C2000™ Position Manager SinCos Library

User's Guide

Literature Number: SPRUI54
January 2016

Contents

List of Figures

List of Tables

C2000™ Position Manager SinCos Library

This user's guide describes a software library module that conditions the SinCos signals to determine the angle. The module is part of the C2000 Position Manager software library.

1 Introduction

1.1 The SinCos Transducer

The “SinCos” transducer is a high precision angle measurement device widely used in industrial motor control and motion control. The transducer delivers three differential analogue outputs: two sinusoidal signals in quadrature phase, and an index signal. There are typically a few thousand sinusoidal cycles for each mechanical revolution of the encoder shaft. The index signal appears only once per mechanical revolution and defines an absolute position of the shaft.

1.2 System Description

Industrial servo drives require highly accurate, reliable, low-latency position measurement for feedback control. Among the many types of angle transducer available today, the “SinCos” transducer delivers exceptional accuracy and robustness.

In principle, the SinCos transducer is a rotational sensor that produces a pair of differential analogue outputs, which can be used to measure angle. Both outputs are sinusoidal, and held in quadrature relationship such that as the shaft turns the signal pair describe sine and cosine functions of angle.

In conventional digital quadrature encoders, angle information is obtained by counting the edges of a pair of quadrature pulses; angular resolution being fixed by the number of pulses per mechanical revolution. In SinCos transducers, the angular measurement is obtained by a trigonometric computation using the unique relationship between the sine and cosine inputs. This allows the absolute angle within each quadrature interval to be determined, greatly increasing measurement precision.

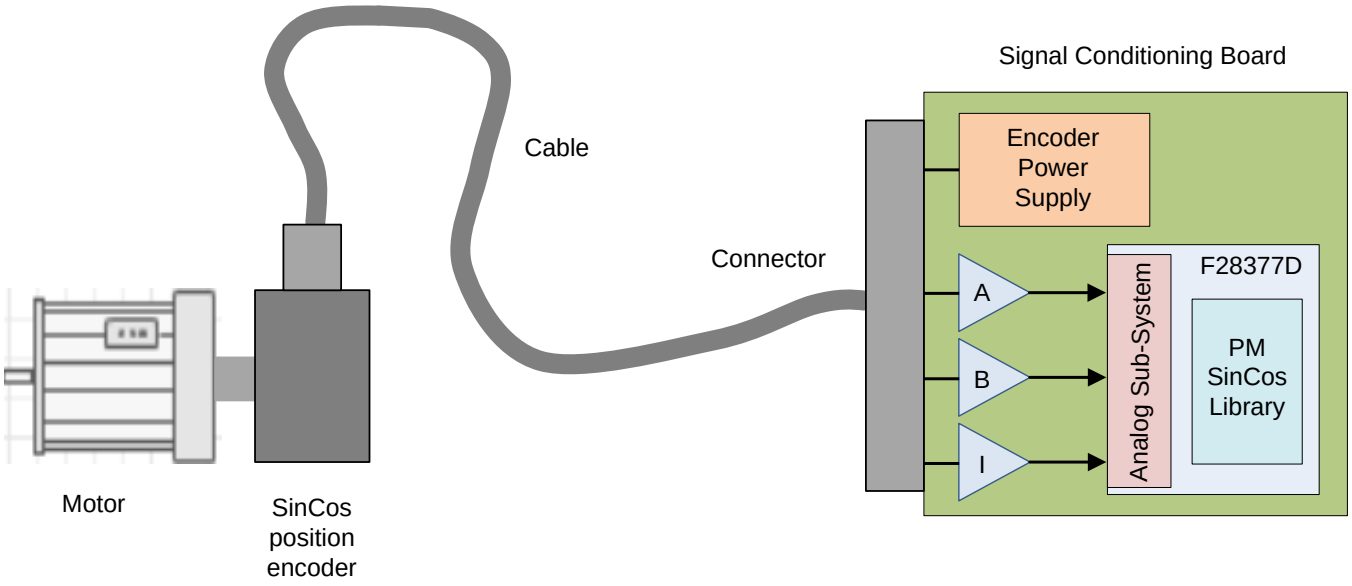

Figure 1. Industrial Servo Drive With SinCos Position Encoder Interface

It is both impractical and unnecessary to maintain high angular precision at high speed. Therefore, at high motor shaft speeds the SinCos algorithm need only count the number of complete sinusoidal revolutions to determine a lower precision angle measurement. Typically, this is done by converting the analogue signals into a pair of quadrature square waves, and counting edges in a similar way to the conventional quadrature encoder. The SinCos library does this using a pair of analogue comparators that compare each of the incoming sinusoids with an adjustable threshold representing the zero crossing point. The comparator outputs correspond to the sign of each sinusoid and the resulting digital signals are similar to those produced by a quadrature encoder. The upper plots in Figure 2 show the ideal sinusoidal inputs after passing through differential amplifiers. The lower plots show the quadrature outputs from the comparator pair.

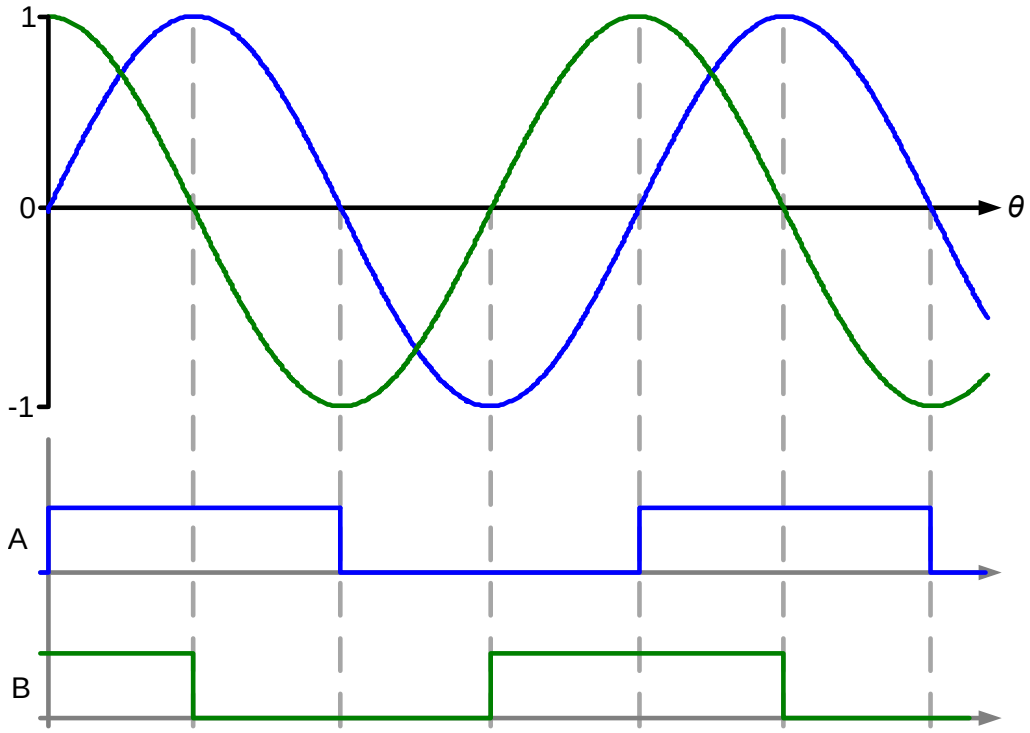

Figure 2. Principle of Operation

In addition to the sine and cosine signals, many SinCos transducers deliver an “index” pulse once each mechanical revolution. The index pulse is similar to that produced by the quadrature encoder; its’ purpose being to provide a datum so that absolute anglular position can be determined. The index pulse causes the quadrature count either to be reset to zero, or loaded with a pre-determined maximum count, depending on the direction of rotation.

1.3 SinCos Implementation Details

The internal analogue sub-system of the F2837x is ideal for interfacing to SinCos transducers. The presence of multiple ADCs which can be triggered from the same source allows simultaneous samples of both sine and cosine channels to be taken. In addition, there are up to eight pairs of analogue comparators, each with its own programmable threshold voltage. These can be used to generate digital quadrature waveforms from the sine and cosine inputs, which can be fed to one of the internal QEP (Quadrature Encoder Peripheral) modules where coarse angle and speed measurement takes place.

1.3.1 Hardware Interface and Connections

The sincos library expects three inputs signals: sine, cosine, and index. The sincos transducer typically delivers these as differential output signals, each of which must be connected to a differential amplifier to produce a single ended signal with appropriate offset and scaling such that the signal lies within range of the ADC inputs. For implementation details, see the IDDK schematics (delivered in the controlSUITE download).

Each signal enters the device on one of the AIO pins. The sine and cosine inputs must be taken to ADC inputs in such way that they may be sampled simultaneously. On the F2837x device, this is acheved by connecting them to separate physical ADCs. Note that version 1.0 of the library expects the ADCs to be configured in 12-bit mode. ADC channels must be selected so that the input signals are also connected to separate internal comparators. The index signal does not need to be sampled, so it is immaterial to which physical ADC or ADC channel it is connected, however, it must be connected to a separate internal comparator sub-system.

Figure 3 shows the interconnections on the IDDK to interface with the SinCos transducer. GPIO pin numbers indicated in the diagram correspond to the TMDXIDDK379D board

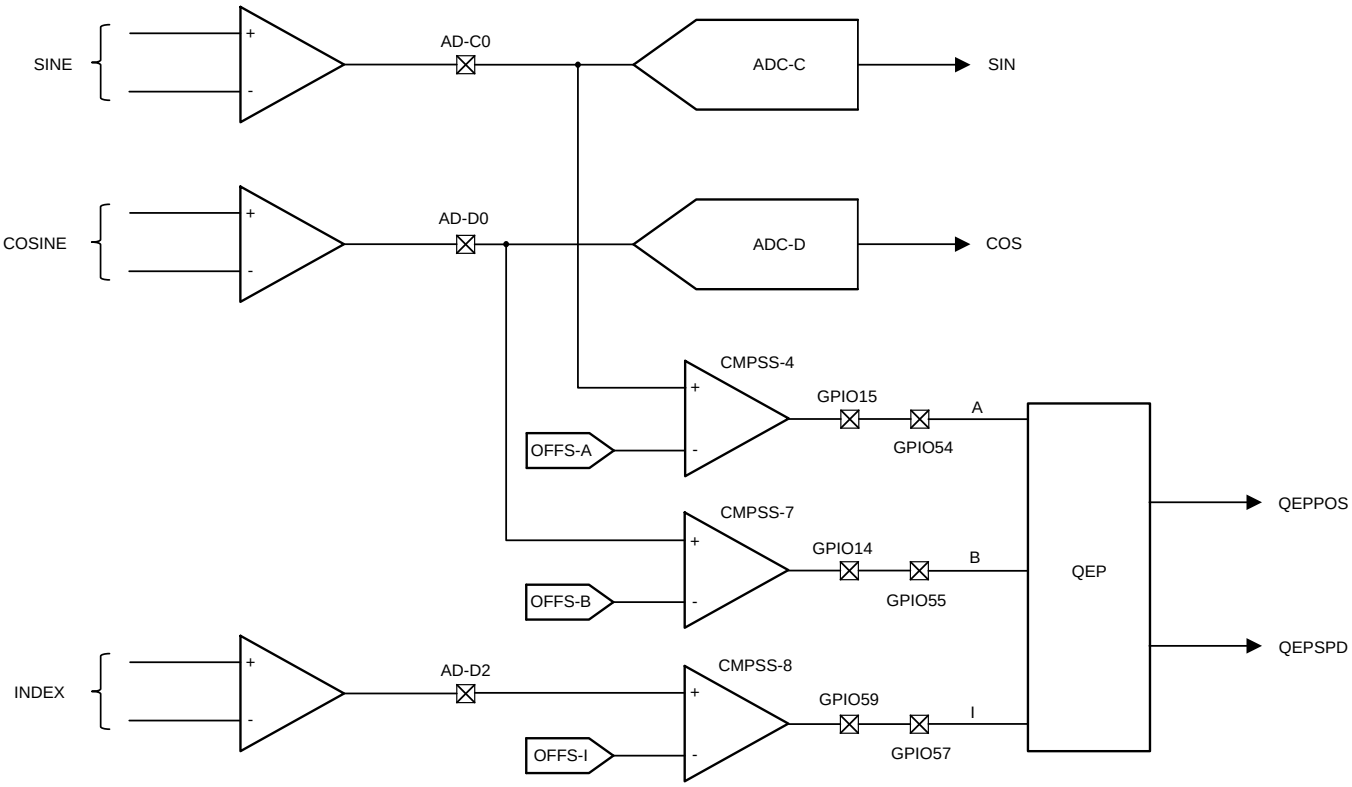

Figure 3. SinCos Implementation Diagram Using TMS320F28379D

In this example, the comparator outputs are connected via the output X-bars to separate GPIO pins. From each pin, a hardware connection is made to an input pin for one of the internal QEP peripherals. Sine, cosine, and index must be connected to QEPA, QEPB, and QEPI, respectively.

1.3.2 Software Implementation Details

Calculation of transducer shaft angle is performed in the *PM_sincos_calcAngle()* function in the SinCos library. Figure 4 and Figure 5 are equivalent software block diagrams of this function. Labels in blue indicate elements in the SinCos interface structure.

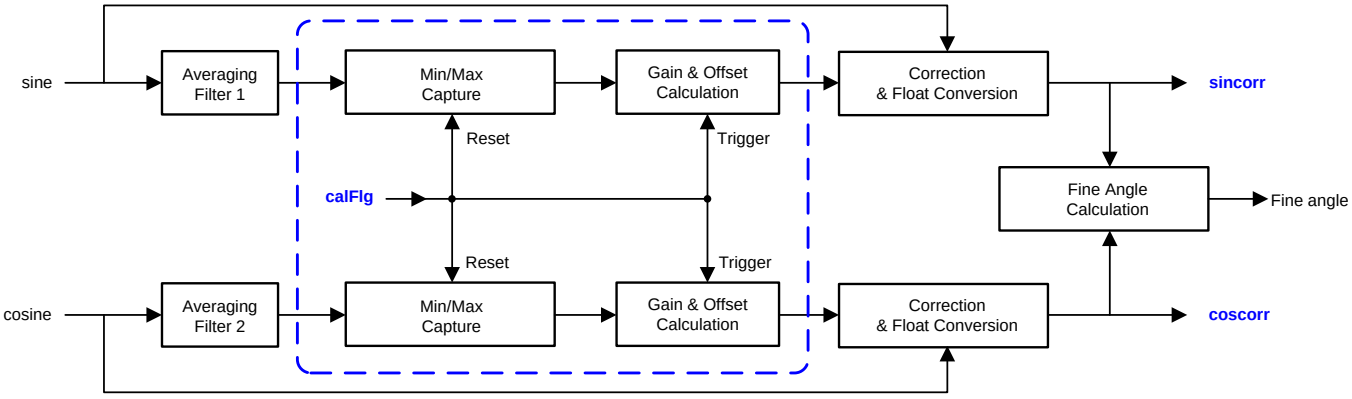

Figure 4. SinCos Fine Angle Calculation

Incoming sine and cosine data are simultaneously sampled and converted by two ADCs. Each data stream is then filtered by a four-point moving averaging filter to reduce the influence of random noise. These filtered data are compared against stored records such that maximum and minimum externals are captured over a pre-defined number of quadrature edges.

Once the shaft has moved through the required number of quadrature cycles, an internal calibration flag causes the offset and gain correction coefficients for each channel to be computed and the data records reset. On start-up (prior to availability of the first extremal set), default gain and offset values are applied. The gain and offset correction blocks apply the coefficients to each incoming data point, and it is these corrected data streams that are used for calculation of fine angle.

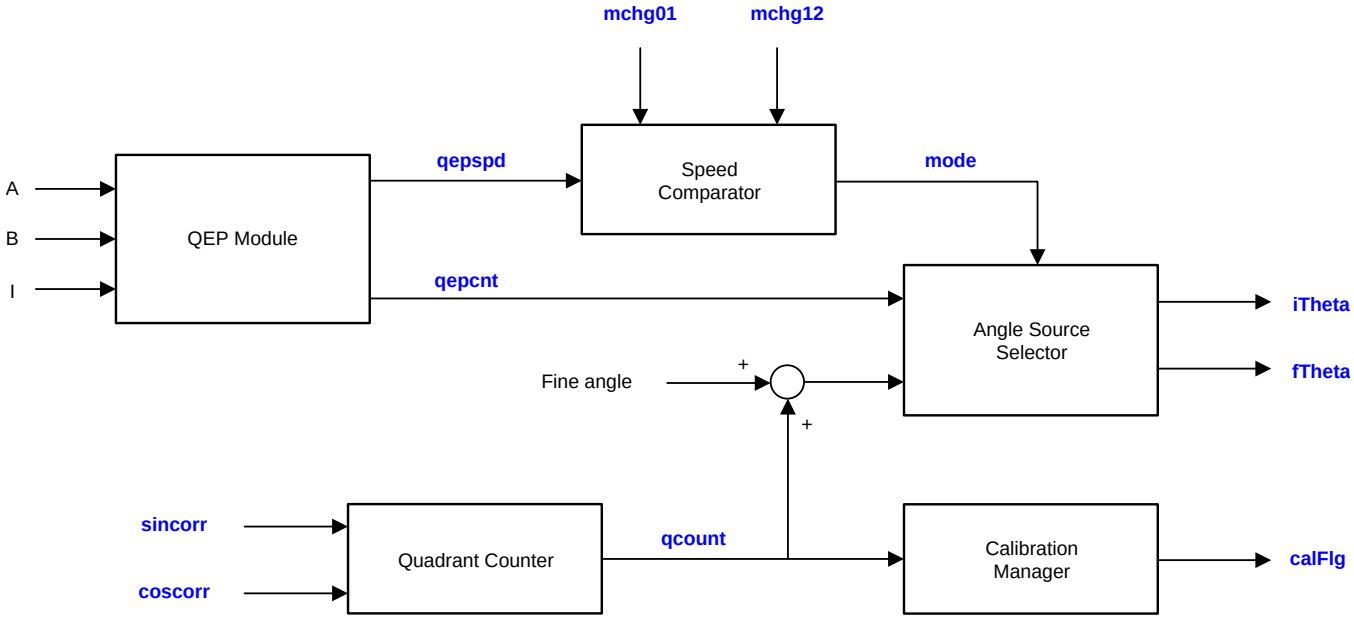

Figure 5. SinCos Quadrature Counter and Mode Control

The corrected data is also used to determine quadrature edge count at low speed. The accumulated edge count is added to the fine angle to obtain precise shaft position. The quadrature edge count is also used as a trigger for application of the calibration coefficients. A calibration manager decides when to apply the new calibration coefficients and sets “calFlg” accordingly. The user code may inspect “calFlg” to decide when to update offsets in the comparator subsystem (see Section 4.2).

Shaft speed information is provided by the QEP module. A software speed comparator determines which operating mode takes effect based on the instantaneous measured speed and two user selectable speed thresholds. The source of the angle measurement delivered by the SinCos module depends on the operating mode.

1.3.2.1 Angle Calculation

Angle information is delivered in two formats: IQ15 and floating-point. Both are available as elements in the SinCos interface structure (‘itheta’ and ‘ftheta’, respectively). In IQ15 format, the data is separated into integer and quotient (fractional) parts in the same 32-bit data word.

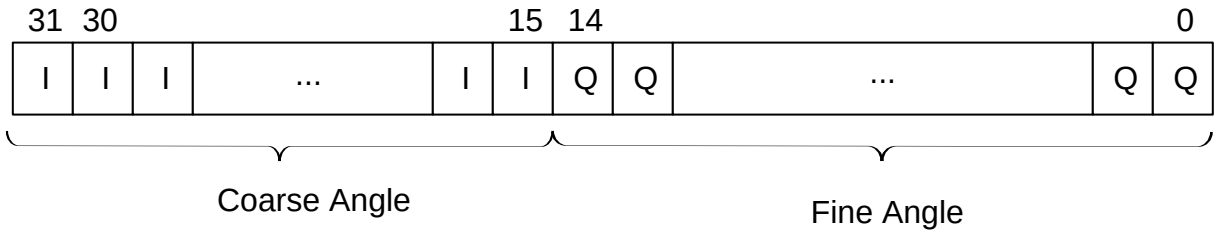

Figure 6. Angle Calculation

The integer part represents the “coarse” angle, and is simply the quadrature edge count. The count is taken either from the software quadrature edge detector (modes 0 and 1) or from the QEP position counter (mode 2).

The quotient part corresponds to “fine” angle; computed using an arctangent of the incoming sine and cosine data. The result is the interpolated per-unit angle between quadrature edges that is concatenated with the coarse angle to produce a 32-bit angular data point. The floating-point angle is obtained by an IQ15 to float conversion of the above data.

1.3.2.2 Mode Selection

The PM_sincos angle calculation software is designed to operate at a fixed rate. Typically, the computation would be called from an interrupt service routine running at around 10 – 20 kHz. The angle calculation runs in one of three modes depending on the speed of the transducer shaft.

Table 1. Mode Selection

Mode	Shaft Speed	Coarse Angle	Fine Angle	Calibration
0	Low	Software	Software	Yes
1	Medium	Software	Software	No
2	High	QEP	None	No

At low speed (modes 0 and 1), quadrature edge counting is performed in software using the corrected sine and cosine readings. Angle calculation is based entirely on measurements of the sine and cosine waveforms. The upper and lower bounds of each channel are captured over multiple electrical cycles, and from this the electrical gain and offset are computed. This gain and offset information is used to calibrate the incoming data streams. The corrected channel data are used for angle calculation and for determination of the quadrant.

As shaft speed increases, there is a point beyond which insufficient data points are available in each electrical cycle to ensure accurate calibration. Therefore, at higher speeds (modes 1 and 2), re-calibration is disabled and the coefficients are no longer updated.

At high speed (mode 2), accurate quadrant detection using the measured sine and cosine information is no longer feasible. Angular information comes entirely from the QEP which receives quadrature pulse streams and an index reference from the internal comparator sub-systems. Interpolation between quadrature edges to determine fine precision angle is not performed in this mode and the fine angle is always zero.

Speed information is always determined from the QEP, which continuously monitors the number of quadrature edges in a fixed time interval regardless of which mode is active. The transition speeds between modes are defined by two elements in the SinCos software structure (mchg01 and mchg12) and are user adjustable.

1.3.2.3 Error Detection

Version 1 of the SinCos library implements basic error detection. Errors are latched in the integer “status” element in the interface C structure. The user code may write to this element at any time to set or reset bits. The allocation of bits within the status element is as follows.

- Bit 0 QEP phase or counter error. This bit latches the setting of either the PHE or PCE bits in the QFLG register of the selected QEP module.
- Bit 1 Loss or one or both input signals. This bit is set (in mode 0 only) if the calculated range of either sine or cosine channel falls below a fixed minimum threshold of 100. This bit implies either a very low gain channel or loss of one or both input signals.
- Bit 2 Relative gain out of bounds. This bit is set (in mode 0 only) if the amplitude of the cosine channel gain relative to that of the sine channel exceeds ±5%.

Bit 3 Quadrature phase error. This bit is set (in modes 0 and 1 only) if an illegal quadrature edge transition is detected – such that, if both inputs change sign together. Changes in calibration coefficients can generate spurious transitions at higher speeds, so if this error appears it may be an indication that the mode 0-1 threshold is too high. Phase error in mode 2 is captured in bit 0.

Bit 4 Interrupt over-lap error. This bit is set if a call to the SinCos calculation routine is called before the previous call has been completed. This can be an indication of insufficient CPU bandwidth.

2 Installing the PM_sincos Library

2.1 PM_sincos Library Package Contents

The PM_sincos Library consists of the following components:

- Header files and software library for the SinCos interface
- Documentation – PM_sincos Library User Guide
- Example project showing the SinCos interface implementation on TMDXIDDK379D hardware

2.2 How to Install the PM_sincos Library

```
<base> install directory is
C:\ti\controlSUITE\libs\app_libs\position_manager\vX.X
```

The following sub-directory structure is used:

<base>\Doc	Documentation
<base>\Float	Contains implementation of the library and corresponding include file
<base>\Examples	Example using PM_sincos library

3 Module Summary

This section describes the contents of PM_sincos_lib.h – the include file for the PM_sincos library.

3.1 PM_sincos Library Functions

The PM_sincos library consists of the following functions that enable the user to interface with encoders. Table 2 lists the functions existing in the PM_sincos library and a summary of cycles taken for execution. Cycle count is measured including C function call overhead.

Table 2. PM_sincos Library Functions

Name	Description	CPU Cycles	Type
PM_sincos_calcAngle	This function computes the shaft angle based on the sine and cosine information. The data is available in the SINCOS structure in both IQ15 and floating-point format.	642	Run time
PM_sincos_updateCalData	This function updates the offset and gain corrections used in the above function with values computed in the most recent incoming data. The function is run periodically at low rate to correct for temperature drift and other variations.	666	Run time
PM_sincos_initLib	This function initializes the elements of the SINCOS structure to default values. It also resets the measurement filter buffers used for calibration.	521	Initialization time
PM_sincos_reset	This function resets those elements of the SINCOS structure used for angle calibration. It is useful for “on-the-fly” initialization; for example, if the transducer reaches a datum point and measurement should re-start from a known state.	77	Run time

3.2 Data Structures

The PM Sincos library defines the SINCOS data structure as below:

```
/* SinCos transducer struct */
typedef volatile struct {
    int initFlg;                    // [1] initialisation & first pass-through complete
    int calFlg;                     // [2] first calibration complete
    unsigned int sindata;           // [3] ch0 - raw data
    unsigned int cosdata;           // [4] ch1 - raw data
    unsigned int ch0offs;           // [5] ch0 offset as integer
    unsigned int ch1offs;           // [6] ch1 offset as integer
    _iq15 sinoffs;                  // [7] ch0 offset as IQ
    _iq15 cosoffs;                  // [8] ch1 offset as IQ
    _iq15 singain;                  // [9] ch0 gain (fixed at 1)
    _iq15 cosgain;                  // [10] ch1 gain (relative to singain)
    _iq15 sincorr;                  // [11] ch0 after static correction
    _iq15 coscorr;                  // [12] ch1 after static correction
    _iq15 qcount;                   // [13] quadrature count from angle
    _iq15 qmaxpos;                  // [14] maximum position count
    int qdir;                       // [15] rotation direction: 0 = CW, 1 = CCW.
    int qcflg;                      // [16] calibration update ready flag
    long qepcnt;                    // [17] raw QEP count
    long qepspd;                    // [18] capture counter from QEP
    int mode;                       // [19] operating mode (0 = normal, 1 = no cal., 2 = coarse
only)
long mchg01;                        // [20] speed transition between modes 0 <-> 1
    long mchg12;                    // [21] speed transition between modes 1 <-> 2
    _iq15 itheta;                   // [22] encoder angle in fixed-point
    float ftheta;                   // [23] encoder angle in float-point
    int status;                     // [24] sincos status & error code
    volatile struct EQEP_REGS *qep;  // [25] pointer to QEP register structure
} SINCOS;
```

Table 3. Module Interface Definition

Module Element	Description	Type
initFlg	Initialization status flag. Used to prevent anomalous behavior with un-initialized variables. 0 – first time execution of PM_sincos_calcAngle(). 1 – PM_sincos_calcAngle() has been called at least once.	int
calFlg	First calibration complete flag. See section... 0 – First calibration incomplete. 1 – First calibration complete.	int
ch0offs	Offset threshold for sine channel DAC in CMPSS.	unsigned int
ch1offs	Offset threshold for cosine channel DAC in CMPSS.	unsigned int
sinoffs	Offset correction to sine channel.	_iq15
cosoffs	Offset correction to cosine channel.	_iq15
singain	Gain correction to sine channel (fixed at 1.0).	_iq15
cosgain	Gain correction to cosine channel relative to sine channel.	_iq15
sincorr	Since channel data after static correction	_iq15
coscorr	Cosine channel data after static correction	_iq15
qcount	Software quadrant count in modes 1 & 2.	_iq15
qmaxpos	Maximum encoder count – “qcount” will be reset to this value on under-flow.	_iq15
qdir	Count direction flag (current measurement): 0 – CW 1 – CCW	int
qcflg	Calibration ready flag. Indicates to host software that new gain/offset calibration data is available. Host software should poll this flag and respond by calling the PM_sincos_updateCalData() function. 0 – no new calibration data.	int

Table 3. Module Interface Definition (continued)

Module Element	Description	Type
	1 – new calibration data ready.	
qepcnt	Quadrature edge counter from QEP	long
qepspd	Shaft speed indication from QEP	long
mode	Mode ID. See section... 0 – low speed with calibration 1 – low speed, calibration disabled 2 – high speed, no fine angle	int
mchg01	Speed threshold between modes 0 and 1.	int
mchg12	Speed threshold between modes 1 and 2.	int
itheta	Angle position measurement in _IQ15 format.	_iq15
ftheta	Angle position measurement in floating point format.	float
status	Error code: Bit 0 = QEP phase or counter error Bit 1 = Loss of one or both input signals Bit 2 = Relative gain out of bounds Bit 3 = Quadrature transition error	int
qep	Pointer to QEP register structure	pointer

3.3 Details of Function Usage

Detailed description of various library functions in PM_sincos library and their usage can be found in the following sections.

Table 4. Summary of Instructions

PM_sincos_calcAngle

Description This function computes the mechanical shaft angle in scaler per-unit. The angle is available in both IQ15 and floating-point format in the “itheta” and “ftheta” elements, respectively, in the SINCOS structure.

Definition

```
int PM_sincos_calcAngle(*p);
```

Parameters Input: *p: A pointer to an instance of the SINCOS structure.

Return: An integer representing the status code.

Usage

```
PM_sincos_calcAngle(&mySincos);
```

PM_sincos_updateCalData

Description Calculates the gain and offset calibration coefficients and updates active values in the SinCos structure.

This function takes the accumulated max/min values for each of the incoming data streams and computes the difference and mid-point. From this information, offset and gain corrections for each input channel are computed. Offsets are updated in “sinoffs” and “cosoffs” structure elements, and a relative gain correction is applied to the “cosgain” element. The internal max/min data records for each channel are reset at the end of the function.

Definition

```
int PM_sincos_updateCalData(SINCOS *p);
```

Parameters Input: *p: A pointer to an instance of the SINCOS structure.

Return: An integer representing the current status code.

Usage

```
PM_sincos_updateCalData(&mySincos);
```

PM_sincos_initLib

Description This function sets the elements of the specified SINCOS data structure to their default values. The function clears all data from both input filters, and initializes the selected QEP peripheral. Typically, this function would be called once, prior to using the library.

Definition

```
void PM_sincos_initLib(SINCOS *p);
```

Parameters Input: *p: A pointer to an instance of the SINCOS structure.

Return: None

Usage

```
PM_sincos_nitLib(&mySincos);
```

PM_sincos_reset

Description This function resets the dynamic SINCOS variables to default values as follows:

Name	Reset Value	Description
calFlg	0	Force new initial calibration
QPOSCNT	0x00000000	QEP hardware position counter (via qep struct element)
qepcnt	0	QEP counter
qepspd	0	Software speed
mode	0	Operating mode
qcount	0	Software quadrature count
itheta	_IQ15(0.0)	Fixed-point angle
ftheta	0.0f	Floating-point angle
status	0	SinCos status

Definition

```
void PM_sincos_reset(SINCOS *p);
```

Parameters Input: *p: A pointer to an instance of the SINCOS structure.

Return: None

Usage

```
PM_sincos_reset(&mySincos);
```

4 Using the PM_sincos Library

4.1 Adding SinCos Lib to the Project

1. Include library in {ProjectName}-Includes.h.

   Add the PM_sincos header file to your project:

   ```
   #include "PM_sincos_lib.h"
   ```

2. Add the PM_sincos library path in the include paths under Project Properties → CCS Build → C2000 Compiler → Include Options.

   Path for the library:

   C:\ti\controlSUITE\libs\app_libs\position_manager\v01_00_00_00\sincos\Float\lib

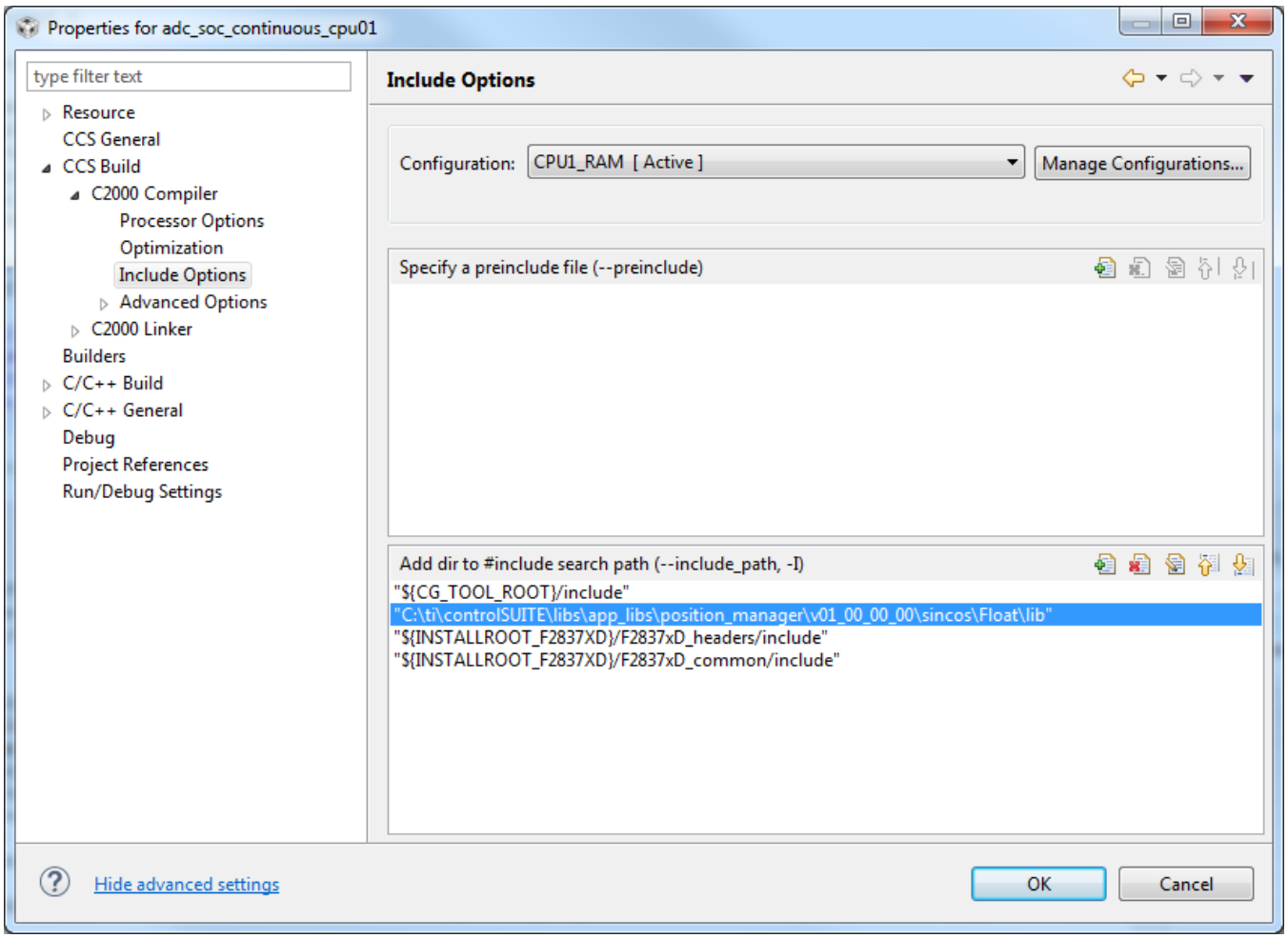

Figure 7. Compiler Options for a Project Using PM Sincos Library

NOTE: Exact location may vary depending on where controlSUITE is installed and which other libraries the project is using.

3. Link the SinCos Library (PM_sincos.lib) to the project.

   Path for the library:

   C:\ti\controlSUITE\libs\app_libs\position_manager\v01_00_00_00\sincos\Float\lib

Figure 8 is a snapshot that shows the changes to the linker options that are required to include the PM_sincos library.

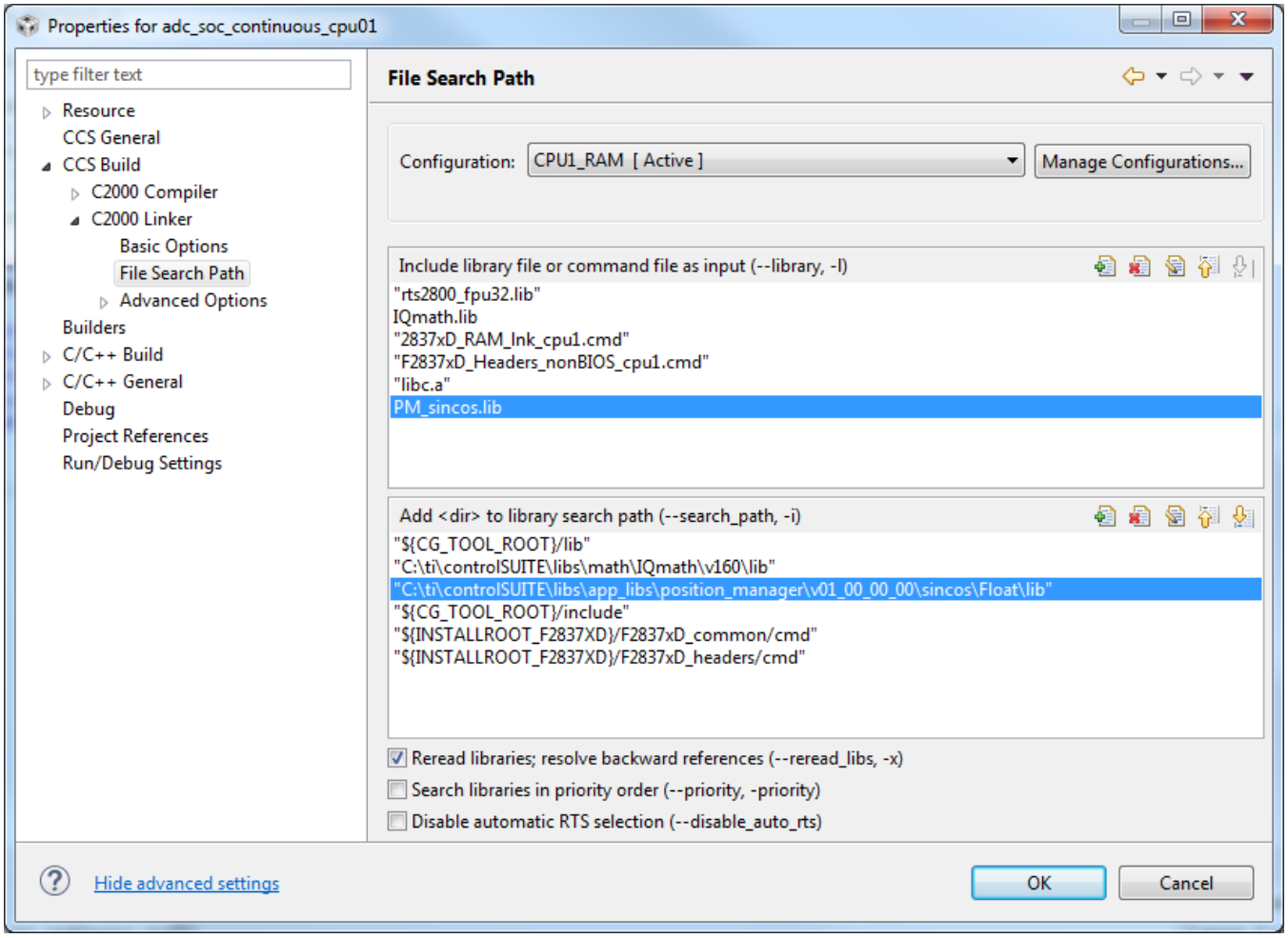

Figure 8. Adding PM_sincos Library to the Linker Options in Code Composer Studio™ (CCS) Project

NOTE: Exact location may vary depending on where controlSUITE is installed and which other libraries the project is using.

4. Include the IQ math header file to your project:

   ```
   #include "IQmathLib.h"
   ```

5. Link the IQ math library (IQmath.lib) to the project:

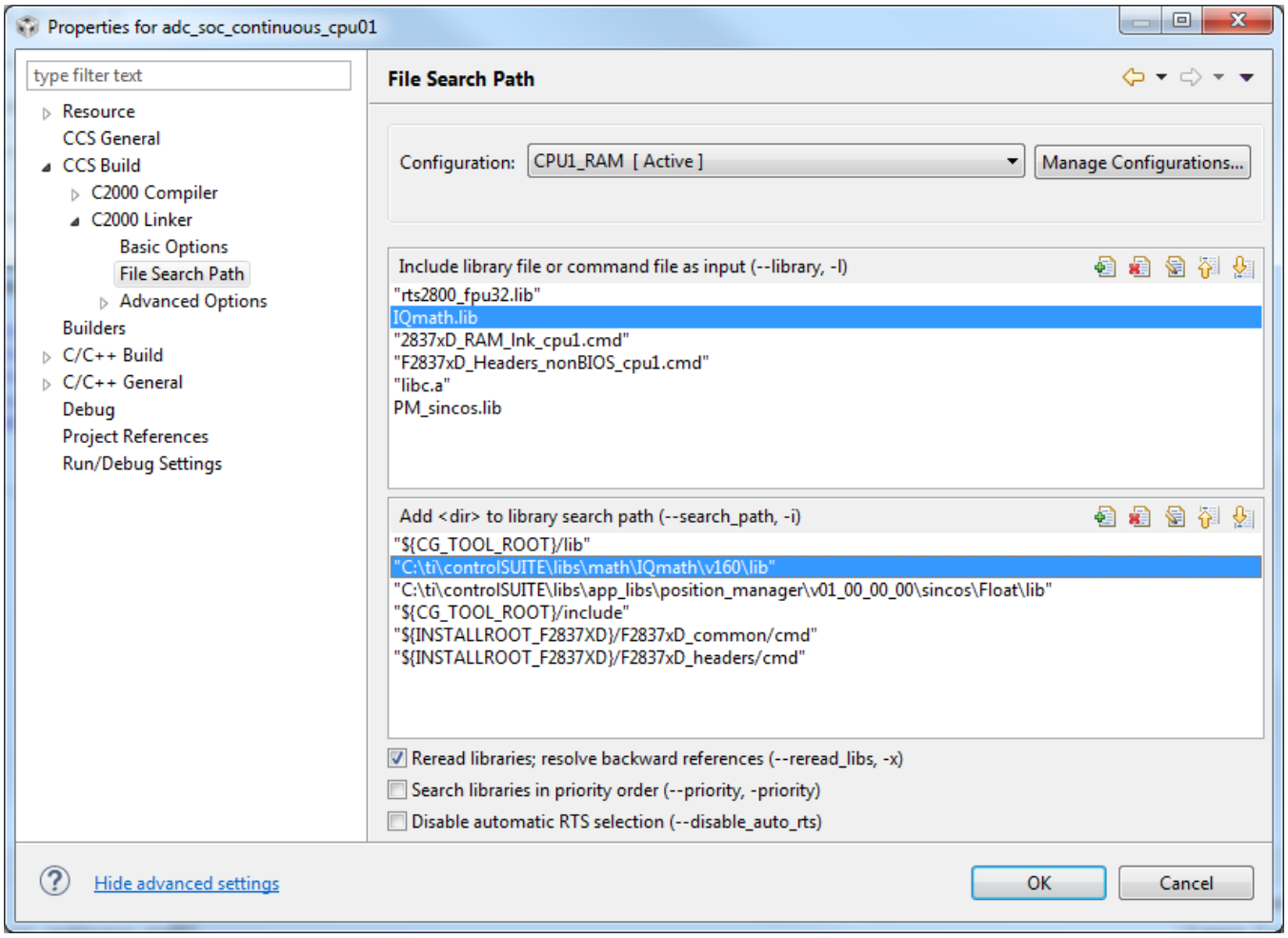

Figure 9. Adding the IQ Math Library to the Linker Options in CCS Project

4.2 Steps for Initialization

The following steps are needed for initialization and proper functioning of SinCos library functions. For more details, see the example provided along with the library.

1. Add the SinCos header file to {ProjectName}-Main.c.

```
#include "sincos.h"
```

2. Create and add module structure to {ProjectName}-Main.c for SinCos interface.

```
SINCOS mySincos;
```

3. Initialize GPIO pins. This is done in the source file “sincos.c” that can be added to the CCS project.

```
GPIO_SetupPinMux(14, GPIO_MUX_CPU1, 6);
GPIO_SetupPinMux(15, GPIO_MUX_CPU1, 6);
GPIO_SetupPinMux(59, GPIO_MUX_CPU1, 5);

// configure GPIOs 54 & 55 for QEP input
GpioCtrlRegs.GPBGMUX2.bit.GPIO54 = 1;
GpioCtrlRegs.GPBGMUX2.bit.GPIO55 = 1;
GpioCtrlRegs.GPBMUX2.bit.GPIO57 = 1;
GpioCtrlRegs.GPBMUX2.bit.GPIO54 = 1;
GpioCtrlRegs.GPBMUX2.bit.GPIO55 = 1;
GpioCtrlRegs.GPBMUX2.bit.GPIO57 = 1;

GpioCtrlRegs.GPBQSEL2.bit.GPIO54 = 2;
```

```
GpioCtrlRegs.GPBQSEL2.bit.GPIO55 = 2;
GpioCtrlRegs.GPBQSEL2.bit.GPIO56 = 2;
GpioCtrlRegs.GPBQSEL2.bit.GPIO57 = 2;
GpioCtrlRegs.GPBCTRL.bit.QUALPRD3 = 3;
```

4. Initialize ADCs as required.

```
AdcSetMode(ADC_ADCC, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
AdcSetMode(ADC_ADCD, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);

   // configure ADCC
   AdccRegs.ADCSOC0CTL.bit.CHSEL = 14;
   AdccRegs.ADCSOC0CTL.bit.TRIGSEL = 5;
   AdccRegs.ADCSOC0CTL.bit.ACQPS = ADC_AQPS - 1;

   // configure ADCD
   AdcdRegs.ADCSOC0CTL.bit.CHSEL = 0;
   AdcdRegs.ADCSOC0CTL.bit.TRIGSEL = 5;
   AdcdRegs.ADCSOC0CTL.bit.ACQPS = ADC_AQPS - 1;

   // Power up the ADCs
   AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;
   AdcdRegs.ADCCTL1.bit.ADCPWDNZ = 1;
   DELAY_US(1000);
```

5. Initialize the comparator subsystems.

```
// configure CMPSS4 for sine input
Cmpss4Regs.COMPCTL.bit.COMPDACE = 1;
Cmpss4Regs.COMPCTL.bit.COMPHSOURCE = 0;
Cmpss4Regs.COMPDACCTL.bit.SELREF = 0;
Cmpss4Regs.COMPDACCTL.bit.SWLOADSEL = 0;
Cmpss4Regs.COMPDACCTL.bit.FREESOFT = 3;
Cmpss4Regs.DACHVALS.bit.DACVAL = 2048;
Cmpss4Regs.COMPHYSCTL.bit.COMPHYS = 0;

// Configure Digital Filter
Cmpss4Regs.CTRIPHFILCLKCTL.bit.CLKPRESCALE = 0xF;
Cmpss4Regs.CTRIPHFILCTL.bit.SAMPWIN = 0x8;
Cmpss4Regs.CTRIPHFILCTL.bit.THRESH = 0x5;
Cmpss4Regs.CTRIPHFILCTL.bit.FILINIT = 1;
Cmpss4Regs.COMPCTL.bit.CTRIPHSEL = 2;
Cmpss4Regs.COMPCTL.bit.CTRIPOUTHSEL = 2;

OutputXbarRegs.OUTPUT3MUX0TO15CFG.bit.MUX6 = 0;
OutputXbarRegs.OUTPUT3MUXENABLE.bit.MUX6 = 1;

// configure CMPSS7 for cosine input
Cmpss7Regs.COMPCTL.bit.COMPDACE = 1;
Cmpss7Regs.COMPCTL.bit.COMPHSOURCE = 0;
Cmpss7Regs.COMPDACCTL.bit.SELREF = 0;
Cmpss7Regs.COMPDACCTL.bit.SWLOADSEL = 0;
Cmpss7Regs.COMPDACCTL.bit.FREESOFT = 3;
Cmpss7Regs.DACHVALS.bit.DACVAL = 2048;
Cmpss7Regs.COMPHYSCTL.bit.COMPHYS = 0;

// Configure Digital Filter
Cmpss7Regs.CTRIPHFILCLKCTL.bit.CLKPRESCALE = 0xF;
Cmpss7Regs.CTRIPHFILCTL.bit.SAMPWIN = 0x8;
Cmpss7Regs.CTRIPHFILCTL.bit.THRESH = 0x5;
Cmpss7Regs.CTRIPHFILCTL.bit.FILINIT = 1;
Cmpss7Regs.COMPCTL.bit.CTRIPHSEL = 2;
Cmpss7Regs.COMPCTL.bit.CTRIPOUTHSEL = 2;

OutputXbarRegs.OUTPUT4MUX0TO15CFG.bit.MUX12 = 0;
OutputXbarRegs.OUTPUT4MUXENABLE.bit.MUX12 = 1;

// configure CMPSS 8 for index pulse input
```

```
Cmpss8Regs.COMPCTL.bit.COMPDACE = 1;
Cmpss8Regs.COMPCTL.bit.COMPHSOURCE = 0;
Cmpss8Regs.COMPDACCTL.bit.SELREF = 0;
Cmpss8Regs.DACHVALS.bit.DACVAL = 4000;
Cmpss8Regs.COMPHYSCTL.bit.COMPHYS = 2;
Cmpss8Regs.COMPCTL.bit.CTRIPHSEL = 0;
Cmpss8Regs.COMPCTL.bit.CTRIPOUTHSEL = 0;

OutputXbarRegs.OUTPUT2MUX0TO15CFG.bit.MUX14 = 0;
OutputXbarRegs.OUTPUT2MUXENABLE.bit.MUX14 = 1;
```

6. Assign the QEP structure pointer to a physical module.

```
mySincos.qep = &EQep2Regs;
```

7. Initialize the SINCOS structure in the main program code.

```
PM_sincos_initLib(&mySincos);
```

8. Add a call to the SinCos calculation routine to the desired interrupt service routine.

```
PM_sincos_calcAngle(&mySincos);
```

   The ADC results must be read and stored in the “cosdata” and “sindata” structure elements before the call is made. For details on how this is typically done, see the example program.

9. Add code to modify the comparator offsets following calibration update. The availability of updated calibration data is indicated using the “qcflg” structure element. An example is shown below.

```
if (mySincos.qcflg == 1)
   {
        Cmpss4Regs.DACHVALS.bit.DACVAL = mySincos.ch0offs;
        Cmpss7Regs.DACHVALS.bit.DACVAL = mySincos.ch1offs;
        mySincos.qcflg = 0;
   };
```

4.3 Resource Requirements

The following resources of the F2837xD MCU are consumed by PM_sincos library (see Figure 3).

Table 5. F2837xD MCU Resources

Resource Name	Type	Purpose	Usage Restrictions
Dedicated Resources			
AD-C0	AIO	Sine channel input	AIO dedicated for SinCos
AD-D0	AIO	Cosine channel input	AIO dedicated for SinCos
AD-D2	AIO	Index input	AIO dedicated for SinCos
GPIO15	IO	CMPSS-4 high output	IO dedicated for SinCos
GPIO14	IO	CMPSS-7 high output	IO dedicated for SinCos
GPIO59	IO	CMPSS-8 high output	IO dedicated for SinCos
Configurable Resources			
QEP	Module and IOs	One QEP instance to support high speed mode	Any QEP instance can be chosen. The module and three corresponding IOs will then be dedicated for SinCos
Shared Resources			
CPU and Memory	Module	Check CPU and Memory utilization for various functions	Application to ensure enough CPU cycles and memory are allocated

5 Test Summary

The SinCos library was tested at Texas Instruments laboratories using the following hardware:

- Industrial Drive Development Kit; TMDXIDDK379D
- Lika encoder, model; HS58S18/17-P9-RM2
- Lika encoder, model; CB59-V-2048/11P12

5.1 Accuracy Assessment

Comparative angular measurements were made using the HS58S combined SinCos/BiSS encoder. Figure 10 shows the typical angular difference between these two encoder types over one full mechanical revolution. The horizontal axis is shaft angle in degrees; the vertical axis is angular error in degrees.

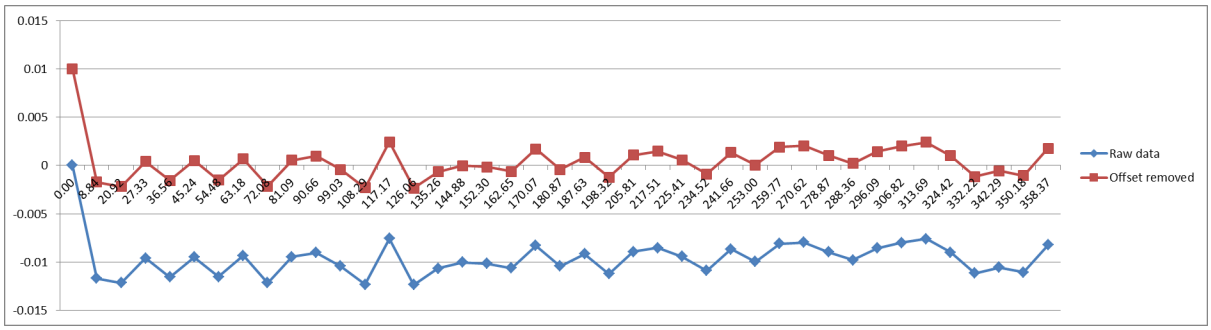

Figure 10. Typical Angular Difference

The initial error of approximately 0.01° is attributable to the default calibration values used on start-up. Measurements from the first data point at approximately 8.84° and thereafter were made after the calibration coefficients had been automatically updated (see Section 1.3.2). Maximum observed angular error was 0.00244°

5.2 Noise Assessment

Figure 11 shows the measured angle using the SinCos library for a fixed shaft position. Measurements were made at a fixed frequency of 16 kHz, and without motor control. The cable length was approximately 1m. The horizontal axis is time in milliseconds; the vertical axis is measured angle in degrees. Measurement variance was 3.056 x 10-9°.

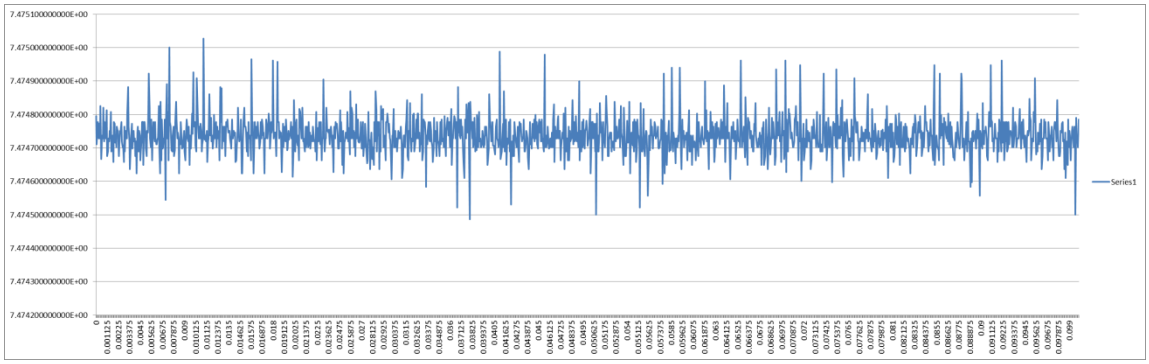

Figure 11. Measured Angle

6 FAQ

Question: Does TI share the source for the PM_sincos library to customers?

Answer: TI does not share the SinCos library source code with customers. For any specific requests, contact your TI sales contact.

Question: Does TI provide application level interface functions for SinCos?

Answer: Basic usage examples are provided along with the library. Any additional application layer functionality should be developed by users using the basic driver interface functions provided in the library.

Question: How can I get technical support for PM_sincos library?

Answer: Contact the local sales team or you can post questions on the C2000 e2e forum, which is located at: http://e2e.ti.com/support/microcontrollers/c2000/.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

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

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 relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license 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 significant portions 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. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

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

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

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