CLA Math Library

USER'S GUIDE

Copyright

Texas Instruments
13905 University Boulevard
Sugar Land, TX 77479
http://www.ti.com/c2000

Revision Information

This is version V4.05.00.00 of this document, last updated on Jun 6, 2025.

Table of Contents

Copyright 2
Revision Information 2
1 Introduction 4
2 Other Resources 5
3 Library Structure 6
3.1 Build Options used to build the library 7
3.2 Header Files 7
4 Using the CLA Math Library 8
4.1 Library Build Configurations 9
4.2 Examples Build Configurations 11
4.3 Integrating the Library into your Project 11
4.4 Using the Tables in the CLA Data ROM 13
5 Mathematical Functions 16
5.1 Arc-Cosine 17
5.2 Arc-Cosine (F2805x Specific) 19
5.3 Arc-Sine 20
5.4 Arc-Tangent of a ratio 21
5.5 Arc-Tangent of a Ratio per Unit 23
5.6 Arc-Tangent 25
5.7 Cosine 26
5.8 Cosine Per-Unit 27
5.9 Divide 28
5.10 Exponential 29
5.11 Exponential of a Ratio 30
5.12 Exponential (Base 10) 31
5.13 Exponential (Base N) 32
5.14 Inverse Square Root 33
5.15 Natural Logarithm 34
5.16 Logarithm(Base 10) 35
5.17 Logarithm(Base N) 36
5.18 Sine 37
5.19 Sine Per-Unit 38
5.20 Sine and Cosine 39
5.21 Square Root 41
5.22 Complex FFT 256 Point 42
5.23 Complex FFT 512 Point 43
5.24 Complex FFT 1024 Point 44
6 Benchmarks 45
7 Revision History 46
IMPORTANT NOTICE 48

1 Introduction

The Texas Instruments® TMS320C28x Control Law Accelerator math library is a collection of optimized floating-point math functions for controllers with the CLA. This source code library includes several C callable assembly math functions. This revision of the library is meant to work with the CLA C compiler (codegen version v22.6.0.LTS and above). All source code is provided so it can be modified to suit the user's requirements.

Chapter 2 provides a host of resources on the CLA in general, the C compiler as well as training material.

Chapter 3 describes the directory structure of the package.

Chapter 4 provides step-by-step instructions on how to integrate the library into a project and use any of the math routines.

Chapter 5 describes each function in the library.

Chapter 6 lists The performance of each of the library routines.

Chapter 7 provides a revision history of the library.

Examples are provided with this package to show the user how to integrate the library into their projects and use any of the routines. They can be found in the *examples* directory. For the current revision, all examples have been written for the *F2805x*, *F2806x*, *F2807x*, *F28004x*, *F2837x*, *F2838x*, *F28P65x*, and *F28P55x* devices and tested on their respective *controlCard* platforms. Each example has a script **"SetupDebugEnv.js"** that can be launched from the *Scripting Console* in CCS. These scripts will set-up the watch variables for the example. In some examples graphs (.graphProp) are provided; these can be imported into CCS during debug.

2 Other Resources

There is a live Wiki page for answers to CLA frequently asked questions(FAQ). Links to other CLA references such as training videos will be posted here as well. http://processors.wiki.ti.com/index.php/Control_Law_Accelerator_(C2000_CLA)_FAQ.

The following Wiki provides details on the C compiler for the CLA (available with codegen v22.6.0.LTS and above): http://processors.wiki.ti.com/index.php/C2000_CLA_C_Compiler.

The same information may be found in the device specific **Firmware Development Package Users Guide** in C2000Ware. Please note that although the examples provided in this package were developed for the F2805x, F2806x, F2807x, F28004x, F2837x and F2838x devices, the library can be used on any device that has a CLA accelerator.

Also check out the TI Piccolo page: http://www.ti.com/piccolo

And don't forgete the TI community website: http://e2e.ti.com

Building the CLA library and examples requires **Codegen Tools v6.2.4 or later**. The library and examples in this revision were built with **Codegen Tools v22.6.0.LTS**.

3 Library Structure

Build Options used to build the library 7
Header Files 7

By default, the library and source code is installed into the following directory:

```
C:\ti\c2000\C2000Ware_X_XX_XX_XX\libraries\math\CLAmath\c28
```

Figure. 3.1 shows the directory structure while the subsequent table 3.1 provides a description for each folder.

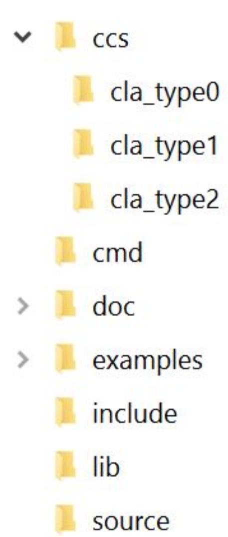

Figure 3.1: Directory Structure of the CLAMath Library

Folder	Description
<base>	
<base>/ccs	Project files for the library. Allows the user to reconfigure, modify and re-build the library to suit their particular needs.
<base>/cmd	Linker command files used in the examples.
<base>/doc	Documentation for the current revision of the library including revision history
<base>/examples	Examples that illustrate the library functions.
<base>/include	Header files for the CLAMath library
<base>/lib	Pre-built CLAMath libraries
<base>/source	Source files and project for the library. Allows the user to reconfigure, modify and re-build the library to suit their particular needs

Table 3.1: CLAMath Library Directory Structure Description

3.1 Build Options used to build the library

The cla0 math library was built with C28x Codegen Tools v22.6.0.LTS with the following options:

```
-v28 -ml -mt --cla_support=cla0 -g --diag_warning=225
```

The cla1 math library was built with C28x Codegen Tools v22.6.0.LTS with the following options:

```
-v28 -ml -mt --cla_support=cla1 -g --diag_warning=225
```

The cla2 math library was built with C28x Codegen Tools v22.6.0.LTS with the following options:

```
-v28 -ml -mt --cla_support=cla2 -g --diag_warning=225
```

The fpu32 variants of the libraries required the **–fpu_support=fpu32** option enabled.

3.2 Header Files

A library header file is supplied in the <base>/include folder. This file contains coefficient table declarations, math function mapping, assembly function prototypes and inline C functions. The function mappings map math.h functions to equivalent CLA math library functions. They also map applicable C28x, TMU, and FPU instrinsics to their equivalent CLA math function or CLA intrinsic. The inline C functions provide the same CLA math operations as the assembly functions, but are written in inline C to enable their use in the type 2 CLA background task.

4 Using the CLA Math Library

Library Build Configurations 9
Examples Build Configurations 11
Integrating the Library into your Project 11
Using the Tables in the CLA Data ROM 13

The source code and project for the CLA math library is provided. If you import the library project into CCS you will be able to view and modify the source code for all the math routines and lookup tables (see Fig. 4.1)

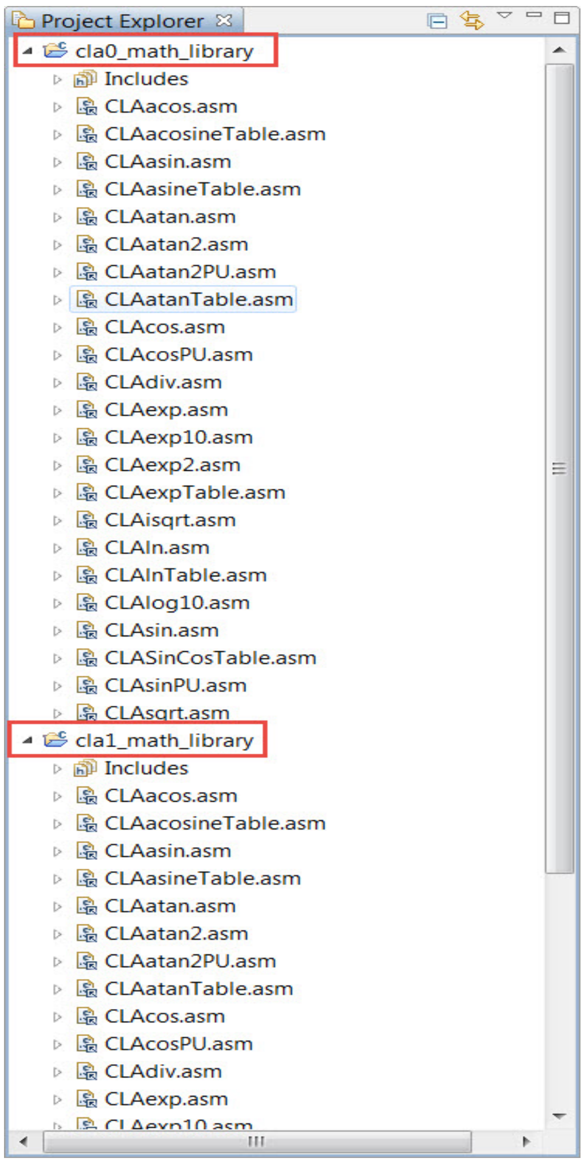

Figure 4.1: CLA Math Library Project View

4.1 Library Build Configurations

There are three libraries provided, one for the type 0 CLA, another for the type 1 CLA, and another for te type 2 CLA. Each library project has eight build configurations (Fig. 4.2)

- **CLAMATHLIB_STD** - the standard build with COFF output format
- **CLAMATHLIB_STD_EABI** - the standard build with EABI output format
- **CLAMATHLIB_FPU32_SUPPORT** - for devices with the hardware floating point unit turned on (projects that use the **–fpu_support=fpu32** option) with COFF output format
- **CLAMATHLIB_FPU32_SUPPORT_EABI** - for devices with the hardware floating point unit turned on (projects that use the **–fpu_support=fpu32** option) with EABI output format
- **CLAMATHLIB_DATAROM_STD** - for devices with the lookup tables in CLA data ROM with COFF output format
- **CLAMATHLIB_DATAROM_STD_EABI** - for devices with the lookup tables in CLA data ROM with EABI output format
- **CLAMATHLIB_DATAROM_FPU32_SUPPORT** - for devices with the lookup tables in CLA data ROM and the hardware floating point unit turned on (projects that use the **–fpu_support=fpu32** option) with COFF output format
- **CLAMATHLIB_DATAROM_FPU32_SUPPORT_EABI** - for devices with the lookup tables in CLA data ROM and the hardware floating point unit turned on (projects that use the **–fpu_support=fpu32** option) with EABI output format

Some devices, like the F2838x, F2837x, F2805x, F2807x, F28004x, F28003x, F28P55x, F28P65x have all the lookup tables in a special data ROM (CLA Data ROM) which is accessible to the CLA. The user is encouraged to use the datarom variant of the library on these devices as it frees up RAM space that would otherwise have been used to store the tables.

Each build configuration, when compiled, generates the following libraries:

NOTE: THE LIBRARY OBJECTS **cla0_math.lib, cla1_math.lib,** AND **cla2_math.lib** ARE INDEX LIBRARIES FOR THE CLA0, CLA1, AND CLA2 VERSIONS OF THE GENERATED CLA MATH LIBRARIES. THIS MEANS THAT IF ANY OF THESE OBJECTS ARE LINKED IN YOUR APPLICATION, THE LINKER USES THE INDEX LIBRARY TO CHOOSE THE APPROPRIATE VERSION OF THE LIBRARY TO USE BASED ON THE PROJECT SETTINGS.

1. **cla0_math.lib** - index library for CLA type0 (used to select appropriate library version, below)
2. **cla0_math_library.lib** - the standard build (ISA C2800)
3. **cla0_math_library_eabi.lib** - the standard build (ISA C2800)
4. **cla0_math_library_fpu32.lib** - floating point unit supported (ISA C28xFPU32)
5. **cla0_math_library_fpu32_eabi.lib** - floating point unit supported (ISA C28xFPU32)
6. **cla0_math_library_datarom.lib** - tables in CLA data ROM (ISA C2800)
7. **cla0_math_library_datarom_eabi.lib** - tables in CLA data ROM (ISA C2800)
8. **cla0_math_library_datarom_fpu32.lib** - tables in CLA data ROM and floating point unit supported (ISA C28xFPU32)
9. **cla0_math_library_datarom_fpu32_eabi.lib** - tables in CLA data ROM and floating point unit supported (ISA C28xFPU32)

1. **cla1_math.lib** - index library for CLA type1 (used to select appropriate library version, below)

2. **cla1_math_library.lib** - the standard build (ISA C2800)
3. **cla1_math_library_eabi.lib** - the standard build (ISA C2800)
4. **cla1_math_library_fpu32.lib** - floating point unit supported (ISA C28xFPU32)
5. **cla1_math_library_fpu32_eabi.lib** - floating point unit supported (ISA C28xFPU32)
6. **cla1_math_library_datarom.lib** - tables in CLA data ROM (ISA C2800)
7. **cla1_math_library_datarom_eabi.lib** - tables in CLA data ROM (ISA C2800)
8. **cla1_math_library_datarom_fpu32.lib** - tables in CLA data ROM and floating point unit supported (ISA C28xFPU32)
9. **cla1_math_library_datarom_fpu32_eabi.lib** - tables in CLA data ROM and floating point unit supported (ISA C28xFPU32)

1. **cla2_math.lib** - index library for CLA type2 (used to select appropriate library version, below)
2. **cla2_math_library.lib** - the standard build (ISA C2800)
3. **cla2_math_library_eabi.lib** - the standard build (ISA C2800)
4. **cla2_math_library_fpu32.lib** - floating point unit supported (ISA C28xFPU32)
5. **cla2_math_library_fpu32_eabi.lib** - floating point unit supported (ISA C28xFPU32)
6. **cla2_math_library_datarom.lib** - tables in CLA data ROM (ISA C2800)
7. **cla2_math_library_datarom_eabi.lib** - tables in CLA data ROM (ISA C2800)
8. **cla2_math_library_datarom_fpu32.lib** - tables in CLA data ROM and floating point unit supported (ISA C28xFPU32)
9. **cla2_math_library_datarom_fpu32_eabi.lib** - tables in CLA data ROM and floating point unit supported (ISA C28xFPU32)

NOTE: IF YOU TRY TO LINK IN THE STANDARD BUILD LIBRARY INTO A PROJECT WHICH HAS FPU32 SUPPORT TURNED ON YOU WILL GET A COMPILER ERROR ABOUT MISMATCHING INSTRUCTION SET ARCHITECTURES, HENCE THE NEED FOR THE FPU32_SUPPORT BUILD CONFIGURATIONS

NOTE: IF YOU ARE USING THE F2838X DEVICE AND ARE USING FPU64 FLOATING POINT SUPPORT YOU WILL NEED TO USE THE FPU32 VERSION OF THE CLA MATH LIBRARY OR YOU WILL GET A COMPILER ERROR ABOUT PROJECT WHICH HAS FPU32 SUPPORT TURNED ON YOU WILL GET A COMPILER ERROR ABOUT MISMATCHING MISMATCHING INSTRUCTION SET ARCHITECTURES, HENCE THE NEED FOR THE FPU32_SUPPORT BUILD CONFIGURATIONS

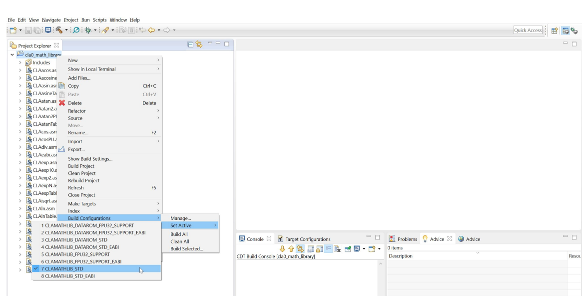

Figure 4.2: Library Build Configurations

4.2 Examples Build Configurations

Each example has multiple build configurations. Each example supports *F2805x*, *F2806x*, *F2807x*, *F28004x*, *F2837x* and *F2838x*. Additionally, each example supports **FLASH**, **RAM**, and **FLASH_NO_ROM** build configurations. For devices which have the CLA math tables in ROM, the **FLASH** and **RAM** build configurations use the CLA math tables provided in the ROM of the device and use the CLA math library build which does include the CLA math tables. The **FLASH_NO_ROM** build configuration is provided for devices which have the CLA math tables available in ROM but use the CLA math library build which includes the CLA math tables. This is provided for demonstation purposes. It is recommended to use the ROM tables if available. For devices which do not have CLA math tables in ROM, the default **FLASH** and **RAM** build configurations do not use ROM tables, but use the CLA math library build which includes the CLA math tables. See Fig. 4.3.

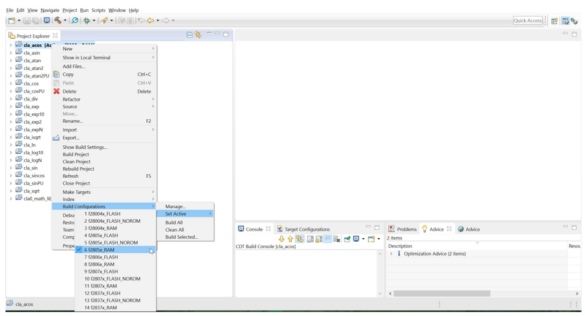

Figure 4.3: Examples Build Configurations

Note: The F2806x does not have a CLA data ROM, therefore, the datarom variant of the math library cannot be used.

4.3 Integrating the Library into your Project

To begin integrating the library into your project you need to follow these easy steps

1. Go to **Project Properties->Linked Resources** and add a new variable (see Fig. 4.4), CLA-MATH_ROOT, and point it to the root directory of the CLA Math library in C2000Ware, i.e. the c28 folder.

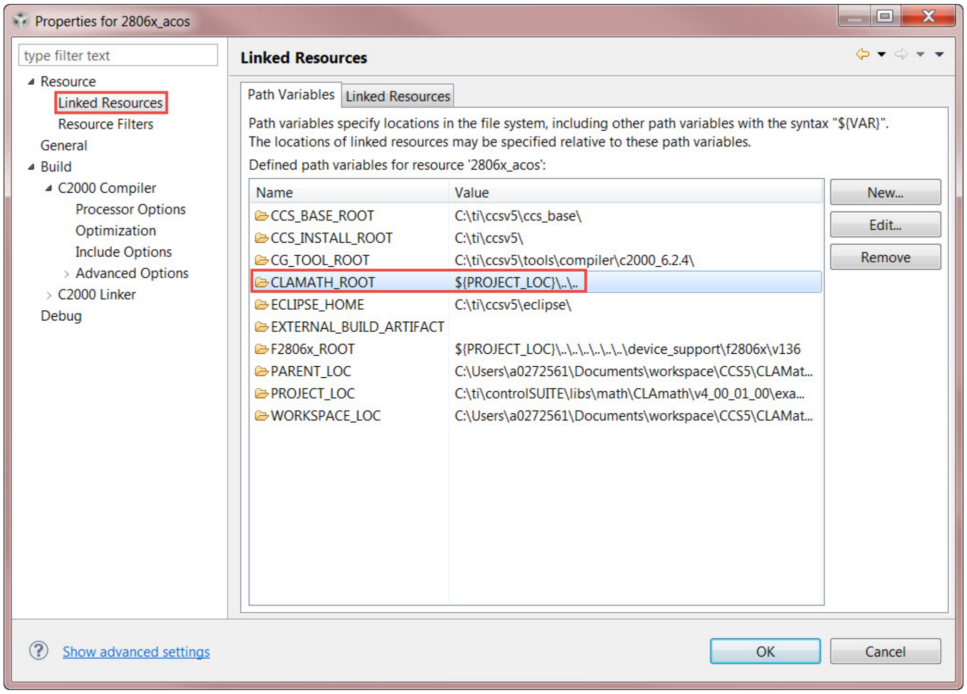

Figure 4.4: Creating a new build variable

2. Add the new path, **CLAMATH_ROOT/include**, to the *Include Options* section of the project properties (Fig. 4.5). This option tells the compiler where to find the library header files.

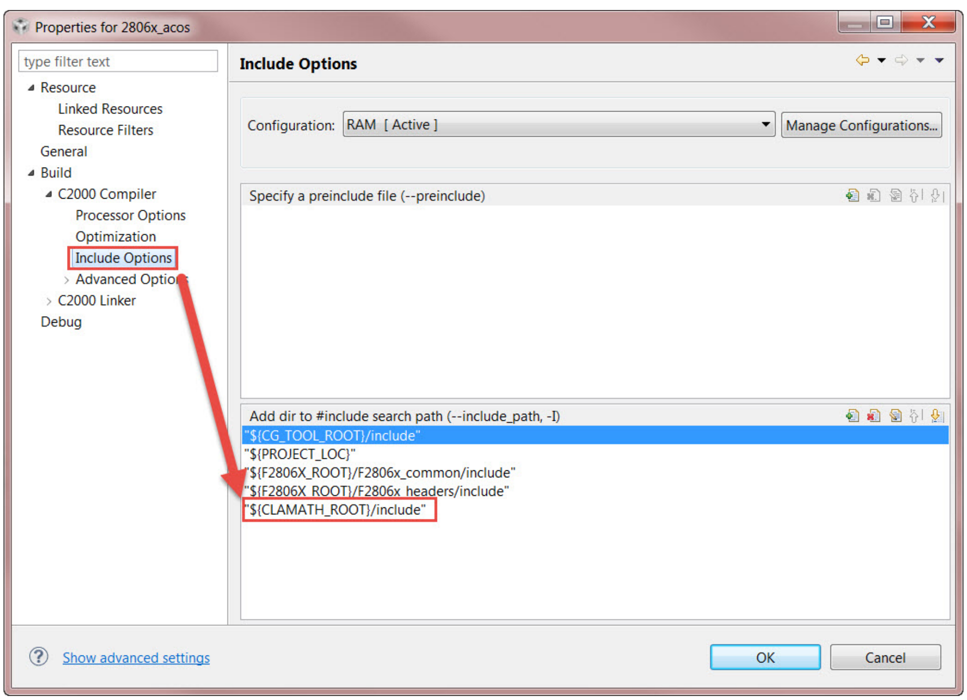

Figure 4.5: Adding the Library Header Path to the Include Options

3. Select the proper CLA type compiler support. Enable the **–cla_support** option in **Processor Options** as shown in Fig. 4.6

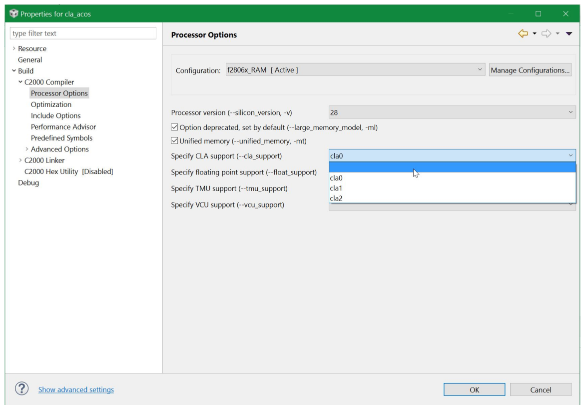

Figure 4.6: Turning on CLA support

4. Add the name of the library and its location to the **File Search Path** as shown in Fig. 4.7.

   NOTE: IF YOUR PROJECT HAS FPU32 SUPPORT TURNED ON YOU WILL NEED TO ADD THE **cla<N>_math_library_fpu32.lib** LIBRARY IN THE UPPER BOX

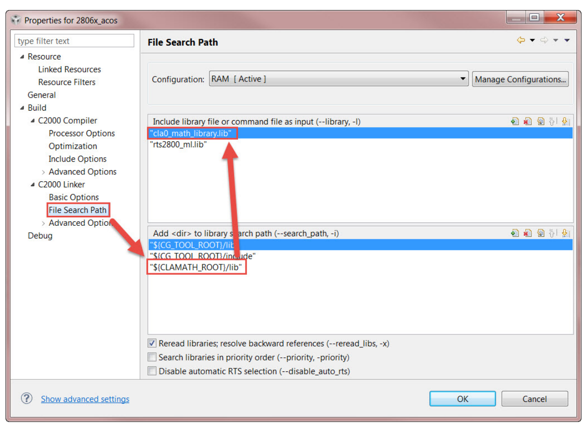

Figure 4.7: Adding the library and location to the file search path

For devices, that have the math tables in CLA data rom refer to the section section 4.4.

4.4 Using the Tables in the CLA Data ROM

The lookup tables for the CLA Math library may be present in the CLA Data ROM of the target device (check target device TRM to determine which tables have been placed in ROM), and can be used by the math routines; this will save the user from having to load the tables to Flash (and subsequently copy them over to RAM at runtime).

Devices that have the tables in ROM will have a ROM symbols table in either of two places depending on the framework under which this library is distributed,

C2000Ware:
C:/ti/c2000/C2000Ware_x_xx_xx_xx/libraries/boot_rom

Each device has its own sub-folder, with sub-folders for device revisions or release versions. The user will find a folder “rom_symbols_lib” which ultimately contain the symbols library (the .lib file).

For example, under the F2837x sub-folder (and revision folder) you will find the **2837x_c1bootROM_CLADataROMSymbols.lib**, which maps the table symbols to their physical address in device ROM.

The user should use the *datarom* variant - this configuration of the library has the tables removed from the build - of the library as shown in Fig. Figure 4.8.

NOTE: THE USER MUST ADD THE SYMBOLS LIBRARY AND ITS LOCATION TO THE FILE SEARCH PATH. THE EXAMPLE SHOWN IN THE FIGURE IS FOR CPU1 OF THE F2837X. THE USER MUST ADD THE CORRECT SYMBOLS LIBRARY FOR THE DEVICE IN QUESTION.

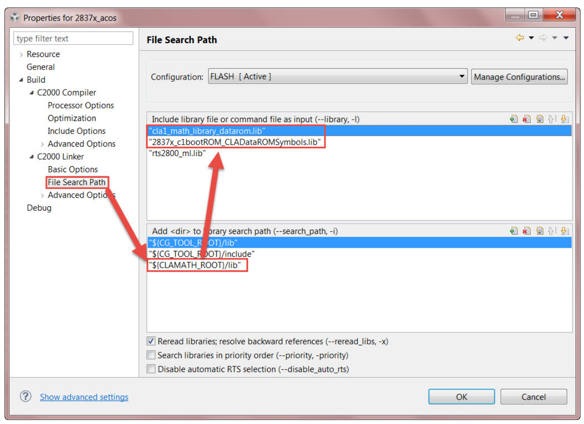

Figure 4.8: Adding the symbols library and datarom variant of the math library to the file search path

If the user does not use the *datarom* variant of the library but the standard build instead, it is then essential that the symbols library be placed higher in order than the CLA Math library and the **–priority** option box be checked.

The linker searches libraries in priority order (when the **–priority** box is checked) to find the referenced function or tables; in order for it to link to the tables in ROM, as opposed to those in the math Library, the symbols library must be placed above the math library.

The user may check the right tables are being pulled in by inspecting the *.map* file for an executable. In Figure 4.9 you see both, an example of the tables in RAM (on the left), and that of the tables in data ROM (on the right) for the F2806x; check against the addresses listed in the device TRM to ensure the correct datarom tables are being used.

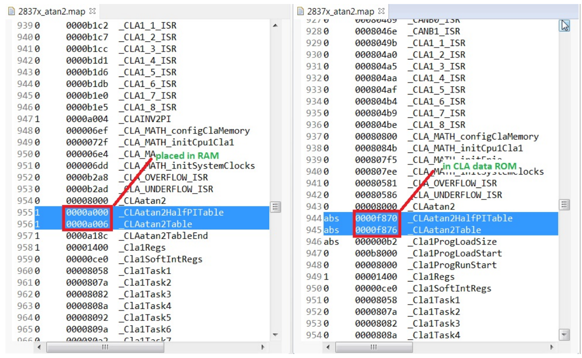

Figure 4.9: Verifying the Tables are in the CLA Data ROM

For devices which have the CLA math tables in ROM, the examples use a predefined symbol *CLA_MATH_TABLES_IN_ROM* to specify if the ROM tables are used or not. If the ROM tables are not used, then *CLA_MATH_TABLES_IN_ROM=0* and the CLA math tables are linked in the f28xxx_cla_c_lnk.cmd linker command file and they are copied to RAM using a memcpy by the C28x in the f28xxx_examples_setup.c file.

All the F2806x examples packaged with this library use the *datarom* variant of the library, and the tables in ROM. In order to use the standard build of the math library, replace the datarom variant with either the standard or fpu32 supported library builds, remove the symbols library file, and set the variable, *CLA_MATH_TABLES_IN_ROM*, in the linker command file to 0.

In the event that the symbols library for a given device (known to have the lookup tables in CLA data ROM) is not present, it is possible to add the required symbols, and their address, directly to the linker command file.

For example, if I wish to call the **CLAasin()** using the ROM lookup tables on a TMS320F28075 without the use of the symbols library, I could add the symbols of the tables (and other variables) required by the arc-sine routine directly to the linker command file as follows,

```
_CLAasinHalfPITable = 0xf9fc;
_CLAasinTable       = 0xfa00;
_CLAasinTableEnd    = 0xfb86;
```

The location of these symbols can be found out from the device TRM, usually under the chapter on the Boot ROM. You are required to use the *datarom* variant of the library for this approach to work. If the standard library (the one with the tables included) is used instead, an error message about conflicting symbols will be produced by the linker.

5 Mathematical Functions

Arc-Cosine . . . 17
Arc-Cosine (F2805x Specific) . . . 19
Arc-Sine . . . 20
Arc-Tangent of a ratio . . . 21
Arc-Tangent of a Ratio per Unit . . . 23
Arc-Tangent . . . 25
Cosine . . . 26
Cosine Per-Unit . . . 27
Divide . . . 28
Exponential . . . 29
Exponential rasied to a Ratio . . . 30
Exponential(Base 10) . . . 31
Exponential (Base N) . . . 32
Inverse Square Root . . . 33
Natural Logarithm . . . 34
Logarithm(Base 10) . . . 35
Logarithm(Base N) . . . 36
Sine . . . 37
Sine Per-Unit . . . 38
Sine and Cosine . . . 39
Square Root . . . 41
Complex FFT 256 Point . . . 42
Complex FFT 512 Point . . . 43
Complex FFT 1024 Point . . . 44

The following functions are included in this release of the CLAmath Library. The source code for these functions can be found in the *source/CLAMathLib* folder.

Trigonometric	
CLAcos	CLAsin
CLAcosPU	CLAsinePU
CLAacos	CLAasin
CLAacos_spc	CLAatan
CLAatan2	CLAatan2PU
CLAsincos	
Logarithmic	
CLAln	CLAlog10
CLAlogN	
Exponential	
CLAexp	CLAexp10
CLAexp2	CLAexpN
Miscellaneous	
CLAdiv	CLAisqrt
CLAsqrt	
Complex FFT	
CLA$_{C}FFT_{r}un256Pt$	CLA$_{C}FFT_{r}un512Pt$
CLA$_{C}FFT_{r}un1024Pt$	CLA$_{C}FFT_{u}npack256Pt$
CLA$_{C}FFT_{u}npack512Pt$	

Table 5.1: List of Functions

5.1 Arc-Cosine

Prototype:
float CLAacos(float fVal)

Parameters:
fVal Input Value ($-1 \le fVal \le 1$)

Returns:
Angle in radians ($0 \le Angle \le \pi$)

Description:
This function calculates the arc-cosine of an argument value i.e. $acos(fVal)$ or $\cos^{-1}(fVal)$, in the following manner

1. Calculate absolute of the input X
2. Use the upper 6-bits of input “X” value as an index into the table to obtain the coefficients for a second order equation
3. Calculate the angle using the following equation:

$$\begin{aligned}\cos^{-1}(Ratio) &= A0 + A1 * fVal + A2 * fVal * fVal \\ &= A0 + fVal(A1 + A2 * fVal)\end{aligned}$$

4. The final angle is determined as follows:

$$
\begin{aligned}
&if(X < 0)\\
&\quad Angle = Pi - Angle
\end{aligned}
$$

Note:

Do not use this function on a F2805x device, with the **DATAROM** variant of the CLA math library. Use the CLAacos_spc function instead.

Equation:

$\theta = cos^{-1}(fVal)$

5.2 Arc-Cosine (F2805x Specific)

Prototype:
float CLAacos_spc(float fVal)

Parameters:
fVal Input Value ($-1 \leq fVal \leq 1$)

Returns:
Angle in radians ($0 \leq Angle \leq \pi$)

Description:
This is a device specific variant of the arc-cosine function. It is meant to be used on the F2805x line of devices, if using the tables in the CLA data ROM i.e. using the **DATAROM** variants of the CLA Math library.

The arc-cosine table in the F2805x ROM has 65 triplets instead of the usual 64; this routine will skip over the first triplet and proceed with its calculations as though it were operating on a lookup table with 64 triplets. It calculates the arc-cosine of an argument value i.e. $acos(fVal)$ or $\cos^{-1}(fVal)$, in the following manner

1. Calculate absolute of the input X
2. Use the upper 6-bits of input "X" value as an index into the table to obtain the coefficients for a second order equation
3. Calculate the angle using the following equation:

$$\begin{aligned} \cos^{-1}(Ratio) &= A0 + A1 * fVal + A2 * fVal * fVal \\ &= A0 + fVal(A1 + A2 * fVal) \end{aligned}$$

4. The final angle is determined as follows:

$$\begin{aligned} &if(X < 0) \\ &\quad Angle = Pi - Angle \end{aligned}$$

Equation:
$\theta = cos^{-1}(fVal)$

5.3 Arc-Sine

Prototype:

float CLAasin(float fVal)

Parameters:

fVal Input Value ($-1 \leq fVal \leq 1$)

Returns:

Angle in radians ($-\frac{\pi}{2} \leq Angle \leq \frac{\pi}{2}$)

Description:

This function calculates the arc-sine of an argument i.e. $asin(fVal)$ or $\sin^{-1}(fVal)$ in the following manner

1. Calculate absolute of the input X
2. Use the upper 6-bits of input "X" value as an index into the table to obtain the coefficients for a second order equation
3. Calculate the angle using the following equation:

$$\begin{aligned} \sin^{-1}(Ratio) &= A0 + A1 * fVal + A2 * fVal * fVal \\ &= A0 + fVal(A1 + A2 * fVal) \end{aligned}$$

4. The final angle is determined as follows:

$$\begin{aligned} &if(X < 0) \\ &\quad Angle = -Angle \end{aligned}$$

Equation:

$\theta = \sin^{-1}(fVal)$

5.4 Arc-Tangent of a ratio

Prototype:
float CLAatan2(float fVal1, float fVal2)

Parameters:

fVal1 First Input Value (normal range of floating point values)

fVal2 Second Input Value (normal range of floating point values)

Returns:

Angle in radians ($-\pi \leq Angle \leq \pi$)

Description:

This function calculates the arc-tangent of the ratio of two input variables i.e. $atan(\frac{fVal1}{fVal2})$ or $\tan^{-1}(\frac{fVal1}{fVal2})$ in the following manner

1.
$$\begin{aligned} if(|fVal1| >= |fVal2|) & \\ Numerator &= |fVal2| \\ Denominator &= |fVal1| \\ else & \\ Numerator &= |fVal1| \\ Denominator &= |fVal2| \end{aligned}$$

2. Ratio = $\frac{Numerator}{Denominator}$
NOTE: RATIO RANGE = 0.0 TO 1.0
3. Use the upper 6-bits of the "Ratio" value as an index into the table, **CLAatan2Table**, to obtain the coefficients for a second order equation
4. Calculate the angle using the following equation:

$$\begin{aligned} \tan^{-1}(Ratio) &= A0 + A1 * Ratio + A2 * Ratio * Ratio \\ &= A0 + Ratio(A1 + A2 * Ratio) \end{aligned}$$

5. The final angle is determined as follows:

$$\begin{aligned} if(fVal1 >= 0 \; and \; fVal2 &>= 0 \; and \; |fVal1| >= |fVal2|) \\ Angle &= arctan(\frac{|fVal2|}{|fVal1|}) \\ if(fVal1 >= 0 \; and \; fVal2 &>= 0 \; and \; |fVal1| < |fVal2|) \\ Angle &= PI/2 - arctan(\frac{|fVal2|}{|fVal1|}) \\ if(fVal1 < 0 \; and \; fVal2 &>= 0 \; and \; |fVal1| < |fVal2|) \\ Angle &= PI/2 + arctan(\frac{|fVal2|}{|fVal1|}) \\ if(fVal1 < 0 \; and \; fVal2 &>= 0 \; and \; |fVal1| >= |fVal2|) \\ Angle &= PI - arctan(\frac{|fVal2|}{|fVal1|}) \\ if(fVal2 < 0) & \end{aligned}$$

$$Angle = -Angle$$

Equation:

$\theta = \tan^{-1}(\frac{fVal1}{fVal2})$

5.5 Arc-Tangent of a Ratio per Unit

Prototype:
float CLAatan2PU(float fVal1, float fVal2)

Parameters:
fVal1 First Input Value (normal range of floating point values)
fVal2 Second Input Value (normal range of floating point values)

Returns:
Angle per 2π radians $(-0.5 \le Angle \le 0.5)$

Description:
This function calculates the arc-tangent of a ratio per unit i.e. $\frac{atan(\frac{fVal1}{fVal2})}{2*\pi}$ or $\frac{\tan^{-1}(\frac{fVal1}{fVal2})}{2*\pi}$ in the following manner

1.
$$\begin{aligned} if(|fVal1| >= |fVal2|) & \\ Numerator &= |fVal2| \\ Denominator &= |fVal1| \\ else & \\ Numerator &= |fVal1| \\ Denominator &= |fVal2| \end{aligned}$$

2. $Ratio = \frac{Numerator}{Denominator}$
   NOTE: RATIO RANGE = 0.0 TO 1.0
3. Use the upper 6-bits of the "Ratio" value as an index into the table, **CLAatan2Table**, to obtain the coefficients for a second order equation
4. Calculate the angle using the following equation:
$$\begin{aligned} \tan^{-1}(Ratio) &= A0 + A1 * Ratio + A2 * Ratio * Ratio \\ &= A0 + Ratio(A1 + A2 * Ratio) \end{aligned}$$

5. The final angle is determined as follows:
$$\begin{aligned} if(fVal1 >= 0 \; and \; fVal2 &>= 0 \; and \; |fVal1| >= |fVal2|) \\ Angle &= arctan(\frac{|fVal2|}{|fVal1|}) \\ if(fVal1 >= 0 \; and \; fVal2 &>= 0 \; and \; |fVal1| < |fVal2|) \\ Angle &= PI/2 - arctan(\frac{|fVal2|}{|fVal1|}) \\ if(fVal1 < 0 \; and \; fVal2 &>= 0 \; and \; |fVal1| < |fVal2|) \\ Angle &= PI/2 + arctan(\frac{|fVal2|}{|fVal1|}) \\ if(fVal1 < 0 \; and \; fVal2 &>= 0 \; and \; |fVal1| >= |fVal2|) \\ Angle &= PI - arctan(\frac{|fVal2|}{|fVal1|}) \\ if(fVal2 < 0) & \end{aligned}$$

$$\begin{aligned} Angle &= -Angle \\ AnglePU &= \frac{Angle}{2 \times \pi} \end{aligned}$$

Equation:

$\theta_{PU} = \frac{\tan^{-1}(\frac{fVal1}{fVal2})}{2*pi}$

5.6 Arc-Tangent

Prototype:
float CLAatan(float fVal)

Parameters:
fVal Input Value (normal range of floating point values)

Returns:
Angle in radians ($-\frac{\pi}{2} \leq Angle \leq \frac{\pi}{2}$)

Description:
This function calculates the arc-tangent of the argument i.e. $atan(fVal)$ or $\tan^{-1}(fVal)$ in the following manner

1. $$\begin{aligned} if(1.0 \quad & >= \quad |fVal|) \\ Numerator &= |fVal| \\ Denominator &= 1.0 \\ else & \\ Numerator &= 1.0 \\ Denominator &= |fVal| \end{aligned}$$

2. $Ratio = \frac{Numerator}{Denominator}$
   NOTE: RATIO RANGE = 0.0 TO 1.0
3. Use the upper 6-bits of the "Ratio" value as an index into the table, **CLAatan2Table** to obtain the coefficients for a second order equation
4. Calculate the angle using the following equation:

$$\begin{aligned} \tan^{-1}(Ratio) &= A0 + A1 * Ratio + A2 * Ratio * Ratio \\ &= A0 + Ratio(A1 + A2 * Ratio) \end{aligned}$$

5. The final angle is determined as follows:

$$\begin{aligned} if(fVal >= 0 \; and \; 1.0 \quad & >= \quad abs(fVal)) \\ Angle &= \tan^{-1}(\frac{abs(fVal)}{1.0}) \\ if(fVal >= 0 \; and \; 1.0 \quad & < \quad abs(fVal)) \\ Angle &= PI/2 - \tan^{-1}(\frac{1.0}{abs(fVal)}) \\ if(fVal < 0) & \\ Angle &= -Angle \end{aligned}$$

Equation:
$\theta = \tan^{-1}(fVal)$

5.7 Cosine

Prototype:
float CLAcos(float fAngleRad)

Parameters:
fAngleRad Input angle in radians ($-2\pi \leq Angle \leq 2\pi$)

Returns:
cosine of the angle(float) ($-1 \leq Result \leq 1$)

Description:
This function calculates the cosine of an anlge i.e. $cos(rad)$, where rad is the input angle in radians and rad = K + X.

Using Taylor series expansion around the value K we get,

$$
\begin{aligned}
cos(rad) = cos(K) \quad & - \quad sin(K) \times X \\
& - \quad cos(K) \times \frac{X^2}{2!} \\
& + \quad sin(K) \times \frac{X^3}{3!} \\
& + \quad cos(K) \times \frac{X^4}{4!} \\
& - \quad sin(K) \times \frac{X^5}{5!} \\
cos(rad) = cos(K) \quad & + \quad X \times (-1.0 \times sin(K) \\
& + \quad X \times (-0.5 \times cos(K) \\
& + \quad X \times (0.166666 \times sin(K) \\
& + \quad X \times (0.04166666 \times cos(K) \\
& + \quad X \times (-0.00833333 \times sin(K)))))) \\
cos(rad) = cos(K) \quad & + \quad X \times (-sin(K) \\
& + \quad X \times (Coef0 \times cos(K) \\
& + \quad X \times (Coef1\_pos \times sin(K) \\
& + \quad X \times (Coef2 \times cos(K) \\
& + \quad X \times (Coef3\_neg \times sin(K))))))
\end{aligned}
$$

Equation:
$Y = cos(fAngleRad)$

5.8 Cosine Per-Unit

Prototype:

float CLAcosPU(float fAngleRadPU)

Parameters:

fAngleRadPU Input angle in radians(per 2π units) ($-1 \leq Angle \leq 1$)

Returns:

Cosine of the angle ($-1 \leq Result \leq 1$)

Description:

This function calculates the cosine of a per-unit angle i.e. $cos(radPU)$, where radPU is the angle in radians(per 2π units) and radPU= K + X

Therefore rad= $radPU * 2 * \pi$

Using Taylor series expansion around the value K we get,

$$
\begin{aligned}
cos(rad) = cos(K) &- sin(K) \times X \\
&- cos(K) \times \frac{X^2}{2!} \\
&+ sin(K) \times \frac{X^3}{3!} \\
&+ cos(K) \times \frac{X^4}{4!} \\
&- sin(K) \times \frac{X^5}{5!} \\
cos(rad) = cos(K) &+ X \times (-1.0 \times sin(K) \\
&+ X \times (-0.5 \times cos(K) \\
&+ X \times (0.166666 \times sin(K) \\
&+ X \times (0.04166666 \times cos(K) \\
&+ X \times (-0.00833333 \times sin(K))))) \\
cos(rad) = cos(K) &+ X \times (-sin(K) \\
&+ X \times (Coef0 \times cos(K) \\
&+ X \times (Coef1\_pos \times sin(K) \\
&+ X \times (Coef2 \times cos(K) \\
&+ X \times (Coef3\_neg \times sin(K)))))
\end{aligned}
$$

Equation:

$Y = cos(fAngleRadPU)$

5.9 Divide

Prototype:

float CLAdiv(float fNum, float fDen)

Parameters:

fNum Numerator (normal range of floating point values)

fDen Denominator (normal range of floating point values $\neq 0$)

Returns:

(float) $\frac{fNum}{fDen}$ (normal range of floating point values)

Description:

This function uses the Newton Raphson approximation to converge on the answer.

$$\begin{aligned}
Y^{'} &\approx \frac{1}{Den} \\
Y^{'} &= Y^{'} \times Den \\
Y^{''} &= Y^{'} - Y^{'} \times (2.0 - Y^{'} \times Den) \\
Y^{'''} &= Y^{''} \times Den \\
Y^{'''} &= Y^{''} - Y^{''} \times (2.0 - Y^{''} \times Den) \\
Y &= Y^{'''} \times Num
\end{aligned}$$

Equation:

$Y = \frac{fNum}{fDen}$

5.10 Exponential

Prototype:

float CLAexp(float fVal)

Parameters:

fVal Input argument (non-negative range of floating point values)

Returns:

Exponential raised to the input argument (positive range of floating point values)

Description:

This function calculates the exponential of the input argument i.e. e^x, where x is the input value. It is calculated as follows:

1. Calculate absolute of x
2. Identify the integer and mantissa of the input
3. Obtain the $e^{integer(x)}$ from the table **CLAExpTable**
4. Calculate the value of $e^{(mantissa)}$ by using the polynomial approx:

$$e^{X_m} = 1 + X_m \times (1 + X_m \times 0.5(1 + (\frac{X_m}{3}) \times (1 + \frac{X_m}{4} \times (1 + \frac{X_m}{5} \times (1 + \frac{X_m}{6} \times (1 + \frac{X_m}{7}))))))$$

5. The value of e^x is the product of results from (3) and (4)

Equation:

$Y = e^{fVal}$

5.11 Exponential of a Ratio

Prototype:

float CLAexp2(float fNum, float fDen)

Parameters:

fNum First argument (normal range of floating point values)

fDen Second argument (normal range of floating point values $\neq 0$)

Returns:

Value of the exponential raised to the ratio of the two input arguments (positive range of floating point values)

Description:

This function calculates the exponential of the ratio of two numbers i.e. $e^{\frac{A}{B}}$, where A and B are the two input arguments. These are the steps in the calculation:

1. Calculate absolute of $x = \frac{A}{B}$
2. Identify the integer and mantissa of the input
3. Obtain the $e^{integer(x)}$ from the table **CLAExpTable**
4. Calculate the value of $e^{(mantissa)}$ by using the following polynomial approx:

$$e^{X_m} = 1 + X_m \times (1 + X_m \times 0.5(1 + (\frac{X_m}{3}) \times (1 + \frac{X_m}{4} \times (1 + \frac{X_m}{5} \times (1 + \frac{X_m}{6} \times (1 + \frac{X_m}{7}))))))$$

5. The value of e^x is the product of results from (3) and (4)

Equation:

$Y = e^{\frac{fNum}{fDen}}$

5.12 Exponential (Base 10)

Prototype:
float CLAexp10(float fVal)
Parameters:
fVal Input argument (non-negative range of floating point values)
Returns:
Base 10 exponential of the input argument (positive range of floating point values)
Description:
This function calculates the base 10 exponential function of the input argument i.e. 10^x, where x is the input value. It is calculated as follows:

1. x = $|\frac{x}{log_(10)(e)}|$
2. Identify the integer and mantissa of the input
3. Obtain the $e^{integer(x)}$ from the table **CLAExpTable**
4. Calculate the value of $e^{(mantissa)}$ by using the polynomial approx:

$$e^{X_m} = 1 + X_m \times (1 + X_m \times 0.5(1 + (\frac{X_m}{3}) \times (1 + \frac{X_m}{4} \times (1 + \frac{X_m}{5} \times (1 + \frac{X_m}{6} \times (1 + \frac{X_m}{7}))))))$$

5. The value of e^x is the product of results from (3) and (4).

   It can be proven that $10^x = e^{\frac{x}{log_{10}e}}$ and since we have divided x by $log_{10}(e)$ in step (1), the result we obtain will be the desired 10^x

Equation:
$Y = 10^{fVal}$

5.13 Exponential (Base N)

Prototype:
float CLAexpN(float fVal, float N)

Parameters:
fVal Power argument (non-negative range of floating point values)

N Base argument (non-negative range of floating point values)

Returns:
Base N exponential of the first input argument (positive range of floating point values)

Description:
This function calculates the base N exponential function of the input argument i.e. N^x, where x and N are the input values. It is calculated as follows:

1. Find the natural logarithm of N, $log_e(N)$
2. Multiply by the first argument fVal (x), $x * log_e(N)$
3. Calculate $N^x = e^{x*log_e(N)}$

Equation:
$Y = N^{fVal}$

5.14 Inverse Square Root

Prototype:
float CLAisqrt(float fVal)

Parameters:
fVal Input number (positive range of floating point values)

Returns:
Inverse Square root of input argument (positive range of floating point values)

Description:
This function calculates the inverse square root of the input argument i.e. $\frac{1}{\sqrt{X}}$, where X is the input argument

This function uses the Newton Raphson approximation to converge on the answer.

$$\begin{aligned} Y^{'} &\approx \frac{1}{\sqrt{X}} \\ Y^{''} &= Y^{'} \times (1.5 - Y^{'} \times Y^{'} \times X \times 0.5) \\ Y^{'''} &= Y^{''} \times (1.5 - Y^{''} \times Y^{''} \times X \times 0.5) \\ Y &= Y^{'''} \end{aligned}$$

Equation:
$Y = \frac{1}{\sqrt{fVal}}$

5.15 Natural Logarithm

Prototype:
float CLAln(float fVal)

Parameters:
fVal Input argument (positive range of floating point values)

Returns:
Natural log of the input argument (non-negative range of floating point values)

Description:
This function calculates the natural log of the input argument i.e. $log_e(x)$, where x is the input value.

1. Calculate absolute of x
2. Identify the exponent of the input, store it float.
3. Identify the mantissa, X_m and use it to look up the polynomial coefficients in the table **CLALnTable**
4. Subtract the bias from the exponent and multiply it by Ln(2)
5. Calculate the value of $log_e(1 + mantissa)$ by using the polynomial approx: $log_e(1 + X_m) = a_0 + X_m \times (a_1 + X_m \times a_2)$
6. $Result = log_e(1 + X_m) + (Exponent - 127) \times (log_e(2))$

Equation:
$Y = log_e(fVal)$

5.16 Logarithm(Base 10)

Prototype:

float CLAlog10(float fVal)

Parameters:

fVal Input argument (positive range of floating point values)

Returns:

Base 10 log of the input argument (non-negative range of floating point values)

Description:

This function calculates the Log(base 10) of the input argument i.e. $log_{10}(x)$, where x is the input value

1. Calculate absolute of x
2. Identify the exponent of the input, store it float.
3. Identify the mantissa, X_m and use it to look up the polynomial coefficients in the table **CLALnTable**
4. Subtract the bias from the exponent and multiply it by Ln(2)
5. Calculate the value of $log_e(1 + mantissa)$ by using the polynomial approx: $log_e(1 + X_m) = a_0 + X_m \times (a_1 + X_m \times a_2)$
6. $Result = \frac{log_e(1+X_m)+(Exponent-127)\times(log_e(2))}{log_e(10)}$

Equation:

$Y = log_{10}(fVal)$

5.17 Logarithm(Base N)

Prototype:
float CLAlogN(float fVal, float N)

Parameters:
fVal Power argument (positive range of floating point values)

N Base argument (positive range of floating point values)

Returns:
Base N log of the input argument (non-negative range of floating point values)

Description:
This function calculates the Log(base N) of the input argument i.e. $log_N(x)$, where x is the input value

1. Calculate $log_e(x)$, where x is fVal
2. Calculate $log_e(N)$
3. The final result, $log_N(x) = \frac{log_e(x)}{log_e(N)}$

Equation:
$Y = log_N(fVal)$

5.18 Sine

Prototype:

float CLAsin(float fAngleRad)

Parameters:

fAngleRad Input angle in radians ($-2\pi \leq Angle \leq 2\pi$)

Returns:

Sine of the input angle ($-1 \leq Result \leq 1$)

Description:

This function calculates the sine of an input angle i.e. $sin(rad)$, where rad is the input angle in radians and rad = K + X

Using Taylor series expansion around the value K we get,

$$
\begin{aligned}
Sin(rad) = Sin(K) \quad & + \quad Cos(K) \times X \\
& - \quad Sin(K) \times \frac{X^2}{2!} \\
& - \quad Cos(K) \times \frac{X^3}{3!} \\
& + \quad Sin(K) \times \frac{X^4}{4!} \\
& + \quad Cos(K) \times \frac{X^5}{5!} \\
Sin(rad) = Sin(K) \quad & + \quad X \times (Cos(K) \\
& + \quad X \times (-0.5 \times Sin(K) \\
& + \quad X \times (-0.166666 \times Cos(K) \\
& + \quad X \times (0.04166666 \times Sin(K) \\
& + \quad X \times (0.00833333 \times Cos(K)))))) \\
Sin(rad) = Sin(K) \quad & + \quad X \times (Cos(K) \\
& + \quad X \times (Coef0 \times Sin(K) \\
& + \quad X \times (Coef1 \times Cos(K) \\
& + \quad X \times (Coef2 \times Sin(K) \\
& + \quad X \times (Coef3 \times Cos(K))))))
\end{aligned}
$$

Equation:

$Y = sin(fAngleRad)$

5.19 Sine Per-Unit

Prototype:

float CLAsinPU(float fAngleRadPU)

Parameters:

fAngleRadPU Input angle in radians(per 2π units) $(-1 \leq Angle \leq 1)$

Returns:

Sine of the angle $(-1 \leq Result \leq 1)$

Description:

This function calculates the sine of a per-unit angle i.e. $sin(radPU)$, where where radPU is the input angle in radians (per unit 2π) and radPU = K + X

Therefore rad= radPU*2*π

Using Taylor series expansion around the value K we get,

$$
\begin{aligned}
Sin(rad) = Sin(K) &+ Cos(K) \times X \\
&- Sin(K) \times \frac{X^2}{2!} \\
&- Cos(K) \times \frac{X^3}{3!} \\
&+ Sin(K) \times \frac{X^4}{4!} \\
&+ Cos(K) \times \frac{X^5}{5!} \\
Sin(rad) = Sin(K) &+ X \times (Cos(K) \\
&+ X \times (-0.5 \times Sin(K) \\
&+ X \times (-0.166666 \times Cos(K) \\
&+ X \times (0.04166666 \times Sin(K) \\
&+ X \times (0.00833333 \times Cos(K))))) \\
Sin(rad) = Sin(K) &+ X \times (Cos(K) \\
&+ X \times (Coef0 \times Sin(K) \\
&+ X \times (Coef1 \times Cos(K) \\
&+ X \times (Coef2 \times Sin(K) \\
&+ X \times (Coef3 \times Cos(K))))))
\end{aligned}
$$

Equation:

$Y = sin(fAngleRadPU)$

5.20 Sine and Cosine

Prototype:

void CLAsincos(float fAngleRad, float *y$_{s}in, float * y_{c}os)$

Parameters:

fAngleRad Input angle in radians ($-2\pi \leq Angle \leq 2\pi$)

y_sin Pointer to the sine of the angle

y_cos Pointer to the cosine of the angle

Returns:

Sine**and *Cosine*** of the input angle ($-1 \leq Result \leq 1$)

Description:

This function calculates the sine and cosine of an input angle i.e. $sin(rad)$, where rad is the input angle in radians and rad = K + X

Using Taylor series expansion around the value K we get,

$$
\begin{aligned}
Sin(rad) = Sin(K) &+ Cos(K) \times X \\
&- Sin(K) \times \frac{X^2}{2!} \\
&- Cos(K) \times \frac{X^3}{3!} \\
&+ Sin(K) \times \frac{X^4}{4!} \\
&+ Cos(K) \times \frac{X^5}{5!} \\
Sin(rad) = Sin(K) &+ X \times (Cos(K) \\
&+ X \times (-0.5 \times Sin(K) \\
&+ X \times (-0.166666 \times Cos(K) \\
&+ X \times (0.04166666 \times Sin(K) \\
&+ X \times (0.00833333 \times Cos(K)))))) \\
Sin(rad) = Sin(K) &+ X \times (Cos(K) \\
&+ X \times (Coef0 \times Sin(K) \\
&+ X \times (Coef1 \times Cos(K) \\
&+ X \times (Coef2 \times Sin(K) \\
&+ X \times (Coef3 \times Cos(K))))))
\end{aligned}
$$

$$
\begin{aligned}
Cos(rad) = Cos(K) &- Sin(K) \times X \\
&- Cos(K) \times \frac{X^2}{2!} \\
&+ Sin(K) \times \frac{X^3}{3!} \\
&+ Cos(K) \times \frac{X^4}{4!}
\end{aligned}
$$

$$
\begin{aligned}
& - \quad Sin(K) \times \frac{X^5}{5!} \\
Cos(rad) = Cos(K) & + \quad X \times (-1.0 \times Sin(K) \\
& + \quad X \times (-0.5 \times Cos(K) \\
& + \quad X \times (0.166666 \times Sin(K) \\
& + \quad X \times (0.04166666 \times Cos(K) \\
& + \quad X \times (-0.00833333 \times Sin(K)))))) \\
Cos(rad) = Cos(K) & + \quad X \times (-Sin(K) \\
& + \quad X \times (Coef0 \times Cos(K) \\
& + \quad X \times (Coef1\_pos \times Sin(K) \\
& + \quad X \times (Coef2 \times Cos(K) \\
& + \quad X \times (Coef3\_neg \times Sin(K))))))
\end{aligned}
$$

Equation:

$Y\_sin = sin(fAngleRad)$
$Y\_cos = cos(fAngleRad)$

5.21 Square Root

Prototype:

float CLAsqrt(float fVal)

Parameters:

fVal Input number (positive range of floating point values)

Returns:

Square root of input argument (positive range of floating point values)

Description:

This function calculates the square root of the input argument i.e. $\sqrt{X}$, where X is the input value

This function uses the Newton Raphson approximation to converge on the answer.

$$\begin{aligned} Y^{'} &\approx \frac{1}{\sqrt{X}} \\ Y^{''} &= Y^{'} \times (1.5 - Y^{'} \times Y^{'} \times X \times 0.5) \\ Y^{'''} &= Y^{''} \times (1.5 - Y^{''} \times Y^{''} \times X \times 0.5) \\ Y &= Y^{'''} \times X \end{aligned}$$

Equation:

$Y = \sqrt{fVal}$

5.22 Complex FFT 256 Point

Prototype:

void CLA_CFFT_run256Pt( )

Parameters:

None

Returns:

None

Description:

This function performs an in-place Complex FFT algorithm in the following manner

1. The FFT buffer must be a global(to both the C28 and CLA) and be named "IOBuffer". If the user desires to change the name, the macro IOBUFFER must be altered in the source assembly to reflect the new name and the code rebuilt.
2. The FFT buffer must be aligned to a 12-bit address, usually the starting address of one of the CLA data RAMs.
3. This is an in-place algorithm where in the FFT buffer "IOBuffer" contains the output at the end of the compute.
4. The complex data has real-first ordering i.e. the real part occupies the lower double word.
5. This function is not re-entrant as it uses global variable to store temporary values.

5.23 Complex FFT 512 Point

Prototype:

void CLA_CFFT_run512Pt( )

Parameters:

None

Returns:

None

Description:

This function performs an in-place Complex FFT algorithm in the following manner

1. The FFT buffer must be a global(to both the C28 and CLA) and be named "IOBuffer". If the user desires to change the name, the macro IOBUFFER must be altered in the source assembly to reflect the new name and the code rebuilt.
2. The FFT buffer must be aligned to a 12-bit address, usually the starting address of one of the CLA data RAMs.
3. This is an in-place algorithm where in the FFT buffer "IOBuffer" contains the output at the end of the compute.
4. The complex data has real-first ordering i.e. the real part occupies the lower double word.
5. This function is not re-entrant as it uses global variable to store temporary values.

5.24 Complex FFT 1024 Point

Prototype:
void CLA_CFFT_run1024Pt( )

Parameters:
None

Returns:
None

Description:
This function performs an in-place Complex FFT algorithm in the following manner

1. The FFT buffer must be a global(to both the C28 and CLA) and be named "IOBuffer". If the user desires to change the name, the macro IOBUFFER must be altered in the source assembly to reflect the new name and the code rebuilt.
2. The FFT buffer must be aligned to a 12-bit address, usually the starting address of one of the CLA data RAMs.
3. This is an in-place algorithm where in the FFT buffer "IOBuffer" contains the output at the end of the compute.
4. The complex data has real-first ordering i.e. the real part occupies the lower double word.
5. This function is not re-entrant as it uses global variable to store temporary values.

6 Benchmarks

All the CLA assembly instructions execute in a single cycle. The benchmark numbers were obtained by simply counting the number of instructions in each of the routines. The benchmarks include the return but not the function call. The call instruction could add between 1 to 4 cycles since the compiler, depending on the optimization level, often places some of the routine's instructions in the delay slot of the call instruction.

Type	Function	Assembly Cycles[1]	Inline C Cycles[2]
Trigonometric	CLAcos	28	23
	CLAsin	28	24
	CLAsincos	43	38
	CLAatan	41	32
	CLAatan2	44	72
	CLAatan2PU	46	77
	CLAcosPU	28	25
	CLAsinePU	28	25
	CLAacos	24	23
	CLAacos_spc	24	
	CLAasin	22	22
Logarithmic	CLAln	28	30
	CLAlog10	29	35
	CLAlogN	67	64
Exponential	CLAexp	41	30
	CLAexp10	43	31
	CLAexp2	53	31
	CLAexpN	68	55
Miscellanous	CLAdiv	13	9
	CLAisqrt	14	14
	CLAsqrt	16	14
FFT	Complex FFT 256	27323	-
	Complex FFT 512	64538	-
	Complex FFT 1024	133881	-
	Real FFT 512	37537	-
	Real FFT 1024	85012	-

Table 6.1: Benchmark for the CLA Math Library Routines.

[1] numbers include the return but not the function call

[2] functions were profiled with optimization level -O1 using CGT 18.12.1.LTS and were inlined, therefore cycle counts do not including function calls and function returns

7 Revision History

V4.05.00.00: Minor Update

- Updated cla$_{a}cosexampletosupportF28P55xdevice$

V4.04.00.00: Minor Update

- Updated cla$_{a}cosexampletosupportF28P65xdevice$

V4.03.03.00: Minor Update

- Added f28003x example support with f28003x build configurations for example projects.
- Updated compiler versions for libraries and examples to CGT 22.6.0.LTS.
- Added Complex FFT functions CLA_CFFT_run256Pt, CLA_CFFT_run512Pt, CLA_CFFT_run1024Pt and examples for each.

V4.03.02.00: Minor Update

- Update example projects to change optimization level from off to 0
- Add ceil and floor functions

V4.03.01.00: Minor Update

- Linker command file for F2838x changed to account for shared memory regions.

V4.03.00.00: Minor Update

- Added f2838x example support with f2838x build configurations for example projects.
- Updated compiler versions for libraries and examples to CGT 18.12.1.LTS.
- Updated benchmarking results in user guide for CGT 18.12.1.LTS compiler version.

V4.02.02.00: Minor Update

- Modified CLAmath.h header file function mappings. For type 2 CLA, the function mappings now default to the inline CLA math functions.
- Added function mapping for tanf to CLAmath.h header file function mappings.

V4.02.01.00: Minor Update

- Added inline C functions for each of the CLA math functions in CLAmath.h.
- Added function mappings from math.h functions to CLA math functions in CLAmath.h.
- Added function mappings for C28x, TMU, and FPU intrinsics to CLA math functions and CLA intrinsics in CLAmath.h.
- Created device agnostic examples, which support multiple devices.
- Added type 2 CLA library project.
- Added EABI build configurations for libraries.
- Updated documentation for modified examples and benchmarking of inline C functions.

V4.02.00.00: Moderate Update

- Fixed CLAatan2PU bug, where the legacy scratchpad section “CLAScratch” was being used, causing the function to fail as the linker would not find an explicit placement for the section, and would assign it to the first available memory hole - which was probably not accessible to the CLA.
- Documented the use of the boot ROM symbols library to access the lookup tables in the CLA data ROM of devices that have them.

V4.01.00.00: Moderate Update

- Refactored both library projects to use CGT 15.12.1.LTS
- Modified existing assembly source code, linker command files and examples to use the new CLA C compiler memory convention
- Added CLAsincos(), CLAexpN(), and CLAlogN() and examples for each.

V4.00.01.00: Minor Update

- Created two library projects for CLA Type 0 and Type 1
- Updated all projects (library and examples) to work with CCSv5 and CGT v6.2.4

- Fixed issue with table lookup in the acos and asin routines
- Added F2805x specific acos routine (used with datarom variant of the CLA library)
- Added FLASH and RAM build configurations for all examples
- Deleted first triplet in the acos lookup table (this was an incorrect entry). The total number of triplets is now 64
- Changed declaration of tables in the header file from pointers to arrays. This allows the user to use the tables in custom C code.
- Added examples for the F2837x which use the Type 1 CLA
- Fixed bug in the CLAdiv() and CLAsqrt() where the ZF bit kept its state across multiple calls

V4.00: Major Update

- Source library re-built with CLA **C** compiler (codegen v6.1.0)
- Math macros from the previous release were retained and modified into C-callable assembly functions

V3.00: Major Update

- Twelve optimized floating point macros performing trigonometric, exponential and logarithmic operations were added to the CLAmath library
- Added a new macro library, *CLAmathBasic*, that implements 13 simple operations like basic arithmetic, type conversion and conditional statements

V2.00: Moderate Update

Two more functions, *atan* and *atan2* added to the list of available macros

V1.00a: Minor Update

Source code has not been altered. Changes made to prepare the package for controlSUITE release and improved usability in CCSv4.

V1.00: Initial Release

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

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