C28x Control Law Accelerator

Math Library

(CLAmath)

Module User's Guide (SPRC993)

C28x Foundation Software

V2.00

December 11, 2009

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

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

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible or liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that products or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.
www.ti.com/sc/docs/stdterms.htm

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

Contents

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 CLA assembly macros of selected floating-point math functions. All source code is provided so it can be modified for your particular requirements.

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://tiexpressdsp.com/index.php/Category:Control_Law_Accelerator_Type0

Also check out the TI Piccolo page:

http://www.ti.com/piccolo

And don't forget the TI community website:

http://e2e.ti.com/

Building CLA code requires Codegen Tools V5.2 or later.

Debugging in Code Composer Studio V4:

Debugging CLA code requires CCS V4.0.2. V4.02 can be installed by going to the update manager and upgrading your CCS 4.0 install.

Debugging in Code Composer Studio V3.3:

C2000 evaluation version 3.3.83.19 or later supports CLA debug. For updating other versions of CCS, please check the Wiki and community website for updates.

3. Installing the Macro Library

3.1. Where the Files are Located (Directory Structure)

As installed, the *C28x CLAmath Library* is partitioned into a well-defined directory structure. By default, the library and source code is installed into the following directory:

C:\ti\controlSUITE\libs\math\CLAmath

Table 1 describes the contents of the main directories used by library:

Table 1. C28x CLAmath Library Directory Structure

Directory	Description
<base>	Base install directory. By default this is C:\ti\controlSUITE\libs\math\CLAmath\v100a For the rest of this document <base> will be omitted from the directory names.
<base>\doc	Documentation including the revision history from any previous release.
<base>\lib	The macro library include file. The macro library header file. Lookup tables used by the macros (assembly file).
<base>\2803x_examples	Example code for the 2803x family of devices.
<base>\2803x_examples\examples	Code Composer Studio V3.3 based examples
<base>\2803x_examples\examples_ccsv4	Code Composer Studio V4 based examples

4. Using the CLAmath Macro Library

The best place to start is to open up and run some of the example programs provided. This section will walk you through the framework the examples use. Open one of the projects and follow along.

4.1. The C28x CPU and the CLA are Friends!

Also known as: How to Use Header Files to Make Sharing Easy

In each of the examples provided, you will find a header file called CLAShared.h. This file includes the header files, variables, and constants that we want both the C28x and the CLA code to know about.

If possible, open one of these examples and follow along in the CLAShared.h file as we go through some suggested content:

- IQmath: If your project uses IQmath, then include it in CLAShared.h. Now you can define variables as _iq and the CLA will also know what your GLOBAL_Q value is.

- C28x Header files: These make accessing peripherals easy and can be used in both C and assembly code. In the header file software downloads we provide a file called DSP28x_Project.h. This will include all of the peripheral header files along with some example header files with useful definitions.

- The CLAmath header file: CLAmath_type0.h. All of the symbols used by the CLAmath library are included in this header file.

- Any variables or constants that both the C28x and CLA should know about. For example, if they will both access a variable called X, you can add:

  ```
  extern float32 X;
  ```

  The variable X will be created in the C28x C environment, but now the CLA will also know about it.

- Symbols in the CLA assembly file. These might include start/end symbols for each task. The main CPU can then use these symbols to calculate the vector address for each task.

4.2. C28x C Code

Now open the main.c file to see the system setup performed by the C28x main CPU. Notice the C28x C code includes the CLAShared.h header file discussed in the previous section.

The C28x C code is responsible for

- Declaring all the C28x and CLA shared variables
- Assigning these variables to linker sections by using the CODE_SECTION #pragma statement. This will be used by the linker file to place the variables in the proper memory block or message RAM.
- Turning on the clock to the CLA
- Initializing the CLA data and program memory

  Note as the examples are provided, the CLA data and program are loaded directly by Code Composer Studio. This is a great first step while debugging CLA code. Later if you move the load address of these sections to flash, the main CPU will need to copy them to the CLA data and program memory.
- Assigning the CLA program and data memory to the CLA module. At reset these memories belong to the C28x CPU so they can be initialized like any other memory. After initialization, they can then be assigned to the CLA for use.
- Enabling CLA interrupts

4.3. CLA Assembly Code

The CLA assembly code for each example is in the file CLA.asm. Open this file and notice the following:

- The CLAShared.h header file discussed previously is included by using the assembly directive:

```
.cdecls   C,LIST,"CLAShared.h"
```

- The CLAmath macro library is included as well. This will allow you to create instances of the macros in the CLA assembly file.

```
.include   "CLAmathLib_type0.inc"
```

- CLA code is assigned to its own assembly section. In this case the section is called CLAProg. Later we will use this section in the linker command file to indicate where in the memory map the CLA code should be loaded and where it will run.
- A symbol indicates the start of CLA program memory and a symbol indicates the start of each task. These can be used by the main CPU to calculate the contents of the vector register (MVECT) for each CLA task.
- In one or more of the tasks, a CLA macro is instanced. The input and output of each macro instance are tied to variables you will create in the C28x code and reference in the CLAShared.h header file. A single task can include one or more macros. The

macros themselves do not include the MSTOP instruction. Notice that an “MSTOP” instruction has been inserted to indicate the end of each task.
.

4.4. Math Lookup Tables

Many of the functions in the library use look-up tables to increase performance. These tables are located in the “CLAmathTables” assembly section and are available in the file called CLAsincosTable_type0.asm.

Make sure to add this file to your project if you are using the sin, cos or sincos macros.

4.5. Linker File

The linker file needs to specify the load and run memory locations for the following sections:

- CLAProg – this contains the CLA program code. The examples load this code directly into CLA program memory for debug. Later you will want to load this section into flash and copy it to CLA program SARAM just as you would any time critical section of code.

- Cla1ToCpuMsgRAM – The examples place all of the CLA to CPU message traffic in this section using the DATA_SECTION #pragma in the main C28x code. This section must be placed in the CLA to CPU message RAM.

- CpuToCla1MsgRAM – Likewise, the examples place all of the CPU to CLA message traffic in this section using the DATA_SECTION #pragma in the main C28x code. This section must be placed in the CPU to CLA message RAM.

- CLAmathTables – This table is used by the sin, cos and sincos macros. The contents of this section should be placed in the CLA data memory at runtime. The examples load this code directly into CLA data memory for debug. Later you will want to load this section into flash and copy it to CLA data SARAM just as you would any time critical data table.

5. CLAmath Macro Summary

The following functions are included in this release of the CLAmath Library. All of the macros are provided in the CLAmathLib_type0.inc file and can be modified as required for a particular application.

cos	isqrt
division	sin
atan	sincos
atan2	sqrt
	sincos

CLAcos	*Single-Precision Floating-Point COS (radians)*

Description Returns the cosine of a 32-bit floating-point argument "rad" (in radians) using table look-up and Taylor series expansion between the look-up table entries.

Header File

C28x C code:
```
#include  "CLAmathLib_type0.h"
```
CLA Code:
```
.cdecls  C,LIST,"CLAmathLib_type0.h"
```

Macro Declaration
```
CLAcos    .macro  y, rad      ; y = cos(rad)

Input:      rad = radians in 32-bit floating-point
Output:     y   = cos(rad) in 32-bit floating-point
```

Lookup Tables This function requires the CLAsincosTable located in the file called CLAsincosTable_type0.asm. By default, this table is in the assembly section named CLAmathTables. Make sure this table is loaded into the CLA data space.

CLAdiv	*Single-Precision Floating-Point Division*

Description Performs a 32-bit/32-bit = 32-bit single-precision floating point division. This function uses a Newton-Raphson algorithm.

Header File None

Macro Declaration
```
CLAdiv    .macro  Dest, Num, Den

Input:      Num = Numerator 32-bit floating-point
            Den = Denominator 32-bit floating-point
Output:     Dest= Num/Den in 32-bit floating-point
```

Lookup Tables None

CLAisqrt	*Single-Precision Floating-Point 1.0/Square Root*

Description Returns 1.0 /square root of a floating-point argument using a Newton-Raphson algorithm.

Header File None

Macro Declaration

```
CLAisqrt    .macro  y, x

Input:      x = 32-bit floating-point input
Output:     y = 1/(sqrt(x)) in 32-bit floating-point
```

Lookup Tables None

Special Cases If x = FLT_MAX or FLT_MIN, CLAisqrt will set the LUF flag.
If x = -FLT_MIN, CLAsqrt will set both the LUF and LVF flags.
If x = 0.0, CLAsqrt sets the LVF flag.
If x is negative, CLAsqrt will set LVF and return 0.0.

CLAsin	*Single-Precision Floating-Point SIN (radians)*

Description Returns the sine of a 32-bit floating-point argument “rad” (in radians) using table look-up and Taylor series expansion between the look-up table entries.

Header File

C28x C code:
```
#include  "CLAmathLib_type0.h"
```
CLA Code:
```
.cdecls  C,LIST,"CLAmathLib_type0.h"
```

Macro Declaration

```
CLAsin    .macro  y, rad      ; y = cos(rad)

Input:      rad = radians in 32-bit floating-point
Output:     y   = cos(rad) in 32-bit floating-point
```

Lookup Tables This function requires the CLAsincosTable located in the file called CLAsincosTable_type0.asm. By default, this table is in the assembly section named CLAmathTables. Make sure this table is loaded into the CLA data space.

CLAsincos — *Single-Precision Floating-Point SIN and COS (radians)*

Description Returns the sine and cosine of a 32-bit floating-point argument "rad" (in radians) using table look-up and Taylor series expansion between the look-up table entries.

Header File

C28x C code:
```
#include  "CLAmathLib_type0.h"
```
CLA Code:
```
.cdecls  C,LIST,"CLAmathLib_type0.h"
```

Macro Declaration

```
CLAcos    .macro  y1, y2, rad, temp1, temp2

Input:      rad = radians in 32-bit floating-point
Output:     y1  = sin(rad) in 32-bit floating-point
            y2  = cos(rad) in 32-bit floating-point

Temporary:  temp1, temp2.
```
Note: temp1 and temp2 locations are used for temporary storage during the function execution.

Lookup Tables This function requires the CLAsincosTable located in the file called CLAsincosTable_type0.asm. By default, this table is in the assembly section named CLAmathTables. Make sure this table is loaded into the CLA data space.

CLAsqrt — *Single-Precision Floating-Point Square Root*

Description Returns the square root of a floating-point argument X using a Newton-Raphson algorithm.

Header File None

Macro Declaration

```
CLAsqrt    .macro  y, x

Input:      x = 32-bit floating-point input
Output:     y = sqrt(x) in 32-bit floating-point
```

Lookup Tables None

Special Cases
If X = FLT_MAX or FLT_MIN CLAsqrt will set the LUF flag.
If X = -FLT_MIN CLAsqrt will set both the LUF and LVF flags.
If X = 0.0, CLAsqrt sets the LVF flag.
If X is negative, CLAsqrt will set LVF and return 0.0.

CLAatan — *Single-Precision Floating-Point ATAN (radians)*

Description

Returns the arc tangent of a floating-point argument x . The return value is an angle in the range [-π, π] radians.

Header File

C28x C code:
```
#include  "CLAmathLib_type0.h"
```
CLA Code:
```
.cdecls  C,LIST,"CLAmathLib_type0.h"
```

Macro Declaration

```
CLAatan    .macro  y,x      ; y = atan(x)

Input:      x in 32-bit floating-point
Output:     y = atan(x) in 32-bit floating-point
```

Lookup Tables

This function requires the CLAatan2Table located in the file called CLAatanTable_type0.asm. By default, this table is in the assembly section named CLAmathTables. Make sure this table is loaded into the CLA data space.

CLAatan2 — *Single-Precision Floating-Point ATAN2 (radians)*

Description

Returns the 4-quadrant arctangent of floating-point arguments y/x. The return value is an angle in the range [-π, π] radians

Header File

C28x C code:
```
#include  "CLAmathLib_type0.h"
```
CLA Code:
```
.cdecls  C,LIST,"CLAmathLib_type0.h"
```

Macro Declaration

```
CLAatan2    .macro  z,y,x      ; z = atan(y/x)

Inputs:     x in 32-bit floating-point
            y in 32-bit floating-point
Output:     z = atan2(y,x) in 32-bit floating-point
```

Lookup Tables

This function requires the CLAatan2Table located in the file called CLAatanTable_type0.asm. By default, this table is in the assembly section named CLAmathTables. Make sure this table is loaded into the CLA data space.

6. Revision History

V2.00: Moderate update

Two more functions , atan and atan2 have been added to the list of available CLA math macros .

V1.00a: Minor update

This update is a minor update that does not change any of the source code. The changes were to prepare the package to be included in ControlSUITE and improve usability in Code Composer Studio V4.

- Changed the default location for the ControlSUITE release of this package. The new release location is:

  C:\ti\controlSUITE\libs\math\CLAmath

- Added example projects ported to work with Code Composer Studio V4.

  CCS 3.3 examples can be found in the directory:
  C:\ti\controlSUITE\libs\math\CLAmath\v100a\2803x_examples\examples

  CCS 4 examples can be found in the directory:
  C:\ti\controlSUITE\libs\math\CLAmath\v100a\2803x_examples\examples_ccsv4

V1.00: Initial release