

3 RUN-TIME LIBRARY

Overview

The C and C++ run-time libraries are collections of functions, macros, and class templates that you can call from your source programs. Many functions are implemented in the TigerSHARC family assembly language. C and C++ programs depend on library functions to perform operations that are basic to the C and C++ programming environments. These operations include memory allocation, character and string conversions, and math calculations. The libraries also include a number of signal processing functions that ease DSP code development. Using the library simplifies your software development by providing code for a variety of common needs.

The ccts compiler provides a broad collection of library functions including those required by the ANSI standard and many others of value for DSP programming. In addition to the Standard C Library, this release of the compiler software includes the Abridged C++ Library, a conforming subset of the Standard C++ Library. The Abridged C++ Library includes the Embedded C++ and Embedded Standard Template Libraries.

For more information on the algorithms on which many of the C library's math functions are based, see the Cody and Waite text "Software Manual for the Elementary Functions" from Prentice Hall (1980). For more information on the C++ library portion of the ANSI/ISO Standard for C++, see the Plauger text "Draft Standard C++ Library" from Prentice Hall (1994) (ISBN: 0131170031).

Overview

The sections of this chapter present the following information on the compiler:

- [“C and C++ Run-Time Libraries Guide” on page 3-3](#) contains introductory information about the ANSI/ISO standard C and C++ libraries. It also provides information about the ANSI-standard and ADI-special header files and built-in functions that are included with this release of the ccts compiler.
- [“Run-time Library Reference” on page 3-29](#) contains reference information about the C run-time functions that are included with this release of the ccts compiler.

The C++ library reference information in HTML format is included on the software distribution CD-ROM. To access the reference files from the VisualDSP++ environment, use the Help Topics command (Help menu) and select the Reference book icon. From the C++ Run-Time Library topic, you can open any of the library files. You can also manually access the HTML files using a web browser.

C and C++ Run-Time Libraries Guide

The C and C++ run-time libraries contain routines that you can call from your source program. This section describes how to use the libraries and provides information on the following topics:

- [“Calling Library Functions” on page 3-3](#)
- [“Linking Library Functions” on page 3-4](#)
- [“Working With Library Source Code” on page 3-6](#)
- [“Working With Library Header Files” on page 3-7](#)
- [“Run-time Library Reference” on page 3-29](#)
- [“Abridged C++ Library Support” on page 3-21](#)

For information about the C library’s contents, see [“Run-time Library Reference” on page 3-29](#). For information about the Abridged C++ library’s contents, see an overview of the [“Abridged C++ Library Support” on page 3-21](#) and on-line Help.

Calling Library Functions

To use a C/C++ library function, call the function by name and give the appropriate arguments. The name and arguments for each function appear on the function’s reference page. The reference pages appear in the [“Run-time Library Reference”](#) section beginning on [page 3-29](#) and in the “C/C++ Run-Time Library” topic of the on-line Help. Like other functions you use, library functions should be declared. Declarations are supplied in header files. For more information about the header files, see [page 3-7](#).

C and C++ Run-Time Libraries Guide

Function names are C/C++ function names. If you call a C or C++ run-time library function from an assembly language program, you must use the assembly version of the function name: prefix an underscore on the name. For more information on the naming conventions, see [“C/C++ and Assembly Interface” on page 2-116](#).

 You can use the archiver, `elfar`, described in the *VisualDSP++ 2.0 Linker and Utilities Manual for TigerSHARC DSPs*, to build library archive files of your own functions.

Linking Library Functions

The C/C++ Run-Time Library is actually organized as three libraries:

- C Run-Time library—Comprises of all the functions that are defined by the ANSI standard
- C++ Run-Time library
- DSP Run-Time library—Contains additional library functions supplied by Analog Devices that provide services commonly required by DSP applications.

Two variants of each library are supplied, one of which is suitable for running on the ADSP-TS001 DSP and the other which is suitable for running on the ADSP-TS101 DSP. [Table 3-1](#) catalogs the names of each of the libraries and also lists other files that are used while linking a program.

Table 3-1. C and C++ Files and Libraries

Description	ADSP-TS001 DSP	ADSP-TS101 DSP
C Run-time library functions	libc_TS001.dlb	libc_TS101.dlb
Threadsafe C Run-time library functions	libc_TS001_mt.dlb	libc_TS101_mt.dlb
C++ Run-time library functions	libcpp_TS001.dlb	libcpp_TS101.dlb
Threadsafe C++ Run-time library functions	libcpp_TS001_mt.dlb	libcpp_TS101_mt.dlb
C++ Run-time library support functions	libcpprt_TS001.dlb	libcpprt_TS101.dlb
Threadsafe C++ Run-time library support functions	libcpprt_TS001_mt.dlb	libcpprt_TS101_mt.dlb
DSP Run-time library functions	libdsp_TS001.dlb	libdsp_TS101.dlb
I/O library functions	libio_TS001.dlb	libio_TS101.dlb
Threadsafe I/O library functions	libio_TS001_mt.dlb	libio_TS101_mt.dlb
Simulator services	libsim.dlb	libsim.dlb
Start-up file - calls setup routines and main	ts_hdr_TS001.doj	ts_hdr_TS101.doj
Start-up file for multi-threaded applications—calls setup routines and main	ts_hdr_TS001_mt.doj	ts_hdr_TS101_mt.doj
Exit routine	ts_exit_TS001.doj	ts_exit_TS101.doj
Exit routine for multi-threaded applications	ts_exit_TS001_mt.doj	ts_exit_TS101_mt.doj

C and C++ Run-Time Libraries Guide

When you call a run-time library function, the call creates a reference that the linker resolves when linking your program. One way to direct the linker to the library's location is to use the default Linker Description File (ADSP-TS<your_target>.ldf).

If you are not using the default LDF, then either add the appropriate library/libraries to the LDF used for your project, or use the compiler's -l switch to specify the library to be added to the link line. For example, the switches -lc_TS001 -ldsp_TS001 add libc_TS001.dlb and libdsp_TS001.dlb to the list of libraries to be searched by the linker. For more information on the LDF file, see the *VisualDSP++ 2.0 Linker and Utilities Manual for TigerSHARC DSPs*

Working With Library Source Code

The source code for some of the functions in the run-time library is provided with your VisualDSP++ software. By default, the installation program copies the source code to a subdirectory of the directory where the run-time libraries are kept, named ...\\TS\\lib\\src. The directory contains the source for the main program start-up and exit functions and for the functions associated with the signal handler dispatching mechanism.

The source code is provided so you can customize any particular function for your own needs. To modify these files, you need proficiency in TigerSHARC assembly language and an understanding of the run-time environment, as explained in [“C/C++ Run-Time Model” on page 2-97](#).

Before you make any modifications to the source code, copy the source code to a file with a different filename and rename the function itself. Test the function before you use it in your system to verify that it is functionally correct. If you do not intend to modify any of the run-time library functions and are not interested in using the source file reference, you can delete this directory and its contents to conserve disk space.



Analog Devices supports the run-time library functions only as provided.

Working With Library Header Files

When you use a library function in your program, you should also include the function's header with the `#include` preprocessor command. The header file for each function is identified in the Synopsis section of the function's reference page. Header files contain function prototypes. The compiler uses these prototypes to check that each function is called with the correct arguments. A list of ANSI standard header files with brief descriptions appear in [Table 3-3 on page 3-9](#); other header files are described throughout this section.

The header file defines the non-suffixed names (e.g. `sin`) as the 32-bit versions (e.g. `sinf`) if the compiler is treating doubles as 32 bits or as the 64-bit version (for example `sind`). This lets you use the non-suffixed names with arguments of type `double`, regardless of whether doubles are 32- or 64-bits.

The routines suffixed with `f` always require 32-bit arguments, and the routines suffixed with `d` always require 64-bit arguments, regardless of whether the `double` type is 32 or 64 bits.

Some of these routines set the `errno` global variable, as required by the C standard. To use `errno`, you must include the header file `errno.h`.

By default, the ccts compiler makes the `double` type 32 bits, for high performance. You can select a compiler option to make the `double` type 64 bits for conformance with the C standard, but this makes computations with `double` values slower. In either case, the `float` type is 32 bits, and the `long double` type is 64 bits.

For more information about the supported data types, refer to [“Data Type Sizes” on page 2-45](#).



In the detailed descriptions, the old “K&R” style has been used for describing the function interfaces, so that the parameter descriptions can be written separately from the function definition itself. The header files have proper ANSI/ISO prototypes.

Standard C Library Header Files

The following C standard header files are supplied with this release of the TigerSHARC compiler. You should use a C standard text to augment the information supplied in this chapter.

[Table 3-2](#) lists the header files that contain ANSI standard run-time environment macros for error handling, standard definitions, limits, and floating-point variables. These do not contain individual functions; they consist of macros and type definitions. See the [“Standard C Library Header File Descriptions” on page 3-10](#) for more detailed descriptions.

Table 3-2. ANSI Standard Run-Time Environment Macro Header Files

Header File	Description
<code>errno.h</code>	Error handling
<code>float.h</code>	Floating-point implementation parameters
<code>limits.h</code>	Implementation limits
<code>stddef.h</code>	Standard definitions

Eleven C standard run-time header files that contain ANSI standard functions are supplied with the present release of the ccts compiler. A list of the Standard C Library function headers appears in [Table 3-3](#). See [“Standard C Library Header File Descriptions” on page 3-10](#) for more detailed descriptions.

Table 3-3. ANSI Standard Run-Time Function Header Files

Header File	Description
<code>assert.h</code>	Diagnostics
<code>ctype.h</code>	Character handling
<code>locale.h</code>	Localization
<code>math.h</code>	Basic math functions
<code>setjmp.h</code>	Non-local jumps
<code>signal.h</code>	Signal handling
<code>stdarg.h</code>	Variable arguments
<code>stdio.h</code>	Input/output
<code>stdlib.h</code>	Additional math functions
<code>string.h</code>	String handling
<code>time.h</code>	Time and data handling

C and C++ Run-Time Libraries Guide

Standard C Library Header File Descriptions

`assert.h`

The `assert.h` header contains the `assert` macro.

`ctype.h`

The `ctype.h` header contains functions for character handling, such as `isalpha`, `tolower`, and so forth.

`errno.h`

The `errno.h` header file provides access to `errno` and also defines macros for associated error codes.

`float.h`

The `float.h` header contains definitions of size and precision values for each C floating point data type.

`limits.h`

The `limits.h` header contains definitions of maximum and minimum values for each C data type other than floating-point.

`locale.h`

The `locale.h` header contains definitions for expressing numeric, monetary, time, and other data.

`math.h`

The `math.h` header includes trigonometric, power, logarithmic, exponential, and other miscellaneous functions. The library contains the functions specified by the C standard along with implementations for `float` and `long double`.

This header file also provides prototypes for a number of additional math functions provided by Analog Devices, such as `favg`, `fmax`, `fclip`, `copysign`.

 Some of the functions exist as both integer and floating point. The floating point functions typically have an `f` prefix. Make sure you use the correct one. The C language provides for implicit type conversion, so the following sequence produces surprising results with no warnings:

```
float x,y;
y = abs(x);
```

The value in `x` is truncated to an integer prior to calculating the absolute value, then reconverted to floating point for the assignment to `y`.

A number of functions (including `fabs`, `favg`, `fmax`, `fmin`, `fclip`, and `copysign`) are implemented via intrinsics (provided the header file has been `#include'd`) that map to single machine instructions.

 If the header is not included, the library implementation is used instead, at a considerable loss in efficiency.

Individual function descriptions focus on the 32-bit float version. When the full range is given for domain, the 64-bit float interfaces are not limited to 3.4×10^{38} .

setjmp.h

The `setjmp.h` header contains `setjmp` and `longjmp` for non-local jumps.

signal.h

The `signal.h` header provides function prototypes for the standard ANSI `signal.h` routines and also for several TigerSHARC family extensions, such as `interrupt()`.

C and C++ Run-Time Libraries Guide

The signal handling functions process conditions (hardware signals) that can occur during program execution. They determine the way that your C program responds to these signals. The functions are designed to process such signals as external interrupts and timer interrupts.

`stdarg.h`

The `stdarg.h` header contains definitions needed for functions that accept a variable number of arguments. Callers of such functions must include a prototype.

`stddef.h`

The `stddef.h` header contains a few common definitions useful for portable programs, such as `size_t`.

`stdio.h`

The `stdio.h` header contains a subset of the C standard's I/O functionality. Always include the header file in your source if you use any of its facilities because the header file contains dual support for when type `double` is 32 bits and for when type `double` is 64 bits. Failure to include the header file will result in a linker failure as the compiler must see a correct function prototype in order to generate the correct calling sequence.

The following facilities are not available in this release:

- The stream positioning functions `fgetpos`, `fseek`, `fsetpos`, `ftell`, `rewind`
- The file handling functions `remove`, `rename`, `tmpfile`, `tmpnam`
- Support for values of type `long long` by the `printf` and `scanf` functions
- Support for values of type `long double` when type `double` is the same size as type `float` by the `printf` and `scanf` functions

A faster set of functions is available for applications that print only to standard output. These functions are linked into an application if you compile with the switch `-flags-link -MD__USING_LIBSIM=1`. This switch forces the linker to link against the library `libsिम.dlb`. This library contains a limited set of `stdio.h` facilities that are executed on the host of the VisualDSP++ debugger rather than inside the debugger's target. This type of execution leads to smaller applications and faster output. The following functions are supported by the `libsिम.dlb` library for this release only:

`printf`, `sprintf`, `fprintf`

All three of these functions from `libsिम.dlb` use the `L` modifier, as in `%LF`, to specify a conversion for a long double (64-bit float) argument. They also use the `ll` modifier, as in `%lld`, to specify a conversion for a long long (64-bit integer) argument. The `fprintf` routine currently ignores its `FILE*` stream argument and always prints to standard output.

stdlib.h

The `stdlib.h` header offers general utilities specified by the C standard. These include some integer math functions such as `abs`, `div`, and `rand`; general string-to-numeric conversions; memory allocation functions such as `malloc` and `free`; and termination functions such as `exit`. This library also contains miscellaneous functions such as `bsearch` and `qsort`.

This header also provides prototypes for a number of additional integer math functions provided by Analog Devices, such as `avg`, `max`, `clip`; also `count_ones` and `addbitrev`.



Some of the functions included in this file exist as both integer and floating point. The floating point functions typically have an `f` prefix. Make sure you are using the correct type.

C and C++ Run-Time Libraries Guide

A number of functions (including `abs`, `avg`, `max`, `min`, `clip`, `count_ones`, and `addbitrev`) are implemented via intrinsics (provided the header file has been `#include'd`) which map to single machine instructions.



If the header is not included, the library implementation is used instead, at a considerable loss in efficiency.

`string.h`

The `string.h` header contains string handling functions, including `strcpy` and `memcpy`.

`time.h`

The `time.h` header contains standard definitions for time-handling functions. TigerSHARC does not support the underlying functionality.

DSP Header Files

In addition to the ANSI standard library of C functions, this release of the compiler contains a broad collection of Analog Devices extensions to the standard library for DSP programming.

`complex` — Basic Complex Arithmetic Functions

The `complex` header file contains utility functions that provide the type definitions and basic arithmetic operations on `complex_float` and `complex_long_double` variables.

The complex functions are listed with their 32 bits float interface (i.e. float in C) only. The old style K&R form of representing the parameters is used in this section, for convenience purposes only.

The following structures are used to represent complex numbers in rectangular coordinates:

```
typedef struct
{
    float re;
    float im;
} complex_float;

typedef struct
{
    long double re;
    long double im;
} complex_long_double;
```

filter— DSP Filters and Transformations

The filter header file contains various filters used in digital signal processing, including FIR, IIR, and so forth. A-law and μ -law companders are also provided. The header file also contains functions that perform key transformations used in DSP applications, including FFT and convolve. See [“complex — Basic Complex Arithmetic Functions” on page 3-14](#) for definitions of the complex types.

For efficiency, the `twiddle` table is calculated once, during initialization, and then provided to the FFT routine as a separate parameter. You must declare the variable and initialize it prior to calling an FFT function. An initialization function, `twidfft`, is provided.

Various different versions of the FFT are provided, including `cfft`, `rfft`, `ifft`, `cfft2d`, `rfft2d`, `ifft2d`. The number of points is provided as an argument. The library uses radix 2 or radix 4 implementations as appropriate.

Several FFTs of different sizes can all be accommodated with the same `twiddle` factor table. To do this, allocate the table at the maximum size.

C and C++ Run-Time Libraries Guide

Each FFT has an additional parameter, the “stride” of the twiddle table. To use the whole table, specify a stride of 1. If your FFT uses only half the points of the largest, the stride should be 2 (this takes only every other element).

The functions described by this header make certain assumptions about their arguments, in order to achieve high efficiency:

- The FFT routines require that the `in`, `t`, and `out` arrays be quad-word aligned, and the `w` (twiddle) array be long-word aligned.
- The filter routines require that the input array (and coefficients) for FIR be quad-word aligned.
- The A-law and μ -law companders require that the input and output arrays be quad-word aligned.



Failure to observe these constraints will result in incorrect operation.

libsim — Simulator Services

The `libsim` header file defines services that are provided by the VisualDSP++ debugging environment. The header file contains the function `__emuc1k`, which returns the current simulator cycle count. The header file also contains several utility print routines that may be useful for assembly language development since they are much easier to call than `printf`. [Table 3-4](#) lists the utility print routines and provides a brief description of each.

Table 3-4. TigerSHARC Print Routines

Function	Description
<code>__print_int</code>	prints 32-bit int
<code>__print_uint</code>	prints unsigned 32-bit int

Table 3-4. TigerSHARC Print Routines

Function	Description
<code>__print_float</code>	prints 32-bit float
<code>__print_double</code>	prints 64-bit long double float
<code>__print_longlong</code>	prints 64-bit signed int
<code>__print_ulonglong</code>	prints 64-bit unsigned int

-  The list of routines contains names suitable for calling from a C program. If you call them from an assembly program, add an additional leading underscore to the name.
-  These functions conform to the C calling conventions, as described in [“C/C++ Run-Time Model” on page 2-97](#).

matrix — Matrix Functions

The matrix header file contains matrix functions for operating on real and complex matrices, both matrix-scalar and matrix-matrix operations. See [“complex — Basic Complex Arithmetic Functions” on page 3-14](#) for definitions of the complex types.

The matrix functions are listed with their 32 bits float interface (i.e. float in C) only. The old style K&R form of representing the parameters is used in this section for convenience purposes only.

The matrix functions are each provided in three forms—for the three floating point types. For brevity, only the float form is listed in the detailed descriptions.

C and C++ Run-Time Libraries Guide

For example, under `cvecsadd`, the following list is provided in the Synopsis:

```
cvecsaddf(float a[], float b, float c[], int n)
```

But, the library also provides:

```
cvecsadd(double a[], double b, double c[], int n)
cvecsadd(long double a[], long double b, long double c[], int
n)
```

The following structures are used to represent complex numbers:

```
typedef struct
{
    float re;
    float im;
} complex_float;

typedef struct
{
    long double re;
    long double im;
} complex_long_double;
```

The functions described by this header make certain assumptions about their arguments, in order to achieve high efficiency:

- Input array arguments are constant, i.e., their contents do not change during the course of the routine. In particular, this means the input arguments do not overlap with any output argument.
- Input array arguments are quad-word aligned, and output array arguments are at least double-word aligned. The compiler provides this alignment for all top-level arrays; you must not pass an argument which points at an arbitrary, non-aligned location in an array.



Failure to observe these constraints will result in incorrect operation.

stats — Statistical Functions

The stats header file contains statistical functions such as `autocoh` and `crosscoh`.

The statistics routines require that the argument array be quad-word aligned.

- Input array arguments are constant, i.e., their contents do not change during the course of the routine. In particular, this means the input arguments do not overlap with any output arguments.
- Input array arguments are quad-word aligned, and output array arguments are at least double-word aligned. The compiler provides this alignment for all top-level arrays; you must not pass an argument that points at an arbitrary, non-aligned location in an array.



Failure to observe these constraints will result in incorrect operation.

vector — Vector Functions

The vector contains functions for operating on real and complex vectors, both vector-scalar and vector-vector operations. See [“complex — Basic Complex Arithmetic Functions” on page 3-14](#) for definition of the complex types.

The vector functions are listed with their 32 bits float interface (i.e. float in C) only. The old style K&R form of representing the parameters is used in this section for convenience purposes only.

The vector functions are each provided in three forms—for the three floating point types. For brevity, only the float form is listed in the detailed descriptions.

C and C++ Run-Time Libraries Guide

For example, under `cvecsadd`, the following list is provided in the Synopsis:

```
cvecsaddf(float a[], float b, float c[], int n)
```

but the library also provides

```
cvecsadd(double a[], double b, double c[], int n)
cvecsadd(long double a[], long double b, long double c[], int
n)
```

The following structures are used to represent complex numbers:

```
typedef struct
{
    float re;
    float im;
} complex_float;

typedef struct
{
    long double re;
    long double im;
} complex_long_double;
```

The functions described by this header make certain assumptions about their arguments, in order to achieve high efficiency:

- input array arguments are constant, i.e., their contents will not change during the course of the routine. In particular, this means the input arguments do not overlap with any output argument.
- input array arguments are quad-word aligned, and output array arguments are at least double-word aligned. The compiler provides this alignment for all top-level arrays; you must not pass an argument which points at an arbitrary, non-aligned location in an array.



Failure to observe these constraints will result in incorrect operation.

window — Window Generators

The window header file contains various functions to generate windows based on various methodologies.

For all window functions, a stride parameter a can be used to space the window values. The window length parameter n equates to the number of elements in the window. Therefore, for a stride a of 2 and a length n of 10, an array of length 20 is required, where every second entry is untouched.

Abridged C++ Library Support

When in C++ mode, the ccts compiler can call a large number of functions from the Abridged Library, a conforming subset of the C++ library.

The Abridged Library has two major components: Embedded C++ Library (EC++) and Embedded Standard Template Library (ESTL). The Embedded C++ Library is a conforming implementation of the Embedded C++ Library as specified by the Embedded C++ Technical Committee.

This section lists and briefly describes the following components of the Abridged Library:

- [“Embedded C++ Library Header Files” on page 3-22](#)
- [“C++ Header Files for C Library Facilities” on page 3-24](#)
- [“Embedded Standard Template Library Header Files” on page 3-26](#)

For more information on the Abridged Library, see online Help.

Embedded C++ Library Header Files

<complex>

The `complex` header defines the generic class `complex`, which supports complex arithmetic and assignment. Predefined types include `complex_float` and `complex_long_double`.

 This implementation does not support the full set of complex operations as specified by the C++ standard. In particular, it does not support either the transcendental functions or the I/O operators `<<` and `>>`.

 The `complex` header and the C library header file `complex.h` refer to two different and incompatible implementations of the complex data type.

<exception>

The `exception` header defines the `exception` and `bad_exception` classes and several functions for exception handling.

<fract>

The `fract` header defines the `fract` data type, which supports fractional arithmetic, assignment, and type-conversion operations. The header file is fully described under [“C++ Fractional Type Support” on page 2-88](#).

<fstream>

The `fstream` header defines the `filebuf`, `ifstream`, and `ofstream` classes for external file manipulations.

<iomanip>

The `iomanip` header declares several `iostream` manipulators. Each manipulator accepts a single argument.

<ios>

The `ios` header defines several classes and functions for basic iostream manipulations. Note that most of the iostream header files include `ios.h`.

<iosfwd>

The `iosfwd` header declares forward references to various iostream template classes defined in other standard headers.

<iostream>

The `iostream` header declares most of the iostream objects used for the standard stream manipulations.

<istream>

The `istream` header defines the `istream` class for iostream extractions. Note that most of the iostream header files include `istream.h`.

<new>

The `new` header declares several classes and functions for memory allocations and deallocations.

<ostream>

The `ostream` header defines the `ostream` class for iostream insertions.

<sstream>

The `sstream` header defines the `stringbuf`, `istringstream`, and `ostringstream` classes for various string object manipulations.

<stdexcept>

The `stdexcept` header defines a variety of classes for exception reporting.

C and C++ Run-Time Libraries Guide

<streambuf>

The `streambuf` header defines the `streambuf` classes for basic operations of the `iostream` classes. Note that most of the `iostream` header files include `streambuf.h`.

<string>

The `string` header defines the `string` template and various supporting classes and functions for string manipulations.

 Objects of the `string` type should not be confused with null-terminated C strings.

<strstream>

The `strstream` header defines the `strstreambuf`, `istrstream`, and `ostrstream` classes for `iostream` manipulations on allocated, extended, and freed character sequences.

C++ Header Files for C Library Facilities

For each C standard library header there is a corresponding standard C++ header. If the name of a C standard library header file is `foo.h`, then the name of the equivalent C++ header file will be `cfoo`. For example, the C++ header file `<cstdio>` provides the same facilities as the C header file `<stdio.h>`. [Table 3-5](#) lists the C++ header files that provide access to the C library facilities.

Normally, the C standard headers files may be used to define names in the C++ global namespace while the equivalent C++ header files define names in the `std` namespace. However, the `std` namespace is not supported in this release of the compiler, and the effect of including one of the C++ header files listed in [Table 3-5](#) is the same as including the equivalent C standard library header file.

Table 3-5. C++ Header Files for C Library Facilities

Header	Description
<cassert>	Enforces assertions during function executions
<cctype>	Classifies characters
<cerrno>	Tests error codes reported by library functions
<cfloating>	Tests floating-point type properties
<climits>	Tests integer type properties
<locale>	Adapts to different cultural conventions
<cmath>	Provides common mathematical operations
<csignal>	Executes non-local goto statements
<csignal>	Controls various exceptional conditions
<cstdarg>	Accesses a various number of arguments
<stddef>	Defines several useful data types and macros
<stdio>	Performs input and output
<stdlib>	Performs a variety of operations
<string>	Manipulates several kinds of strings

Embedded Standard Template Library Header Files

Templates and the associated header files are not part of the Embedded C++ standard, but are supported by the ccts compiler in C++ mode. The fifteen Embedded Standard Template Library headers are:

<algorithm>

The `algorithm` header defines numerous common operations on sequences.

<deque>

The `deque` header defines a deque template container.

<functional>

The `functional` header defines numerous function objects.

<hash_map>

The `hash_map` header defines two hashed map template containers.

<hash_set>

The `hash_set` header defines two hashed set template containers.

<iterator>

The `iterator` header defines common iterators and operations on iterators.

<list>

The `list` header defines a list template container.

<map>

The `map` header defines two map template containers.

<memory>

The `memory` header defines facilities for managing memory.

<numeric>

The `numeric` header defines several numeric operations on sequences.

<queue>

The `queue` header defines two queue template container adapters.

<set>

The `set` header defines two set template containers.

<stack>

The `stack` header defines a stack template container adapter.

<utility>

The `utility` header defines an assortment of utility templates

<vector>

The `vector` header defines a vector template container.

The Embedded C++ library also includes several headers for compatibility with traditional C++ libraries:

C and C++ Run-Time Libraries Guide

`fstream.h`

The `fstream.h` header defines several iostreams template classes that manipulate external files.

`iomanip.h`

The `iomanip.h` header declares several iostreams manipulators that take a single argument.

`iostream.h`

The `iostream.h` header declares the iostreams objects that manipulate the standard streams.

`new.h`

The `new.h` header declares several functions that allocate and free storage.

Run-time Library Reference

The run-time library is a collection of functions that you can call from your programs. The following sections describe the format used to present the functions.

Notation Conventions. An interval of numbers is indicated by the minimum and maximum, separated by a comma, and enclosed in two square brackets, two parentheses, or one of each. A square bracket indicates that the endpoint is included in the set of numbers; a parenthesis indicates that the endpoint is not included.

Reference Format. The reference pages for the library functions use the following format.

Name and Purpose of the function

Synopsis—Required header file and functional prototype. For ease of documentation, the interface for some functions is presented using the “K&R” style; the header file contains a proper ANSI/ISO prototype.

Description—Function specification

Algorithm—High level mathematical representation of the function

Domain—Range of values supported by the function

Notes—Miscellaneous information

Run-time Library Reference

abs

absolute value

Synopsis

```
#include <stdlib.h>
int abs (int x)
float fabsf (float x)
double fabs (double x)
long double fabsl (long double x)
```

Description

Gives the absolute value of a number.

Algorithm

Return $|x|$

Domain

Full range for given type.

acos

arc cosine

Synopsis

```
#include <math.h>

float acosf (float x)
double acos (double x)
long double acosd (long double x)
```

Description

This function computes the arc cosine of number x . The output is in the range $[0 \text{ to } \pi]$ radians.

Algorithm

return = $\cos^{-1}(x)$

Domain

$x = [-1.0 \dots 1.0]$

Run-time Library Reference

a_compress

a-law compression

Synopsis

```
#include <filter.h>
void a_compress(in, out, n)
const int in[]; /* Input array */
int out[];      /* Output array */
int n;          /* number of elements to be compressed */
```

Description

This function performs A-law compression (ITU rec. G.711) on the elements of the input vector `in` and outputs expanded data into the vector `out`. The function has been optimized and requires that both the input and output vectors are quad-word aligned.

Algorithm

$C(k)$ =a-law compression of $A(k)$ for $k=0$ to n

Domain

content of input array: -4096 to 4095

addbitrev

bit-reversed adder

Synopsis

```
#include <stdlib.h>
int addbitrev (int a, int b)
```

Description

Adds the two arguments using the bit-reversed adder. This is binary addition in which the carries are propagated to the right, rather than to the left. Therefore:

`addbitrev(0x50, 0x40)` yields `0x30`

This is useful in algorithms such as the FFT and interleaver.

Algorithm

See description.

Run-time Library Reference

a_expand

a-law expansion

Synopsis

```
#include <filter.h>
void a_expand(in, out, n)
const int in[]; /* Input array */
int out[];      /* Output array */
int n;          /* number of elements to be expanded */
```

Description

This function performs A-law expansion (ITU rec. G.711) on the elements of the input vector `in` and outputs expanded data into the vector `out`. The function has been optimized and requires that both the input and output vectors are quad-word aligned.

Algorithm

$C(k)$ =a-law expansion of $A(k)$ for $k=0$ to $n-1$.

Domain

content of input array: 0 to 255

arg

get phase of a complex number

Synopsis

```
#include <complex.h>
float argf(a)
complex_float a; /* Complex input a */
```

Description

This function computes the phase of complex input *a* and returns the result.

Algorithm

$$c = \operatorname{atan}\left(\frac{\operatorname{Im}(a)}{\operatorname{Re}(a)}\right)$$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

asin

arc sine

Synopsis

```
#include <math.h>
float asinf (float x)
double asin (double x)
long double asind (long double x)
```

Description

This function computes the arc sine of number x . The output is in the range $[0 \text{ to } \pi]$ radians.

Algorithm

return = $\sin^{-1}(x)$

Domain

$x = [-1.0 \dots 1.0]$

atan

arc tangent

Synopsis

```
#include <math.h>
float atanf (float x)
double atan (double x)
long double atand (long double x)
```

Description

This function computes the arc tangent of number x . The output is in the range of $[-\pi/2$ to $+\pi/2]$ radians.

Algorithm

return = $\tan^{-1}(x)$

Domain

$x = [-3.4 \times 10^{38} \dots 3.4 \times 10^{38}]$ for `atanf()`
 $x = [-1.7 \times 10^{308} \dots 1.7 \times 10^{308}]$ for `atand()`

Run-time Library Reference

atan2

arc tangent of quotient

Synopsis

```
#include <math.h>
float atan2f (float y, float x)
double atan2 (double y, double x)
long double atan2d (long double y, long double x)
```

Description

This function computes the arc tangent of number y/x . The output is in the range of $[-\pi$ to $+\pi$] radians.

If $y = x = 0$, this function returns 0.

Algorithm

$$return = \tan^{-1}\left(\frac{y}{x}\right)$$

Domain

$x, y = [-3.4 \times 10^{38} \dots 3.4 \times 10^{38}]$ for `atan2f()`

$x, y = [-1.7 \times 10^{308} \dots 1.7 \times 10^{308}]$ for `atan2d()`

NOT $x = y = 0$

atof

string to double conversion

Synopsis

```
#include <stdlib.h>
double atof (const char *string)
```

Description

Converts a character string to a `double` value where the input string is a sequence of characters that can be interpreted as a numerical value of the specified type.

Algorithm

The string argument is as follows:

[whitespace][sign][digits][.digits][{e|E}[sign]digits]

where *whitespace* can consist of spaces and/or tab characters, *sign* is either plus (+) or minus (-), and *digits* are one or more decimal digits. If no digits appear before the decimal point then at least one must appear after the decimal point. The decimal digits may be followed by an exponent denoted by one of the following characters: *e*, or *E* followed by a decimal integer which may be signed.

The function stops reading the input string at the first character that it cannot recognize as part of the valid argument defined above.

Domain

The number should fit within the dynamic range of a double.

Run-time Library Reference

atoi

string to integer conversion

Synopsis

```
#include <stdlib.h>
int atoi (const char *string)
```

Description

Converts a character string to a integer fixed-point value where the input string is a sequence of characters that can be interpreted as a numerical value of the specified type.

Algorithm

The string argument is as follows:

[whitespace][sign][base]digits

where *whitespace* can consist of spaces and/or tab characters, *sign* is either plus (+) or minus (-), *base* is 0 for octal and 0x for hexadecimal, and *digits* are one or more decimal digits (or letters from a to f for hexadecimal numbers).

The function stops reading the input string at the first character that it cannot recognize as part of a valid argument defined above.

Domain

-2,147,483,648 to 2,147,483,647

atol

string to long conversion

Synopsis

```
#include <stdlib.h>
long atol (const char *string)
```

Description

Converts a character string to a long integer fixed-point value where the input string is a sequence of characters that can be interpreted as a numerical value of the specified type.

Algorithm

The string argument is as follows:

[whitespace][sign][base]digits

where *whitespace* can consist of spaces and/or tab characters, *sign* is either plus (+) or minus (-), *base* is 0 for octal and 0x for hexadecimal, and *digits* are one or more decimal digits (or letters from a to f for hexadecimal numbers).

The function stops reading the input string at the first character that it cannot recognize as part of a valid argument defined above.

Domain

-2,147,483,648 to 2,147,483,647

Run-time Library Reference

atold

string to long double conversion

Synopsis

```
#include <stdlib.h>
long double atold (const char *string)
```

Description

Converts a character string to a double-precision floating-point value where the input string is a sequence of characters that can be interpreted as a numerical value of the specified type.

Algorithm

The string argument is as follows:

[whitespace][sign][digits][.digits][{e|E}[sign]digits]

where *whitespace* can consist of spaces and/or tab characters, *sign* is either plus (+) or minus (-), and *digits* are one or more decimal digits. If no digits appear before the decimal point then at least one must appear after the decimal point. The decimal digits may be followed by an exponent denoted by one of the following characters: *e*, or *E* followed by a decimal integer which may be signed.

The function stops reading the input string at the first character that it cannot recognize as part of a valid argument defined above.

Domain

The number should fit within the dynamic range of a 64 bits double.

autocoh

autocoherence

Synopsis

```
#include <stats.h>
void autocohf(a,n,m,c)
const float a[]; /* Input vector a */
int n;           /* Input samples */
int m;           /* Lag count */
float c[];       /* Output vector c */
```

Description

This function computes the autocohereence of the input elements contained within input vector *a*, and stores the result to output vector *c*.

Algorithm

$$c_k = \frac{1}{n} * \left(\sum_{j=0}^{n-k-1} (a_j - \bar{a}) * (a_{j+k} - \bar{a}) \right)$$

where $k=\{0,1,...,m-1\}$ and $\bar{a}$ is the mean value of input vector *a*.

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

autocorr

autocorrelation

Synopsis

```
#include <stats.h>
void autocorrf(a,n,m,c)
const float a[];    /* Input vector a          */
int n;              /* Number of input samples */
int m;              /* Lag count                */
float c[];          /* Output vector c          */
```

Description

This function computes the autocorrelation of the input elements contained within input vector *a*, and stores the result to output vector *c*.

Algorithm

$$c_k = \frac{1}{n} * \left(\sum_{j=0}^{n-k-1} a_j * a_{j+k} \right)$$

where $k=\{0,1,...,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

avg

returns the mean of two values

Synopsis

Integer forms:

```
#include <stdlib.h>
int avg(int a, int b)
long avgl(long a, long b)
```

Floating point forms:

```
#include <math.h>
float favgf(float a, float b)
double favg(double a, double b)
long double favgd(long double a, long double b)
```

Description

This function adds two arguments and divides the result by 2.

Algorithm
$$(a + b) / 2$$
Domain

Run-time Library Reference

bsearch

binary search

Synopsis

```
#include <stdlib.h>
void *bsearch (const void *key, const void *array,
               size_t nb_elem, size_t width, int (*compare)())
```

Description

Performs a binary search on a sorted array.

Algorithm

Call the compare function with the key and the current node (i.e. initially the one in the middle of the array). If the compare returns 0, then return the current node; if the compare returns -1, apply the search recursively on the portion of the array to the left of the current position; if the compare returns 1, apply the search recursively on the portion of the array to the right of the current location.

Domain

N/A.

cabs

complex absolute value

Synopsis

```
#include <complex.h>
float cabsf(a)
complex_float a; /* Complex input */
```

Description

This function computes the complex absolute value of a complex input and returns the result.

Algorithm

$$c = \sqrt{\text{Re}^2(a) + \text{Im}^2(a)}$$

Domain

$$a^2 + b^2 < 3.4 \times 10^{38}$$

Run-time Library Reference

cadd

complex addition

Synopsis

```
#include <complex.h>
complex_float caddf(a,b)
complex_float a; /* Complex input a */
complex_float b; /* Complex input b */
```

Description

This function adds two complex values a and b , placing the result in the complex value c .

Algorithm

$$\begin{aligned}\operatorname{Re}(c) &= \operatorname{Re}(a) + \operatorname{Re}(b) \\ \operatorname{Im}(c) &= \operatorname{Im}(a) + \operatorname{Im}(b)\end{aligned}$$

Domain

$$-3.4 \times 10^{38} \text{ to } +3.4 \times 10^{38}$$

cdiv

complex division

Synopsis

```
#include <complex.h>
complex_float cdivf(a,b)
complex_float a; /* Complex input a */
complex_float b; /* Complex input b */
```

Description

This function computes the quotient of the division of two complex values, placing the result in the complex value c.

Algorithm

$$\text{Re}(c) = \frac{\text{Re}(a) * \text{Re}(b) + \text{Im}(a) * \text{Im}(b)}{\text{Re}^2(b) + \text{Im}^2(b)}$$

$$\text{Im}(c) = \frac{\text{Re}(a) * \text{Im}(b) - \text{Im}(a) * \text{Re}(b)}{\text{Re}^2(b) + \text{Im}^2(b)}$$

Domain

-3.4 x 10³⁸ to +3.4 x 10³⁸

Run-time Library Reference

ceil

ceiling

Synopsis

```
#include <math.h>
float ceilf (float x)
double ceil (double x)
long double ceild (long double x)
```

Description

This function calculates the next highest whole number that is greater than or equal to the floating-point number x .

Algorithm

return = smallest int $\geq x$

Domain

$x = [-3.4 \times 10^{38} \dots 3.4 \times 10^{38}]$ for `ceilf()`
 $x = [-1.7 \times 10^{308} \dots 1.7 \times 10^{308}]$ for `ceild()`

cexp

complex exponential

Synopsis

```
#include <complex.h>
complex_float cexpf(a)
float a; /* Value of input */
```

Description

This function computes the complex exponential of real input *a* and stores the result in complex output *c*.

Algorithm

$$\text{Re}(c) = \cos(a)$$

$$\text{Im}(c) = \sin(a)$$

Domain

a = [-1,647,095 ... 1,647,095] for cexpf()

a = [-843,314,850 ... 843,314,850]for cexpd()

Run-time Library Reference

cfft

N point complex input FFT

Synopsis

```
#include <filter.h>

void cfft(in[], t[], out[], w[], wst, n)
const complex_float in[]; /* input sequence */
complex_float t[];        /* temporary working buffer */
complex_float out[];      /* output sequence */
const complex_float w[];  /* twiddle sequence */
int wst;                  /* twiddle factor stride */
int n;                    /* number of FFT points */
```

Description

This function transforms the time domain complex input signal sequence to the frequency domain by using the accelerated version of the ‘Discrete Fourier Transformation’ known as a ‘Fast Fourier Transform’ or FFT. The CFFT function will ‘decimate in frequency’ by the best choice FFT algorithm, radix-4 or mixed-radix, depending on the input sequence length. If input data can be overwritten, the optimum memory usage can be achieved by setting the output pointer to the input array.

For efficiency, the "twiddle table" is calculated once, during initialization, and then provided to the FFT routine as a separate parameter. You must declare the variable and initialize it prior to calling an FFT function. An initialization function, `twidfft`, is provided.

If the twiddle table has been allocated at a larger size than needed for a particular call of `cfft`, then the stride parameter will need to be set appropriately; otherwise, it should be one. [For more information, see “twidfft” on page 3-152.](#)

Algorithm

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}$$

When the sequence length, n , equals power of four, the radix4 method is used. When the sequence length is a power of two (see domain), the mixed radix method is used. At the first stage of the decimation in frequency, CFFT uses the radix2 method to generate 2 sequences with $n/2$ points each. $n/2$ is now a power of four which creates the condition to employ the faster, radix4 method over these two sequences.

Domain

Input sequence length n must be equal to either a power of two, or a power of four, and at least 16.

cfft2d

NxN point 2-d complex input FFT

Synopsis

```
#include <filter.h>

void cfft2d(*in, *t, *out, w[], wst, n)
const complex_float *in; /* pointer to input matrix a[n][n] */
complex_float *t;        /* pointer to working buffer t[n][n] */
complex_float *out;       /* pointer to matrix c[n][n] */
const complex_float w[]; /* twiddle sequence */
int wst;                  /* twiddle factor stride */
int n;                    /* number of FFT points */
```

Description

This function computes the two dimensional Fast Fourier Transform of the complex input matrix `a[n][n]`, and stores the result to the complex matrix `c[n][n]`.

If the input data can be overwritten, optimum memory usage is achieved by setting the output pointer to the input array.

For efficiency, the "twiddle table" is calculated once, during initialization, and then provided to the FFT routine as a separate parameter. You must declare the variable and initialize it prior to calling an FFT function. An initialization function, `twidfft`, is provided.

If the twiddle table has been allocated at a larger size than needed for a particular call of `cfft2d`, then the stride parameter will need to be set appropriately; otherwise, it should be one. [For more information, see “twidfft” on page 3-152.](#)

Algorithm

$$c(i, j) = \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} a(k, l) * e^{-2\pi j(i*k + j*l)/n}$$

where $i=\{0,1,\dots,n-1\}$, $j=\{0,1,2,\dots,n-1\}$

Domain

Input sequence length n must be equal to either a power of two, or a power of four, and at least 16.

Run-time Library Reference

clip

clip

Synopsis

```
#include <stdlib.h>
int clip (int parm1, int parm2)
float fclipf (float parm1, float parm2)
double fclip (double parm1, double parm2)
long double fclipd (long double parm1, long double parm2)
```

Description

Clip a value if it is too large.

Algorithm

```
if ( |parm1| < |parm2| )
    return( parm1 )
else
    return( |parm2| * signof(parm1))
```

Domain

Full range.

cmatmadd

complex matrix + matrix addition

Synopsis

```

#include <matrix.h>
void cmatmaddf(a,b,n,m,c)
const complex_float *a; /* Pointer to input matrix a[][] */
const complex_float *b; /* Pointer to input matrix b[][] */
int n;                  /* Number of rows in matrix a[][] */
int m;                  /* Number of columns in matrix a[][] */
complex_float *c;       /* Pointer to matrix c[][] */

```

Description

This function computes the addition of input complex matrix $a[][]$ with input complex matrix $b[][]$, and stores the result to output complex matrix $c[][]$. The dimensions of complex matrix $a[][]$ are n and m and the dimensions of complex matrix b are n and m . The resulting output complex matrix $c[][]$ is of dimensions n and m .

Algorithm

$$\text{Re}(c_{i,j}) = \text{Re}(a_{i,j}) + \text{Re}(b_{i,j})$$

$$\text{Im}(c_{i,j}) = \text{Im}(a_{i,j}) + \text{Im}(b_{i,j})$$

where $i=\{0,1,2,\dots,n-1\}$, $j=\{0,1,2,\dots,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

cmatmmlt

complex matrix * matrix multiplication

Synopsis

```
#include <matrix.h>
void cmatmmltf(a,n,k,b,m,c)
const complex_float *a; /* Pointer to input matrix a[][] */
int n; /* Number of rows in matrix a[][] */
int k; /* Number of columns in matrix a[][] */
const complex_float *b; /* Pointer to input matrix b[][] */
int m; /* Number of columns in matrix b[][] */
complex_float *c; /* Pointer to matrix c[][] */
```

Description

This function computes the multiplication of input complex matrix $a[][]$ with input complex matrix $b[][]$, and stores the result to output complex matrix $c[][]$. The dimensions of complex matrix $a[][]$ are n and k and the dimensions of complex matrix b are k and m . The resulting output complex matrix $c[][]$ is of dimensions n and m .

Algorithm

$$\text{Re}(c_{i,j}) = \sum_{l=0}^{k-1} (\text{Re}(a_{i,l}) * \text{Re}(b_{l,j}) - \text{Im}(a_{i,l}) * \text{Im}(b_{l,j}))$$
$$\text{Im}(c_{i,j}) = \sum_{l=0}^{k-1} (\text{Re}(a_{i,l}) * \text{Im}(b_{l,j}) + \text{Im}(a_{i,l}) * \text{Re}(b_{l,j}))$$

where $i=\{0,1,2,\dots,n-1\}$, $j=\{0,1,2,\dots,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

cmatmsub

complex matrix - matrix subtraction

Synopsis

```

#include <matrix.h>
void cmatmsubf(a,b,n,m,c)
const complex_float *a; /* Pointer to input matrix a[][] */
const complex_float *b; /* Pointer to input matrix b[][] */
int n;                  /* Number of rows in matrix a[][] */
int m;                  /* Number of columns in matrix a[][] */
complex_float *c;       /* Pointer to matrix c[][] */

```

Description

This function computes the subtraction of input complex matrix $a[][]$ with input complex matrix $b[][]$, and stores the result to output complex matrix $c[][]$. The dimensions of complex matrix $a[][]$ are n and m and the dimensions of complex matrix b are n and m . The resulting output complex matrix $c[][]$ is of dimensions n and m .

Algorithm

$$\text{Re}(c_{i,j}) = \text{Re}(a_{i,j}) - \text{Re}(b_{i,j})$$

$$\text{Im}(c_{i,j}) = \text{Im}(a_{i,j}) - \text{Im}(b_{i,j})$$

where $i=\{0,1,2,\dots,n-1\}$, $j=\{0,1,2,\dots,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

cmatsadd

complex matrix + scalar addition

Synopsis

```
#include <matrix.h>
void cmatsaddf(a,b,n,m,c)
const complex_float *a; /* Pointer to input matrix a[][] */
const complex_float *b; /* Pointer to input scalar b */
int n; /* Number of rows in matrix a[][] */
int m; /* Number of columns in matrix a[][] */
complex_float *c; /* Pointer to matrix c[][] */
```

Description

This function adds the complex scalar value b to each element of complex matrix $a[][]$, placing the result in complex matrix $c[][]$. The dimensions of complex matrix $a[][]$ are n and m . The resulting output complex matrix $c[][]$ is of dimensions n and m .

Algorithm

$$\text{Re}(c_{i,j}) = \text{Re}(a_{i,j}) + \text{Re}(b)$$

$$\text{Im}(c_{i,j}) = \text{Im}(a_{i,j}) + \text{Im}(b)$$

where $i=\{0,1,2,\dots,n-1\}$, $j=\{0,1,2,\dots,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

cmatsmult

complex matrix * scalar multiplication

Synopsis

```
#include <matrix.h>
void cmatsmultf(a,b,n,m,c)
const complex_float *a; /* Pointer to input matrix a[][] */
const complex_float *b; /* Pointer to input scalar b */
int n; /* Number of rows in matrix a[][] */
int m; /* Number of columns in matrix a[][] */
complex_float *c; /* Pointer to matrix c[][] */
```

Description

This function computes the multiplication of input complex matrix $a[][]$ with input complex scalar b , and stores the result to output complex matrix $c[][]$. The dimensions of complex matrix $a[][]$ are n and m . The resulting output complex matrix $c[][]$ is of dimensions n and m .

Algorithm

$$\text{Re}(c_{i,j}) = \text{Re}(a_{i,j}) * \text{Re}(b) - \text{Im}(a_{i,j}) * \text{Im}(b)$$

$$\text{Im}(c_{i,j}) = \text{Re}(a_{i,j}) * \text{Im}(b) + \text{Im}(a_{i,j}) * \text{Re}(b)$$

where $i=\{0,1,2,\dots,n-1\}$, $j=\{0,1,2,\dots,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

cmatssub

complex matrix - scalar subtraction

Synopsis

```
#include <matrix.h>
void cmatssubf(a,b,n,m,c)
const complex_float *a; /* Pointer to input matrix a[][] */
const complex_float *b; /* Pointer to input scalar b */
int n; /* Number of rows in matrix a[][] */
int m; /* Number of columns in matrix a[][] */
complex_float *c; /* Pointer to matrix c[][] */
```

Description

This function computes the subtraction of input complex matrix $a[][]$ with input complex scalar b , and stores the result to output complex matrix $c[][]$. The dimensions of complex matrix $a[][]$ are n and m . The resulting output complex matrix $c[][]$ is of dimensions n and m .

Algorithm

$$\text{Re}(c_{i,j}) = \text{Re}(a_{i,j}) - \text{Re}(b)$$

$$\text{Im}(c_{i,j}) = \text{Im}(a_{i,j}) - \text{Im}(b)$$

where $i=\{0,1,2,\dots,n-1\}$, $j=\{0,1,2,\dots,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

cmlt

complex multiply

Synopsis

```
#include <complex.h>
complex_float cmltf(a,b)
complex_float a; /* Complex input a */
complex_float b; /* Complex input b */
```

Description

This function multiplies two complex values *a* and *b*, returning the complex result value *c*.

Algorithm

$$\begin{aligned}\text{Re}(c) &= \text{Re}(a) * \text{Re}(b) - \text{Im}(a) * \text{Im}(b) \\ \text{Im}(c) &= \text{Re}(a) * \text{Im}(b) + \text{Im}(a) * \text{Re}(b)\end{aligned}$$

Domain

$$-3.4 \times 10^{38} \text{ to } +3.4 \times 10^{38}$$

Run-time Library Reference

conj

complex conjugate

Synopsis

```
#include <complex.h>
complex_float conjf(a)
complex_float a; /* Complex input a */
```

Description

This function conjugates the complex input *a* and stores the result in complex output *c*.

Algorithm

$$\begin{aligned}\text{Re}(c) &= \text{Re}(a) \\ \text{Im}(c) &= -\text{Im}(a)\end{aligned}$$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

convolve

convolution

Synopsis

```

#include <filter.h>
void convolve(vin1, vlen1, vin2, vlen2, vout)
const float vin1[]; /* pointer to input sequence 1 */
int vlen1;          /* length of the input sequence 1 */
const float vin2[]; /* pointer to input sequence 2 */
int vlen2;          /* length of the input sequence 2 */
float vout[];       /* pointer to output sequence */

```

Description

This function convolves two sequences pointed to by `vin1` and `vin2`. If `vin1` points to the sequence whose length is `vlen1` and `vin2` points to the sequence whose length is `vlen2`, then resulting sequence pointed to by `vout` has length: `vlen1 + vlen2 - 1`

Algorithm

Convolution between two sequences `cin1` and `cin2` is described as:

$$cout(n) = \sum_{k=0}^{k=vlen2-1} cin1(n+k) \bullet cin2(vlen2-k-1)$$

for `n=0` to `(vlen-1)`

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

conv2d

2-d convolution

Synopsis

```
#include <filter.h>
void conv2d(min1, mrow1, mcol1, min2, mrow2, mcol2, mout )
const float *min; /* pointer to input matrix 1 */
int mrow1; /* number of rows in matrix 1 */
int mcol1; /* number of columns in matrix 1 */
const float *min2; /* pointer to input matrix 2 */
float *mrow2; /* number of rows in matrix 1 */
int mcol2; /* number of columns in matrix 2 */
int *mout; /* pointer to matrix */
```

Description

This function computes 2-dimensional convolution of the NxN matrix.

Algorithm

Two dimensional input matrix `min1` is convolved with input matrix `min2`, placing the result in a matrix pointed to by `mout`.

$$mout(r,c) = \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} min2[k-1-i][k-1-j] \bullet min1[r+i][c+j]$$

for $r=0$ to $nr-k+1$ and for $c=0$ to $nc-k+1$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

copysign

copysign

Synopsis

```
#include <math.h>
float copysignf (float parm1, float parm2)
double copysign (double parm1, double parm2)
long double copysignl (long double parm1, long double parm2)
```

Description

Copies the sign of the second argument to the first argument.

Algorithm

```
return( |parm1| * copysignof(parm2))
```

Domain

Full range.

Run-time Library Reference

cos

cosine

Synopsis

```
#include <math.h>
float cosf (float x)
double cos (double x)
long double cosd (long double x)
```

Description

This function calculates the cosine of number x where x is measured in radians. The output is in the range $[-1$ to $1]$. If x is outside of the domain, this function returns 0.0.

Algorithm

return = $\cos(x)$

Domain

$x = [-1,647,095 \dots 1,647,095]$ for `cosf()`
 $x = [-843,314,850 \dots 843,314,850]$ for `cosd()`

cosh

hyperbolic cosine

Synopsis

```
#include <math.h>
float coshf (float x)
double cosh (double x)
long double coshd (long double x)
```

Description

This function calculates the hyperbolic cosine of a number x where x is measured in radians. If x is outside the domain, this function returns 3.4×10^{38} for a float-type return value and 1.7×10^{308} for a double-type return value.

Algorithm

return = cosh(x)

Domain

$x = [-(\ln(3.4 \times 10^{38}) - \ln(2)) \dots (\ln(3.4 \times 10^{38}) - \ln(2))]$ for `atanf()`
 $x = [-(\ln(1.7 \times 10^{308}) - \ln(2)) \dots (\ln(1.7 \times 10^{308}) - \ln(2))]$ for `atand()`

Run-time Library Reference

cot

cotangent

Synopsis

```
#include <math.h>
float cotf (float x)
double cot (double x)
long double cotd (long double x)
```

Description

This function calculates the cotangent of number x where x is measured in radians. If x is outside of the domain, this function returns 0.0.

Algorithm

return = $\cot(x)$

Domain

$x = [-6,588,397 \dots 6,588,397]$ for $\cotf()$
 $x = [-421,657,424 \dots 421,657,424]$ for $\cotd()$

count_ones

count one bits in word

Synopsis

```
#include <stdlib.h>
int count_ones(int word)
int lcount_ones(long word)
int llcount_ones(long long word)
```

Description

This function counts the number of one bits in `word`.

Algorithm

return = sum(j=0 to 31) bit[j] of `word`

Run-time Library Reference

crosscoh

cross-coherence

Synopsis

```
#include <stats.h>
void crosscohf(a,b,n,m,c)
const float a[]; /* Input vector a          */
const float b[]; /* Input vector b          */
int n;           /* Number of input samples */
int m;           /* Lag count                */
float c[];       /* Output vector c          */
```

Description

This function computes the cross-coherence of the input elements contained within input vector *a* and input vector *b*, and stores the result to output vector *c*.

Algorithm

$$c_k = \frac{1}{n} * \left(\sum_{j=0}^{n-k-1} (a_j - \bar{a}) * (b_{j+k} - \bar{b}) \right)$$

where $k=\{0,1,...,m-1\}$, $\bar{a}$ is the mean value of input vector *a* and $\bar{b}$ is the mean value of input vector *b*.

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

crosscorr

cross-correlation

Synopsis

```

#include <stats.h>
void crosscorr(float a[], float b[], int n, int m, float c[])
/* Input vector a */
/* Input vector b */
/* Number of input samples */
/* Lag count */
/* Pointer to output vector c */

```

Description

This function computes the cross-correlation of the input elements contained within input vector *a* and input vector *b*, and stores the result to output vector *c*.

Algorithm

$$c_k = \frac{1}{n} * \left(\sum_{j=0}^{n-k-1} a_j * b_{j+k} \right)$$

where $k=\{0,1,...,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

csub

complex subtraction

Synopsis

```
#include <complex.h>
complex_float csubf(a,b)
complex_float a; /* Complex input a */
complex_float b; /* Complex input b */
```

Description

This function subtracts the value *b* from the value *a* (both values are complex), placing the result in the complex value *c*.

Algorithm

$$\begin{aligned}\text{Re}(c) &= \text{Re}(a) - \text{Re}(b) \\ \text{Im}(c) &= \text{Im}(a) - \text{Im}(b)\end{aligned}$$

Domain

$$-3.4 \times 10^{38} \text{ to } +3.4 \times 10^{38}$$

cvecdot

complex vector dot product

Synopsis

```
#include <vector.h>
complex_float cvecdotf(a,b,n)
const complex_float a[]; /* Input vector a */
const complex_float b[]; /* Input vector b */
int n;                    /* Element count */
```

Description

This function computes the complex dot product of two complex input vectors *a* and *b*, returning the complex result value.

Algorithm

return =

$$\left\{ \begin{array}{l} \text{Real} = \sum_{i=0}^{n-1} \text{Re}(a_i) * \text{Re}(b_i) - \text{Im}(a_i) * \text{Im}(b_i) \\ \text{Imaginary} = \sum_{i=0}^{n-1} \text{Re}(a_i) * \text{Im}(b_i) + \text{Im}(a_i) * \text{Re}(b_i) \end{array} \right.$$

Domain

-3.4 x 10³⁸ to +3.4 x 10³⁸

Run-time Library Reference

cvecsadd

complex vector + scalar addition

Synopsis

```
#include <vector.h>
void cvecsaddf(a,b,c,n)
const complex_float a[]; /* Input vector */
complex_float b;          /* Input scalar */
complex_float c[];        /* Output vector */
int n;                   /* Element count */
```

Description

This function adds input complex scalar *b* to each element of input complex vector, and the results are stored in output complex vector.

Algorithm

$$\text{Re}(c_i) = \text{Re}(a_i) + \text{Re}(b)$$

$$\text{Im}(c_i) = \text{Im}(a_i) + \text{Im}(b)$$

where $i=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

cvecssmlt

complex vector * scalar multiplication

Synopsis

```
#include <vector.h>
void cvecssmltf(a,b,c,n)
const complex_float a[]; /* Input vector */
complex_float b;          /* Input scalar */
complex_float c[];        /* Output vector*/
int n;                    /* Element count*/
```

Description

This function multiplies each element of input complex vector by input complex scalar *b*, the results are stored in output complex vector.

Algorithm

$$\text{Re}(c_i) = \text{Re}(a_i) * \text{Re}(b) - \text{Im}(a_i) * \text{Im}(b)$$

$$\text{Im}(c_i) = \text{Re}(a_i) * \text{Im}(b) + \text{Im}(a_i) * \text{Re}(b)$$

where $i=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

cvecssub

complex vector - scalar subtraction

Synopsis

```
#include <vector.h>
void cvecssubf(a,b,c,n)
const complex_float a[]; /* Input vector */
complex_float b;          /* Input scalar */
complex_float c[];        /* Output vector */
int n;                   /* Element count */
```

Description

This function subtracts input complex scalar b from each element of input complex vector, the results are stored in output complex vector.

Algorithm

$$\begin{aligned}\text{Re}(c_i) &= \text{Re}(a_i) - \text{Re}(b) \\ \text{Im}(c_i) &= \text{Im}(a_i) - \text{Im}(b)\end{aligned}$$

where $i=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

cvecvadd

complex vector + vector addition

Synopsis

```
#include <vector.h>
void cvecvaddf(a,b,c,n)
const complex_float a[]; /* Input vector a */
const complex_float b[]; /* Input vector b */
complex_float c[];        /* Output vector */
int n;                    /* Element count */
```

Description

This function adds two input vectors, the results are stored in output vector.

Algorithm

$$\text{Re}(c_i) = \text{Re}(a_i) + \text{Re}(b_i)$$

$$\text{Im}(c_i) = \text{Im}(a_i) + \text{Im}(b_i)$$

where $i=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

cvecvsub

complex vector - vector subtraction

Synopsis

```
#include <vector.h>
void cvecvsubf(a,b,c,n)
const complex_float a[]; /* Input vector a */
const complex_float b[]; /* Input vector b */
complex_float c[];        /* Output vector */
int n;                    /* Element count */
```

Description

This function subtracts input vector *b* from input vector *a*, and the results are stored in output vector.

Algorithm

$$\text{Re}(c_i) = \text{Re}(a_i) - \text{Re}(b_i)$$

$$\text{Im}(c_i) = \text{Im}(a_i) - \text{Im}(b_i)$$

where $i=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

cvecvmlt

complex vector * vector multiplication

Synopsis

```
#include <vector.h>
void cvecvmlt(f(a,b,c,n)
const complex_float a[]; /* Input vector a */
const complex_float b[]; /* Input vector b */
complex_float c[];        /* Output vector */
int n;                    /* Element count */
```

Description

This function multiplies two input vectors: a and b, and the results are stored in output vector.

Algorithm

$$\text{Re}(c_i) = \text{Re}(a_i) * \text{Re}(b_i) - \text{Im}(a_i) * \text{Im}(b_i)$$

$$\text{Im}(c_i) = \text{Re}(a_i) * \text{Im}(b_i) + \text{Im}(a_i) * \text{Re}(b_i)$$

where $i=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

div

integer division

Synopsis

```
#include <stdlib.h>
div_t div (int numerator, int denominator)
```

Description

Performs an integer division and returns the quotient and remainder.

Algorithm

$$\text{Quotient} = \left\lfloor \frac{|\text{num}|}{|\text{denom}|} \right\rfloor \times \text{signof}\left(\frac{\text{num}}{\text{denom}}\right)$$

$$\text{Remainder} = \text{num} - (\text{Quotient} \times \text{denom})$$

Domain

numerator: -2,147,483,648 to 2,147,483,647

denominator: -2,147,483,648 to -1 and 1 to 2,147,483,647

exp

exponential

Synopsis

```
#include <math.h>
float expf (float x)
double exp (double x)
long double expd (long double x)
```

Description

This function calculates the exponent of number x . If x is outside of the domain, this function returns 0 for large negative x . For large positive x , this function returns 3.4×10^{38} for a float-type return value and 1.7×10^{308} for a double-type return value.

Algorithm

return = $e^{(x)}$

Domain

$x = [-\ln(3.4 \times 10^{38}) \dots \ln(3.4 \times 10^{38})]$ for `expf()`
 $x = [-\ln(1.7 \times 10^{308}) \dots \ln(1.7 \times 10^{308})]$ for `expd()`

Run-time Library Reference

__emuclk

Get simulator cycle count

Synopsis

```
#include <libsim.h>
int __emuclk(void)
```

Description

This function returns the current value of the simulator cycle count. Note that the function name has two leading underscores.

Algorithm

See description.

fir

finite impulse response filter

Synopsis

```
#include <filter.h>

void fir(x,y,n,s)
const float x[]; /* Input sample vector x          */
float y[];       /* Output sample vector y         */
int n;          /* Number of input samples                        */
fir_state *s     /* Pointer to filter state structure              */
```

The FIR filter function uses the following structure to maintain the state of the filter:

```
typedef struct
{
    float *h; /* filter coefficients          */
    float *d; /* start of delay line              */
    float *p; /* read/write pointer               */
    int k;    /* no. of coefficients              */
    int l;    /* interpolation/decimation index   */
} fir_state;

void fir_init(fir_state *s, float h[], float d[], int k, int l);
```

Description

This function computes a FIR (Finite Impulse Response Filter) using the coefficients stored in vector `s.h` applied to the elements of vector `x`. The results are stored in vector `y`.

For efficiency, this filter uses a separate variable of type `fir_state` to maintain the filter state. You must declare space for this variable, and initialize it prior to using the filter. An initialization function, `fir_init` is provided.

Run-time Library Reference

Algorithm

$$y(k) = \sum_{i=0}^{p-1} h(i) * x(k-i) \text{ for } k = 0, 1, \dots, n$$

Domain

$$-3.4 \times 10^{38} \text{ to } +3.4 \times 10^{38}$$

fir_decima

FIR decimation filter

Synopsis

```
#include <filter.h>
void fir_decima(x,y,ng,s)
const float x[]; /* Input sample vector x          */
float y[];        /* Output sample vector y          */
int ng;           /* Number of samples to generate  */
fir_state *s      /* Pointer to filter state structure */
```

Description

This function computes a FIR (Finite Impulse Response Filter) using the coefficients stored in vector `s.h` applied to the elements of vector `x`. The results are stored in vector `y` and computed according to a decimation index `s.l`. The number of desired output filter samples to generate is specified by the parameter `ng`, while the size of vector `s.h` is `s.k`, the size of input vector `x` is `ng`.

Algorithm

$$y(k) = \sum_{i=0}^{p-1} x(k * l - i) * h(i)$$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

fir_interp

FIR interpolation filter

Synopsis

```
#include <filter.h>
void fir_interp(x,y,n,s)
const float x[]; /* Input sample vector x */
float y[]; /* Output sample vector y */
int n; /* Number of input samples */
fir_state *s /* Pointer to filter state structure */
```

Description

This function computes a FIR (Finite Impulse Response Filter) using the coefficients stored in vector `s.h` applied to the elements of vector `x`. The results are stored in vector `y` and computed according to a interpolation index, `s.l`. The number of input samples `x` is specified by the parameter `n`, the size of vector `s.h` is specified by `s.k`, and `s.p` and `s.l` must be integers, and the size of output vector `y` is `n`.

Algorithm

$$y_m(k) = \sum_{i=0}^{p/l-1} x(k-i) * h(i * l + m)$$

where `m={0,1,2,...,l}`

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

floor

floor

Synopsis

```
#include <math.h>
float floorf (float x)
double floor (double x)
long double floord (long double x)
```

Description

This function calculates and rounds down to the next lowest whole number that is less than or equal to number x .

Algorithm

return = largest int $\leq x$

Domain

$x = [-3.4 \times 10^{38} \dots 3.4 \times 10^{38}]$ for floorf()
 $x = [-1.7 \times 10^{308} \dots 1.7 \times 10^{308}]$ for floord()

Run-time Library Reference

fmod

floating point remainder

Synopsis

```
#include <math.h>
float fmodf (float x, float y)
double fmod (double x, double y)
long double fmodd (long double x, long double y)
```

Description

This function calculates the floating-point remainder of x/y . If $x = y = 0$, this function returns 0.

Algorithm

$$return = \left(\left\lfloor \frac{x}{y} \right\rfloor - floor \left\lfloor \frac{x}{y} \right\rfloor \right) \cdot sign(x)$$

Domain

$x, y = [-3.4 \times 10^{38} \dots 3.4 \times 10^{38}]$ for `fmodf()`

$x, y = [-1.7 \times 10^{308} \dots 1.7 \times 10^{308}]$ for `fmodd()`

NOT $x = y = 0$

frexp

get mantissa and exponent

Synopsis

```
#include <math.h>
float frexpf (float x, int *n)
double frexp (double x, int *n)
long double frexpd (long double x, int *n)
```

Description

This function separates out the mantissa (fractional component) and exponent of a floating-point number x . The normalized mantissa is returned as the value of the function; the exponent is stored through the pointer argument n .

Algorithm

return = f ; n passed through 2nd parameter pointer. $x = f * 2^n$

Domain

$x = [-3.4 \times 10^{38} \dots 3.4 \times 10^{38}]$ for `frexpf()`
 $x = [-1.7 \times 10^{308} \dots 1.7 \times 10^{308}]$ for `frexpd()`

gen_bartlett

generate bartlett window

Synopsis

```
#include <window.h>
void gen_bartlett(w,a,N)
float w[]; /* Window vector */
int a; /* Address stride in samples for window vector */
int N; /* Length of window vector */
```

Description

This function generates a vector containing the Bartlett window. The length is specified by parameter *N*. Note that this window is similar to the Triangle window but has the following different properties [2]:

- The Bartlett window always returns a window with two zeros on either end of the sequence, so that for odd *n*, the center section of a *N*+2 Bartlett window equals a *N* Triangle window.
- For even *n*, the Bartlett window is still the convolution of two rectangular sequences. There is no standard definition for the Triangle window for even *n*; the slopes of the Triangle window are slightly steeper than those of the Bartlett window.

Algorithm

$$w[n] = 1 - \left| \frac{n - \frac{N-1}{2}}{\frac{N-1}{2}} \right|$$

where $n = \{0, 1, 2, \dots, N-1\}$

Domain

$a > 0; N > 0$

Run-time Library Reference

gen_blackman

generate blackman window

Synopsis

```
#include <window.h>
void gen_blackman(w,a,N)
float w[]; /* Window vector */
int a;     /* Address stride in samples for window vector */
int N;     /* Length of window vector */
```

Description

This function generates a vector containing the Blackman window. The length is specified by parameter N.

Algorithm

$$w[n] = 0.42 - 0.5 \cos\left(\frac{2\pi n}{N-1}\right) + 0.08 \cos\left(\frac{4\pi n}{N-1}\right)$$

where $n = \{0, 1, 2, \dots, N-1\}$

Domain

$a > 0$; $N > 0$

gen_gaussian

generate gaussian window

Synopsis

```

#include <window.h>
void gen_gaussian(w,alpha,a,N)
float w[];    /* Window vector */
float alpha; /* Gaussian alpha parameter */
int a;        /* Address stride in samples for window vector */
int N;        /* Length of window vector */

```

Description

This function generates a vector containing the Gaussian window. The length is specified by parameter N.

Algorithm

$$w(n) = \exp \left[-\frac{1}{2} \left(\alpha \frac{n - N/2 - 1/2}{N/2} \right)^2 \right]$$

where $n = \{0, 1, 2, \dots, N-1\}$ and a is an input parameter

Domain

$a > 0$; $N > 0$; $\alpha > 0.0$

Run-time Library Reference

gen_hamming

generate hamming window

Synopsis

```
#include <window.h>
void gen_hamming(w,a,N)
float w[]; /* Window vector */
int a; /* Address stride in samples for window vector */
int N; /* Length of window vector */
```

Description

This function generates a vector containing the Hamming window. The length is specified by parameter N.

Algorithm

$$w[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N-1}\right)$$

where $n = \{0, 1, 2, \dots, N-1\}$

Domain

$a > 0$; $N > 0$

gen_hanning

generate hanning window

Synopsis

```
#include <window.h>
void gen_hanning(w,a,N)
float w[]; /* Window vector */
int a; /* Address stride in samples for window vector */
int N; /* Length of window vector */
```

Description

This function generates a vector containing the Hanning window. The length is specified by parameter N. This window is also known as the Cosine window.

Algorithm

$$w[n] = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N+1}\right)$$

where $n = \{1, 1, 2, \dots, N+1\}$

Domain

$a > 0$; $N > 0$

Run-time Library Reference

gen_harris

generate harris window

Synopsis

```
#include <window.h>
void gen_harris(w,a,N)
float w[]; /* Window vector */
int a;     /* Address stride in samples for window vector */
int N;     /* Length of window vector */
```

Description

This function generates a vector containing the Harris window. The length is specified by parameter N. This window is also known as the Blackman-Harris window.

Algorithm

$$w[n] = 0.35875 - 0.48829 * \cos\left(\frac{2\pi n}{N-1}\right) + 0.14128 * \cos\left(\frac{4\pi n}{N-1}\right) + 0.01168 * \cos\left(\frac{6\pi n}{N-1}\right)$$

where $n = \{0, 1, 2, \dots, N-1\}$

Domain

$a > 0$; $N > 0$

gen_kaiser

generate kaiser window

Synopsis

```

#include <window.h>
void gen_kaiser(w,beta,a,N)
float w[]; /* Window vector */
float beta; /* Kaiser beta parameter */
int a; /* Address stride in samples for window vector */
int N; /* Length of window vector */

```

Description

This function generates a vector containing the Kaiser window. The length is specified by parameter N. The β value is specified by parameter b.

Algorithm

$$w[n] = \frac{I_0 \left[\beta \left(1 - \left[\frac{n-\alpha}{\alpha} \right]^2 \right)^{1/2} \right]}{I_0(\beta)}$$

where $n = \{0, 1, 2, \dots, N-1\}$, $a = (N - 1) / 2$, and $I_0(\beta)$ represents the zeroth-order modified Bessel function of the first kind.

Domain

$a > 0$; $N > 0$;

Run-time Library Reference

gen_rectangular

generate rectangular window

Synopsis

```
#include <window.h>
void gen_rectangular(w,a,N)
float w[]; /* Window vector */
int a;    /* Address stride in samples for window vector */
int N;    /* Length of window vector */
```

Description

This function generates a vector containing the Rectangular window. The length is specified by parameter N.

Algorithm

$w[n] = 1$ where $n = \{0, 1, 2, \dots, N-1\}$

Domain

$a > 0$; $N > 0$

gen_triangle

generate triangle window

Synopsis

```

#include <window.h>
void gen_triangle(w,a,N)
float w[]; /* Window vector */
int a; /* Address stride in samples for window vector */
int N; /* Length of window vector */

```

Description

This function generates a vector containing the Triangle window. The length is specified by parameter `N`. Refer to the Bartlett window regarding the relationship between it and the Triangle window.

Algorithm

For even n the following equation applies:

$$w[n] = \begin{cases} \frac{2n+1}{N} & n < N/2 \\ \frac{2N-2n-1}{N} & n > N/2 \end{cases}$$

where $n = \{0, 1, 2, \dots, N-1\}$

For odd n the following equation applies:

$$w[n] = \begin{cases} \frac{2n+2}{N+1} & n < N/2 \\ \frac{2N-2n}{N+1} & n > N/2 \end{cases}$$

where $n = \{0, 1, 2, \dots, N-1\}$

Run-time Library Reference

Domain

$a > 0; N > 0$

gen_vonhann

generate von hann window

Synopsis

```
#include <window.h>
void gen_vonhann(w,a,N)
float w[]; /* Window vector */
int a;     /* Address stride in samples for window vector */
int N;     /* Length of window vector */
```

Description

This function is identical to gen_hanning window.

Domain

$a > 0$; $N > 0$

Run-time Library Reference

histogram

histogram

Synopsis

```
#include <stats.h>
void histogramf(a,c,max,min,n,m)
const float a[]; /* Pointer to input vector a */
int c[];          /* Pointer to output vector c */
float max;        /* Maximum value of the bin */
float min;        /* Minimum value of the bin */
int n;            /* Number of input samples */
int m;            /* Number of bins */
```

Description

This function computes the histogram of the input elements contained within input vector *a*, and stores the result to output vector *c*.

Algorithm

It bins the element of input vector *x* into *m* equally spaced container, and returns the number of elements in each container.

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

ifft

N point inverse FFT

Synopsis

```
#include <filter.h>
void ifft(in[], t[], out[], w[], wst, n)
const complex_float in[]; /* input sequence          */
complex_float t[];        /* temporary working buffer */
complex_float out[];      /* output sequence          */
const complex_float w[];  /* twiddle sequence         */
int wst;                  /* twiddle factor stride    */
int n;                    /* number of FFT points     */
```

Description

This function transforms the frequency domain complex input signal sequence to the time domain by using the accelerated version of the ‘Discrete Fourier Transformation’ known as an ‘Inverse Fast Fourier Transform’ or IFFT. It will ‘decimate in frequency’ by the best choice FFT algorithm, radix4 or mixed-radix, depending on the input sequence length. If input data can be overwritten, the optimum memory usage is achieved by setting the output pointer to the input array.

For efficiency, the “twiddle table” is calculated once, during initialization, and then provided to the FFT routine as a separate parameter. You must declare the variable and initialize it prior to calling an FFT function. An initialization function, `twidfft`, is provided.

If the twiddle table has been allocated at a larger size than needed for a particular call of `ifft`, then the stride parameter will need to be set appropriately; otherwise, it should be one. [For more information, see “twidfft” on page 3-152.](#)

Run-time Library Reference

Algorithm

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk}$$

It uses core FFT functions implemented as direct radix4, or direct mixed radix algorithm. To get the inverse effect, it first swaps the real and imaginary parts of the input, performs the direct radix4 or mixed radix transformation and finally swaps the real and imaginary parts of the output.

Domain

Input sequence length n must equal to either a power of two, or a power of four, and at least 16.

ifft2d

NxN point 2-d inverse input FFT

Synopsis

```

#include <filter.h>
void ifft2d(*in, *t, *out, w[], wst, n)
const complex_float *in; /* pointer to input matrix a[n][n] */
complex_float *t;        /* pointer to working buffer t[n][n] */
complex_float *out;       /* pointer to matrix c[n][n] */
const complex_float w[]; /* twiddle sequence */
int wst;                  /* twiddle factor stride */
int n;                    /* number of FFT points */

```

Description

This function computes two dimensional inverse Fast Fourier Transform of the complex input matrix `a[n][n]`, and stores the result to the complex matrix `c[n][n]`.

If input data can be overwritten, the optimum memory usage is achieved by setting the output pointer to the input array.

For efficiency, the “twiddle table” is calculated once, during initialization, and then provided to the FFT routine as a separate parameter. You must declare the variable and initialize it prior to calling an FFT function. An initialization function, `twidfft`, is provided.

If the twiddle table has been allocated at a larger size than needed for a particular call of `ifft2d`, then the stride parameter will need to be set appropriately; otherwise, it should be one. [For more information, see “twidfft” on page 3-152.](#)

Run-time Library Reference

Algorithm

$$c(i, j) = \frac{1}{n^2} \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} a(k, l) * e^{2\pi j(i*k + j*l)/n}$$

where $i=\{0,1,\dots,n-1\}$, $j=\{0,1,2,\dots,n-1\}$

Domain

Input sequence length n must equal to either a power of two, or a power of four, and at least 16.

iir

infinite impulse response filter

Synopsis

```
#include <filter.h>
void iir(x,y,n,s)
const float x[]; /* Input sample vector x          */
float y[];       /* Output sample vector y         */
int n;           /* Number of input samples                        */
iir_state *s     /* Pointer to filter state structure */
```

The IIR filter function uses the following structure to maintain the state of the filter:

```
typedef struct
{
    float *c; /* coefficients          */
    float *d; /* start of delay line          */
    int k;    /* no. of bi-quad stages      */
} iir_state;
```

These structure can be initialized with the following function/macro:

```
void iir_init(iir_state *s, float c[], float d[], int k);
```

Description

Using a bi-quad implementation, this function computes an IIR (Infinite Impulse Response Filter) using the coefficients stored in vector `s.c`, delay node points stored in input buffer `s.d`, and applied to the elements of vector `x`. The results are stored in vector `y`.

For efficiency, this filter uses a separate variable of type `iir_state` to maintain the filter state. You must declare space for this variable, and initialize it prior to using the filter. An initialization function, `iir_init` is provided.

Run-time Library Reference

Algorithm

$$H(z) = \frac{B_0 + B_1 z^{-1} + B_2 z^{-2}}{1 - A_1 z^{-1} - A_2 z^{-2}}$$

where

$$D_m = A_2 * D_{m-2} + A_1 * D_{m-1} + x_m$$

$$Y_m = B_2 * D_{m-2} + B_1 * D_{m-1} + B_0 * D_m$$

where $m=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Notes

The B and A coefficients are passed through the `s.c` vector in the `iir_state` structure. `s.c[n+0] = B2`, `s.c[n+1] = B1`, `s.c[n+2] = B0`, `s.c[n+3] = A2`, `s.c[n+4] = A1`. The value of A0 is implied to be 1.0 and A1 and A2 must be scaled accordingly!

log

natural logarithm

Synopsis

```
#include <math.h>
float logf (float x)
double log (double x)
long double logd (long double x)
```

Description

This function calculates the natural logarithm of number x . If $x = 0$, this function will return -3.4×10^{38} for a `float` return value and -1.7×10^{308} for a `long double` return value.

Algorithm

return = $\ln(x)$

Domain

$x = [0.0 \dots 3.4 \times 10^{38}]$ for `logf()`
 $x = [0.0 \dots 1.7 \times 10^{308}]$ for `logd()`

log10

logarithm base 10

Synopsis

```
#include <math.h>
float log10f (float x)
double log10 (double x)
long double log10d (long double x)
```

Description

This function calculates the base 10 logarithm of number x . If $x = 0$, this function will return -3.4×10^{38} for a `float` return value and -1.7×10^{308} for a `long double` return value.

Algorithm

return = $\log(x)$

Domain

$x = [0.0 \dots 3.4 \times 10^{38}]$ for `log10f()`

$x = [0.0 \dots 1.7 \times 10^{308}]$ for `log10d()`

matsadd

real matrix + scalar addition

Synopsis

```

#include <matrix.h>
void matsaddf(a,b,n,m,c)
const float *a; /* Pointer to input matrix a[][] */
float b;        /* value of input scalar b; */
int n;          /* Number of rows in matrix a[][] */
int m;          /* Number of columns in matrix a[][] */
float *c;       /* Pointer to matrix c[][] */

```

Description

This function computes the addition of input matrix $a[][]$ with input scalar b , and stores the result to matrix $c[][]$. The dimensions of matrix $a[][]$ are n and m . The resulting matrix $c[][]$ is of dimensions n and m .

Algorithm

$$c_{i,j} = a_{i,j} + b$$

where $i=\{0,1,2,\dots,n-1\}$, $j=\{0,1,2,\dots,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

matmadd

real matrix + matrix addition

Synopsis

```
#include <matrix.h>
void matmaddf(a,b,n,m,c)
const float *a; /* Pointer to input matrix a[][] */
const float *b; /* Pointer to input matrix b[][] */
int n;          /* Number of rows in matrix a[][] and b[][] */
int m;          /* Nb of columns in matrix a[][] and b[][] */
float *c;       /* Pointer to matrix c[][] */
```

Description

This function adds the input matrix `a[][]` to the input matrix `b[][]` placing the result into the output matrix `c[][]`. The dimensions of matrix `a[][]` are `n` and `m` and the dimensions of matrix `b` are `n` and `m`. The resulting matrix `c[][]` is of dimensions `n` and `m`.

Algorithm

$$c_{i,j} = a_{i,j} + b_{i,j}$$

where $i=\{0,1,2,\dots,n-1\}$, $j=\{0,1,2,\dots,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

matmmlt

real matrix * matrix multiplication

Synopsis

```

#include <matrix.h>
void matmmltf(a,n,k,b,m,c)
const float *a; /* Pointer to input matrix a[][] */
int n;          /* Number of rows in matrix a[][] */
int k;          /* Number of columns in matrix a[][] */
const float *b; /* Pointer to input matrix b[][] */
int m;          /* Number of columns in matrix b[][] */
float *c;       /* Pointer to matrix c[][] */

```

Description

This function computes the multiplication of input matrix `a[][]` with input matrix `b[][]`, and stores the result to matrix `c[][]`. The dimensions of matrix `a[][]` are `n` and `k` and the dimensions of matrix `b` are `k` and `m`. The resulting matrix `c[][]` is of dimensions `n` and `m`.

Algorithm

$$c_{i,j} = \sum_{l=0}^{k-1} a_{i,l} * b_{l,j}$$

where $i=\{0,1,2,\dots,n-1\}$, $j=\{0,1,2,\dots,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

matmsub

real matrix - matrix subtraction

Synopsis

```
#include <matrix.h>
void matmsubf(a,b,n,m,c)
const float *a; /* Pointer to input matrix a[][] */
const float *b; /* Pointer to input matrix b[][] */
int n;          /* Number of rows in matrix a[][] and b[][] */
int m;          /* Number of columns in matrix a[][] and b[][] */
float *c;       /* Pointer to matrix c[][] */
```

Description

This function computes the subtraction of input matrix $a[][]$ with input matrix $b[][]$, and stores the result to matrix $c[][]$. The dimensions of matrix $a[][]$ are n and m and the dimensions of matrix b are n and m . The resulting matrix $c[][]$ is of dimensions n and m .

Algorithm

$$c_{i,j} = a_{i,j} - b_{i,j}$$

where $i=\{0,1,2,\dots,n-1\}$, $j=\{0,1,2,\dots,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

matmlt

real matrix * scalar multiplication

Synopsis

```

#include <matrix.h>
void matmltf(a,b,n,m,c)
const float *a; /* Pointer to input matrix a[][] */
float b;        /* value of input scalar b; */
int n;          /* Number of rows in matrix a[][] */
int m;          /* Number of columns in matrix a[][] */
float *c;       /* Pointer to matrix c[][] */

```

Description

This function computes the multiplication of input matrix `a[][]` with input scalar `b`, and stores the result to matrix `c[][]`. The dimensions of matrix `a[][]` are `n` and `m`. The resulting matrix `c[][]` is of dimensions `n` and `m`.

Algorithm

$$c_{i,j} = a_{i,j} * b$$

where $i=\{0,1,2,\dots,n-1\}$, $j=\{0,1,2,\dots,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

matssub

real matrix - scalar subtraction

Synopsis

```
#include <matrix.h>
void matssubf(a,b,n,m,c)
const float *a; /* Pointer to input matrix a[][] */
float b; /* value of input scalar b; */
int n; /* Number of rows in matrix a[][] */
int m; /* Number of columns in matrix a[][] */
float *c; /* Pointer to matrix c[][] */
```

Description

This function computes the subtraction of input matrix $a[][]$ with input scalar b , and stores the result to matrix $c[][]$. The dimensions of matrix $a[][]$ are n and m . The resulting matrix $c[][]$ is of dimensions n and m .

Algorithm

$$c_{i,j} = a_{i,j} - b$$

where $i=\{0,1,2,\dots,n-1\}$, $j=\{0,1,2,\dots,m-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

max

maximum

Synopsis

```
#include <stdlib.h>
int max (int parm1, int parm2)
float fmaxf (float parm1, float parm2)
double fmax (double parm1, double parm2)
long double fmaxd (long double parm1, long double parm2)
```

Description

Return the larger of its 2 arguments.

Algorithm

```
if (parm1 > parm2)
    return(parm1)
else
    return(parm2)
```

Domain

Full range.

Run-time Library Reference

mean

mean

Synopsis

```
#include <stats.h>
float meanf(a,n)
const float a[]; /* Input vector a */
int n;           /* Number of input samples */
```

Description

This function computes the mean of the input elements contained within input vector *a* and returns the result.

Algorithm

$$c = \frac{1}{n} * (\sum_{i=0}^{n-1} a_i)$$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

min

minimum

Synopsis

```
#include <stdlib.h>
int min (int parm1, int parm2)
float fminf (float parm1, float parm2)
double fmin (double parm1, double parm2)
long double fmind (long double parm1, long double parm2)
```

Description

Return the smaller of its 2 arguments.

Algorithm

```
if (parm1 < parm2)
    return(parm1)
else
    return(parm2)
```

Domain

Full range.

Run-time Library Reference

modf

get fraction and integer

Synopsis

```
#include <math.h>
float modff (float x, float *i)
double modf (double x, double *i)
long double modfd (long double x, long double *i)
```

Description

This function calculates the fractional part and integer part of number x . The fractional part is returned as the value of the function, and the integer part is stored through the pointer argument i .

Algorithm

$$return = (|x| - floor|x|) \cdot sign(x)$$
$$i = (floor|x| \cdot sign(x))$$

Domain

$$x = [-3.4 \times 10^{38} \dots 3.4 \times 10^{38}] \quad \text{for } modf()$$
$$x = [-1.7 \times 10^{308} \dots 1.7 \times 10^{308}] \quad \text{for } modd()$$

mu_compress

μ-law compression

Synopsis

```
#include <filter.h>
void mu_compress(in, out, n)
const int in[]; /* Input array */
int out[];      /* Output array */
int n;          /* number of elements to be compressed */
```

Description

This function performs μ-law compression (ITU rec. G.711) on the elements of the input vector `in` and outputs expanded data into the vector `out`. The function has been optimized and requires that both the input and output vectors are quad-word aligned.

Algorithm

$C(k)$ = mu_low compression of $A(k)$ for $k=0$ to $n-1$.

Domain

content of input array: -8192 to 8191

Run-time Library Reference

mu_expand

μ-law expansion

Synopsis

```
#include <filter.h>
void mu_expand(in, out, n)
const int in[]; /* Input array */
int out[];      /* Output array */
int n;          /* number of elements to be expanded */
```

Description

This function performs μ-law expansion (ITU rec. G.711) on the elements of the input vector `in` and outputs expanded data into the vector `out`. The function has been optimized and requires that both the input and output vectors are quad-word aligned.

Algorithm

$C(k) = \text{mu_law expansion of } A(k) \text{ for } k=0 \text{ to } n-1$

Domain

content of input array: 0 to 255

norm

normalization

Synopsis

```
#include <complex.h>
complex_float normf(a)
complex_float a; /* Complex input a */
```

Description

This function normalizes the complex input *a* and stores the result in complex output *c*.

Algorithm

$$\text{Re}(c) = \frac{\text{Re}(a)}{\sqrt{\text{Re}^2(a) + \text{Im}^2(a)}}$$

$$\text{Im}(c) = \frac{\text{Im}(a)}{\sqrt{\text{Re}^2(a) + \text{Im}^2(a)}}$$

Domain

-3.4 x 10³⁸ to +3.4 x 10³⁸

Run-time Library Reference

polar

construct from polar coordinates

Synopsis

```
#include <complex.h>
complex_float polarf(mag, phase)
float mag;      /* Magnitude */
float phase;    /* Phase      */
```

Description

This function transforms a polar coordinate to a normal coordinate.

Algorithm

$$\text{Re}(c) = r \cdot \cos(\theta)$$

$$\text{Im}(c) = r \cdot \sin(\theta)$$

Domain

phase = [-1,647,095 ... 1,647,095] for polarf()

phase = [-843,314,850 ... 843,314,850] for polard()

mag = -3.4×10^{38} to $+3.4 \times 10^{38}$

pow

power

Synopsis

```
#include <math.h>
float powf (float x, float y)
double pow (double x, double y)
long double powd (long double x, long double y)
```

Description

This function calculates x to the power y . If $x < 0$ and y is not an integral value, this function returns 0. If $x = 0$ and $y = 0$, this function returns 0. If overflow occurs, this function returns 3.4×10^{38} for a float-type return value and 1.7×10^{308} for a double-type return value; if underflow occurs, this function returns -3.4×10^{38} for a float return value and -1.7×10^{308} for a long double return value.

Algorithm

$$\text{return} = x^y$$

Domain

$x, y = [-3.4 \times 10^{38} \dots 3.4 \times 10^{38}]$ except $x < 0, y \neq i$ where i is an integer for `powf()`

$x, y = [-1.7 \times 10^{308} \dots 1.7 \times 10^{308}]$ except $x < 0, y \neq i$ where i is an integer for `powd()`

qsort

quick sort

Synopsis

```
#include <stdlib.h>
void qsort(void base, size_t nelem, size_t size,
int (*compare) (const void *, const void *));
```

Description

The `qsort` function sorts an array of `nelem` objects, pointed to by `base`. The size of each object is specified by `size`.

The contents of the array are sorted into ascending order according to a comparison function pointed to by `compare`, which is called with two arguments that point to the objects being compared. The function shall return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second.

- If two elements compare as equal, their order in the sorted array is unspecified. `qsort` executes a binary-search operation on a pre-sorted array
- `key` is a pointer to the element to search for.
- `base` points to the start of the array.
- `nelem` is the number of elements in the array.
- `size` is the size of each element of the array.
- `compare` points to the function used to compare two elements. It takes as parameters a pointer to the key and a pointer to an array element and should return a value less than, equal to, or greater than zero, according to whether the first parameter is less than, equal to, or greater than the second.

Algorithm

See description.

Domain

N/A

Run-time Library Reference

rand

random number generator

Synopsis

```
#include <stdlib.h>
int rand (void)
```

Description

Returns a pseudo-random number.

Algorithm

The algorithm is based on a linear congruential generator.

Domain

N/A

rfft

N point real input FFT

Synopsis

```
#include <filter.h>
void rfft(in[], t[], out[], w[], wst, n)
const float in[];          /* input/output sequence */
complex_float t[];         /* temporary working buffer*/
complex_float out[];       /* working buffer */
|const complex_float w[]; /* twiddle sequence */
int wst;                   /* twiddle factor stride */
int n;                     /* number of FFT points */
```

Description

This function transforms the time domain real input signal sequence to the frequency domain by using the accelerated version of the ‘Discrete Fourier Transformation’ known as a ‘Fast Fourier Transform’ or FFT. It will ‘decimate in frequency’ by the best choice FFT algorithm, radix-4 or mixed-radix, depending on the input sequence length. At the initial stage of the transformation, RFFT takes advantage of the fact that the imaginary part of the input equals zero, which in turn eliminates half of the multiplications in the butterfly.

If input data can be overwritten, the optimum memory usage is achieved by setting the output pointer to the input array and providing that the input array is 2N.

For efficiency, the “twiddle table” is calculated once, during initialization, and then provided to the FFT routine as a separate parameter. You must declare the variable and initialize it prior to calling an FFT function. An initialization function, `twidfft`, is provided.

Run-time Library Reference

If the twiddle table has been allocated at a larger size than needed for a particular call of `rfft`, then the stride parameter will need to be set appropriately; otherwise, it should be one. [For more information, see “twidfft” on page 3-152.](#)

Algorithm

See [“cfft” on page 3-52](#). Output sequence will have $n/2$ complex members due to a fact that `rfft` of real input data produces symmetric output around half sampling frequency point.

Domain

Input sequence length n must equal to either a power of two, or a power of four, and at least 16.

rfft2d

NxN point 2-d real input FFT

Synopsis

```

#include <filter.h>
void rfft2d(*in, *t, *out, w[], wst, n)
const float *in;          /* pointer to input matrix a[n][n]    */
complex_float *t;         /* pointer to working buffer t[n][n] */
complex_float *out;       /* pointer to matrix c[n][n]         */
const complex_float w[]; /* twiddle sequence                  */
int wst;                  /* twiddle factor stride              */
int n;                   /* number of FFT points              */

```

Description

This function computes a two dimensional Fast Fourier Transform of the real input matrix `a[n][n]`, and stores the result in the complex matrix `c[n][n]`.

If input data can be overwritten, the optimum memory usage is achieved by setting the output pointer to the input array and providing that the input array is 2 times NxN.

For efficiency, the “twiddle table” is calculated once, during initialization, and then provided to the FFT routine as a separate parameter. You must declare the variable and initialize it prior to calling an FFT function. An initialization function, `twidfft`, is provided.

If the twiddle table has been allocated at a larger size than needed for a particular call of `rfft2d`, then the stride parameter will need to be set appropriately; otherwise, it should be one. [For more information, see “twidfft” on page 3-152.](#)

Run-time Library Reference

Algorithm

$$c(i, j) = \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} a(k, l) * e^{-2\pi j(i*k + j*l)/n}$$

where $i=\{0,1,\dots,n-1\}$, $j=\{0,1,2,\dots,n-1\}$

Domain

Input sequence length n must equal to either a power of two, or a power of four, and at least 16.

rms

root mean square

Synopsis

```
#include <stats.h>
float rmsf(a,n)
const float a[]; /* Pointer to input vector a */
int n;           /* number of input samples */
```

Description

This function computes the root mean square of the input elements contained within input vector *a* and returns the result.

Algorithm

$$c = \sqrt{\frac{\sum_{i=0}^{n-1} a_i^2}{n}}$$

Domain

-3.4 x 10³⁸ to +3.4 x 10³⁸

Run-time Library Reference

rsqrt

reciprocal square root

Synopsis

```
#include <math.h>
float rsqrtf (float x)
double rsqrt (double x)
long double rsqrtl (long double x)
```

Description

This function calculates the reciprocal of the square root of the number x .
If x is negative, this function returns 0.

Algorithm

$$\text{return} = 1/(\sqrt{x})$$

Domain

$x = [0.0 \dots 3.4 \times 10^{38}]$ for `rsqrtf()`
 $x = [0.0 \dots 1.7 \times 10^{308}]$ for `rsqrtl()`

sign

sign

Synopsis

```
#include <math.h>
float signf (float parm1, float parm2)
double sign (double parm1, double parm2)
long double signd (long double parm1, long double parm2)
```

Description

Copies the sign of the second argument to the first argument.

Algorithm

```
return( |parm1| * signof( parm2))
```

Domain

Full range.

Run-time Library Reference

sin

sine

Synopsis

```
#include <math.h>
float sinf (float x)
double sin (double x)
long double sind (long double x)
```

Description

This function calculates the sine of number x where x is measured in radians. The output is in the range $[-1$ to $1]$. If x is outside of the domain, this function returns 0.0.

Algorithm

return = $\sin(x)$

Domain

$x = [-1,647,095 \dots 1,647,095]$ for `sinf()`
 $x = [-843,314,850 \dots 843,314,850]$ for `sind()`

sinh

hyperbolic sine

Synopsis

```
#include <math.h>
float sinhf (float x)
double sinh (double x)
long double sinhd (long double x)
```

Description

This function calculates the hyperbolic sine of number x where x is measured in radians. If x is outside the domain, this function returns 3.4×10^{38} for a float-type return value and 1.7×10^{308} for a double-type return value.

Algorithm

$$return = \sinh(x)$$

Domain

$$x = [-(\ln(3.4 \times 10^{38}) - \ln(2)) \dots (\ln(3.4 \times 10^{38}) - \ln(2))] \quad \text{for } \sinhf()$$

$$x = [-(\ln(1.7 \times 10^{308}) - \ln(2)) \dots (\ln(1.7 \times 10^{308}) - \ln(2))] \quad \text{for } \sinhd()$$

Run-time Library Reference

sqrt

square root

Synopsis

```
#include <math.h>
float sqrtf (float x)
double sqrt (double x)
long double sqrtld (long double x)
```

Description

This function calculates the square root number x . If x is negative, this function returns 0.

Algorithm

return = $\sqrt{x}$

Domain

$x = [0.0 \dots 3.4 \times 10^{38}]$ for `sqrtf()`
 $x = [0.0 \dots 1.7 \times 10^{308}]$ for `sqrtld()`

srand

set seed for random number generator

Synopsis

```
#include <stdlib.h>
void srand (unsigned int newseed)
```

Description

Sets the seed for the pseudo-random number generator `rand`.

Algorithm

generator_seed = *newseed*

Domain

0 to RAND_MAX

Run-time Library Reference

strtod

string to double conversion

Synopsis

```
#include <stdlib.h>
double strtod (const char *string, char **end)
```

Description

Converts a character string to a `double` value where the input string is a sequence of characters that can be interpreted as a numerical value of the specified type.

Algorithm

The string argument is as follows:

[whitespace][sign][digits][.digits][{e|E}[sign]digits]

where *whitespace* can consist of spaces and/or tab characters, *sign* is either plus (+) or minus (-), and *digits* are one or more decimal digits. If no digits appear before the decimal point then at least one must appear after the decimal point. The decimal digits may be followed by an exponent denoted by one of the following characters: *e*, or *E* followed by a decimal integer which may be signed.

The function stops reading the input string at the first character that it cannot recognize as part of a valid argument defined above. It sets **end* to that location.

Domain

The number should fit within the dynamic range of a `double`.

strtof

string to float conversion

Synopsis

```
#include <stdlib.h>
float strtof (const char *string, char **end)
```

Description

Converts a character string to a single-precision floating-point value where the input string is a sequence of characters that can be interpreted as a numerical value of the specified type.

Algorithm

The string argument is as follows:

[whitespace][sign][digits][.digits][{e|E}[sign]digits]

where whitespace can consist of spaces and/or tab characters, sign is either plus (+) or minus (-), and digits are one or more decimal digits. If no digits appear before the decimal point then at least one must appear after the decimal point. The decimal digits may be followed by an exponent denoted by one of the following characters: e, or E followed by a decimal integer which may be signed.

The function stops reading the input string at the first character that it cannot recognize as part of a valid argument defined above. It sets *end to that location.

Domain

The number should fit within the dynamic range of a 32 bits float.

strtoi

string to integer conversion

Synopsis

```
#include <stdlib.h>
int strtoi (const char *string, char **end, int base)
```

Description

Converts a character string to a integer fixed-point value where the input string is a sequence of characters that can be interpreted as a numerical value of the specified type.

Algorithm

The string argument is as follows:

[whitespace][sign][base]digits

where *whitespace* can consist of spaces and/or tab characters, *sign* is either plus (+) or minus (-), *base* is 0 for octal and 0x for hexadecimal, and *digits* are one or more decimal digits (or letters from a to f for hexadecimal numbers). The function stops reading the input string at the first character that it cannot recognize as part of a valid argument defined above. It sets **end* to that location.

If the third argument *base* is zero, then the function tries to determine the base from the information it finds inside the string (see format description above). If the third argument *base* is non-zero, then it has to be between 2 and 36 inclusively. If the third argument *base* doesn't correspond to any of the above values, it is set to 10. If the third argument *base* is between 11 and 36 (inclusively) the letters of the alphabet from a to z are used as needed to represent the extra digits in the corresponding base.

Domain

-2,147,483,648 to 2,147,483,647

strtol

string to long integer conversion

Synopsis

```
#include <stdlib.h>
long strtol (const char *string, char **end, int base)
```

Description

Converts a character string to a integer fixed-point value where the input string is a sequence of characters that can be interpreted as a numerical value of the specified type.

Algorithm

The string argument is as follows:

[whitespace][sign][base]digits

where *whitespace* can consist of spaces and/or tab characters, *sign* is either plus (+) or minus (-), *base* is 0 for octal and 0x for hexadecimal, and *digits* are one or more decimal digits (or letters from a to f for hexadecimal numbers). The function stops reading the input string at the first character that it cannot recognize as part of a valid argument defined above. It sets **end* to that location.

If the third argument *base* is zero, then the function tries to determine the base from the information it finds inside the string (see format description above). If the third argument *base* is non-zero, then it has to be between 2 and 36 inclusively. If the third argument *base* doesn't correspond to any of the above values, it is set to 10. If the third argument *base* is between 11 and 36 (inclusively) the letters of the alphabet from a to z will be used as needed to represent the extra digits in the corresponding base.

Domain

-2,147,483,648 to 2,147,483,647

Run-time Library Reference

strtold

string to long double conversion

Synopsis

```
#include <stdlib.h>
long double strtold (const char *string, char **end)
```

Description

Converts a character string to a double-precision floating-point value where the input string is a sequence of characters that can be interpreted as a numerical value of the specified type.

Algorithm

The string argument is as follows:

[whitespace][sign][digits][.digits][{e|E}[sign]digits]

where *whitespace* can consist of spaces and/or tab characters, *sign* is either plus (+) or minus (-), and *digits* are one or more decimal digits. If no digits appear before the decimal point then at least one must appear after the decimal point. The decimal digits may be followed by an exponent denoted by one of the following characters: e, or E followed by a decimal integer which may be signed.

The function stops reading the input string at the first character that it cannot recognize as part of a valid argument defined above. It sets **end* to that location.

Domain

The number should fit within the dynamic range of a 64 bits double.

strtoul

string to unsigned long integer conversion

Synopsis

```
#include <stdlib.h>
unsigned long strtoul (const char *string, char **end, int base)
```

Description

Converts a character string to a integer fixed-point value where the input string is a sequence of characters that can be interpreted as a numerical value of the specified type.

Algorithm

The string argument is as follows:

[whitespace][base]digits

where *whitespace* can consist of spaces and/or tab characters, *base* is 0 for octal and 0x for hexadecimal, and *digits* are one or more decimal digits (or letters from 'a' to 'f' for hexadecimal numbers). The function stops reading the input string at the first character that it cannot recognize as part of a valid argument defined above. It sets **end* to that location.

If the third argument *base* is zero, then the function tries to determine the base from the information it finds inside the string (see format description above). If the third argument *base* is non-zero, then it has to be between 2 and 36 inclusively. If the third argument *base* doesn't correspond to any of the above values, it is set to 10. If the third argument *base* is between 11 and 36 (inclusively) the letters of the alphabet from a to z will be used as needed to represent the extra digits in the corresponding base.

Domain

0 to 4,294,967,295

Run-time Library Reference

tan

tangent

Synopsis

```
#include <math.h>
float tanf (float x)
double tan (double x)
long double tand (long double x)
```

Description

This function calculates the tangent of number x where x is measured in radians. If x is outside of the domain, this function returns 0.0

Algorithm

return = $\tan(x)$

Domain

$x = [-6,588,397 \dots 6,588,397]$ for `tanf()`
 $x = [-421,657,424 \dots 421,657,424]$ for `tand()`

tanh

hyperbolic tangent

Synopsis

```
#include <math.h>
float tanhf (float x)
double tanh (double x)
long double tanhd (long double x)
```

Description

This function calculates the hyperbolic tangent of number x where x is measured in radians. If x is outside the domain, this function returns 3.4×10^{38} for a float-type return value and 1.7×10^{308} for a double-type return value.

Algorithm

$$return = \tanh(x)$$
Domain

$$x = [-(\ln(3.4 \times 10^{38}) - \ln(2)) \dots (\ln(3.4 \times 10^{38}) - \ln(2))] \quad \text{for } \tanhf()$$

$$x = [-(\ln(1.7 \times 10^{308}) - \ln(2)) \dots (\ln(1.7 \times 10^{308}) - \ln(2))] \quad \text{for } \tanhd()$$

Run-time Library Reference

twidfft

generate FFT twiddle factors

Synopsis

```
#include <filter.h>
void twidfft(w[], n)
complex_float w[]; /* twiddle sequence */
int n;             /* number of FFT points */
```

Description

Various different versions of the FFT are provided, including `cffft`, `rffft`, `iffft`, `cffft2d`, `rffft2d`, `iffft2d`. The number of points is provided as an argument; when appropriate, the library uses radix 2 or radix 4 implementations.

For efficiency, the `twiddle table` is calculated once, during initialization, and then provided to the FFT routine as a separate parameter. You must declare the variable and initialize it prior to calling an FFT function. The function `twidfft` is the initialization function.

Several FFTs of different sizes can all be accommodated with the same twiddle factor table. Simply allocate the table at the maximum size. Each FFT has an additional parameter, the “stride” of the twiddle table. To use the whole table, specify a stride of 1. If your FFT uses only half the points of the largest, the stride should be 2 (this takes only every other element).

Algorithm

This function takes FFT length n as an input parameter and generates the lookup table of complex twiddle coefficients. $3/4$ of one period of sine/cosine is described by $3/4 n$ complex samples in the lookup table.

The samples are generated as follows:

$$twid_re(k) = \cos\left(\frac{2\pi}{n} k\right)$$

$$twid_im(k) = \sin\left(\frac{2\pi}{n} k\right)$$

for $k=0$ to $3/4\ n$

Domain

n must equal to either a power of two, or a power of four and at least 16.

Run-time Library Reference

var

variance

Synopsis

```
#include <stats.h>
float varf(a,n)
const float a[]; /* Pointer to input vector a */
int n;           /* number of input samples */
```

Description

This function computes the variance of the input elements contained within input vector *a* and returns the result.

Algorithm

$$c = \frac{n * \sum_{i=0}^{n-1} a_i^2 - (\sum_{i=0}^{n-1} a_i)^2}{n(n-1)}$$

Domain

-3.4 x 10³⁸ to +3.4 x 10³⁸

vecdot

real vector dot product

Synopsis

```
#include <vector.h>
float vecdotf(a,b,n)
const float a[]; /* Input vector a */
const float b[]; /* Input vector b */
int n;           /* Element count */
```

Description

This function computes the dot product of the two input vectors *a* and *b*, and returns the scalar result.

Algorithm

$$return = \sum_{i=0}^{n-1} a_i * b_i$$

where $i=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

vecsadd

real vector + scalar addition

Synopsis

```
#include <vector.h>
void vecsaddf(a,b,c,n)
const float a[]; /* Input vector a */
float b;          /* Input scalar */
float c[];        /* Output vector */
int n;           /* Element count */
```

Description

This function adds input scalar *b* to each element of input vector *a*. The results are stored in the output vector *c*.

Algorithm

$$c_i = a_i + b$$

where *i* = {0,1,2,...,n-1}

Domain

-3.4 x 10³⁸ to +3.4 x 10³⁸

vecsmult

real vector * scalar multiplication

Synopsis

```
#include <vector.h>
void vecsmultf(a,b,c,n)
const float a[]; /* Input vector a */
float b;          /* Input scalar */
float c[];        /* Output vector */
int n;           /* Element count */
```

Description

This function multiplies each element of input vector *a* by input scalar *b*. The results are stored in the output vector *c*.

Algorithm

$$c_i = a_i * b$$

where $i=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

vecssub

real vector - scalar subtraction

Synopsis

```
#include <vector.h>
void vecssubf(a,b,c,n)
const float a[]; /* Input vector a */
float b;         /* Input scalar  */
float c[];       /* Output vector */
int n;          /* Element count */
```

Description

This function subtracts input scalar *b* from each element of input vector *a*.
The results are stored in the output vector *c*.

Algorithm

$$c_i = a_i - b$$

where $i=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

vecvadd

real vector + vector addition

Synopsis

```
#include <vector.h>
void vecvaddf(a,b,c,n)
const float a[]; /* Input vector a */
const float b[]; /* Input vector b */
float c[];        /* Output vector */
int n;           /* Element count */
```

Description

This function adds two input vectors. The results are stored in the output vector *c*.

Algorithm

$$c_i = a_i + b_i$$

where $i=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

vecvmlt

real vector * vector multiplication

Synopsis

```
#include <vector.h>
void vecvmlt(f,a,b,c,n)
const float a[]; /* Input vector a */
const float b[]; /* Input vector b */
float c[];        /* Output vector */
int n;            /* Element count */
```

Description

This function multiplies two input vectors *a* and *b*. The results are stored in the output vector *c*.

Algorithm

$$c_i = a_i * b_i$$

where $i=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

vecvsub

real vector - vector subtraction

Synopsis

```
#include <vector.h>
void vecvsubf(a,b,c,n)
const float a[]; /* Input vector a */
const float b[]; /* Input vector b */
float c[];        /* Output vector */
int n;           /* Element count */
```

Description

This function subtracts input vector *b* from input vector *a*. The results are stored in the output vector *c*.

Algorithm

$$c_i = a_i - b_i$$

where $i=\{0,1,2,\dots,n-1\}$

Domain

-3.4×10^{38} to $+3.4 \times 10^{38}$

Run-time Library Reference

zero_cross

count zero crossing

Synopsis

```
#include <stats.h>
int zero_crossf(a,n)
const float a[]; /* Pointer to input vector a */
int n;           /* Number of input samples */
```

Description

This function computes the number of times that a signal crosses over the zero line and returns the result.

Algorithm

The actual algorithm is different from what is shown below because we need to handle the case where an element of the array is zero, but this gives you an understanding;

if (a(i) > 0 && a(i+1) < 0)|| (a(i) < 0 && a(i+1) > 0)
number of zeros increased by one

Domain

-3.4 x 10³⁸ to +3.4 x 10³⁸