since the error is much smaller. No underflow with small *__x* and *__y*. No overflow if result is in range.

static int isfinite (double __x) `[static]`

The **isfinite()** function returns a nonzero value if *__x* is finite: not plus or minus infinity, and not NaN.

int isinf (double __x)

The function **isinf()** returns 1 if the argument *__x* is positive infinity, -1 if *__x* is negative infinity, and 0 otherwise.

Note:

The GCC 4.3 can replace this function with inline code that returns the 1 value for both infinities (gcc bug #35509).

int isnan (double __x)

The function **isnan()** returns 1 if the argument *__x* represents a 'not-a-number' (NaN) object, otherwise 0.

double ldexp (double __x, int __exp)

The **ldexp()** function multiplies a floating-point number by an integral power of 2. It returns the value of *__x* times 2 raised to the power *__exp*.

double log (double __x)

The **log()** function returns the natural logarithm of argument *__x*.

double log10 (double __x)

The **log10()** function returns the logarithm of argument *__x* to base 10.

long lrint (double __x)

The **lrint()** function rounds *__x* to the nearest integer, rounding the halfway cases to the even integer direction. (That is both 1.5 and 2.5 values are rounded to 2). This function is similar to rint() function, but it differs in type of return value and in that an overflow is possible.

Returns:

The rounded long integer value. If *__x* is not a finite number or an overflow was, this realization returns the `LONG_MIN` value (0x80000000).

long lround (double __x)

The **lround()** function rounds *__x* to the nearest integer, but rounds halfway cases away from zero (instead of to the nearest even integer). This function is similar to **round()** function, but it differs in type of return value and in that an overflow is possible.

Returns:

The rounded long integer value. If *__x* is not a finite number or an overflow was, this realization returns the `LONG_MIN` value (0x80000000).

double modf (double __x, double * __iptr)

The **modf()** function breaks the argument *__x* into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a double in the object pointed to by *__iptr*.

The **modf()** function returns the signed fractional part of *__x*.

Note:

This implementation skips writing by zero pointer. However, the GCC 4.3 can replace this

function with inline code that does not permit to use NULL address for the avoiding of storing.

float modff (float __x, float * __iptr)
An alias for **modf()**.

double pow (double __x, double __y)
The function **pow()** returns the value of *__x* to the exponent *__y*.

double round (double __x)
The **round()** function rounds *__x* to the nearest integer, but rounds halfway cases away from zero (instead of to the nearest even integer). Overflow is impossible.

Returns:
The rounded value. If *__x* is an integral or infinite, *__x* itself is returned. If *__x* is `NaN`, then `NaN` is returned.

int signbit (double __x)
The **signbit()** function returns a nonzero value if the value of *__x* has its sign bit set. This is not the same as '*__x* < 0.0', because IEEE 754 floating point allows zero to be signed. The comparison '-0.0 < 0.0' is false, but 'signbit (-0.0)' will return a nonzero value.

double sin (double __x)
The **sin()** function returns the sine of *__x*, measured in radians.

double sinh (double __x)
The **sinh()** function returns the hyperbolic sine of *__x*.

double sqrt (double __x)
The **sqrt()** function returns the non-negative square root of *__x*.

float sqrtf (float)
An alias for **sqrt()**.

double square (double __x)
The function **square()** returns *__x* * *__x*.

Note:
This function does not belong to the C standard definition.

double tan (double __x)
The **tan()** function returns the tangent of *__x*, measured in radians.

double tanh (double __x)
The **tanh()** function returns the hyperbolic tangent of *__x*.

double trunc (double __x)
The **trunc()** function rounds *__x* to the nearest integer not larger in absolute value.

Author

Generated automatically by Doxygen for avr-libc from the source code.