Motion Engineering, Inc.
33 South La Patera Lane
Santa Barbara, CA 93117-3214
ph (805) 681-3300
fax (805) 681-3311
tech@motioneng.com
www.motioneng.com

Release Note

DSP Series
Sinusoidal Commutation v2.0b4

Option H001-0022 **Motion Library Version 2.5.05** **Dec 18 98**
Firmware Version 2.40, Rev G4, Opt 1

Changes from Previous Release v2.0b3

Removed I/O monitoring (Contact MEI if your application requires it)

Added Bidirectional limit switch checking option in firmware (with no command velocity requirements). Refer to Section 5.2.4 for implementation details.

Default Configuration

- Standard hardware limit checking (with command velocity requirement in direction of active limit)
- Backwards-compatible with previous SinComm Release v2.0b3, Firmware 2.40, Rev G3, Opt 1

1.0 Introduction

This document provides an overview of concepts for external sinusoidal commutation, and detailed implementation instructions for DSP Series motion controllers.

The latest DSP Series hardware revisions support sinusoidal commutation as an option. DAC outputs from 2 axes are used to provide 'A' and 'B' phase sinusoidal signals that are phase-shifted by 120 degrees. The third commutation signal is generated by balance loops in the power stage of the servo amplifier. Thus, each externally commutated motor requires 2 controller axes. For example, an 8-axis card can drive up to 4 axes of sinusoidally commutated motion. In addition, the firmware also supports mixed commutated and non-commutated axes on the same controller card.

2.0 Overview of Sinusoidal Commutation

A brief review of commutation basics will be helpful before discussing external sinusoidal commutation. Motors rotate due to the torque produced by 2 interacting magnetic fields. The resultant torque is proportional to the magnitudes of the rotor and stator fields, times the sine of the angle between the 2 vectors. Consequently, maximum torque is produced when the stator and rotor angles are at 90 degrees. Maintaining this stator phase advance is a goal of the sinusoidal commutation scheme (also known as "vector control'). In the case of the permanent magnet brushless motor, the rotor has permanent magnets which provide a fixed magnetic field. The external stator windings are switched to maintain a revolving magnetic field. Ideally, the 3 phases are not switched, but controlled in a way to provide a uniformly revolving stator magnetic vector.

The process of rotating the magnetic field as a function of rotor position is called *commutation*. Historically, there have been 2 types of commutation schemes:

- the motor performs the commutation itself (self-commutating)
- the motor amplifier performs the commutation (used in most brushless motor applications) Traditionally, brushless servo motors use Hall sensors to provide coarse detection of the angular orientation of the rotor.

These 2 commutation techniques work well in many applications. However, systems that need *low torque ripple* and *excellent velocity regulation at low speeds* require external sinusoidal commutation.

External sinusoidal commutation optimally energizes the motor windings to obtain peak motor performance. When a high-resolution encoder is used for feedback, the controller can precisely energize each motor winding in a sine type relationship. The result is a motor with *nearly constant torque* throughout the motor cycle, unlike traditional Hall sensor commutation.

External sinusoidal commutation offers you these benefits:

- Low torque ripple
- Improved motor smoothness at low speeds
- More efficient drives, with less heat dissipation
- Potential reduction in system costs (because Hall sensors are not required)

3.0 Hardware Requirements

Sinusoidal commutation requires 2 controller axes per motor for the DSP series cards. In addition, only MEI boards beginning with the following revision numbers have the hardware to support sinusoidal commutation:

Controller	***Rev***
PCX/DSP	4
STD/DSP	8
104/DSP	6
LC/DSP	8
V6U/DSP	2

4.0 Installation

4.1 Wiring for Sinusoidal Commutation

Since 2 MEI axes are used for each motor, we recommend that the pair of axes used for commutation begin with the *even-numbered axis*. This will enable the even-numbered axis to drive Phase A, and the odd-numbered axis to drive Phase B. This scheme simplifies the wiring by enabling all of the necessary signals to originate from one header.

Motion should be commanded to the axis configured to drive Phase A. Beginning with the 2.5.05 release, the Phase B DAC may now be directed *to any other axis* using software commands.

Note	You must assign the Phase B DAC **a higher axis number** than the Phase A DAC (axis).

The following connector pinouts provide an example of encoder and drive connections for PCX/DSP, VME/DSP (both V6U and V3U), and STD/DSP controllers:

Pin	***Signal***
1	Gnd
2	5V
3	Encoder A+
4	Encoder A-
5	Encoder B+
6	Encoder B-
7	Index +
8	Index -
9	Phase A Analog Output
22	Phase B Analog Output

An example of connector pinouts for the 104/DSP and LC/DSP controllers are:

Pin	***Signal***
1	5V
2	Encoder A+
3	Encoder A-
4	Encoder B+
5	Encoder B-
6	Index +
7	Index -
8	Phase A Analog Output
9	Phase B Analog Output
22	Gnd

The DSP Series Controllers use single-ended analog outputs. Optimum noise rejection can be achieved by using shielded cables and adhering to single-point grounding practices.

4.2 Loading the Software

Copy the files from the distribution media to your hard drive. The following directory structure is suggested:

Directory	*Files*
c:\mei\sinecomm	Release notes (and subdirectories below)
c:\mei\sinecomm**firmware**	Firmware files and utilities to configure controller
c:\mei\sinecomm**samples**	Sample programs
c:\mei\sinecomm**utils**	Utility programs

4.3 Loading the Sinusoidal Commutation Firmware

The sinusoidal commutation **firmware must be downloaded** to the controller by using the *config.exe* program (or by using *Motion Console*). You can use the *config.exe* program to download firmware to the controller, configure the DAC offsets, and perform some basic tests of the axes.

Safety Note	Before running *config.exe*, **disconnect all of the cables** from the DSP Series controller and **turn off the power** to any external devices (amplifiers, etc.). When *config.exe* is executed, all **previous configurations** stored in the DSP's boot memory **will be lost**.

Note	Jogging, analog feedback, parallel feedback and I/O Monitoring are not supported in the standard commutation release. If you have an application that require these features with commutation, please contact MEI for more information.

1. To download the firmware, switch to the *mei\sinecomm\firmware* directory where *config.exe* is stored, and from the DOS prompt, execute:

   ```
   config -f 8axisc.abs
   ```

2. The *config.exe* program will first load the standard *8axis.abs* firmware and run tests in order to determine the number of axes on the card and the internal offsets of the axes. After the tests have been completed, *config.exe* will load *8axisc.abs* with the appropriate internal axis offsets.

   Note that the configured axis offsets are saved *into the firmware downloaded to the DSP Series board*, and are not saved to the firmware files on the hard drive. If there are any problems, the *config.exe* program will display error messages.

3. If your board is configured for a base address other than the default (0x300), you will need to set an environment variable called DSP to the desired address. For example, to set the DSP base address setting to 0x280, from the DOS prompt, execute:

   ```
   set DSP=base:0x280
   ```

   The *setup.exe* and *config.exe* programs will then automatically read the DSP variable and access the board at address 0x280. Note that you may also insert this command into the *autoexec.bat* file for automatic execution at system boot time.

5.0 Operation

Successful motor commutation requires initialization wherein the armature's field vector is determined from the feedback system. The feedback system and motion controller then track the field vector position for all subsequent moves. During these moves, the DSP Series controller calculates the 90 degree (stator) current phase advance required for closed loop operation, and its PID algorithm determines the magnitude of the current vector.

Two techniques are described in Section 5.2.1 for initial armature phase-finding. Before running the phase-finding software, you must determine the exact number of *encoder counts per electrical cycle*.

On encoder specification sheets, this term is often called the *number of encoder counts per revolution*.
On motor specification sheets, this term is often called the *number of electrical cycles (or pole-pairs) per revolution*.

Use the open loop utility programs (*counts.exe*, *cycles.exe*) to verify the parameters to be used in your initialization code.

5.1 Sinusoidal Commutation Developers Toolkit

The developer's toolkit includes 3 software tools:

srate.exe Sample rate modification program.

counts.exe Open loop commutation program. **Counts.exe** counts the number of encoder counts per electrical cycle, and also provides motor phasing information (ABC or ACB).

cycles.exe Open loop commutation program. **Cycles.exe** counts the number of electrical cycles per motor revolution, and also provides motor phasing information (ABC or ACB).

srate.exe - Sample Rate Modification for Commutation

Use *srate.exe* to change the sample rate of the card. At the DOS prompt, execute:

```
srate
```

Follow the prompts to change the sample rate of the card. If the card cannot attain the specified rate, *srate.exe* will set the sample rate to the highest possible value.

Operation of rotary motors at very high speed (>100 electrical cycles per second) and low sampling rates may provide too few points to properly define the drive's 3 sinusoidal electrical cycles. The result can be excess current use and degraded motor life (due to higher heat dissipation). MEI recommends at least 8 sample points per electrical cycle to maintain low motor current levels.

The DSP controller’s firmware incorporates a velocity-based *phase advance* which reduces sample-rate latency effects. If your application requires commutation rates greater than 150 electrical cycles per second, please contact MEI for technical support.

counts.exe - Open Loop Commutation Program

Use *counts.exe* to commutate a motor in open-loop mode, which is typically useful when initializing the motor as part of a startup sequence. *Counts.exe* requires 3 parameters:

- AXIS an integer specifying the axis to commutate (Phase A)
  and an integer specifying the Phase B axis.
- VOLTAGE a double precision value specifying the voltage (proportional to current) to apply during the open loop portion of the commutation.

Counts.exe will commutate the motor for one electrical cycle and report the *approximate* number of encoder counts per electrical cycle. Please note that this number is only an approximation. *Counts.exe* will also return motor phasing information.

For safety, *counts.exe* will leave the amplifier in a disabled state and will also zero the PID coefficients. After executing *counts.exe* and before running the commutation initialization program, **you must restore the PID coefficients**.

cycles.exe - Open Loop Commutation Program

Use *cycles.exe* to commutate a motor in open loop mode. *Cycles.exe* requires 4 parameters:

- AXIS -an integer specifying the axis to commutate (Phase A)
  -an integer specifying the Phase B axis.
- VOLTAGE -a double value specifying the voltage (proportional to current) to apply during the open loop portion of the commutation.
- COUNTS -an integer specifying the number of encoder counts per revolution in quadrature for the motor.

Cycles.exe will commutate the motor in open loop mode for the distance specified (as the number of encoder counts). When the commutation is completed, *cycles.exe* will display both the phasing of the motor relative to the encoder and the number of electrical cycles per revolution. You will need these values to initialize the closed loop commutation.

For safety, *cycles.exe* will leave the amplifier in a disabled state and will zero the PID coefficients.
After executing *cycles.exe* and before running the commutation initialization program, **you must restore the PID coefficients**.

5.2 Sinusoidal Commutation Development Software

The source code file *sincomm.c* and header file *sincomm.h* contain all of the code that you need to initialize a motor for commutation.

Section 5.2.1 describes 2 phase-finding techniques and their input parameters.
Section 5.2.2 describes the commutation parameters.
Section 5.2.3 describes sample programs (2 programs are listed in Section 6.0).
Section 5.2.4 describes the bidirectional limit switch safety feature, and how to implement it.

5.2.1 Closed-Loop Commutation Initialization Programs

MEI currently supports 2 commutation initialization methods in the 2.5.05 Release:

Stepper Mode Method: Set the magnetic vector, wait for motor to settle, then initialize.

Dither Mode Method: Dither the stator magnetic vector until rotor position is known, then initialize.

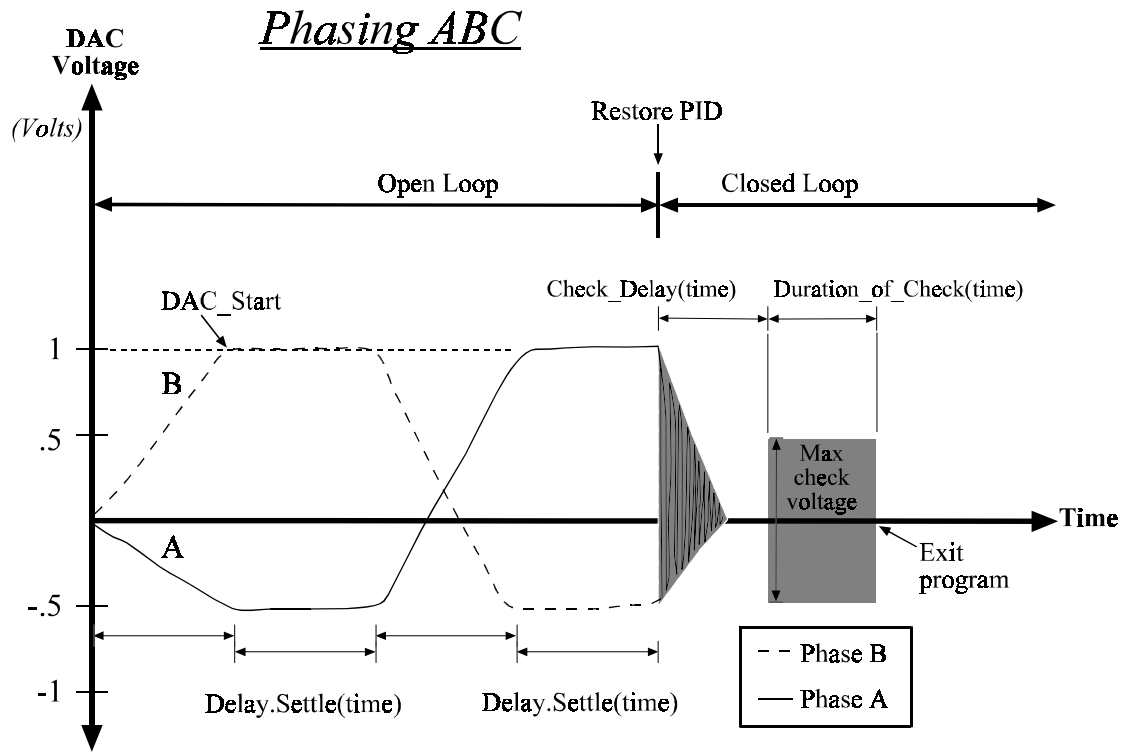
Note	Important! We recommend that you initially **use low DAC voltage levels** to protect the motor windings. Before you set the *phase-finding* open loop current parameters, you should verify the safe continuous current levels for your motor/drive combination. It is also good practice to install fuses for each of the 3 motor windings during this initial development period, to prevent damage to your motor.

Note	The **dsp_reset(...)** command will reconfigure the card from boot memory, and the commutation index (determined from the phase-finding sequence) will be lost. **Amplifiers should be turned off during a reset!** After the reset, the commutation phase initialization procedure must be implemented.

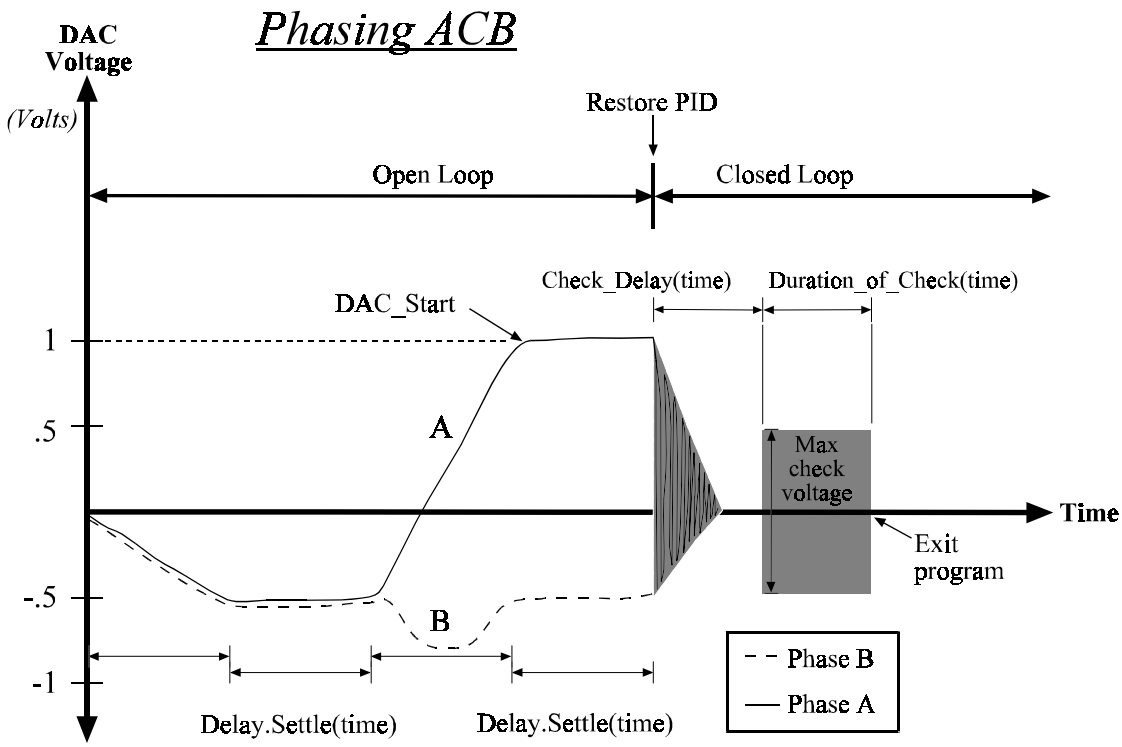
Stepper Mode Method (comm_init_set_vector):

Generally, the *Stepper Mode* initialization method is the preferred method for initialization. The *Stepper Mode* initialization method sets the stator magnetic vector and draws the armature vector into a known location within one electrical cycle. The rotor is initially pulled into a magnetic position 120 electrical degrees in advance of the "A" phase position. Next, the rotor is drawn back to the "A" phase position. This 2 position "stepper" technique is used to avoid initializing the rotor at a null position 180 electrical degrees from the assumed rotor position. Initialization occurs following a waiting period for the motor to settle. The following 2 diagrams illustrate the method.

Note	Initialization of large inertial loads with minimal friction (i.e., low damping coefficients) may require several minutes to settle (see parameter `delay.settle` in next figure).



This figure shows the A & B *phase voltage history* during *step phase-finding* for an ABC phasing configuration.



This figure shows the A & B *phase voltage history* during *step phase-finding* for an ACB phasing configuration.

Dither Mode Method (comm_init_dither):

Use the *Dither Mode* initialization method in situations where drawing the motor to a stator pole will cause excessive initial movement of the load. An example of this might be initializing near a limit switch. However, the *Dither Mode* method requires that you be more aware of the response of the system.

The *Dither Mode* initialization method locates the armature position by setting a known stator vector orientation, and then waits to see the direction of the initial armature acceleration. Based on the initial acceleration, the angular extent of the region containing the armature vector can be continually reduced by repeating this process. The goal of having small position change during the dithering cycle may require initial testing for optimum open loop voltage and time period for acceleration averaging (see additional descriptions in the upcoming commutation parameters table). Because force/torque induced acceleration is the critical measurement for successful dithering, systems subjected to external forces (i.e., gravitation force on vertical axes, large cable carrier forces, etc.) will not be suitable to use the *Dither Mode* initialization method.

5.2.2 Commutation Parameters

The required input parameters for the commutation data structure are listed below.

Input Parameters	
Axis	An integer specifying the axis to commutate. This corresponds to the "A" phase axis. (e.g., 0 - 7)
Phase_B_Axis	A second controller axis used for the “B” phase signal. (e.g., 1- 7) Note that the phase “B” axis number **must be greater than** the phase “A” axis number.
Counts_Per_Comm_Cycle	The number of encoder counts (in quadrature) per revolution of motor. For a linear motor, this would be the *number of counts per one electrical cycle*. For rotary motors, *counts_per_comm_cycle* does not necessarily correspond to the number of counts per electrical cycle. (e.g., 4096)
Electrical_Cycles	The integer number of electrical cycles (pole pairs) per motor revolution. The software will automatically account for systems requiring up to 5 electrical cycles, where the modulus of [*counts_per_comm_cycle*]/[*electrical_cycles*] is zero. Linear motors use the integer value, one. (e.g., 1 - 5)
Phasing	The motor/amplifier phasing may be ABC (where the A phase immediately leads the B phase) or ACB (A phasing immediately leading the C phase). Use macros: PHASING_ABC or PHASING_ACB.
DAC_Start	Is the open loop DAC output used during *Stepper Mode* phase-finding and also the initial voltage used during *Dither Mode* phase-finding. Verify safe continuous motor current levels BEFORE setting the *DAC_Start* level. (0 - 32767, Max. 32,767 corresponds to 10V. Typical value= 3000)
DAC_Step	Is the DAC voltage increment used during *Dither Mode* phase-finding, to step from the value *DAC_Start until* rotor movement is detected. (DAC output level will be limited below *DAC_Limit*). Note that *DAC_Step* is not used for *Stepper Mode* phase-finding. (0 -32767, typical= 500)
DAC_Limit	The maximum DAC output limit to be used during *Dither Mode* phase-finding. (0 - 32767, typical value= 6000)
Max_Check_Voltage	*Max_Check_Voltage* is a motor safety feature that checks for unsuccessful phasing finding. Following the phase-finding procedure and while servoing on position, the A and B phase DACs should be providing relatively small voltage outputs to the amplifier. If the absolute values of these voltage outputs to the amplifier exceed *Max_Check_Voltage*, then the software will disable the amplifier. (0 - 32767, Max. 32,767 corresponds to 10V. Typical value: 3000)
Check_Delay	Following the phase-finding procedure, *Check_Delay* is the delay period (in seconds) used for settling prior to checking the DAC outputs (using the *Max_Check_Voltage*). Due to the rollover of an *unsigned16*, *Check_Delay* must not exceed [65535/*dsp_sample_rate*]). (Suggested values: 1 - 5 seconds)
Duration_of_Check	Following the phase-finding procedure and settling time (*Check_Delay*), *Duration_of_Check* is the time period (in seconds) during which the DSP will verify that the A and B phase DACs do not exceed *Max_Check_Voltage*. Due to the roll-over of an *unsigned16*, *Duration_of_Check* must not exceed [65535/*dsp_sample_rate*]). (Suggested values: 1 - 5 seconds)
Delay.Settle	The time period (in seconds) used to wait for motion to settle during *Stepper Mode* initialization moves. Note that this value is used 4 times during the *Stepper Mode* initialization sequence. (Suggested values: 5 - 60 seconds)
Delay.Dither.Vel	The time period (in sample periods) over which velocity is averaged from position updates during *Dither Mode* initialization. (Suggested values: 5 - 10 samples)
Delay.Dither.Acc	The time period (in sample periods) over which acceleration is averaged from *Delay.Dither.Vel* updates in velocity. Like *Delay.Dither.Vel*, *Delay.Dither.Acc* is only used for *Dither Mode* initialization. (Suggested values: 5 - 10 samples)

5.2.3 Sample Closed-Loop Commutation Initialization Programs

To phase-find the rotor position, use one of the following 4 example programs (using your system's specifications):

- **step_abc.c** Sets the magnetic stator vector and draws the rotor (field vector) into a known position. Resets position and closes servo loop. The sequence uses ABC phasing.
- **step_acb.c** Does the same as **step_abc.c**, only with ACB phasing.
- **dith_abc.c** Open loop dithers stator vector while monitoring rotor position (acceleration). By successive approximation, it determines the phase position of the rotor. It also resets position and closes the servo loop. Uses ABC phasing.
- **dith_acb.c** Does the same as **dith_abc.c**, only with ACB phasing.

5.2.4 Bidirectional Limit Switch Protection Feature

The latest version of MEI commutation firmware offers bidirectional limit switch protection, both during open-loop phase-finding and during normal closed loop operation. Previous commutation firmware and standard DSP firmware (non-commutation) required that an activated limit switch must also be *in the direction of commanded velocity*, to generate a hardware limit event. (This simplifies limit recovery.)

In order to offer backwards-compatibility, the directional limit switch option is retained as the default configuration for the new firmware. A single function call to DSP external memory can configure any axis combination to have full limit switch protection, independent of the direction of command velocity. This ensures greater hardware safety, particularly during initial testing with external sinusoidal commutation. However, the full limit switch protection feature eliminated the I/O monitoring capability, and requires extra steps for limit switch recovery.

How to Configure Bidirectional Limit Switch Protection:

Write a configuration word to the address 0x36E8. The default state is 0. Bit 0 enables Axis 0, bit 1 enables Axis 1, and so on.

```
/* Configure Axes 0 - 7 for special limit switch protection  */
int16 bit_num;
bit_num = 0x00FF;
dsp_write_dm(0x36E8, bit_num)
```

How to Recover from a Limit Switch Event:

Option 1 Manually move the stage/motor from the active limit.

Option 2 Restore the default limit configuration [`dsp_write_dm(0x36E8, 0x0000)`], and implement an open-loop move away from the active limit.

6.0 Code Listings

6.1 step_abc.c

```
/* Step_ABC.C

:Sinusoidal commutation configuration and initialization.

Program uses Stepper Type motor initialization with ABC Phasing

Written for Version 2.5

Warning!  This is a sample program to assist in the integration of the
 DSP-Series controller with your application.  It may not contain all
 of the logic and safety features that your application requires.
*/


# include <stdio.h>
# include <stdlib.h>
# include "sincomm.h"

#define lmtaddr 0x36E8

void error(int16 error_code)
{
   char buffer[MAX_ERROR_LEN];

   switch (error_code)
   {
       case DSP_OK:
           /* No error, so we can ignore it. */
           break ;

       default:
           error_msg(error_code, buffer) ;
           fprintf(stderr, "ERROR: %s (%d).\n", buffer, error_code) ;
           exit(1);
           break;
   }
}

int main()
{
   Commutation_Init CInit;

   int16   error_code;
   int16   bit_num;          /* bit word to configure optional HW limit protection */

   error_code = do_dsp();    /* initialize communication with the controller */
   error(error_code);        /* any problems initializing? */

  /* standard limit configuration: HW protected only in direction of cmd vel*/
  //   bit_num = 0x0000;

  /*  configure Axes 0-7 for special limit switch protection in both
           directions without cmd velocity requirement */
           bit_num = 0x00FF;
           dsp_write_dm(lmtaddr, bit_num);

   /*
    *  Set up user-defined SinComm settings */
         CInit.Axis                        = 0;            /* Control & A phase axis */
         CInit.Phase_B_Axis                = 1;            /* B phase axis */
         CInit.Counts_Per_Comm_Cycle       = 4096;         /* Counts per rev */
         CInit.Electrical_Cycles           = 2;            /* Elec. cycles per rev */
         CInit.Phasing                     = PHASING_ABC;  /* Phasing */
         CInit.Dac_Start                   = 3277;         /* Stepper output value, 1.V */
         CInit.Dac_Step                    = 500;          /* For dithering only, 0.015V */
         CInit.Dac_Limit                   = 16000;        /* For dithering only, 5.V */
         CInit.Max_Check_Voltage           = 6553;         /* Safety check, 2 volts */
         CInit.Check_Delay                 = 2;            /* Delay prior to check, sec */
         CInit.Duration_of_Check           = 1;            /* DAC checking time, sec*/
         CInit.Delay.Settle                = 3;            /* Delay used 4 times, sec*/
```

```
   /*
    *  Configure axis for sinusoidal commutation.  Sinusoidal commutation requires
    *  two axes.  The first axis is Phase A, the second axis is
    *  Phase B.  Phase C is generated by the servo drive.  After sinusoidal
    *  commutation is initialized, all motion will be commanded to the
    *  Phase A axis.
    */

   error(comm_configure(&CInit));
   error(comm_init_set_vector(&CInit));

   error(set_position(CInit.Axis, 0.0));/* zero out position */

    return 0;
}
```

6.2 dith_abc.c

```
/* Dith_ABC.C

:Sinusoidal commutation configuration and initialization.

Program uses Dithering Type motor initialization with ABC Phasing

Warning!  This is a sample program to assist in the integration of the
 DSP-Series controller with your application.  It may not contain all
 of the logic and safety features that your application requires.

Written for Version 2.5
*/


# include <stdio.h>
# include <stdlib.h>
# include "sincomm.h"

#define lmtaddr 0X36E8

void error(int16 error_code)
{
   char buffer[MAX_ERROR_LEN];

   switch (error_code)
   {
       case DSP_OK:
           /* No error, so we can ignore it. */
           break ;

       default:
           error_msg(error_code, buffer) ;
           fprintf(stderr, "ERROR: %s (%d).\n", buffer, error_code) ;
           exit(1);
           break;
   }
}

int main()
{
   Commutation_Init CInit;

   int16   error_code;
   int16   bit_num;          /* bit word to configure optional HW limit protection */


   error_code = do_dsp();    /* initialize communication with the controller */
   error(error_code);        /* any problems initializing? */


  /* standard limit configuration: HW protected only in direction of cmd vel*/
  //   bit_num = 0x0000;
```

```
  /*  configure Axes 0-7 for special limit switch protection in both
          directions without cmd velocity requirement */
          bit_num = 0x00FF;
          dsp_write_dm(lmtaddr, bit_num);

   /*
    *  Set up user-defined SinComm settings
    */
         CInit.Axis                          = 0;            /* Control & A phase axis */
         CInit.Phase_B_Axis                  = 1;            /* B phase axis */
         CInit.Counts_Per_Comm_Cycle         = 4096;         /* Counts per rev */
         CInit.Electrical_Cycles             = 2;            /* Elec. cycles per rev */
         CInit.Phasing                       = PHASING_ABC;  /* Phasing */
         CInit.Dac_Start                     = 3277;         /* Stepper output value, 1.V */
         CInit.Dac_Step                      = 500;          /* For dithering only, .015V */
         CInit.Dac_Limit                     = 16000;        /* For dithering only, 5.V */
         CInit.Max_Check_Voltage             = 6553;         /* Safety check, 2 volts */
         CInit.Check_Delay                   = 2;            /* Delay prior to check, sec */
         CInit.Duration_of_Check             = 1;            /* DAC checking time, sec*/
         CInit.Delay.Settle                  = 3;            /* Delay used 4 times, sec*/

   /*
    *  Configure axis for sinusoidal commutation.  Sinusoidal commutation requires
    *  two axes.  The first axis is Phase A, the second axis is
    *  Phase B.  Phase C is generated by the servo drive.  After sinusoidal
    *  commutation is initialized, all motion will be commanded to the
    *  Phase A axis.
    */

   error(comm_configure(&CInit));
   error(comm_init_dither(&CInit));

   error(set_position(CInit.Axis, 0.0));           /* zero out position */

    return 0;
}
```