TMS320C67x FastRTS Library Programmer's Reference

SPRU100A
October 2002

PRINTED WITH SOY INK™

TEXAS INSTRUMENTS

Printed on Recycled Paper

IMPORTANT NOTICE

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

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

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

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

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

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

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Preface

Read This First

About This Manual

Welcome to the TMS320C67x Fast Run-Time-Support Library, or FastRTS library for short. The FastRTS library is a collection of 26 optimized floating-point math functions for the TMS320C67x device. This source code library includes C-callable (ANSI-C-language compatible) optimized versions of the floating-point math functions included in previous run-time-support libraries.

How to Use This Manual

The information in this document describes the contents of the TMS320C67x FastRTS library in several different ways.

- ❑ Chapter 1 provides a brief introduction to the C67x FastRTS library, shows the organization of the routines contained in the library, and lists the features and benefits of the library.
- ❑ Chapter 2 provides information on how to install, use, and rebuild the C67x FastRTS library.
- ❑ Chapter 3 provides a quick overview of all FastRTS functions for easy reference. The information shown for each function includes the name, a brief description, and a page reference for obtaining more detailed information.
- ❑ Chapter 4 provides a list of the routines within the FastRTS library organized into functional categories. The functions are listed in alphabetical order and include syntax, file defined in, description, functions called, and special cases.
- ❑ Appendix A provides information about warranty issues, software updates, and customer support.

Notational Conventions

This document uses the following conventions:

- Program listings, program examples, and interactive displays are shown in a `special typeface`.
- In syntax descriptions, the function appears in a **bold typeface** and the parameters appear in plainface.
- The TMS320C67x is also referred to in this reference guide as the C67x.

Related Documentation From Texas Instruments

The following books describe the TMS320C6x devices and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477-8924. When ordering, please identify the book by its title and literature number. Many of these documents can be found on the Internet at http://www.ti.com.

TMS320C62x/C67x Technical Brief (literature number SPRU197) gives an introduction to the TMS320C62x™ and TMS320C67x™ digital signal processors, development tools, and third-party support.

TMS320C6000 CPU and Instruction Set Reference Guide (literature number SPRU189) describes the TMS320C6000™ CPU architecture, instruction set, pipeline, and interrupts for these digital signal processors.

TMS320C6201/C6701 Peripherals Reference Guide (literature number SPRU190) describes common peripherals available on the TMS320C6201 and TMS320C6701 digital signal processors. This book includes information on the internal data and program memories, the external memory interface (EMIF), the host port interface (HPI), multichannel buffered serial ports (McBSPs), direct memory access (DMA), enhanced DMA (EDMA), expansion bus, clocking and phase-locked loop (PLL), and the power-down modes.

TMS320C6000 Programmer's Guide (literature number SPRU198) describes ways to optimize C and assembly code for the TMS320C6000™ DSPs and includes application program examples.

TMS320C6000 Assembly Language Tools User's Guide (literature number SPRU186) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the TMS320C6000™ generation of devices.

TMS320C6000 Optimizing Compiler User's Guide (literature number SPRU187) describes the TMS320C6000™ C compiler and the assembly optimizer. This C compiler accepts ANSI standard C source code and produces assembly language source code for the TMS320C6000 generation of devices. The assembly optimizer helps you optimize your assembly code.

TMS320C6000 Chip Support Library (literature number SPRU401) describes a set of application programming interfaces (APIs) used to configure and control all on-chip peripherals.

Trademarks

The Texas Instruments logo and Texas Instruments are registered trademarks of Texas Instruments Incorporated. Trademarks of Texas Instruments include: TI, Code Composer Studio, TMS320, TMS320C6000, and TMS320C67x.

Contents

Tables

Notes, Cautions, and Warnings

Chapter 1

Introduction

This chapter provides a brief introduction to the C67x Fast Run-Time-Support (FastRTS) Library, shows the organization of the routines contained in the FastRTS library, and lists the features and benefits of the FastRTS library.

Topic **Page**

1.1 Introduction to the C67x FastRTS Library

The C67x FastRTS library is an optimized floating-point math function library for C programmers using TMS320C67x devices. These routines are typically used in computationally intensive real-time applications where optimal execution speed is critical. By using these routines instead of the routines found in the existing run-time-support libraries, you can achieve execution speeds considerably faster without rewriting existing code.

The FastRTS library includes all floating-point math routines currently provided in existing run-time-support libraries for C6000. These new functions can be called with the current run-time-support library names or the new names included in the FastRTS library.

Single-precision and double-precision versions of the routines are available:

Table 1-1. FastRTS Library Functions

Single Precision	Double Precision
atanf	atan
atan2f	atan2
cosf	cos
expf	exp
exp2f	exp2
exp10f	exp10
logf	log
log2f	log2
log10f	log10
powf	pow
recipf	recip
rsqrtf	rsqrt
sinf	sin

1.2 Features and Benefits

The FastRTS library provides the following features and benefits:

- Hand-coded assembly-optimized routines
- C-callable routines, which are fully compatible with the TMS320C6000 compiler
- Provided functions are tested against C model and existing run-time-support functions

Chapter 2

Installing and Using FastRTS

This chapter provides information on the contents of the FastRTS archive, and how to install, use, and rebuild the C67x FastRTS library.

Topic **Page**

2.1 FastRTS Library Contents

The C67xFastRTS.exe installs the following file structure:

lib	Directory containing the following library files:
fastrts67x.lib	Little-endian library file
fastrts67xe.lib	Big-endian library file
fastrts67x.src	Source archive file
include	Directory containing the following include files:
fastrts67x.h	Alternative entry header file
recip.h	Header file for reciprocal functions
doc	Directory containing the following document files:
spru100.pdf	PDF document of API
License.doc	License agreement

2.2 How to Install the FastRTS Library

To install the FastRTS libary, follow these steps:

Step 1: Open the file C67xFastRTS.exe.

Step 2: Click Yes to install the library.

Step 3: Click Next to continue with the Install Shield Wizard.

Step 4: Read the Software Licenses and choose either “I accept” or “I don’t accept.”

Step 5: Click Next to continue.

If you selected “I accept,” the installation will continue.
If you selected “I don’t accept,” the installation cancels.

Step 6: Choose the location where you would like to install the library. The wizard will install the header files in the include directory, documentation in the doc directory, and the library and source files in the lib directory.

The default install location is c:\ti. Libraries will be installed in c:\ti\c6700\mthlib, and documentation will be in c:\ti\docs\pdf.

Step 7: Click Next.

Step 8: If the library has already been installed, you will be prompted to decide whether to replace the files or not. Click Yes to update the library.

Step 9: The Install Shield will complete the installation. When the installation is complete, click Finish.

2.3 Using the FastRTS Library

Before using the FastRTS library functions, you need to update your linker command file. If you want to use the FastRTS functions in place of the existing versions of these functions, the FastRTS library must be linked in before the existing run-time-support library.

Ensure that you link with the correct run-time-support library and the FastRTS library for little-endian code by adding the following line in your linker command file before the line linking the current run-time-support library:

```
-lfastrts67x.lib
```

For big-endian code, add the following line in your linker command file before the line linking the current run-time-support library:

```
-lfastrts67xe.lib
```

2.3.1 FastRTS Library Arguments and Data Types

FastRTS Types

Table 2-1 shows the data types handled by the FastRTS.

Table 2-1. FastRTS Data Types

Name	Size (bits)	Type	Minimum	Maximum
IEEE float	32	floating point	1.17549435e-38	3.40282347e+38
IEEE double	64	floating point	2.2250738585072014e-308	1.7976931348623157e+308

FastRTS Arguments

The C67x FastRTS functions operate on single value arguments. The single-precision versions operate on IEEE float arguments and the double-precision versions operate on IEEE double arguments. The functions atan2 and pow require two arguments.

2.3.2 Calling a FastRTS Function From C

In addition to correctly installing the FastRTS software, you must follow these steps to include a FastRTS function in your code:

- Include the function header file corresponding to the FastRTS function:
  - The fastrts67x.h header file must be included if you use the special FastRTS function names.
  - The recip.h header file must be included if the recip, recipdp, recipf, or recipsp function is called.
  - The math.h header file must be included if you are using the standard run-time-support function names.
- Link your code with fastrts67x.lib for little-endian code or fastrts67xe.lib for big-endian code.
- Use the correct linker command file for the platform you use. Remember, the FastRTS library replaces only a subset of the functions in the current run-time-support libraries. Therefore, fastrts67x.lib or fastrts67xe.lib must be linked in before rts6700.lib or rts6700e.lib.

For example, if you call the cos FastRTS function, you would add:

```
#include <math.h>
```

in your C file and compile and link using

```
cl6x main.c –z –o drv.out –lfastrts67x.lib –rts6701.lib
```

Note: Adding FastRTS in Code Composer Studio

If you set up a project under Code Composer Studio, you can add the FastRTS library to your project by selecting Project→Add Files to Project and choosing fastrts67x.lib or fastrts67xe.lib.

2.3.3 Calling a FastRTS Function From Assembly

The C67x FastRTS functions were written to be used from C. Calling the functions from assembly language source code is possible as long as the calling function conforms to the Texas Instruments C67x C compiler calling conventions. For more information, refer to the *Run-Time Environment* chapter of the *TMS320C6000 Optimizing C/C++ Compiler User's Guide*.

2.4 How to Rebuild the FastRTS Library

If you want to rebuild the FastRTS library (for example, because you modified the source file contained in the archive), you must use the mk6x utility as follows for little endian and big endian versions:

```
mk6x fastrts67x.src -l fastrts67x.lib
mk6x –me fastrts67x.src -l fastrts67xe.lib
```

FastRTS Library Functions Tables

This chapter provides tables containing all FastRTS functions, a brief description of each, and a page reference for more detailed information.

Topic **Page**

3.1 Arguments and Conventions Used

The following conventions have been followed when describing the arguments for each individual function:

Table 3-1. Argument Conventions

Argument	Description
x,y,z	Argument reflecting input data
r	Argument reflecting output data

3.2 FastRTS Functions

The routines included in the FastRTS library are provided as both single- and double-precision versions. SP is used in the following tables to identify the single-precision functions. DP is used to identify the double-precision functions. Listed in the table below are current run-time-support library function names and the alternate function names for the Fast RTS library. Either name can be used to call the FastRTS version of the function.

Table 3-2. FastRTS Function Names Comparison

	Current Name		Alternate Name		
Description	**SP**	**DP**	**SP**	**DP**	**Page**
arc tangent of one argument	atanf	atan	atansp	atandp	4-2
arc tangent of two arguments	atan2f	atan2	atan2sp	atan2dp	4-3
cosine of a radian argument	cosf	cos	cossp	cosdp	4-4
exponential base e	expf	exp	expsp	expdp	4-5
exponential base 10	exp10f	exp10	exp10sp	exp10dp	4-6, 4-7
exponential base 2	exp2f	exp2	exp2sp	exp2dp	4-7, 4-8
logarithm base e	logf	log	logsp	logdp	4-8
logarithm base 10	log10f	log10	log10sp	log10dp	4-10
logarithm base 2	log2f	log2	log2sp	log2dp	4-11, 4-12
power = X raised to power Y	powf	pow	powsp	powdp	4-13, 4-14
reciprocal = 1/argument	recipf†	recip†	recipsp	recipdp	4-15
reciprocal of square root	rsqrtf	rsqrt	rsqrtsp	rsqrtdp	4-16, 4-17
sine of a radian argument	sinf	sin	sinsp	sindp	4-17, 4-18

† The FastRTS functions recipf and recip are not defined in the corresponding rts67xx.lib

Some of the RTS functions call the routines listed in Table 3-3 for improved performance. These functions are described in *TMS32067xx Divide and Square Root Floating-Point Functions* (literature number SPRA516).

Table 3-3. Divide and Square Root Floating-Point Functions

	Run-Time-Support Name		Alternate Name		
Description	**SP**	**DP**	**SP**	**DP**	**Page**
division of two arguments	_divf	_divd	divsp	divdp	4-19
square root	sqrtf	sqrt	sqrtsp	sqrtdp	4-20

Chapter 4

FastRTS Reference

This chapter provides a list of the functions within the FastRTS library. The functions are listed in alphabetical order and include syntax, file defined in, description, functions called, and special cases.

Topic **Page**

4.1 General FastRTS Functions

atan/atandp — *Double-Precision Polar Arc Tangent*

Syntax--Standard #include <math.h>

double **atan**(double z);

Syntax--FastRTS #include <fastrts67x.h>

double **atan**(double z); or double **atandp**(double z)

Defined in atan2dp.asm

Description The atan and atandp functions return the arc tangent of a floating-point argument z. The return value is an angle in the range [-π/2, π/2] radians.

Functions none

Special Cases If | z | < 1.49e-8 = 2^{-26}, then the return value is z for small angles.

atanf/atansp — *Single-Precision Polar Arc Tangent*

Syntax--Standard #include <math.h>

float **atanf**(float z);

Syntax--FastRTS #include <fastrts67x.h>

float **atanf**(float z); or float **atansp**(float z);

Defined in atan2sp.asm

Description The atanf and atansp functions return the arc tangent of a floating-point argument z. The return value is an angle in the range [-π/2, π/2] radians.

Functions none

Special Cases If | z | < 2.44e-4 = 2^{-12}, then the return value is z for small angles.

atan2/atan2dp

Double-Precision Cartesian Arc Tangent

Syntax--Standard

#include <math.h>

double **atan2**(double y, double x) ;

Syntax--FastRTS

#include <fastrts67x.h>

double **atan2f**(double y, double x); or double **atan2dp**(double y, double x);

Defined in

atan2dp.asm

Description

The atan2 and atan2dp functions return the arc tangent of the floating-point arguments y/x. The return value is an angle in the range of [$-\pi$, π] radians.

Functions

none

Special Cases

If | y/x | < 1.49e-8 = 2^{-26}, then the return value is y/x for small angles.

If y = 0, then the return value is 0 independent of the value of x (including 0).

If x = 0, then the return value is +/-π/2 as determined by the sign of a non-zero y.

atan2f/atan2sp

Single-Precision Cartesian Arc Tangent

Syntax--Standard

#include <math.h>

float **atan2**(float y, float x) ;

Syntax--FastRTS

#include <fastrts67x.h>

float **atan2f**(float y, float x); or float **atan2sp**(float y, float x);

Defined in

atan2sp.asm

Description

The atan2f and atan2sp functions return the arc tangent of the floating-point arguments y/x. The return value is an angle in the range of [$-\pi$, π] radians.

Functions

none

Special Cases

If | y/x | < 2.44e-4 = 2^{-12}, then the return value is y/x for small angles.

If y = 0, then the return value is 0 independent of the value of x (including 0).

If x = 0, then the return value is +/- π/2 as determined by the sign of a non-zero y.

cos/cosdp — *Double-Precision Cosine*

Syntax--Standard

#include <math.h>

double **cos**(double z) ;

Syntax--FastRTS

#include <fastrts67x.h>

double **cosf**(double z); or double **cosdp**(double z);

Defined in

sindp.asm

Description

The cos and cosdp functions return the cosine of a floating-point argument z. The angle z is expressed in radians. The return value is in the range of [-1.0 and +1.0]. An argument with a large magnitude may produce a result with little or no significance.

Functions

sin (or sindp) using the identity:

cos(z) = **sin**(| z | + π/2) = **sin**(W)

where W = | z | + π/2. The cos routine continues into the **sin** routine without making a call.

Special Cases

If | W | < 9.536743e-7 = 2^{-20}, then the return value is W for small angles.

If | W | > 1.0737e+9 = 2^{+30}, then the return value is zero for large angles.

cosf/cossp — *Single-Precision Cosine*

Syntax--Standard

#include <math.h>

float **cosf**(float z) ;

Syntax--FastRTS

#include <fastrts67x.h>

float **cosf**(float z); or float **cossp**(float z);

Defined in

sinsp.asm

Description

The cosf and cosp functions return the cosine of a floating-point argument z. The angle z is expressed in radians. The return value is in the range of [-1.0 and +1.0]. An argument with a large magnitude may produce a result with little or no significance.

Functions

sinf (or sinsp) using the identity:

cosf(z) = **sinf**(| z | + π/2) = **sinf**(W)

where W = | z | + π/2. The cosf routine continues into the **sinf** routine without making a call.

Special Cases

If | W | < 2.44e-4 = 2^{-12}, then the return value is W for small angles.

If | W | > 1.04858e+6 = 2^{+20}, then the return value is zero for large angles.

exp/expdp — *Double-Precision Exponential Base e*

Syntax--Standard

#include <math.h>

double **exp**(double z) ;

Syntax--FastRTS

#include <fastrts67x.h>

double **expf**(double z); or double **expdp**(double z);

Defined in

expdp.asm

Description

The exp and expdp functions return the exponential function of a real floating-point argument z. The return value is the number e raised to power z. If the magnitude of z is too large, the maximum double-precision floating-point number (1.797693e+308 = 2^{+1024}) is returned.

Functions

none

Special Cases

If | z | < 1.11e-16 = 2^{-53}, then the return value is 1.0 for small arguments.

If z < -708.3964 = minimum $\mathbf{log}_{\mathbf{e}}$ (2.225e-308 = 2^{-1022}), then the return value is 0.0.

If z > +709.7827 = maximum $\mathbf{log}_{\mathbf{e}}$ (1.797693e+308 = 2^{+1024}), then the return value is 1.797693e+308 = 2^{+1024} (maximum double-precision floating-point number).

expf/expsp — *Single-Precision Exponential Base e*

Syntax--Standard

#include <math.h>

float **expf**(float z) ;

Syntax--FastRTS

#include <fastrts67x.h>

float **expf**(float z); or float **expsp**(float z);

Defined in

expsp.asm

Description

The expf and expsp functions return the exponential function of a real floating-point argument z. The return value is the number e raised to power z. If the magnitude of z is too large, the maximum single-precision floating-point number (3.402823e+38 = 2^{+128}) is returned.

Functions

none

Special Cases If | z | < 9.313e-10 = 2^{-30}, then the return value is 1.0.

If z < -87.3365 = minimum $\log_e$ (1.175e-38 = 2^{-126}), then the return value is 0.0.

If z > +88.7228 = maximum $\log_e$ (3.402823e+38 = 2^{+128}), then the return value is 3.402823e+38 = 2^{+128} (maximum single-precision floating-point number).

exp10/exp10dp — *Double-Precision Exponential Base 10*

Syntax--Standard

```
#define _TI_ENHANCED_MATH_H 1
#include <math.h>

double exp10( double z ) ;
```

Syntax--FastRTS

```
#include <fastrts67x.h>

double exp10f( double z ); or double exp10dp( double z );
```

Defined in expdp.asm

Description The exp10 and exp10dp functions return the exponential function of a real floating-point argument z. The return value is the number 10 raised to power z. If the magnitude of z is too large, the maximum double-precision floating-point number (1.797693e+308 = 2^{+1024}) is returned.

Functions _divd (or DIVDP) using the large memory model (32-bit addresses)

The shared **exp** kernel is used (without making a call) in the following identity:

exp10(z) = exp(z * 2.302585093 …) = exp(W) where W = z * 2.302585093 … and 2.302585093 … = $\log_e$ (10).

Special Cases If | W | < 1.11e-16 = 2^{-53}, then the return value is 1.0 for small arguments.

If W < -708.3964 = minimum $\log_e$ (2.225e-308 = 2^{-1022}), then the return value is 0.0.

If W > +709.7827 = maximum $\log_e$ (1.797693e+308 = 2^{+1024}), then the return value is 1.797693e+308 = 2^{+1024} (maximum double-precision floating-point number).

exp10f/exp10sp — *Single-Precision Exponential Base 10*

Syntax--Standard

```
#define _TI_ENHANCED_MATH_H 1
#include <math.h>
```

float **exp10f**(float z) ;

Syntax--FastRTS

```
#include <fastrts67x.h>
```

float **exp10f**(float z); or float **exp10sp**(float z);

Defined in expsp.asm

Description The exp10f and exp10sp functions return the exponential function of a real floating-point argument z. The return value is the number 10 raised to power z. If the magnitude of z is too large, the maximum single-precision floating-point number (3.4028323e+38 = 2^{+128}) is returned.

Functions none

Special Cases If | W | < 9.31e-10 = 2^{-30}, then the return value is 1.0 for small arguments.

If W < -87.3365 = minimum $\log_e$ (1.175e-38 = 2^{-126}), then the return value is 0.0.

If W > +88.7228 = maximum $\log_e$ (3.402823e+38 = 2^{+128}), then the return value is 3.402823e+38 = 2^{+128} (maximum single-precision floating-point number).

exp2/exp2dp — *Double-Precision Exponential Base 2*

Syntax--Standard

```
#define _TI_ENHANCED_MATH_H 1
#include <math.h>
```

double **exp2**(double z) ;

Syntax--FastRTS

```
#include <fastrts67x.h>
```

double **exp2f**(double z); or double **exp2dp**(double z);

Defined in expdp.asm

Description The exp2 and exp2dp functions return the exponential function of a real floating-point argument z. The return value is the number 2 raised to power z. If the magnitude of z is too large, the maximum double-precision floating-point number(1.797693e+308 = 2^{+1024}) is returned.

Functions none

Special Cases If | W | < 1.11e-16 = 2^{-53}, then the return value is 1.0 for small arguments.

If W < -708.3964 = minimum $\mathbf{log}_{\mathbf{e}}$ (2.225e-308 = 2^{-1022}), then the return value is 0.0.

If W > +709.7827 = maximum $\mathbf{log}_{\mathbf{e}}$ (1.797693e+308 = 2^{+1024}), then the return value is 1.797693e+308 = 2^{+1024} (maximum double-precision floating-point number).

exp2f/exp2sp *Single-Precision Exponential Base 2*

Syntax--Standard

```
#define _TI_ENHANCED_MATH_H 1
#include <math.h>
```

float **exp2f**(float z) ;

Syntax--FastRTS

```
#include <fastrts67x.h>
```

float **exp2f**(float z); or float **exp2sp**(float z);

Defined in expsp.asm

Description The exp2f and exp2sp functions return the exponential function of a real floating-point argument z. The return value is the number 2 raised to power z. If the magnitude of z is too large, the maximum single-precision floating-point number (3.402823e+38 = 2^{+128}) is returned.

Functions none

Special Cases If | W | < 9.3e-10 = 2^{-30}, then the return value is 1.0 for small arguments.

If W < -87.3365 = minimum $\mathbf{log}_{\mathbf{e}}$ (1.175e-38 = 2^{-126}), then the return value is 0.0.

If W > +88.7228 = maximum $\mathbf{log}_{\mathbf{e}}$ (3.402823e+38 = 2^{+128}), then the return value is 3.402823e+38 = 2^{+128} (maximum single-precision floating-point number).

log/logdp *Double-Precision Natural Logarithm*

Syntax--Standard

```
#include <math.h>
```

double **log**(double z) ;

Syntax--FastRTS #include <fastrts67x.h>

double **logf**(double z); or double **logdp**(double z);

Defined in logdp.asm

Description The log and logdp functions return the natural logarithm_e of a real floating-point argument z. If z is not positive, the negative maximum double-precision floating-point number (-1.797693e+308 = $-1*2^{+1024}$) is returned.

Functions none

Special Cases If z < = 0, then the return value is -1.797693e+308 = $-1*2^{+1024}$ (largest double-precision floating-point number with a negative sign).

If z < 2.225e-308 = 2^{-1022}, then the return value is -708.3964 = minimum $\mathbf{log_e}$ (+2.225e-308 = 2^{-1022}).

If z > 8.9885e+307 = 2^{+1023}, then the return value is +709.7827 = maximum $\mathbf{log_e}$ (+1.797693e+308 = 2^{+1024}).

logf/logsp

Single-Precision Natural Logarithm

Syntax--Standard #include <math.h>

float **logf**(float z);

Syntax--FastRTS #include <fastrts67x.h>

float **logf**(float z); or float **logsp**(float z);

Defined in logsp.asm

Description The logf and logsp functions return the natural logarithm of a real floating-point argument z. If z is not positive, the negative maximum single-precision floating-point number (-3.402823e+38 = $-1*2^{+128}$) is returned.

Functions none

Special Cases If z < = 0, then the return value is -3.402823e+38 = $-1*2^{+128}$ (largest single-precision floating-point number with a negative sign).

If z < 1.175e-38 = 2^{-126}, then the return value is –87.3365 = minimum $\mathbf{log_e}$ (+1.175e-38 = 2^{-126}).

If z > 1.7014e+38 = 2^{+127}, then the return value is +88.7228 = maximum $\mathbf{log_e}$ (+3.402823e+38 = 2^{+128}).

log10/log10dp

Double-Precision Common Logarithm Base 10

Syntax--Standard

#include <math.h>

double **log10**(double z) ;

Syntax--FastRTS

#include <fastrts67x.h>

double **log10f**(double z); or double **log10dp**(double z);

Defined in

logdp.asm

Description

The log10 and log10dp functions return the common logarithm_{10} of a real floating-point argument z. If z is not positive, the negative maximum double-precision floating-point number (-1.797693e+308 = $-1*2^{+1024}$) is returned.

Functions

none

Special Cases

If z < = 0, then the return value is -1.797693e+308 = $-1*2^{+1024}$ (largest double-precision floating-point number with a negative sign).

If z < 2.225e-308 = 2^{-1022}, then the return value is -708.3964 = minimum $\mathbf{log_e}$ (+2.225e-308 = 2^{-1022}) and scaled by $\log_{10}$ (e) = -307.65265.

log10f/log10sp

Single-Precision Common Logarithm Base 10

Syntax--Standard

#include <math.h>

float **log10f**(float z) ;

Syntax--FastRTS

#include <fastrts67x.h>

float **log10f**(float z); or float **log10sp**(float z);

Defined in

logsp.asm

Description

The log10f and log10sp functions return the common logarithm_{10} of a real floating-point argument z. If z is not positive, the negative maximum single-precision floating-point number (-3.402823e+38 = $-1*2^{+128}$) is returned.

Functions

none

Special Cases

If z < = 0, then the return value is -3.402823e+38 = $-1*2^{+128}$ (largest single-precision floating-point number with a negative sign).

If z < 1.1755e-38 = 2^{-126}, then the return value is –87.3365 = minimum $\mathbf{logf_e}$ (+1.175e-38 = 2^{-126}) and scaled by $\mathbf{logf_{10}}$ (e) = -37.92978.

If z > 1.70e+38 = 2^{+127}, then the return value is +88.7228 = maximum $\mathbf{logf_e}$ (+3.402823e+38 = 2^{+128}) and scaled by $\mathbf{logf_{10}}$ (e) = +38.53184.

log2/log2dp

Double-Precision Binary Logarithm Base 2

Syntax--Standard

#define _TI_ENHANCED_MATH_H 1
#include <math.h>

double **log2**(double z) ;

Syntax--FastRTS

#include <fastrts67x.h>

double **log2f**(double z); or double **log2dp**(double z);

Defined in

logdp.asm

Description

The log2 and log2dp functions return the binary logarithm_2 of a real floating-point argument z. If z is not positive, the negative maximum double-precision floating-point number (-1.797693e+308 = -1*2^{+1024}) is returned.

Functions

none

Special Cases

If z < = 0, then the return value is -1.797693e+308 = -1*2^{+1024} (largest double-precision floating-point number with a negative sign).

If z < 2.225e-308 = 2^{-1022}, then the return value is -708.3964 = minimum $\mathbf{log_e}$ (+2.225e-308 = 2^{-1022}) and scaled by $\mathbf{log_2}$ (e) = -1022.

If z > 8.9885e+307 = 2^{+1023}, then the return value is +709.7827 = maximum $\mathbf{log_e}$ (+1.797693e+308 = 2^{+1024}) and scaled by $\mathbf{log_2}$ (e) = +1024.

log2f/log2sp — *Single-Precision Binary Logarithm Base 2*

Syntax--Standard

```
#define _TI_ENHANCED_MATH_H 1
#include <math.h>
```

float **log2f**(float z) ;

Syntax--FastRTS

```
#include <fastrts67x.h>
```

float **log2f**(float z); or float **log2sp**(float z) ;

Defined in

logsp.asm

Description

The log2f and log2sp functions return the binary logarithm_2 of a real floating-point argument z. If z is not positive, the negative maximum single-precision floating-point number ($-3.402823e+38 = -1*2^{+128}$) is returned.

Functions

none

Special Cases

If z < = 0, then the return value is $-3.402823e+38 = -1*2^{+128}$ (largest single-precision floating-point number with a negative sign).

If $z < 1.175e-38 = 2^{-126}$, then the return value is –87.3365 = minimum $\mathbf{logf_e}$ ($+1.175e-38 = 2^{-126}$) and scaled by $\mathbf{logf_2}$ (e) = -126.

If $z > 1.70e+38 = 2^{+127}$, then the return value is +88.7228 = maximum $\mathbf{logf_e}$ ($+3.402823e+38 = 2^{+128}$) and scaled by $\mathbf{logf_2}$ (e) = +128.

pow/powdp — *Double-Precision Raise to a Power*

Syntax--Standard

#include <math.h>

double **pow**(double x double y) ;

Syntax--FastRTS

#include <fastrts67x.h>

double **powf**(double x double y); or double **powdp**(double x double y);

Defined in

powdp.asm

Description

The pow and powdp functions return x raised to the power y. The functions are equivalent to:

pow(x,y) = $\mathbf{exp_e}$ (y * $\mathbf{log_e}$ (x)) = $\mathbf{exp_e}$ (W) where W = y * $\mathbf{log_e}$ (x)

If x < 0, then y must have an integer value, else NaN is returned. The compound restrictions of $\mathbf{log_e}$ and $\mathbf{exp_e}$ apply to the returned answer.

Functions

$\log_e$, $\exp_e$, and _divd (or divdp) using the large memory model (32-bit addresses)

Special Cases

The following order of tests are observed:

If y = 0, return 1.0 (x is ignored).

If | x | > 8.9885e+307 = 2^{+1023}, the return value is Infinity (y is ignored).

If | x | < 2.225e-308 = 2^{-1022}, then the return value is 0 (y is ignored).

If x < 0, and y is not an integer value, then **NaN** is returned.

If x < 0, and y is a 32-bit integer value, $-1^{y} * |x|^{y}$ is formed.

Form W = y * $\mathbf{log_e}$ (|x|).

If W > 709.089 = maximum $\mathbf{log_e}$ (+8.988e+307 = 2^{+1023}), Infinity is returned.

If W < -708.396 = minimum $\mathbf{log_e}$ (+2.225e-307 = 2^{-1022}), then 0 is returned.

Otherwise, $\mathbf{exp_e}$ (W) = $\mathbf{exp_e}$ (y * $\mathbf{log_e}$ (x)) = x^{y} is returned.

powf/powsp — *Single-Precision Raise to a Power*

Syntax--Standard

#include <math.h>

float **powf**(float x, float y) ;

Syntax--FastRTS

#include <fastrts67x.h>

float **powf**(float x, float y); or float **powsp**(float x, float y);

Defined in

powsp.asm

Description

The powf and powsp functions return x raised to the power y. This is equivalent to:

powf(x,y) = expf $_e$ (y * logf $_e$ (x)) = expf $_e$ (W)

where W = y * logf $_e$ (x). If x < 0, then y must have an integer value, else NaN is returned. The compound restrictions of **logf$_e$** and **expf$_e$** apply to the returned answer.

Functions

logf $_e$, expf $_e$, and _divf (or DIVSP) using the large memory model (32-bit addresses)

Special Cases

The following order of tests are observed:

If y = 0, return 1.0 (x is ignored).

If | x | > 1.701e+38 = 2^{+127}, the return value is Infinity (y is ignored).

If | x | < 1.175e-38 = 2^{-126}, then the return value is 0 (y is ignored).

If x < 0, and y is not an integer value, then **NaN** is returned.

If x < 0, and y is a 32-bit integer value, -1^{y}* $|x|^{y}$ is formed.

Form W = y * **logf $_e$** (|x|).

If W > 88.02969 = maximum **logf $_e$** (+1.701e+38 = 2^{+127}), Infinity is returned.

If W < –87.3365 = minimum **logf $_e$** (+1.175e-38 = 2^{-126}), then 0 is returned.

Otherwise, **expf $_e$** (W) = **expf $_e$** (y * **logf $_e$** (x)) = x^{y} is returned.

recip/recipdp *Double-Precision Reciprocal*

Syntax--FastRTS

#include <fastrts67x.h>
#include <recip.h>

double **recipf**(double z); or double **recipdp**(double z);

Note: recip Function

The recip function is not defined in the rts6700.lib or rts6700e.lib file.

Defined in recipdp.asm

Description The recip and recipdp functions return the reciprocal of a floating-point argument z.

Functions none

Special Cases If | z | < 2.225e-308 = 2^{-1022}, then the return value for small arguments is **NaN** = **Not-a-Number** (exponent and mantissa are all ones) > maximum double-precision floating point value +/- 1.797693e+308 = +/-1* 2^{+1024}.

If | z | > 1.797693e+308 = 2^{+1024}, then the return value is zero for large arguments.

recipf/recipsp *Single-Precision Reciprocal*

Syntax--FastRTS

#include <fastrts67x.h>
#include <recip.h>

float **recipf**(float z); or float **recipsp**(float z);

Note: recipf Function

The recipf function is not defined in the rts6700.lib or rts6700e.lib file.

Defined in recipsp.asm

Description The recipf and recipsp functions return the reciprocal of a floating-point argument z.

Functions none

Special Cases If | z | < 1.1755e-38 = 2^{-126}, then the return value for small arguments is **NaN** = **Not-a-Number** (exponent and mantissa are all ones) > the maximum single-precision floating point value +/- 3.402823e+38 = +/-1 * 2^{+128}.

If | z | > 3.402823e+38 = 2^{+128}, then the return value is zero for large arguments.

rsqrt/rsqrtdp — *Double-Precision Reciprocal Square Root*

Syntax--Standard

```
#define _TI_ENHANCED_MATH_H 1
#include <math.h>

double rsqrt( double z ) ;
```

Syntax--FastRTS

```
#define _TI_ENHANCED_MATH_H 1
#include <fastrts67x.h>

double rsqf( double z ); or double rsqrt( double z );
double rsqrtf( double z ); or double rsqrtdp( double z );
```

Defined in rsqrtdp.asm

Description The rsqrt and rsqrtdp functions return the reciprocal square root of a real floating-point argument z. The absolute value of z is used.

Functions none

Special Cases If | z | < 2.225e-308 = 2^{-1022}, then the return value for small arguments is **NaN** = **Not-a-Number** (exponent and mantissa are all ones) > the maximum double-precision floating point value 1.797693e+308 = 2^{+1024}.

If | z | > 1.797693e+308 = 2^{+1024}, then the return value is zero for large arguments.

rsqrtf/rsqrtsp — *Single-Precision Reciprocal Square Root*

Syntax--Standard

```
#define _TI_ENHANCED_MATH_H 1
#include <math.h>
```

float **rsqrtf**(float z) ;

Syntax--FastRTS

```
#define _TI_ENHANCED_MATH_H 1
#include <fastrts67x.h>
```

float **rsqrtf**(float z); or float **rsqrtsp**(float z);

Defined in

rsqrtsp.asm

Description

The rsqrtf and rsqrtsp functions return the reciprocal square root of a real floating-point argument z. The absolute value of z is used.

Functions

none

Special Cases

If | z | < 1.1755e-38 = 2^{-126}, then the return value for small arguments is **NaN** = **Not-a-Number** (exponent and mantissa are all ones) > maximum single-precision floating point value 3.402823e+38 = 2^{+128}.

If | z | > 3.402823e+38 = 2^{+128}, then the return value is zero for large arguments.

sin/sindp — *Double-Precision Sine*

Syntax--Standard

```
#include <math.h>
```

double **sin**(double z) ;

Syntax--FastRTS

```
#include <fastrts67x.h>
```

double **sinf**(double z); or double **sindp**(double z);

Defined in

sindp.asm

Description

The sin and sindp functions return the sine of a floating-point argument z. The angle z is expressed in radians. The return value is in the range of [-1.0 and +1.0]. An argument with a large magnitude may produce a result with little or no significance.

Functions

none

Special Cases

If | z | < 9.54e-7 = 2^{-20}, then the return value is z for small angles.

If | z | > 3.402823e+38 = 2^{+128}, then the return value is zero for large arguments.

sinf/sinsp — *Single-Precision Sine*

Syntax--Standard

#include <math.h>

float **sinf**(float x) ;

Syntax--FastRTS

#include <fastrts67x.h>

float **sinf**(float x); or float **sinsp**(float x);

Defined in

sinsp.asm

Description

The sinf and sinsp functions return the sine of a floating-point argument z. The angle z is expressed in radians. The return value is in the range of [-1.0 and +1.0]. An argument with a large magnitude may produce a result with little or no significance.

Functions

none

Special Cases

If $| z | < 2.44e-4 = 2^{-12}$, then the return value is z for small angles.

If $| z | > 1.048576e+6 = 2^{+20}$, then the return value is zero for large angles.

4.2 Divide and Square Root Routines

The following routines are found in *TMS32067xx Divide and Square Root Floating-point Functions* (literature number SPRA516) and are included as references.

_divd/divdp — *Double-Precision Division*

Syntax--Standard

#include <ieeed.h>

double **_divd**(double x, double y) ;

Syntax--FastRTS

#include <fastrts67x.h>

double **_divd**(double x, double y); or double **divdp**(double x, double y);

Defined in

divdp.asm

Description

The _divd and divdp functions return the quotient of two real floating-point arguments x and y. The return value is x / y.

Functions

none

Special Cases

If | y | < 2.225e-308 = 2^{-1022}, then the return value is **NaN** = **Not-a-Number** (exponent and mantissa are all ones) > +/- 1.797693e+308 = +/-1 * 2^{+1024} (largest double-precision floating-point number) with the sign of x.

_divf/divsp — *Single-Precision Division*

Syntax--Standard

#include <ieeef.h>

float **_divf**(float x, float y) ;

Syntax--FastRTS

#include <fastrts67x.h>

float **_divf**(float x, float y); or float **divsp**(float x, float y);

Defined in

divsp.asm

Description

The _divf and divsp functions return the quotient of two real floating-point arguments x and y. The return value is x / y.

Functions

none

Special Cases

If | y| < 1.1755e-38 = 2^{-126}, then the return value is NaN = Not-a-Number (exponent and mantissa are all ones) > +/- 3.402823e+38 = +/- 1 * 2^{+128} (largest single-precision floating-point number) with the sign of x.

sqrt/sqrtdp — *Double-Precision Square Root*

Syntax--Standard

#include <math.h>

double **sqrt**(double z) ;

Syntax--FastRTS

#include <fastrts67x.h>

double **sqrt**(double z); or double **sqrtdp**(double z);

Defined in

sqrtdp.asm

Description

The sqrt and sqrtdp functions return the square root of a real floating-point argument z. The absolute value of z is used.

Functions

none

Special Cases

If | z | < 2.225e-308 = 2^{-1022}, then the return value for small arguments is zero.

sqrtf/sqrtsp — *Single-Precision Square Root*

Syntax--Standard

#include <math.h>

float **sqrtf**(float z) ;

Syntax--FastRTS

#include <fastrts67x.h>

float **sqrtf**(float z); or float **sqrtsp**(float z);

Defined in

sqrtsp.asm

Description

The sqrtf and sqrtsp functions return the square root of a real floating-point argument z. The absolute value of z is used.

Functions

none

Special Cases

If | z | < 1.1755e-38 = 2^{-126}, then the return value for small arguments is zero.

Appendix A

Performance Considerations

This appendix describes the sample performance of the C67x FastRTS. It also provides information about software updates and customer support issues.

Topic **Page**

A.1 Performance Considerations

Table A-1 gives samples of execution clock cycles. Times include the call and return overhead. The cycle counts were found with the following arguments: func1(3.15) or func2(3.15, 0.625)

Table A-1. Sample Performance

Function	Data	rts6701	FastRTS	rts/FastRTS ratio
atan	64 FP	1001	382	2.62
atanf	32 FP	282	89	2.83
atan2	64 FP	1119	415	2.70
atan2f	32 FP	551	87	6.33
cos	64 FP	371	154	2.41
cosf	32 FP	200	76	2.63
exp	64 FP	656	217	3.02
expf	32 FP	229	79	2.90
exp10	64 FP	681	230	2.96
exp10f	32 FP	235	80	2.94
exp2	64 FP	681	230	2.96
exp2f	32 FP	235	80	2.94
log	64 FP	937	288	3.25
logf	32 FP	152	73	2.08
log10	64 FP	957	289	3.31
log10f	32 FP	169	74	2.28
log2	64 FP	957	289	3.31
log2f	32 FP	169	74	2.28
pow	64 FP	1256	539	2.33
powf	32 FP	679	224	3.03
recip	64 FP	397	81	4.90
recipf	32 FP	182	32	5.69
rsqrt	64 FP	356	111	3.21
rsqrtf	32 FP	186	42	4.43
sin	64 FP	350	150	2.33
sinf	32 FP	189	73	2.59

A.2 FastRTS Software Updates

C67x FastRTS Software updates may be periodically released incorporating product enhancements and fixes as they become available. You should read the spru100.pdf available in the root directory of every release.

A.3 FastRTS Customer Support

If you have questions or want to report problems or suggestions regarding the C67x FastRTS, contact Texas Instruments at dsph@ti.com.

Appendix B

Glossary

A

address: The location of program code or data stored; an individually accessible memory location.

API: See *application programming interface.*

application programming reference (API): Used for proprietary application programs to interact with communications software or to conform to protocols from another vendor's product.

B

bit: A binary digit, either a 0 or 1.

big endian: An addressing protocol in which bytes are numbered from left to right within a word. More significant bytes in a word have lower numbered addresses. Endian ordering is specific to hardware and is determined at reset. See also *little endian*.

C

clock cycle: A periodic or sequence of events based on the input from the external clock.

code: A set of instructions written to perform a task; a computer program or part of a program.

compiler: A computer program that translates programs in a high-level language into their assembly-language equivalents.

D

digital signal processor (DSP: A semiconductor that turns analog signals—such as sound or light—into digital signals, which are discrete or discontinuous electrical impulses, so that they can be manipulated.

F

FastRTS: TMS320C67x Fast Run-Time-Support

L

least significant bit (LSB): The lowest-order bit in a word.

linker: A software tool that combines object files to form an object module, which can be loaded into memory and executed.

little endian: An addressing protocol in which bytes are numbered from right to left within a word. More significant bytes in a word have higher-numbered addresses. Endian ordering is specific to hardware and is determined at reset. See also *big endian*.

Index