TMS320C67x FastRTS Library

Will soon move to here [1]

Introduction

The C67x FastRTS library is an optimized floating-point math function library for C programmers using TMS320C67x devices. These routines are typically used in computationally intensive real-time applications where optimal execution speed is critical. By using these routines instead of the routines found in the existing run-time-support libraries, you can achieve execution speeds considerably faster without rewriting existing code. The FastRTS library includes all floating-point math routines currently provided in existing run-time-support libraries for C6000. These new functions can be called with the current run-time-support library names or the new names included in the FastRTS library.

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

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

The FastRTS library provides the following features and benefits:

- Hand-coded assembly-optimized routines
- C-callable routines, which can be inlined and are fully compatible with the TMS320C6000

compiler

- Routines which accept single sample or vector inputs
- Provided functions are tested against C model and existing run-time-support

functions

Installation and Usage

FastMath Library Contents

lib	directory that contains Little-endian library (C674xfastMath.lib, C67xfastMath.lib, C67pfastMath.lib)
src	Source archive dirctories for all included fastMath functions
inc	Directory that contains header file C67fastMath.h
docs	Directory containing the following documentation files:
TI_Software_Manifest.pdf	License agreement

How to Install the FastMath Library

Visit the C67x Fast Run-Time Support (RTS) Library [2] on ti.com:

Download the Windows Installer sprc060.zip [3]

To install the FastMath libary, follow these steps:

Step 1: Open the file c67xmathlib_2_01_00_00_Windows_Setup.exe.

Step 2: Click Yes to install the library.

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

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

Step 5: Click Next to continue.If you selected “I accept,” the installation will continue. If you selected “I don’t accept,” the installation cancels.

Step 6: Choose the location where you would like to install the library. The wizard will install the header files in the include directory, documentation in the doc directory, and the library and source files in the lib directory.The default install location is c:\ti. Libraries will be installed in c:\C67mathlib.

Step 7: Click Next.

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

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

Using the FastMath Library

Before using the FastMath library functions, you need to update your linker command file. If you want to use the FastMath functions in place of the existing versions of these functions, the FastRTS library must be linked in before the existing run-time-support library. Ensure that you link with the correct run-time-support library and the FastMath library for little-endian code by adding the following line in your linker command file before the line linking the current run-time-support library:

-l C67xfastMath.lib

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

-l fastrts67xe.lib

Describe

1. single sample asm
2. single sample c intrinsics
3. vector c intrinsics

4. inline single sample c intrinsics

FastRTS Library Arguments and Data Types

FastRTS Types

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

FastRTS Arguments

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

Calling a FastMath Function From C

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

- Include the path to the lib and to the header file in the fastMath library.

```
-i <FASTMATH_INSTALL_DIR>\inc -i<FASTMATH_INSTALL_DIR>\lib
```

- Include the function header file corresponding to the FastMath function:
- The c67fastMath.h header file must be included if you use the special FastMath function names.
- The math.h header file must be included if you are using the standard run-time-support function names.
- Link your code with C67xfastMath.lib for little-endian code.
- Use the correct linker command file for the platform you use. Remember, the FastMath library replaces only a subset of the functions in the current run-time-support libraries. Therefore, C674xfastrts.lib must be linked in before rts6700.lib.

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

1. include <math.h>

in your C file and compile and link using

cl6x main.c –z –o drv.out –lC67xfastMath.lib –rts6701.lib

Note: Adding FastMath in Code Composer Studio If you set up a project under Code Composer Studio, you can add the FastMath library to your project by selecting Project→Add Files to Project and choosing C67xfastMath.lib.

Calling a FastMath Function From Assembly

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

How to Rebuild the FastMath Library

If you want to rebuild the FastMath library (for example, because you modified the source file contained in the archive) Open the C67fastMath.pjt project from the build folder and rebuild the project. This will generate a C67XXfastMath_rebuild.lib in the folder C67XX where XX will depend on the device configration choosen in the project. You can use the rebuilt library to consume the changes made to the source files.

FastMath Library Functions Tables

Arguments and Conventions Used

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

Argument Description

z,r,n Argument reflecting input data

x Argument reflecting output data

The _i ending after the function is the single sample version of the function that performs inlinings. The sp or dp ending is the single sample version without inlining.

FastMath Functions

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

Description	SP	DP	SP	DP
arc tangent	atanf	atan	atansp	atandp
arc tangent of 2 arguments	atan2f	atan2	atansp	atandp
cosine	cosf	cos	cossp	cosdp
exponential	expf	exp	expsp	expdp
exponential base 2	exp2f	exp2	exp2sp	expdp
exponential base 10	exp10f	exp10	exp10sp	exp10dp
logorithm base e	logf	log	logsp	logdp
logorithm base 2	log2f	log2	log2sp	log2dp
logorithm base 10	log10f	log10	log10sp	log10dp
power = X raised to power Y	powf	pow	powsp	powdp
reciprocal = 1/argument	recipf	recip	recipsp	recipdp
square root of reciprocal	rsqrtf	rsqrt	rsqrtsp	rsqrtdp
sine	sinf	sin	sinsp	sindp
división of 2 arguements	_divf	_divd	divsp	divdp

Some of the FastRTS functions call the routines listed for improved performance. The functions which are called inside other function include div, log, and exp and sine.

General FastRTS Functions

atan2dp_v(Double-Precision Polar Arc Tangent (2 argument))

Function:

```
void  atandp_v(double *z, double *z2, double *r, int n)
```

Parameters:

- z : input double vector
- z2 : input 2 double vector
- r : Resultant double vector
- n : Number of elements in the vectors

Description: The atan2dp_v function returns the arc tangent of a floating-point argument z/z2. The return value is an angle in the range [-π/2, π/2] radians.

Special Cases:

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

***Other available functions:**

- static inline double atan2dp_i(double a, double b)
- double atan2dp(double a, double b)

atandp_v(Double-Precision Polar Arc Tangent)

Function:

```
void  atandp_v(double *z, double *r, int n)
```

Parameters:

- z : input double vector
- r : Resultant double vector
- n : Number of elements in the vectors

Description: The atandp_v function returns the arc tangent of a floating-point argument z. The return value is an angle in the range [-π/2, π/2] radians.

Special Cases:

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

***Other available functions:**

- static inline double atandp_i(double a)
- double atandp(double a)

atansp_v(Single-Precision Polar Arc Tangent)

Function:

```
void atansp_v(float *z, float *r, int n)
```

Parameters:

- z : input float vector
- r : Resultant float vector
- n : Number of elements in the vectors

Description: atansp_v function return the arc tangent of a floating-point argument z. The return value is an angle in the range [-π/2, π/2] radians.

Special Cases:

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

***Other available functions:**

- static inline float atansp_i(float a)
- float atansp(float a)

cosdp_v(Double-Precision Cosine)

Function

```
void cosdp_v(double *z, double *r, int n)
```

Parameters:

- z : input double vector
- r : Resultant double vector
- n : Number of elements in the vectors

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

Special Cases:

- If | W | < 9.536743e-7 = 2- 20, then the return value is W for small angles.
- If | W | > 1.0737e+9 = 2+30, then the return value is zero for large angles.

***Other available functions:**

- static inline double cosdp_i(double a)
- double cosdp(double a)

cossp_v (Single-Precision Cosine)

Function:

```
void cossp_v (float *z, float *r, int n)
```

Parameters:

- z : input float vector
- r : Resultant float vector
- n : Number of elements in the vectors

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

Special Cases:

- If | W | < 2.44e-4 = 2- 12, then the return value is W for small angles.
- If | W | > 1.04858e+6 = 2+20, then the return value is zero for large angles.

***Other available functions:**

- static inline float cossp_i(float a)
- float cossp(float a)

sindp_v (Double-Precision Sine)

Function

```
void sindp_v (,double *z, double *r, int n)
```

Parameters:

- z : input double vector
- r : Resultant double vector
- n : Number of elements in the vectors

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

Special Cases:

- If | W | < 9.536743e-7 = 2- 20, then the return value is W for small angles.
- If | W | > 1.0737e+9 = 2+30, then the return value is zero for large angles.

***Other available functions:**

- static inline double sindp_i(double a)
- double sindp(double a)

sinsp_v (Single-Precision Sine)

Function:

```
void sinsp_v (float *z, float *r, int n)
```

Parameters:

- z : input float vector
- r : Resultant float vector
- n : Number of elements in the vectors

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

Special Cases:

- If | W | < 2.44e-4 = 2- 12, then the return value is W for small angles.
- If | W | > 1.04858e+6 = 2+20, then the return value is zero for large angles.

***Other available functions:**

- static inline float sinsp_i(float a)
- float sinsp(float a)

expdp_v (Double-Precision Exponential)

Function

```
void expdp_v (double *z, double *r, int n)
```

Parameters:

- z : input double vector
- r : Resultant double vector
- n : Number of elements in the vectors

Description: The expdp_v function returns the exponential function of a real floating-point argument z. The return value is the number e raised to power z. If the magnitude of z is too large, the maximum double-precision floating-point number (1.797693e+308 = 2+1024) is returned.

Special Cases:

- If | z | < 1.11e-16 = 2- 53, then the return value is 1.0 for small arguments.
- If z < -708.3964 = minimum log e (2.225e-308 = 2- 1022), then the return value is 0.0.
- If z > +709.7827 = maximum log e (1.797693e+308 = 2+1024), then the return value is 1.797693e+308 = 2+1024(maximum double-precision floating-point

number).

***Other available functions:**

- static inline double expdp_i(double a)
- double expdp(double a)

expsp_v (Single-Precision Exponential)

Function:

```
void expsp_v (float *z, float *r, int n)
```

Parameters:

- z : input float vector
- r : Resultant float vector r= (float)x
- n : Number of elements in the vectors

Description: The expdp_v function returns the exponential function of a real floating-point argument z. The return value is the number e raised to power z. If the magnitude of z is too large, the maximum double-precision floating-point number (1.797693e+308 = 2+1024) is returned.

Special Cases:

- If | z | < 9.313e-10 = 2- 30, then the return value is 1.0 for small arguments.
- If z < -87.3365 = minimum log e (1.175e-38 = 2- 126), then the return value is 0.0.
- If z > +88.7228 = maximum log e (3.402823e+38 = 2+128), then the return value is 3.402823e+38 = 2+128 (maximum single-precision floating-point number).

***Other available functions:**

- static inline float expsp_i(float a)
- float expsp(float a)

exp2dp_v (Double-Precision Exponent of 2)

Function

```
void exp2dp(double *z, double *r, int n)
```

Parameters:

- z : input double vector
- r : Resultant double vector r= (float)x
- n : Number of elements in the vectors

Description: The exp2dp_v function returns the exponential function of a real floating-point argument z. The return value is the number 2 raised to power z. If the magnitude of z is too large, the maximum double-precision floating-point number (1.797693e+308 = 2+1024) is returned.

Special Cases:

- If | z | < 1.11e-16 = 2- 53, then the return value is 1.0 for small arguments.
- If z < -708.3964 = minimum log 2 (2.225e-308 = 2- 1022), then the return value is 0.0.
- If z > +709.7827 = maximum log 2 (1.797693e+308 = 2+1024), then the return value is 1.797693e+308 = 2+1024(maximum double-precision floating-point

number).

***Other available functions:**

- static inline double expdp_i(double a)
- double expdp(double a)

exp2sp_v (Single-Precision Exponent of 2)

Function:

```
void exp2sp_v (float *z, float *r, int n)
```

Parameters:

- z : input float vector
- r : Resultant float vector
- n : Number of elements in the vectors

Description: The exp2dp_v function returns the exponential function of a real floating-point argument z. The return value is the number 2 raised to power z. If the magnitude of z is too large, the maximum double-precision floating-point number (1.797693e+308 = 2+1024) is returned.

Special Cases:

- If | z | < 9.313e-10 = 2- 30, then the return value is 1.0 for small arguments.
- If z < -87.3365 = minimum log 2 (1.175e-38 = 2- 126), then the return value is 0.0.
- If z > +88.7228 = maximum log 2 (3.402823e+38 = 2+128), then the return value is 3.402823e+38 = 2+128 (maximum single-precision floating-point number).

***Other available functions:**

- static inline float exp2sp_i(float a)
- float exp2sp(float a)

exp10dp_v (Double-Precision Exponent of 10)

Function

```
void exp10dp(double *z, double *r, int n)
```

Parameters:

- z : input double vector
- r : Resultant double vector
- n : Number of elements in the vectors

Description: The exp10dp_v function returns the exponential function of a real floating-point argument z. The return value is the number 10 raised to power z. If the magnitude of z is too large, the maximum double-precision floating-point number (1.797693e+308 = 2+1024) is returned.

Special Cases:

- If | z | < 1.11e-16 = 2- 53, then the return value is 1.0 for small arguments.
- If z < -708.3964 = minimum log 2 (2.225e-308 = 2- 1022), then the return value is 0.0.
- If z > +709.7827 = maximum log 2 (1.797693e+308 = 2+1024), then the return value is 1.797693e+308 = 2+1024(maximum double-precision floating-point

number).

***Other available functions:**

- static inline double exp10dp_i(double a)
- double exp10dp(double a)

exp10sp_v (Single-Precision Exponent of 10)

Function:

```
void exp10sp_v (float *z, float *r, int n)
```

Parameters:

- z : input float vector
- r : Resultant float vector
- n : Number of elements in the vectors

Description: The exp2dp_v function returns the exponential function of a real floating-point argument z. The return value is the number 10 raised to power z. If the magnitude of z is too large, the maximum double-precision floating-point number (1.797693e+308 = 2+1024) is returned.

Special Cases:

- If | z | < 9.313e-10 = 2- 30, then the return value is 1.0 for small arguments.
- If z < -87.3365 = minimum log 10 (1.175e-38 = 2- 126), then the return value is 0.0.
- If z > +88.7228 = maximum log 10 (3.402823e+38 = 2+128), then the return value is 3.402823e+38 = 2+128 (maximum single-precision floating-point number).

***Other available functions:**

- static inline float exp10sp_i(float a)
- float exp10sp(float a)

logdp_v (Double-Precision Logarithm)

Function

```
void logdp_v (double *z, double *r, int n)
```

Parameters:

- z : input double vector
- r : Resultant double vector
- n : Number of elements in the vectors

Description: The logdp_v function returns the logarithm function of a real floating-point argument z. The return value is the number logarithmic value of z. If the magnitude of z is too large, the maximum double-precision floating-point number (1.797693e+308 = 2+1024) is returned.

Special Cases:

- If | z | < 1.11e-16 = 2- 53, then the return value is 1.0 for small arguments.
- If z < -708.3964 = minimum log e (2.225e-308 = 2- 1022), then the return value is 0.0.
- If z > +709.7827 = maximum log e (1.797693e+308 = 2+1024), then the return value is 1.797693e+308 = 2+1024(maximum double-precision floating-point

number).

***Other available functions:**

- static inline double logdp_i(double a)
- double logdp(double a)

logsp_v (Single-Precision Logarithm)

Function:

```
void logsp_v (float *z, float *r, int n)
```

Parameters:

- z : input float vector logarithm
- r : Resultant float vector r= (float)x
- n : Number of elements in the vectors

Description: The logdp_v function returns the logarithm function of a real floating-point argument z. The return value is the number logarithmic value of z. If the magnitude of z is too large, the maximum double-precision floating-point number (1.797693e+308 = 2+1024) is returned.

Special Cases:

- If | z | < 9.313e-10 = 2- 30, then the return value is 1.0 for small arguments.
- If z < -87.3365 = minimum log e (1.175e-38 = 2- 126), then the return value is 0.0.
- If z > +88.7228 = maximum log e (3.402823e+38 = 2+128), then the return value is 3.402823e+38 = 2+128 (maximum single-precision floating-point number).

***Other available functions:**

- static inline float logsp_i(float a)
- float logsp(float a)

recipdp_v (Double-Precision Reciprocal)

Function

```
void recipdp_v (double *z, double *r, int n)
```

Parameters:

- z : input double vector
- r : Resultant double vector
- n : Number of elements in the vectors

Description: The recipdp_v function returns the reciprocal function of a real floating-point argument z. The return value is the number reciprocal value of z.

Special Cases:

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

***Other available functions:**

- static inline double recipdp_i(double a)
- double recipdp(double a)

recipsp_v (Single-Precision Reciprocal)

Function:

```
void recipsp_v (float *z, float *r, int n)
```

Parameters:

- z : input float vector
- r : Resultant float vector
- n : Number of elements in the vectors

Description: The recipsp_v function returns the reciprocal function of a real floating-point argument z. The return value is the number reciprocal value of z.

Special Cases:

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

***Other available functions:**

- static inline float recipsp_i(float a)
- float recipsp(float a)

rsqrtdp_v (Double-Precision Reciprocal square root)

Function

```
void rsqrtdp_v (double *z, double *r, int n)
```

Parameters:

- z : input double vector
- r : Resultant double vector
- n : Number of elements in the vectors

Description: The rsqrtdp_v function returns the reciprocal square root function of a real floating-point argument z. The return value is the number reciprocal value of z.

Special Cases:

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

***Other available functions:**

- static inline double rsqrtdp_i(double a)
- double rsqrtdp(double a)

rsqrtsp_v (Single-Precision Reciprocal square root)

Function:

```
void rsqrtsp_v (float *z, float *r, int n)
```

Parameters:

- z : input float vector
- r : Resultant float vector
- n : Number of elements in the vectors

Description: The rsqrtsp_v function returns the reciprocal square root function of a real floating-point argument z. The return value is the number reciprocal value of z.

Special Cases:

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

***Other available functions:**

- static inline float rsqrtsp_i(float a)
- float rsqrtsp(float a)

sqrtdp_v (Double-Precision Square root)

Function

```
void sqrtdp_v (double *z, double *r, int n)
```

Parameters:

- z : input double vector
- r : Resultant double vector
- n : Number of elements in the vectors

Description: The sqrtdp_v function returns the square root function of a real floating-point argument z. The return value is the number reciprocal value of z.

Special Cases:

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

***Other available functions:**

- static inline double sqrtdp_i(double a)
- double sqrtdp(double a)

sqrtsp_v (Single-Precision Square root)

Function:

```
void sqrtsp_v (float *z, float *r, int n)
```

Parameters:

- z : input float vector
- r : Resultant float vector
- n : Number of elements in the vectors

Description: The sqrtsp_v function returns the square root function of a real floating-point argument z. The return value is the number reciprocal value of z.

Special Cases:

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

***Other available functions:**

- static inline float sqrtsp_i(float a)
- float sqrtsp(float a)

powdp_v (Double-Precision Raise to a Power)

Function

```
void powdp_v (double *x,double *y, double *r, int n)
```

Parameters:

- x : input double vector(base)
- y : input double vector (power)
- r : Resultant double vector
- n : Number of elements in the vectors

Description: The powdp_v function returns the x power of y function. The return value is the number x^y.

Special Cases: The following order of tests are observed:

- If y = 0, return 1.0 (x is ignored).
- If | x | > 8.9885e+307 = 2+1023, the return value is Infinity (y is ignored).
- If | x | < 2.225e-308 = 2- 1022, then the return value is 0 (y is ignored).
- If x < 0, and y is not an integer value, then NaN is returned.
- If x < 0, and y is a 32-bit integer value, -1y* |x| y is formed.

Form W = y * log e (|x|).

- If W > 709.089 = maximum log e (+8.988e+307 = 2+1023), Infinity is returned.
- If W < -708.396 = minimum log e (+2.225e-307 = 2- 1022), then 0 is returned.

‘*Otherwise, exp e (W) = exp e (y * log e (x)) = x y is returned.

***Other available functions:**

- static inline double powdp_i(double a)
- double powdp(double a)

powsp_v (Single-Precision Raise to a Power)

Function:

```
void powsp_v (float *x, ,float *y,  float *r, int n)
```

Parameters:

- x : input float vector(base)
- y : input float vector(power)
- r : Resultant float vector
- n : Number of elements in the vectors

Description: The powsp_v function returns the x power of y function . The return value is the x power of y.

Special Cases:

- The following order of tests are observed:
- If y = 0, return 1.0 (x is ignored).
- If | x | > 1.701e+38 = 2+127, the return value is Infinity (y is ignored).
- If | x | < 1.175e-38 = 2- 126, then the return value is 0 (y is ignored).
- If x < 0, and y is not an integer value, then NaN is returned.
- If x < 0, and y is a 32-bit integer value, -1y* |x| y is formed.
- Form W = y * logf e (|x|).

If W > 88.02969 = maximum logf e (+1.701e+38 = 2+127), Infinity is returned. If W < –87.3365 = minimum logf e (+1.175e-38 = 2- 126), then 0 is returned. ‘*Otherwise, expf e (W) = expf e (y * logf e (x)) = x y is returned.

***Other available functions:**

- static inline float powsp_i(float a)
- float powsp(float a)

divdp_v (Double-Precision Division)

Function

```
void divdp_v (double *x,double *y, double *r, int n)
```

Parameters:

- x : input double vector(dividend)
- y : input double vector (divisor)
- r : Resultant double vector
- n : Number of elements in the vectors

Description: The divdp_v function returns the division function of a real floating-point argument x by y. The return value is the number resulting from x/y.

Special Cases:

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

***Other available functions:**

- static inline double divdp_i(double a, double b)
- double divdp(double a, double b)

divsp (Single-Precision Division)

Function:

```
void divsp_v (float *x, ,float *y,  float *r, int n)
```

Parameters:

- x : input float vector(dividend)
- y : input float vector(divisor)
- r : Resultant float vector r = (float)x
- n : Number of elements in the vectors

Description: The divsp_v functions return the quotient of two real floating-point arguments x and y. The return value is x / y.

Special Cases:

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

***Other available functions:**

- static inline float divsp_i(float a, float b)
- float divsp(float a, float b)

*** The _i ending after the function is the single sample version of the function that performs inlinings. The sp or dp ending is the single sample version without inlining.**

Performance

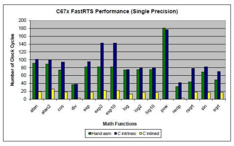
Performance benchmarks of C67X FastMath(RTS Library) vs the RTS library has been compared below

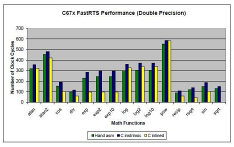
C67x FastRTS vs. RTS Benchmarking

Single Precision				
Function	RTS6740**	FastRTS	FastRTS (Intrinsics)	FastRTS (Inlining) Pipelining w/128 Calls
atan	651	91	101	19
atan2	1267	89	100	26
cos	167	74	94	18
div	540	37	38	3
exp	184	82	95	18
exp2	190	82	143	22
exp10	190	82	143	22
log	128	74	75	13
log2	153	75	79	17
log10	153	75	79	17
pow	937	181	177	82*
recip	542	32	42	3
rsqrt	165	44	78	18
sin	142	69	82	17
sqrt	161	49	70	17

Double Precision				
Function	RTS6740**	FastRTS	FastRTS (Intrinsics)	FastRTS (Inlining) Pipelining w/128 Calls
atan	1915	320	356	322
atan2	2016	453	479	421
cos	327	155	190	103
div	853	99	118	61
exp	1136	229	287	96
exp2	1083	243	297	96
exp10	1157	243	297	96
log	1871	302	359	326
log2	1887	305	370	337
log10	1891	305	370	337
pow	1570	552	587	583
recip	856	90	108	61
rsqrt	325	120	139	45
sin	309	150	188	104
sqrt	337	130	151	81*

Notes: The benchmarking was performed using the RTS, fastRTS, fastRTS w/intrinsics and fastRTS w/inlining of the different math functions. The inlining was performed to allow for pipelining in a loop iterated 128 times. The average clock cycles was then determined for the respective fastRTS (Inlining) column. The functions atan2 and pow require 2 arguments. *Special case modified to optimize. ** Average clocks measured may vary depending on the range of the data





References

[1] http://processors.wiki.ti.com/index.php/TMS320C67x_FastRTS_Library
[2] http://focus.ti.com/docs/toolsw/folders/print/sprc060.html
[3] http://www-s.ti.com/sc/techlit/sprc060.zip

Article Sources and Contributors

TMS320C67x FastRTS Library *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?oldid=73935 *Contributors*: A0272049, Momalave

Image Sources, Licenses and Contributors

Image:FastMath_performance.JPG *Source*: http://ap-fpdsp-swapps.dal.design.ti.com/index.php?title=File:FastMath_performance.JPG *License*: unknown *Contributors*: A0272049