

# ***TMS320C62x/64x FastRTS Library Programmer's Reference***

Literature Number: SPRU653  
February 2003



## 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

# 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), multi-channel 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

---

---

---

|          |                                                                                                                                                                                  |            |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| <b>1</b> | <b>Introduction</b> .....                                                                                                                                                        | <b>1-1</b> |
|          | <i>Provides an introduction to the C62x/64x FastRTS, shows the organization of the routines contained in the library, and explains the features and benefits of the library.</i> |            |
| 1.1      | Introduction to the C62x/64x FastRTS Library .....                                                                                                                               | 1-2        |
| 1.2      | Features and Benefits .....                                                                                                                                                      | 1-3        |
| <b>2</b> | <b>Installing and Using FastRTS</b> .....                                                                                                                                        | <b>2-1</b> |
|          | <i>Provides information on how to install and rebuild the FastRTS library.</i>                                                                                                   |            |
| 2.1      | FastRTS Library Contents .....                                                                                                                                                   | 2-2        |
| 2.2      | How to Install the FastRTS Library .....                                                                                                                                         | 2-3        |
| 2.3      | Using the FastRTS Library .....                                                                                                                                                  | 2-4        |
| 2.3.1    | FastRTS Library Arguments and Data Types .....                                                                                                                                   | 2-4        |
| 2.3.2    | Calling a FastRTS Function From C .....                                                                                                                                          | 2-5        |
| 2.3.3    | Calling a FastRTS Function From Assembly .....                                                                                                                                   | 2-5        |
| 2.4      | How to Rebuild the FastRTS Library .....                                                                                                                                         | 2-6        |
| <b>3</b> | <b>FastRTS Library Functions Tables</b> .....                                                                                                                                    | <b>3-1</b> |
|          | <i>Provides a list of the functions in the Fast Run-Time-Support (FastRTS) Library, organized alphabetically in two functional categories.</i>                                   |            |
| 3.1      | Arguments and Conventions Used .....                                                                                                                                             | 3-2        |
| 3.2      | FastRTS Functions .....                                                                                                                                                          | 3-3        |
| <b>4</b> | <b>FastRTS Reference</b> .....                                                                                                                                                   | <b>4-1</b> |
|          | <i>Provides a list of the functions in the Fast Run-Time-Support (FastRTS) Library, organized alphabetically in two functional categories.</i>                                   |            |
| 4.1      | General FastRTS Functions .....                                                                                                                                                  | 4-2        |
| 4.2      | Conversion Routines .....                                                                                                                                                        | 4-11       |
| <b>A</b> | <b>Performance Considerations</b> .....                                                                                                                                          | <b>A-1</b> |
|          | <i>Describes performance considerations related to the C67x FastRTS and provides information about software updates and customer support issues.</i>                             |            |
| A.1      | Performance Considerations .....                                                                                                                                                 | A-2        |
| A.2      | FastRTS Software Updates .....                                                                                                                                                   | A-4        |
| A.3      | FastRTS Customer Support .....                                                                                                                                                   | A-4        |
| <b>B</b> | <b>Glossary</b> .....                                                                                                                                                            | <b>B-1</b> |
|          | <i>Defines terms and acronyms used in this book.</i>                                                                                                                             |            |

# Tables

---

---

---

---

|     |                                                      |     |
|-----|------------------------------------------------------|-----|
| 1-1 | FastRTS Library Functions .....                      | 1-2 |
| 2-1 | FastRTS Data Types .....                             | 2-4 |
| 3-1 | Argument Conventions .....                           | 3-2 |
| 3-2 | FastRTS Function Names Comparison .....              | 3-4 |
| 3-3 | FastRTS Conversion Functions Names Comparisons ..... | 3-4 |
| A-1 | Sample Performance .....                             | A-2 |

# 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.

| <b>Topic</b>                                                  | <b>Page</b> |
|---------------------------------------------------------------|-------------|
| <b>1.1 Introduction to the C62x/64x FastRTS Library</b> ..... | <b>1-2</b>  |
| <b>1.2 Features and Benefits</b> .....                        | <b>1-3</b>  |



## 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*

| <b>Single Precision</b> | <b>Double Precision</b> | <b>Others</b> |
|-------------------------|-------------------------|---------------|
| _addf                   | _addd                   | _cvtfd        |
| _divf                   | _divd                   | _cvtfd        |
| _fixfi                  | _fixdi                  |               |
| _fixfli                 | _fixdli                 |               |
| _fixfu                  | _fixdu                  |               |
| _fixful                 | _fixdul                 |               |
| _fltif                  | _fltfd                  |               |
| _ftlif                  | _ftlid                  |               |
| _fltuf                  | _fltud                  |               |
| _fltulf                 | _fltuld                 |               |
| _mpyf                   | _mpyd                   |               |
| recipf                  | recipd                  |               |
| _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

# 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.

| <b>Topic</b>                                        | <b>Page</b> |
|-----------------------------------------------------|-------------|
| <b>2.1 FastRTS Library Contents .....</b>           | <b>2-2</b>  |
| <b>2.2 How to Install the FastRTS Library .....</b> | <b>2-3</b>  |
| <b>2.3 Using the FastRTS Library .....</b>          | <b>2-4</b>  |
| <b>2.4 How to Rebuild the FastRTS Library .....</b> | <b>2-6</b>  |

## 2.1 FastRTS Library Contents

The C62x64xFastRTS.exe installs the following file structure:

|                |                                                               |
|----------------|---------------------------------------------------------------|
| <i>lib</i>     | Directory containing the following library files:             |
|                | <i>fastrts62x64x.lib</i> Little-endian C62x/C64x library file |
|                | <i>fastrts62x64xe.lib</i> Big-endian C62x/C64x library file   |
|                | <i>fastrts62x64x.src</i> Source archive file                  |
| <i>include</i> | Directory containing the following include files:             |
|                | <i>fastrts62x64x.h</i> Alternative entry header file          |
|                | <i>recip.h</i> Header file for reciprocal functions           |
| <i>doc</i>     | Directory containing the following document files:            |
|                | <i>spru653.pdf</i> PDF document of API (this document)        |

## 2.2 How to Install the FastRTS Library

To install the FastRTS library, 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 runtime-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
```



# 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.

| <b>Topic</b>                                    | <b>Page</b> |
|-------------------------------------------------|-------------|
| <b>3.1 Arguments and Conventions Used .....</b> | <b>3-2</b>  |
| <b>3.2 FastRTS Functions .....</b>              | <b>3-3</b>  |

### 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*

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

<sup>†</sup> 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 | <code>_cvtdf</code>  | <code>dpdp</code> | <code>x = (float)r;</code>  | 4-11     |
| SP to DP Conversion | <code>_cvtfld</code> | <code>spdp</code> | <code>r = (double)x;</code> | 4-11     |

# 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.

| <b>Topic</b>                               | <b>Page</b> |
|--------------------------------------------|-------------|
| <b>4.1 General FastRTS Functions .....</b> | <b>4-2</b>  |
| <b>4.2 Conversion Routines .....</b>       | <b>4-11</b> |

## 4.1 General FastRTS Functions

### **add/adddp** *Double-precision floating-point addition*

---

|                          |                                                                                                                       |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>Syntax - Standard</b> | double <b>_addd</b> (double x, double y);                                                                             |
| <b>Syntax - FastRTS</b>  | #include < fastrts62x64x.h><br>double <b>_addd</b> (double x, double y); or double <b>adddp</b> (double x, double y); |
| <b>Defined in</b>        | adddp.asm                                                                                                             |
| <b>Description</b>       | The sum of two input 64-bit floating-point (FP) numbers is generated.                                                 |
| <b>Special Cases</b>     | Underflow returns zero; overflow returns + or - infinity.                                                             |

### **addf/addsp** *Single-precision floating-point addition*

---

|                          |                                                                                                                 |
|--------------------------|-----------------------------------------------------------------------------------------------------------------|
| <b>Syntax - Standard</b> | float <b>_addf</b> (float x, float y);                                                                          |
| <b>Syntax - FastRTS</b>  | #include < fastrts62x64x.h><br>float <b>_addf</b> (float x, float y); or float <b>addsp</b> (float x, float y); |
| <b>Defined in</b>        | addsp.asm                                                                                                       |
| <b>Description</b>       | The sum of two input 32-bit floating-point (FP) numbers is generated.                                           |
| <b>Special Cases</b>     | Underflow returns zero; overflow returns + or - infinity.                                                       |

### **divd/divdp** *Double-precision floating-point division*

---

|                          |                                                                                                                             |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax - Standard</b> | double <b>_divd</b> (double x, double y);                                                                                   |
| <b>Syntax - FastRTS</b>  | #include < fastrts62x64x.h ><br>double <b>_divd</b> (double x, double y); or double <b>divdp</b> (double x, double y);      |
| <b>Defined in</b>        | divdp.asm                                                                                                                   |
| <b>Description</b>       | The quotient of two input 64-bit FP numbers is generated.                                                                   |
| <b>Special Cases</b>     | 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*

---

|                          |                                                                                                                                                     |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax - Standard</b> | <code>float _divf(float x, float y);</code>                                                                                                         |
| <b>Syntax - FastRTS</b>  | <code>#include &lt;fastrts62x64x.h &gt;</code><br><code>float _divf(float x, float y);</code> or <code>float <b>divsp</b>(float x, float y);</code> |
| <b>Defined in</b>        | <code>divsp.asm</code>                                                                                                                              |
| <b>Description</b>       | The quotient of two input 32-bit FP numbers is generated.                                                                                           |
| <b>Special Cases</b>     | 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*

---

|                          |                                                                                                                                                                                                                                                                                                                                     |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax - Standard</b> | <code>int _fixdi(double x);</code>                                                                                                                                                                                                                                                                                                  |
| <b>Syntax - FastRTS</b>  | <code>#include &lt;fastrts62x64x.h&gt;</code><br><code>int _fixdi(double x);</code> or <code>int <b>dpint</b>(double x);</code>                                                                                                                                                                                                     |
| <b>Defined in</b>        | <code>dpint.asm</code>                                                                                                                                                                                                                                                                                                              |
| <b>Description</b>       | An input 64-bit FP number is converted to a 32-bit signed integer.                                                                                                                                                                                                                                                                  |
| <b>Special Cases</b>     | Numbers with magnitude less than 1.0 return zero. Numbers greater than 32 bits return one of the following saturation values: <ul style="list-style-type: none"><li><input type="checkbox"/> <code>0x7fff_ffff</code> for positive numbers</li><li><input type="checkbox"/> <code>0x8000_0000</code> for negative numbers</li></ul> |

**`_fixfi/spint`**

*Single-precision floating-point to 32-bit signed integer*

---

|                          |                                                                                                                               |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax - Standard</b> | <code>int _fixfi(float x);</code>                                                                                             |
| <b>Syntax - FastRTS</b>  | <code>#include &lt;fastrts62x64x.h&gt;</code><br><code>int _fixfi(float x);</code> or <code>int <b>spint</b>(float x);</code> |
| <b>Defined in</b>        | <code>spint.asm</code>                                                                                                        |
| <b>Description</b>       | An input 32-bit FP number is converted to a 32-bit signed integer.                                                            |

## fixdli/dplong

---

**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_fff_fff` 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

**\_fltld/intdp**

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

---

**Syntax - Standard**      double **\_fltld** (int x);

**Syntax - FastRTS**      #include <fastrts62x64x.h>  
double **\_fltld** (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.

**\_fltld/longdp**

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

---

**Syntax - Standard**      double **\_fltld** (long x);

**Syntax - FastRTS**      #include <fastrts62x64x.h>  
double **\_fltld** (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.

---

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

---

**Syntax - Standard** float **\_ftlif** (long x);  
**Syntax - FastRTS** #include <fastrts62x64x.h>  
float **\_ftlif** (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.

---

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

---

**Syntax - Standard** double **\_fttud** (unsigned int x);  
**Syntax - FastRTS** #include <fastrts62x64x.h>  
double **\_fttud**(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.

---

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

---

**Syntax - Standard** float **\_fttuf** (unsigned int x);  
**Syntax - FastRTS** #include <fastrts62x64x.h>  
float **\_fttuf**(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.

---

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

---

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

## \_fltulf/ulongsp

---

|                    |                                                                              |
|--------------------|------------------------------------------------------------------------------|
| <b>Defined in</b>  | ulongdp.asm                                                                  |
| <b>Description</b> | 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*

---

|                          |                                                                                                                          |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax - Standard</b> | float <b>_fltulf</b> (unsigned long int x);                                                                              |
| <b>Syntax - FastRTS</b>  | #include <fastrts62x64x.h><br>float <b>_fltulf</b> (unsigned long int x); or float <b>ulongsp</b> (unsigned long int x); |
| <b>Defined in</b>        | ulongsp.asm                                                                                                              |
| <b>Description</b>       | An input 40-bit unsigned long integer is converted to a 32-bit SP FP number.                                             |

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

---

|                          |                                                                                                                      |
|--------------------------|----------------------------------------------------------------------------------------------------------------------|
| <b>Syntax - Standard</b> | double <b>_mpyd</b> (double x);                                                                                      |
| <b>Syntax - FastRTS</b>  | #include <fastrts62x64x.h><br>double <b>_mpyd</b> (double x, double y); or double <b>mpydp</b> (double x, double y); |
| <b>Defined in</b>        | mpydp.asm                                                                                                            |
| <b>Description</b>       | The product of two input 64-bit FP numbers is generated.                                                             |
| <b>Special Cases</b>     | Underflow or exponents $< 2$ ( $2^{-1022} = 4.45 \times 10^{-308}$ ) returns zero; overflow returns + or - infinity. |

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

---

|                          |                                                                                                                |
|--------------------------|----------------------------------------------------------------------------------------------------------------|
| <b>Syntax - Standard</b> | float <b>_mpyf</b> (float x);                                                                                  |
| <b>Syntax - FastRTS</b>  | #include <fastrts62x64x.h><br>float <b>_mpyf</b> (float x, float y); or float <b>mpysp</b> (float x, float y); |
| <b>Defined in</b>        | mpysp.asm                                                                                                      |
| <b>Description</b>       | 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

## **\_subf/subsp**

---

**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

---

### **`_cvtfd/dpsp`** *Convert double-precision to single-precision floating point*

---

|                          |                                                                                                                                     |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax - Standard</b> | <code>float _cvtfd(double x);</code>                                                                                                |
| <b>Syntax - FastRTS</b>  | <code>#include &lt; fastrts62x64x.h&gt;</code><br><code>float _cvtfd(double x);</code> or <code>float <b>dpsp</b>(double x);</code> |
| <b>Defined in</b>        | <code>dpsp.asm</code>                                                                                                               |
| <b>Description</b>       | An input 64-bit FP number is converted to a 32-bit FP number.                                                                       |
| <b>Special Cases</b>     | Underflow returns zero; overflow returns + or - SP infinity.                                                                        |

---

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

---

|                          |                                                                                                                                     |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>Syntax - Standard</b> | <code>double _cvtfd(float x);</code>                                                                                                |
| <b>Syntax - FastRTS</b>  | <code>#include &lt; fastrts62x64x.h&gt;</code><br><code>double _cvtfd(float x);</code> or <code>double <b>spdp</b>(float x);</code> |
| <b>Defined in</b>        | <code>spdp.asm</code>                                                                                                               |
| <b>Description</b>       | An input 32-bit FP number is converted to a 64-bit FP number.                                                                       |
| <b>Special Cases</b>     | Underflow returns zero; overflow returns + or - infinity.                                                                           |

# Performance Considerations

---

---

---

This appendix describes the sample performance of the C62x/64x FastRTS. It also provides information about software updates and customer support issues.

| <b>Topic</b>                                | <b>Page</b> |
|---------------------------------------------|-------------|
| <b>A.1 Performance Considerations .....</b> | <b>A-2</b>  |
| <b>A.2 FastRTS Software Updates .....</b>   | <b>A-4</b>  |
| <b>A.3 FastRTS Customer Support .....</b>   | <b>A-4</b>  |



## 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

| Function | Alternate Name | Data  | C64x        |         |                          | C62x        |         |                          |
|----------|----------------|-------|-------------|---------|--------------------------|-------------|---------|--------------------------|
|          |                |       | rts6400.lib | FastRTS | RTS/<br>FastRTS<br>Ratio | rts6200.lib | FastRTS | RTS/<br>FastRTS<br>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   | dpoint         | 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                     |
| _fltld   | intdp          | 64 FP | 35          | 21      | 1.67                     | 40          | 21      | 1.90                     |
| _fltlf   | longsp         | 32 FP | 16          | 14      | 1.14                     | 17          | 15      | 1.13                     |
| _fltld   | 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)

| Function | Alternate Name | Data  | C64x        |         |                          | C62x        |         |                          |
|----------|----------------|-------|-------------|---------|--------------------------|-------------|---------|--------------------------|
|          |                |       | rts6400.lib | FastRTS | RTS/<br>FastRTS<br>Ratio | rts6200.lib | FastRTS | RTS/<br>FastRTS<br>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                     |
| _cvtf    | 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.

# 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*.



- 40-bit signed long integer to single-precision floating point 4-7
- 40-bit unsigned long integer to double-precision floating point 4-7
- 40-bit unsigned long integer to single-precision floating point 4-8
- double-precision floating-point to 32-bit signed integer 4-3
- double-precision floating-point to 32-bit unsigned integer 4-4
- double-precision floating-point to 40-bit signed long integer 4-4
- double-precision floating-point to 40-bit unsigned long integer 4-5
- double-precision to single-precision floating point 4-11
- single-precision floating-point to 32-bit signed integer 4-3
- single-precision floating-point to 32-bit unsigned integer 4-5
- single-precision floating-point to 40-bit signed long integer 4-4
- single-precision floating-point to 40-bit unsigned long integer 4-6
- single-precision to double-precision floating point 4-11

customer support A-4

## D

- data types, FastRTS 2-4
- digital signal processor (DSP), defined B-1
- divdp function 4-2
- division
  - double-precision 4-2
  - single-precision 4-3
- divsp function 4-3
- double-precision functions
  - \_addd 4-2
  - \_cvtfd 4-11
  - \_divd 4-2
  - \_fixdi 4-3
  - \_fixdli 4-4
  - \_fixdu 4-4

- \_fixdul 4-5
- \_fltld 4-6
- \_ftlid 4-6
- \_fltud 4-7
- \_ftlud 4-7
- \_mpyd 4-8
- \_subd 4-9
- adddp 4-2
- divdp 4-2
- dpint 4-3
- dplong 4-4
- dpoint 4-4
- dpulong 4-5
- intdp 4-6
- longdp 4-6
- mpydp 4-8
- recipd 4-9
- recipdp 4-9
- spdp 4-11
- subdp 4-9
- uintdp 4-7
- ulongdp 4-7

- double-precision routines, table listing 1-2
- dpint function 4-3
- dplong function 4-4
- dpsp function 4-11
- dpoint function 4-4
- dpulong function 4-5

## F

- FastRTS
  - archive contents 2-2
  - argument conventions 3-2
  - arguments and data types 2-4
  - calling a function from assembly 2-5
  - calling a function from C 2-5
  - customer support A-4
  - data types, table 2-4
  - defined B-1
  - features and benefits 1-3
  - function, reference 4-1
  - function names comparison table 3-4
  - how to install 2-3
  - how to rebuild FastRTS 2-6

- include directory 2-5
- introduction 1-2
- performance A-2
- software updates A-4

fastrts62x64x.h header file 2-5

fastrts67x.h header file 2-5

features and benefits 1-3

function

- calling a FastRTS function from assembly 2-5
- calling a FastRTS function from C 2-5
- names comparison table 3-4

## I

- include directory 2-5
- installing FASTRTS 2-3
- intdp function 4-6
- intsp function 4-6

## L

- least significant bit (LSB), defined B-2
- linker, defined B-2
- little endian, defined B-2
- longdp function 4-6
- longsp function 4-7

## M

- mpydp function 4-8
- mpysp function 4-8
- multiplication
  - double-precision 4-8
  - single-precision 4-8

## P

- performance A-2

## R

- rebuilding FastRTS 2-6
- recipd function 4-9
- recipdp function 4-9
- recipf function 4-9
- reciprocal
  - double-precision 4-9
  - single-precision 4-9
- recipsp function 4-9
- routines, FastRTS 1-2

## S

- single-precision functions
  - \_addf 4-2
  - \_cvtidf 4-11
  - \_divf 4-3
  - \_fixfi 4-3
  - \_fixfli 4-4
  - \_fixfu 4-5
  - \_fixful 4-6
  - \_fltif 4-6
  - \_fltlf 4-7
  - \_fltuf 4-7
  - \_fltulf 4-8
  - \_mpyf 4-8
  - \_subf 4-10
  - addsp 4-2
  - divsp 4-3
  - dpsp 4-11
  - intsp 4-6
  - longsp 4-7
  - mpysp 4-8
  - recipf 4-9
  - recipsp 4-9
  - spint 4-3
  - splong 4-4
  - spuint 4-5
  - spulong 4-6
  - subsp 4-10
  - uintsp 4-7
  - ulongsp 4-8

- single-precision routines, table listing 1-2



software updates A-4  
spdp function 4-11  
spint function 4-3  
splong function 4-4  
spuint function 4-5  
spulong function 4-6  
subdp function 4-9  
subsp function 4-10

subtraction  
  double-precision 4-9  
  single-precision 4-10

## U

uintdp function 4-7  
uintsp function 4-7  
ulongdp function 4-7  
ulongsp function 4-8