

C28x Floating Point Unit

fastRTS Library

Module User's Guide

C28x Foundation Software

V1.00

August 6, 2008



IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

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

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

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. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible or liable for any such use.

Resale of TI's products or services with statements different from or beyond the parameters stated by TI for that products or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.
www.ti.com/sc/docs/stdterms.htm

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

Copyright ©2002, Texas Instruments Incorporated

Contents

1.	<i>Introduction</i>	4
2.	<i>Installing the Library</i>	4
2.1.	Where the Files are Located (Directory Structure)	4
2.2.	Build options used to build the library	4
3.	<i>Using the fastRTS Library</i>	5
3.1.	Link Order of the Library	5
3.2.	Header Files	5
3.3.	Linker File	5
4.	<i>How to Rebuild the fastRTS Library</i>	6
5.	<i>Function Summary</i>	6
5.1.	FPU fastRTS Function Summary	6
6.	<i>Function Descriptions</i>	7
	atan	7
	atan2	7
	cos	7
	FS\$DIV	8
	isqrt	8
	sin	9
	sincos	9
	sqrt	9
7.	<i>Benchmarks</i>	10
8.	<i>Revision History</i>	10

Trademarks

TMS320 is the trademark of Texas Instruments Incorporated.

Code Composer Studio is a trademark of Texas Instruments Incorporated.

All other trademark mentioned herein is property of their respective companies

1.Introduction

The Texas Instruments TMS320C28x Floating Point Unit Fast Run-Time Support (RTS) library is a collection of optimized floating-point math functions for controllers with the C28x plus floating-point unit (FPU). This source code library includes C-callable optimized versions of selected floating-point math functions included in the compiler's standard run-time support libraries.

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.

2.Installing the Library

2.1. Where the Files are Located (Directory Structure)

As installed, the *C28x FPU fastRTS Library* is partitioned into a well-defined directory structure. By default, the library and source code is installed into the following directory:

c:\tidcs\c28\C28x_FPU_fastRTS\<version>

Table 1 describes the contents of the main directories used by library:

Table 1. C28x fastRTS Library Directory Structure

Directory	Description
<base>	Base install directory. By default this is c:\tidcs\c28\C28x_FPU_fastRTS\v100 For the rest of this document <base> will be omitted from the directory names.
<base>\doc	Documentation including the revision history from the previous release.
<base>\lib	The built library.
<base>\include	Header file for non-standard functions such as isqrt() and sincos()
<base>\source	Source files for the library. This also includes a Code Composer Studio project that can be used to re-build the library if required.

2.2. Build options used to build the library

The 1.00 library is built with C28x codegen tools V5.0.2 with the following options:

-g -o3 -d"_DEBUG" -d"LARGE_MODEL" -ml -v28 --float_support=fpu32

3. Using the fastRTS Library

3.1. Link Order of the Library

To use the fastRTS functions in place of the existing functions the fastRTS library must be linked before the existing run-time support library. The fastRTS library replaces only a subset of the functions in the current run time support libraries. Therefore, the standard runtime support library should be linked after the fastRTS library.

1. Add the fastRTS and standard RTS libraries to the project using

Project->Add Files to Project

Once the libraries are added to the project they will appear in the link order tab of the build options.

2. Open the build options dialog box under Project->Build Options
3. Under the Linker->Advanced tab, select the `-priority` linker switch.

This will force the linker to resolve symbols to the first library linked.

4. Under the Link Order tab, select the two libraries and add them to the link order.

Use the up/down arrows to arrange them in the proper order. The first library listed will be linked first.

5. Under the Linker->Libraries dialog add the path to the fast RTS library in the search path.

Do not include either of the RTS libraries in the "Incl. Libraries" box. Doing so can cause problems when changing the link order since Code Composer uses both this field and the link order tab to determine which object files are linked first.

6. Save the project. (Project->Save).

3.2. Header Files

Use the same header files you would for the standard RTS library. For functions that are not part of the standard RTS library, use the included `C28x_FPU_FastRTS.h` header file.

3.3. Linker File

Many of the functions in the library use look-up tables to increase performance. These tables are located in the "FPUmathTables" memory section and are available in the boot ROM of the TMS320x2833x devices.

If you do not wish to load a copy of these tables into the device, use the boot ROM memory addresses and label the section as "NOLOAD" as shown below. This facilitates referencing the look-up tables without actually loading the section to the target.

Note that the boot ROM may not be zero-wait state on all devices and therefore using the boot ROM copy may add a few CPU cycles when compared to using the table loaded into SARAM. The impact to performance is minimal. Refer to the benchmarks section.

```

MEMORY
{
    PAGE 0 :
        ...
        FPUTABLES : origin = 0x3FEBDC, length = 0x0006A0
        ...
    }
    SECTIONS
    {
        ...
        FPUMathTables : > FPUTABLES, PAGE = 0, TYPE = NOLOAD
        ...
    }
}

```

Note:

The addresses shown above are for the TMS320x2833x devices.

Note:

Using the fastRTS library may change the behavior of other standard RTS functions. For example, the fmod() function uses division. If the fastRTS library is used then the division portion will come from the FastRTS instead of the standard library.

4.How to Rebuild the fastRTS Library

If you want to rebuild the fastRTS library (for example, because you modified the source contained in the archive), use the supplied Code Composer Studio project in the source directory.

5.Function Summary

5.1. FPU fastRTS Function Summary

The following functions are included in this release of the fast RTS library. Other functions will be added in future releases. These functions are called as in the current runtime support library.

atan	isqrt
atan2	sin
cos	sincos
division	sqrt
	sincos

Note: isqrt() and sincos are not included in the standard RTS library.

6.Function Descriptions

atan	<i>Single-Precision Floating-Point ATAN (radians)</i>
-------------	---

Description Returns the arc tangent of a floating-point argument X. The return value is an angle in the range $[-\pi, \pi]$ radians.

Header File `#include <math.h>`

Declaration `float32 atan (float32 X)`

atan2	<i>Single-Precision Floating-Point ATAN2 (radians)</i>
--------------	--

Description Returns the 4-quadrant arctangent of floating-point arguments X/Y. The return value is an angle in the range $[-\pi, \pi]$ radians.

Header File `#include <math.h>`

Declaration `float32 atan2 (float32 X, float32 Y)`

cos	<i>Single-Precision Floating-Point COS (radians)</i>
------------	--

Description Returns the cosine of a floating-point argument X (in radians) using table look-up and Taylor series expansion between the look-up table entries.

Header File `#include <math.h>`

Declaration `float32 cos (float32 X)`

FS\$DIV	<i>Single-Precision Floating-Point Division</i>
Description	Replaces the single-precision division operation from the standard RTS library. This function uses a Newton-Raphson algorithm.
Header File	None
Example	<pre>float32 X, Y, Z; ... <Initialize X, Y> ... Z = Y/X // invokes FS\$DIV</pre>
Special Cases:	<pre>0.0/0.0 = +infinity +FLT_MAX/+FLT_MAX = 0.0, LUF = 1 -FLT_MAX/+FLT_MAX = -0.0, LUF = 1 +FLT_MAX/-FLT_MAX = 0.0, LUF = 1 -FLT_MAX/-FLT_MAX = -0.0, LUF = 1 +FLT_MIN/+FLT_MAX = 0.0, LUF = 1 -FLT_MIN/+FLT_MAX = -0.0, LUF = 1 +FLT_MIN/-FLT_MAX = 0.0, LUF = 1 -FLT_MIN/-FLT_MAX = -0.0, LUF = 1</pre> <p>Division by 0.0 sets the LVF flag.</p>

isqrt	<i>Single-Precision Floating-Point 1.0/Square Root</i>
Description	<p>Returns 1.0 /square root of a floating-point argument X using a Newton-Raphson algorithm.</p> <p>Note: This function is not included in the standard RTS library. It is typically computed as 1.0L/sqrt(X). To use this function you must modify your code to instead call isqrt(X).</p> <p>When migrating from an IQmath project, you can modify the IQmath header file to use isqrt(X) when configured for FLOAT_MATH.</p>
Header File	<code>#include "C28x_FPU_FastRTS.h"</code>
Declaration	<code>float32 sqrt (float32 X)</code>
Special Cases	<p>isqrt(FLT_MAX) and isqrt(FLT_MIN) set the LUF flag. isqrt(-FLT_MIN) will set both the LUF and LVF flags. isqrt(0.0) sets the LVF flag. If X is negative, isqrt(X) will set LVF and return 0.0.</p>

sin	<i>Single-Precision Floating-Point SIN (radians)</i>
------------	--

Description Returns the sine of a floating-point argument X (in radians) using table look-up and Taylor series expansion between the look-up table entries.

Header File `#include <math.h>`

Declaration `float32 sin (float32 X)`

sincos	<i>Single-Precision Floating-Point SIN and Cosine (radians)</i>
---------------	---

Description Returns both the sine and cosine of a floating-point argument X (in radians) using table look-up and Taylor series expansion between the look-up table entries.

Header File `#include "C28x_FPU_FastRTS.h"`

Declaration `void sincos(float32 X, float32* PtrSin,
float32* PtrCos);`

X Input argument in radians
PtrSin Pointer to the sine result
PtrCos Pointer to the cosine result

sqrt	<i>Single-Precision Floating-Point Square Root</i>
-------------	--

Description Returns the square root of a floating-point argument X using a Newton-Raphson algorithm.

Header File `#include <math.h>`

Declaration `float32 sqrt (float32 X)`

Special Cases `sqrt(FLT_MAX)` and `sqrt(FLT_MIN)` set the LUF flag.
`sqrt(-FLT_MIN)` will set both the LUF and LVF flags.
`sqrt(0.0)` sets the LVF flag.
If X is negative, `sqrt(X)` will set LVF and return 0.0.

7. Benchmarks

The following table lists the execution time in CPU cycles for the fastRTS library routines. These numbers assume that both the code and stack are in zero wait-state memory. These numbers include function-call/return overhead but do not include any cycles for setting up the input data or storing the result.

Function	FPUMathTables in zero-wait SARAM	FPUMathTables in single-wait Boot ROM
atan	47	51
atan2	49	53
cos	38	42
division	24	24
isqrt	25	25
sin	37	41
sincos	44	50
sqrt	28	28

8. Revision History

Changes from Beta1 to V1.00

- Removed the version name from the library name. This makes updating to a new library easier.
- Added sincos() function
- Sin and Cos:
 - Corrected the constant value of 0.166 to 0.166667
 - Changed the truncated $2\pi/512$ value to a rounded value of $2\pi/512$. Previously this value was truncated.
 - In Beta 1, the $\text{int}(\text{Radian} * 512/(2\pi))$ calculation was done using float to 16-bit int. Changed this to 32-bit int to accommodate a larger range of input values.