TMS320C62x/64x FastRTS Library Programmer's Reference

Literature Number: SPRU653
February 2003

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Preface

Read This First

About This Manual

Welcome to the TMS320C62x/64x Fast Run-Time-Support Library, or FastRTS library for short. The FastRTS library is a collection of 28 optimized floating-point functions for the fixed-point TMS320C62x/64x devices. This source code library includes C-callable (ANSI-C-language compatible) optimized versions of the floating-point functions included in previous run-time support libraries.

How to Use This Manual

The information in this document describes the contents of the TMS320C62x/64x FastRTS library in several different ways.

- Chapter 1 provides a brief introduction to the C62x/64x 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 C62x/64x 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 TMS320C62x is also referred to in this reference guide as the C62x. The TMS320C64x is also referred to in this reference guide as the C64x.

Related Documentation From Texas Instruments

The following books describe the TMS320C6000 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

Texas Instruments trademarks include: TI, Code Composer Studio, TMS320, TMS320C6000, TMS320C62x, TMS320C64x, and TMS320C67x.

All other trademarks are the property of their respective owners.

Contents

Tables

Chapter 1

Introduction

This chapter provides a brief introduction to the C62x/64x 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 C62x/64x FastRTS Library

The C62x/64x FastRTS library is an optimized, floating-point function library for C programmers using either TMS320C62x or TMS320C64x devices. These routines are typically used in computationally intensive real-time applications where optimal execution speed is critical. By replacing the current floating-point library (RTS) with the FastRTS library, you can achieve execution speeds considerably faster, without rewriting existing code.

The FastRTS library includes the routines currently provided in existing run-time-support libraries which provide floating-point functionality for the fixed-point C62x and C64x devices. These new functions can be called with the current run-time-support library names or the new names provided in the FastRTS library.

As shown in Table 1-1, single- and double-precision routines are available.

Table 1-1. FastRTS Library Functions

Single Precision	Double Precision	Others
_addf	_addd	_cvtdf
_divf	_divd	_cvtfd
_fixfi	_fixdi	
_fixfli	_fixdli	
_fixfu	_fixdu	
_fixful	_fixdul	
_fltif	_fltid	
_fltlif	_fltlid	
_fltuf	_fltud	
_fltulf	_fltuld	
_mpyf	_mpyd	
recipf	recip	
_subf	_subd	

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 C62x/64x FastRTS library.

Topic **Page**

2.1 FastRTS Library Contents

The C62x64xFastRTS.exe installs the following file structure:

lib Directory containing the following library files:

fastrts62x64x.lib	Little-endian C62x/C64x library file
fastrts62x64xe.lib	Big-endian C62x/C64x library file
fastrts62x64x.src	Source archive file

include Directory containing the following include files:

fastrts62x64x.h	Alternative entry header file
recip.h	Header file for reciprocal functions

doc Directory containing the following document files:

spru653.pdf	PDF document of API (this document)

2.2 How to Install the FastRTS Library

To install the FastRTS libary, follow these steps:

Step 1: Open the file, C62x64xFastRTS.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 location is c:\ti\c6000\cgtools.

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 version 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:

```
-lfastrts62x64x.lib
```

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

```
-lfastrts62x64xe.lib
```

The FastRTS library also contains alternate names for the functions. This allows you to choose where you want use the current RTS or FastRTS functions throughout your code. To exploit this option, link the FastRTS library after the current run-time-support library. The existing routines will automatically be called; however, you can now also explicitly call the FastRTS functions by using the alternate names.

2.3.1 FastRTS Library Arguments and Data Types

2.3.1.1 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
int	16	Integer	-32767	+32767
long int	40	Long integer	-549755813887	+549755813887
unsigned int	16	Unsigned integer	0	+65535
unsigned long	40	Unsigned long integer	0	+1099511627775

2.3.1.2 FastRTS Arguments

The C62x/C64x FastRTS functions operate on single value arguments. The functions add, div, mpy, and sub 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 fastrts62x64x.h header file must be included if you use the special FastRTS function names.
  - The recip.h header file must be included if the recipd, recipdp, recipf, or recipsp functions are called.
- Link your code with fastrts62x64x.lib for little-endian code or fastrts62x64xe.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 current run-time-support libraries. Therefore, fastrts62x64x.lib or fastrts62x64xe.lib must be linked in along with rts6x.lib or rts6xe.lib.

For example, if you call the add FastRTS function, you would add in your C file, and compile and link using:

```
cl6x main.c -z -o drv.out -lfastrts62x64x.lib -rts6201.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 fastrts62x64x.lib for fastrts62x64xe.lib.

2.3.3 Calling a FastRTS Function From Assembly

The C62x/C64x 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* (SPRU212).

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 fastrts62x64x.src -l fastrts62x64x.lib
mk6x -me fastrts62x64x.src -l fastrts62x64xe.lib
```

Chapter 3

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 conventions shown in Table 3-1 have been followed when describing the arguments for each individual function.

Table 3-1. Argument Conventions

Argument	Description
x, y	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 Table 3-2 are current run-time-support library function names and the alternate function names for the FastRTS library. Either name can used to call the FastRTS version of the function.

For practical use, the routines included in the C62x/C64x FastRTS library are invoked automatically by the compiler when the appropriate floating-point operation or conversion is required for the fixed-point C62x/C64x processors. For example, the following addition of two SP floating-point numbers on the C62x/C64x will cause the C compiler to invoke a function call to _addf:

```
float x = 5.0;
float y = 2.5;
float z;
z = x + y;
```

Similarly, an appropriate type cast in C will cause the C compiler to invoke the corresponding run-time-support library function. For example, the following conversion of a signed integer to a SP floating-point number will cause the C compiler to invoke a function call to _fltif:

```
int y = 2;
float z;
z = (float)y;
```

Also included in the following chart is the C syntax that will force the C compiler to invoke the corresponding FastRTS library. The C syntax listing assumes the following data types:

```
float x, y;
double r, s;
int a;
unsigned int b;
long int c;
unsigned long int d;
```

Table 3-2. FastRTS Function Names Comparison

Description	Current Name SP	Current Name DP	Alternate Name SP	Alternate Name DP	C Invocation SP	C Invocation DP	See Page
Floating-point addition	_addf	_addd	addsp	adddp	x = x + y;	r = r + s;	4-2
Floating-point division	_divf	_divd	divdsp	divdp	x = x/y;	r = r/s;	4-2, 4-3
Floating-point to 32-bit signed integer	_fixfi	_fixdi	spint	dpint	a = (int)x;	a = (int)r;	4-3, 4-3
Floating-point to 40-bit signed long integer	_fixfli	_fixdli	splong	dplong	c = (long int)x;	c = (long int)r;	4-4
Floating-point to 32-bit unsigned integer	_fixfu	_fixdu	spuint	dpuint	b = (unsigned)x;	b = (unsigned)r;	4-4, 4-5
Floating-point to 40-bit unsigned long integer	_fixful	_fixdul	spulong	dpulong	d = (unsigned long)x;	d = (unsigned long)r;	4-5, 4-6
32-bit signed integer to floating point	_fltif	_fltid	intsp	intdp	x = (float)a;	r = (double)a;	4-6
40-bit signed long integer to floating point	_fltlif	_fltlid	longsp	longdp	x = (float)c;	r = (double)c;	4-6, 4-7
32-bit unsigned integer to floating point	_fltuf	_fltud	uintsp	uintdp	x = (float)b;	r = (double)b;	4-7
40-bit unsigned long integer to floating point	_fltulf	_fltuld	ulongsp	ulongdp	x = (float)d;	r = (double)d;	4-7, 4-8
Floating-point multiplication	_mpyf	_mpyd	mpysp	mpydp	x = x*y;	r = r*s;	4-8
Floating-point reciprocal	recipf†	recipd†	recipsp	recipdp	x = recipf(y);	r = recipd(s);	4-9
Floating-point subtraction	_subf	_subd	subsp	subdp	x = x - y;	r = r - s;	4-9, 4-10

† The FastRTS functions recipf and recipd are not defined in the corresponding rts62xx.lib and rts64xx.lib.

Two of the functions are for conversion between single-precision and double-precision floating-point numbers. Table 3-3 lists these functions.

Table 3-3. FastRTS Conversion Functions Names Comparisons

Description	Current Name	Alternate Name	C Invocation	See Page
DP to SP Conversion	_cvtdf	dpsp	x = (float)r;	4-11
SP to DP Conversion	_cvtfd	spdp	r = (double)x;	4-11

Chapter 4

FastRTS Reference

This chapter provides a list of 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

_addd/adddp — *Double-precision floating-point addition*

Syntax - Standard double **_addd**(double x, double y);

Syntax - FastRTS #include < fastrts62x64x.h>
double **_addd**(double x, double y); or double **adddp**(double x, double y);

Defined in adddp.asm

Description The sum of two input 64-bit floating-point (FP) numbers is generated.

Special Cases Underflow returns zero; overflow returns + or - infinity.

_addf/addsp — *Single-precision floating-point addition*

Syntax - Standard float **_addf**(float x, float y);

Syntax - FastRTS #include < fastrts62x64x.h>
float **_addf**(float x, float y); or float **addsp**(float x, float y);

Defined in addsp.asm

Description The sum of two input 32-bit floating-point (FP) numbers is generated.

Special Cases Underflow returns zero; overflow returns + or - infinity.

_divd/divdp — *Double-precision floating-point division*

Syntax - Standard double **_divd**(double x, double y);

Syntax - FastRTS #include < fastrts62x64x.h >
double **_divd**(double x, double y); or double **divdp**(double x, double y);

Defined in divdp.asm

Description The quotient of two input 64-bit FP numbers is generated.

Special Cases Underflow returns zero; overflow returns + or - infinity. Zero over zero returns zero; non-zero over zero returns infinity.

_divf/divsp — *Single-precision floating-point division*

Syntax - Standard float **_divf**(float x, float y);

Syntax - FastRTS #include < fastrts62x64x.h >
float **_divf**(float x, float y); or float **divsp**(float x, float y);

Defined in divsp.asm

Description The quotient of two input 32-bit FP numbers is generated.

Special Cases Underflow returns zero; overflow returns + or - infinity. Zero over zero returns zero; non-zero over zero returns infinity.

_fixdi/dpint — *Double-precision floating-point to 32-bit signed integer*

Syntax - Standard int **_fixdi**(double x);

Syntax - FastRTS #include <fastrts62x64x.h>
int **_fixdi**(double x); or int **dpint**(double x);

Defined in dpint.asm

Description An input 64-bit FP number is converted to a 32-bit signed integer.

Special Cases Numbers with magnitude less than 1.0 return zero. Numbers greater than 32 bits return one of the following saturation values:

- 0x7fff_ffff for positive numbers
- 0x8000_0000 for negative numbers

_fixfi/spint — *Single-precision floating-point to 32-bit signed integer*

Syntax - Standard int **_fixfi**(float x);

Syntax - FastRTS #include <fastrts62x64x.h>
int **_fixfi**(float x); or int **spint**(float x);

Defined in spint.asm

Description An input 32-bit FP number is converted to a 32-bit signed integer.

Special Cases Numbers with magnitude less than 1.0 return zero. Numbers greater than 32 bits return one of the following saturation values:

- 0x7fff_ffff for positive numbers
- 0x8000_0000 for negative numbers

_fixdli/dplong *Double-precision floating-point to 40-bit signed long integer*

Syntax - Standard long int **_fixdli**(double x);

Syntax - FastRTS #include <fastrts62x64x.h>
long int **_fixdli**(double x); or long int **dplong**(double x);

Defined in dplong.asm

Description An input 64-bit FP number is converted to a 40-bit signed long integer.

Special Cases Numbers with magnitude less than 1.0 return zero. Numbers greater than 40 bits return one of the following saturation values:

- 0x7f_ffff_ffff for positive numbers
- 0x80_0000_0000 for negative numbers

_fixfli/splong *Single-precision floating-point to 40-bit signed long integer*

Syntax - Standard long int **_fixfli**(float x);

Syntax - FastRTS #include <fastrts62x64x.h>
long int **_fixfli**(float x); or long int **splong**(float x);

Defined in splong.asm

Description An input 32-bit FP number is converted to a 40-bit signed long integer.

Special Cases Numbers with magnitude less than 1.0 return zero. Numbers greater than 40 bits return one of the following saturation values:

- 0x7f_ffff_ffff for positive numbers
- 0x80_0000_0000 for negative numbers

_fixdu/dpuint *Double-precision floating-point to 32-bit unsigned integer*

Syntax - Standard unsigned int **_fixdu**(double x);

Syntax - FastRTS #include <fastrts62x64x.h>
unsigned int **_fixdu**(double x); or unsigned int **dpuint**(double x);

Defined in dpuint.asm

Description An input 64-bit FP number is converted to a 32-bit unsigned integer.

Special Cases Numbers less than 1.0 return zero. Numbers greater than 32 bits return one of the following saturation values:

- 0xffff_ffff for positive numbers
- 0x0000_0000 for negative numbers

_fixfu/spuint *Single-precision floating-point to 32-bit unsigned integer*

Syntax - Standard unsigned int **_fixfu**(float x);

Syntax - FastRTS #include <fastrts62x64x.h>
unsigned int **_fixfu**(float x); or unsigned int **spuint**(float x);

Defined in spuint.asm

Description An input 32-bit FP number is converted to a 32-bit unsigned integer.

Special Cases Numbers less than 1.0 return zero. Numbers greater than 32 bits return one of the following saturation values:

- 0xffff_ffff for positive numbers
- 0x0000_0000 for negative numbers

_fixdul/dpulong *Double-precision floating-point to 40-bit unsigned long integer*

Syntax - Standard long **_fixdul**(double x);

Syntax - FastRTS #include <fastrts62x64x.h>
long **_fixdul**(double x); or long **dpulong**(double x);

Defined in dpulong.asm

Description An input 64-bit FP number is converted to a 40-bit unsigned long integer.

Special Cases Numbers less than 1.0 return zero. Numbers greater than 32 bits return one of the following saturation values:

- 0xff_ffff_ffff for positive numbers
- 0x00_0000_0000 for negative numbers

_fixful/spulong *Single-precision floating-point to 40-bit unsigned long integer*

Syntax - Standard long **_fixful**(float x);

Syntax - FastRTS #include <fastrts62x64x.h>
long **_fixful**(float x); or long **spulong**(float x);

Defined in spulong.asm

Description An input 32-bit FP number is converted to a 40-bit unsigned long integer.

Special Cases Numbers less than 1.0 return zero. Numbers greater than 32 bits return one of the following saturation values:

- 0xff_ffff_ffff for positive numbers
- 0x00_0000_0000 for negative numbers

_fltid/intdp *Convert 32-bit signed integer to double-precision floating point*

Syntax - Standard double **_fltid** (int x);

Syntax - FastRTS #include <fastrts62x64x.h>
double **_fltid** (int x); or double **intdp** (int x);

Defined in intdp.asm

Description An input 32-bit signed integer is converted to a 64-bit SP FP number.

_fltif/intsp *Convert 32-bit signed integer to single-precision floating point*

Syntax - Standard float **_fltif** (int x);

Syntax - FastRTS #include <fastrts62x64x.h>
float **_fltif** (int x); or float **intsp** (int x);

Defined in intsp.asm

Description An input 32-bit signed integer is converted to a 32-bit SP FP number.

_fltlid/longdp *Convert 40-bit signed long integer to double-precision floating point*

Syntax - Standard double **_fltlid** (long x);

Syntax - FastRTS #include <fastrts62x64x.h>
double **_fltlid** (long x); or double **longdp** (long x);

Defined in longdp.asm

Description An input 40-bit signed long integer is converted to a 64-bit SP FP number.

_fltlif/longsp *Convert 40-bit signed long integer to single-precision floating point*

Syntax - Standard float **_fltlif** (long x);

Syntax - FastRTS #include <fastrts62x64x.h>
float **_fltlif** (long x); or float **longsp** (long x);

Defined in longsp.asm

Description An input 40-bit signed long integer is converted to a 32-bit SP FP number.

_fltud/uintdp *Convert 32-bit unsigned integer to double-precision floating point*

Syntax - Standard double **_fltud** (unsigned int x);

Syntax - FastRTS #include <fastrts62x64x.h>
double **_fltud**(unsigned int x); or double **uintdp**(unsigned int x);

Defined in uintdp.asm

Description An input 32-bit unsigned integer is converted to a 64-bit SP FP number.

_fltuf/uintsp *Convert 32-bit unsigned integer to single-precision floating point*

Syntax - Standard float **_fltuf** (unsigned int x);

Syntax - FastRTS #include <fastrts62x64x.h>
float **_fltuf**(unsigned int x); or float **uintsp**(unsigned int x);

Defined in uintsp.asm

Description An input 32-bit unsigned integer is converted to a 32-bit SP FP number.

_fltuld/ulongdp *Convert 40-bit unsigned long integer to double-precision floating point*

Syntax - Standard double **_fltuld** (unsigned long int x);

Syntax - FastRTS #include <fastrts62x64x.h>
double **_fltuld** (unsigned long int x); or double **ulongdp** (unsigned long int x);

Defined in ulongdp.asm

Description An input 40-bit unsigned long integer is converted to a 64-bit SP FP number.

_fltulf/ulongsp *Convert 40-bit unsigned long integer to single-precision floating point*

Syntax - Standard float **_fltulf** (unsigned long int x);

Syntax - FastRTS #include <fastrts62x64x.h>
float **_fltulf** (unsigned long int x); or float **ulongsp** (unsigned long int x);

Defined in ulongsp.asm

Description An input 40-bit unsigned long integer is converted to a 32-bit SP FP number.

_mpyd/mpydp *Double-precision floating-point multiplication*

Syntax - Standard double **_mpyd** (double x);

Syntax - FastRTS #include <fastrts62x64x.h>
double **_mpyd**(double x, double y); or double **mpydp**(double x, double y);

Defined in mpydp.asm

Description The product of two input 64-bit FP numbers is generated.

Special Cases Underflow or exponents < 2 ($2^{-1022} = 4.45 \times 10^{-308}$) returns zero; overflow returns + or - infinity.

_mpyf/mpysp *Single-precision floating-point multiplication*

Syntax - Standard float **_mpyf** (float x);

Syntax - FastRTS #include <fastrts62x64x.h>
float **_mpyf**(float x, float y); or float **mpysp**(float x, float y);

Defined in mpysp.asm

Description The product of two input 32-bit FP numbers is generated.

Special Cases Underflow or exponents < 2 ($2^{-125} = 2.35 \times 10^{-38}$) returns zero; overflow returns + or - infinity.

recipd/recipdp *Double-precision floating-point reciprocal*

Syntax - FastRTS
#include <fastrts62x64x.h>
#include <recip.h>
double **recipd** (double x); or double **recipdp** (double x);

Defined in divdp.asm

Description The reciprocal of an input 64-bit FP number is generated.

Special Cases Underflow returns zero; overflow returns + or - infinity. The reciprocal of zero returns infinity.

recipf/recipsp *Single-precision floating-point reciprocal*

Syntax - FastRTS
#include <fastrts62x64x.h>
#include <recip.h>
float **recipf** (float x); or float **recipsp** (float x);

Defined in divsp.asm

Description The reciprocal of an input 32-bit FP number is generated.

Special Cases Underflow returns zero; overflow returns + or - infinity. The reciprocal of zero returns infinity.

_subd/subdp *Double-precision floating-point subtraction*

Syntax - Standard double **_subd**(double x, double y);

Syntax - FastRTS
#include <fastrts62x64x.h>
double **_subd**(double x, double y); or double **subdp**(double x, double y);

Defined in adddp.asm

Description The difference of two input 64-bit FP numbers is generated.

Special Cases Underflow returns zero; overflow returns + or - infinity.

_subf/subsp

Single-precision floating-point subtraction

Syntax - Standard float **_subf**(float x, float y);

Syntax - FastRTS #include <fastrts62x64x.h>
float **_subf**(float x, float y); or float **subsp**(float x, float y);

Defined in addsp.asm

Description The difference of two input 32-bit FP numbers is generated.

Special Cases Underflow returns zero; overflow returns + or - infinity.

4.2 Conversion Routines

_cvtdf/dpsp — *Convert double-precision to single-precision floating point*

Syntax - Standard float **_cvtdf**(double x);

Syntax - FastRTS #include < fastrts62x64x.h>
float **_cvtdf**(double x); or float **dpsp**(double x);

Defined in dpsp.asm

Description An input 64-bit FP number is converted to a 32-bit FP number.

Special Cases Underflow returns zero; overflow returns + or - SP infinity.

_cvtfd/spdp — *Convert single-precision to double-precision floating point*

Syntax - Standard double **_cvtfd**(float x);

Syntax - FastRTS #include < fastrts62x64x.h>
double **_cvtfd**(float x); or double **spdp**(float x);

Defined in spdp.asm

Description An input 32-bit FP number is converted to a 64-bit FP number.

Special Cases Underflow returns zero; overflow returns + or - infinity.

Appendix A

Performance Considerations

This appendix describes the sample performance of the C62x/64x 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). The table compares the execution clock cycles for the current run-time-support libraries for the C62x and C64x versus the execution clock cycles for the new FastRTS routines.

Table A-1. Sample Performance

			C64x			C62x		
Function	**Alternate Name**	**Data**	**rts6400.lib**	**FastRTS**	**RTS/ FastRTS Ratio**	**rts6200.lib**	**FastRTS**	**RTS/ FastRTS Ratio**
_addf	addsp	32 FP	81	29	2.79	88	24	3.67
_addd	adddp	64 FP	142	75	1.89	141	70	2.01
_divf	divsp	32 FP	109	32	3.41	115	27	4.26
_divd	divdp	64 FP	204	77	2.65	156	72	2.17
_fixfi	spint	32 FP	85	29	2.93	82	22	3.73
_fixdi	dpint	64 FP	165	51	3.24	170	44	3.86
_fixfli	splong	32 FP	154	52	2.96	149	51	2.92
_fixdli	dplong	64 FP	306	238	1.29	306	188	1.63
_fixfu	spuint	32 FP	154	53	2.91	152	53	2.87
_fixdu	dpuint	64 FP	310	239	1.30	310	190	1.63
_fixful	spulong	32 FP	32	14	2.29	31	15	2.07
_fixdul	dpulong	64 FP	30	20	1.50	30	17	1.76
_fltif	intsp	32 FP	37	16	2.31	37	17	2.18
_fltid	intdp	64 FP	35	21	1.67	40	21	1.90
_fltlif	longsp	32 FP	16	14	1.14	17	15	1.13
_fltlid	longdp	64 FP	20	17	1.18	21	17	1.24
_fltuf	uintsp	32 FP	28	15	1.87	31	16	1.94
_fltud	uintdp	64 FP	29	16	1.81	35	20	1.75

Table A-1. Sample Performance (Continued)

			C64x			C62x		
Function	**Alternate Name**	**Data**	**rts6400.lib**	**FastRTS**	**RTS/ FastRTS Ratio**	**rts6200.lib**	**FastRTS**	**RTS/ FastRTS Ratio**
_fltulf	ulongsp	32 FP	30	18	1.67	32	15	2.13
_fltuld	ulongdp	64 FP	34	17	2.00	43	18	2.39
_mpyf	mpysp	32 FP	35	18	1.94	36	17	2.12
_mpyd	mpydp	64 FP	42	19	2.21	46	22	2.09
recipf	recipsp	32 FP	37	17	2.18	37	16	2.31
recipd	recipdp	64 FP	46	19	2.42	49	18	2.72
_subf	subsp	32 FP	37	18	2.06	37	16	2.31
_subd	subdp	64 FP	45	22	2.05	51	22	2.32
_cvtfd	spdp	32 FP	28	16	1.75	33	15	2.20
_cvtdf	dpsp	64 FP	38	17	2.24	39	18	2.17

A.2 FastRTS Software Updates

C62x/C64x FastRTS Software updates may be periodically released incorporating product enhancements and fixes as they become available. You should read the spru653.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 C62x/C64x FastRTS, contact Texas Instruments at dsph@ti.com.

Appendix B

Glossary

A

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: 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

D

F

I

L

M

P

R

S

U